Semper 6

COMMAND

REFERENCE

Synoptics

Trademarks:

Semper is a registered trademark of *Synoptics Limited*
IBM PC is a trademark of International Business Machines Inc
Hewlett-Packard LaserJet+ is a registered trademark of Hewlett-Packard
Postscript is a registered trademark of Adobe Systems Incorporated
Microsoft is a registered trademark of Microsoft Corporation
Mouse Systems is a trademark of MSC Technologies, Inc
Siiicon Graphics Personal IRIS 4D is a trademark of Siiicon Graphics Inc
Sun is a registered trademark of Sun Microsystems Inc
VAX is a registered trademark of Digital Equipment Corporation

Manual number: SEM002
Update number: 0
Printed: August 1989

Table OF CONTENTS

PART 1

Chapter 1: Introduction

Chapter 2: Command Summary

Appendix E: Error Messages

Appendix F: Protected and Fixed Variables

Appendix G: Pixel Connectivity

Appendix H: Illumination

Appendix I: Customer Report Form

Appendix J. Ascii Key Codes

Chapter 1

INTRODUCTION

Overview

Semper 6 is a high-level language that has been developed to suit universal image processing applications. It consists of a large number of commands that can be used individually, or built into a program. It is *flexible* and provides the building blocks to construct your own individual solutions to your image processing problem.

About this manual

This manual provides a complete reference to Semper commands and is designed to complement the Semper on-line help.

How it's organised

The remainder of this manual is divided into two chapters and nine appendices:

- *Chapter 2: Command Summary* gives a list of Semper commands by function. This allows you to identify the group of commands that are required for a particular task.
- *Chapter 3: Semper 6 Commands* provides a detailed description of commands and command syntax. The commands are listed in alphabetic order.
- The appendices at the end of this manual include *Appendix E: Error Messages* which lists and explains Semper error messages.

This manual itself is split into two volumes:

- *Command Reference Part 1* contains introductory chapters and Semper commands *A* to *O*
- *Command Reference* Part 2 contains Semper commands *P* to *Z* and appendices.

What you need to know

This manual assumes a working knowledge of image processing terms and basic experience in the use of Semper 6 and Semper concepts, for example, *picture* and *device management*. If you are new to Semper 6, consult one of the manuals described in the last section of this chapter: *What else to read*.

Conventions used in this manual

Examples of Semper commands are shown in the following font:

```
semper
```

except when they are embedded in the text, in which case they are highlighted using a **bold** font.

This manual shows commands in their *unabbreviated* form. However, you can abbreviate all Semper commands (and parts of commands) to three characters. For example, the command:

```
pan cre pos 2,3
```

means the same to Semper as:

```
panel create position 2,3
```

and both these commands mean the same to Semper as:

```
pantaloons crease positively 2,3
```

In this guide each Semper command is described under its command heading, giving the following details:

- command syntax
- summary of command functions
- examples of use
- full description of command
- notes
- defaults and ranges

The sections below explain the format of the command syntax, notes, defaults and ranges.

Command syntax

Semper command syntax is divided into two parts:

- keys
- options

A *key* is added to a command to make its meaning more specific. Often a key limits the way in which a command works, for example, by applying the command action to a specified subregion only. A key *value* is required for each key.

An *option* does not require a value and is usually an instruction to alter the way in which a command works. For example, the **mask** command includes the options **inside/outside**. These options determine whether the command works inside or outside a mask boundary.

Each command syntax in this manual is given as in the **mask** command below:

mask

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	radius	*<number>*	radius of circular mask
	position	*<x>, <y>*	centre position of circular mask
	width	*<number>*	width parameter for 'soft edge' of circular mask
	with	*<number>*	*Plist* picture containing mask boundary curve
	value	*<n1>, <n2>*	reset pixels to specified value
	mark	*<number>*	mark circle, or polygon if ***with***
		<yes *or* no>	mark circle, or polygon if ***with***
options:	**outside/inside**		reset pixels outside or inside a mask

The text shown in angular brackets, such as *<number>*, describes the type of value to be supplied with a key. For example:

```
mask radius 25
```

is a valid use of the key **radius** with the **mask** command. You can also supply variables and expressions that give a numerical value. For example:

```
mask radius x+y
```

```
mask radius 3*z
```

Keys given in square brackets **[key]** can be omitted and are still understood by Semper. For example, the keys **from** and **to**, meaning *source picture* and *output picture* respectively, are used so often with Semper commands that they do not need to be stated explicitly:

```
mask 1 2
```

is the same as:

```
mask from 1 to 2
```

Sometimes you will only see square brackets [ ] that do not enclose a key. This means that you cannot supply a key, only a value or set of values. For example, the syntax :

calculate

keys:	[ ]	*<expression>*	*expression to be calculated*
	to	*<number>*	*output picture*

means that you must supply the text of an arithmetic expression for the **calculate** command. For example:

```
calculate (:10+:11)/2 to display
```

Note that Semper includes a number of keys and options that are called *general*, in that they are accepted by every command, for example, the options **byte**, **integer**, **fp** and **complex**. For details of these keys and options, refer to *Appendix C: Semper Keys and Options*.

Defaults and Ranges

The defaults and valid range of values for each command are given in a table at the end of each command description as is shown below:

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	source picture	valid picture number
radius	two thirds of the minimum source dimension	real number
position	centre of source picture	within bounds of picture (real numbers)
width	width 0	real number
with	circular mask	valid picture number
value	source picture average around mask boundary	real number
mark	mark off	see *Appendix C*
outside/inside	outside mask	

Defaults

The defaults for each **key** are given in the *defaults and ranges* table. The default of each **option** is not usually shown in the table as, in general, the default is off. A default is shown, however, if the default is *on*, for example, the default for **letter** and **border** is *on* and these are shown in the relevant

table. Note that to turn an option *off* you simply prefix it by **no**, for example, **noletter**, **noborder**, **noverify** etc. A default is also shown for an option if there is a choice, as in the **outside/inside** options in the above table for the **mask** command.

Ranges

The valid range of values that can be supplied with each key are also defined in the defaults and ranges table. Some ranges are not straightforward, for example, the **from** and **to** keys usually expect a *valid picture number*, which can be a simple integer:

```
mask from 42 to 43 inside
```

or can be a combination of device number and picture number separated by a colon:

```
mask 1:42 to 1:43 inside
```

which refers to pictures 42 and 43 on device 1. Alternatively, Semper allows you to refer to pictures 42 and 43 on device 1 as 1042 and 1043, so an equivalent command line is:

```
mask 1042 to 1043 inside
```

Note that many ranges are machine dependent. For example, some machines can only display a limited number of colours, although Semper can define many more. You may see the following range for a key:

integer in range 1 to system limits (type **show system**)

which means that the range is determined by your particular type of installation and the command **show system** provides details of your system limits.

Notes

After the description of each command, you will see a section called **Notes** which draws your attention to some features or restrictions of a command. An explanation of some of the notes is given below:

restrictions:

image sizes must be powers of two: the command requires that the size of the source picture is a power of two, for example, size 256 by 128.

factorisable size: the command requires that the size of a source picture can be factorised into factors of 2, 3, 4 and 5, with at least one factor 4.

unsuitable for direct output to display: the data generated by a command needs to be stored on file before it is sent to the display. This is because the display cannot store some forms of data accurately, for example, data with a range 0.1 to 0.5.

multi-layer pictures:

faulted: the command cannot accept multi-layer pictures.

layers processed independently: this means that each layer is processed as a separate 2-D image

forms used internally: *data form*

This note refers to the form used to process data, regardless of the form of the source data. The possible forms are *byte*, *integer*, *floating point* (*fp*) and *complex*. See *Appendix A: Picture Types* for further detail of picture forms.

display marking: *type of mark*

This note describes how a display is marked by the command if you use the **mark** key. For example, the **mask** command marks the mask boundary on the screen.

variables set: *variable name*

Like most languages, Semper makes use of variables to store values, for example, a variable called *select* holds the number of the current picture and a variable called *cd* holds the number of the current device. This note details any variables whose values are set by a particular command.

variables used: *variable name*

This note details any variables that are used by the command in its processing.

What else to read

For information on the Semper system, read the:

Semper 6 Guide

This guide is a collection of documents that detail each aspect of Semper. It includes the following sections:

- *Beginners' User Guide*
- *Advanced Users' Guide*
- *Quick Reference List*
- *User Interface Guide*
- *Tutor User Guide*

If you are new to Semper 6, first read the *Beginners' User Guide*. This document provides structured examples and illustrations of the Semper language. The *Advanced Users' Guide*

provides a more detailed and comprehensive description of Semper and its facilities. For an introduction to Semper 6 *Plus* user interface creation read the *Tutor User Guide* and the *Semper 6 Plus User Interface Guide*.

If you require a general introduction to the field of image processing, we recommend:

Digital Image Processing
by Rafael C. Gonzalez and Paul Wintz
published by Addison-Wesley, 1987
I.S.B.N. 0-201-11026-1

For an overview of *Remote Sensing* (Semper 6 contains a Remote Sensing package), we recommend:

Remote Sensing Digital Image Analysis: An Introduction
by John A. Richards
published by Springer-Verlag, 1986
I:S.B.N. 3-540-16007-8 Springer-Verlag Berlin Heidelberg New York
I.S.B.N. 0-387-16007-8 Springer-Verlag New York Heidelberg Berlin

On-line help

You can also refer to Semper on-line help for definitive information of a command. For example, for help on the **mask** command, start Semper and type the following command at the terminal:

```
help mask
```

For a description of **mask** syntax, type:

```
help mask.syntax
```

For the complete on-line help available for **mask**, type

```
help/full mask
```

For a list of all the commands and any other topics for which help is available, type:

```
help/topics
```

For further information, refer to the **help** command entry in this manual.

Chapter 2

COMMAND SUMMARY

Overview

This chapter gives a summary of the Semper commands that are detailed in this manual. The commands are grouped according to function, rather than alphabetically. This allows you to isolate the commands that are suitable for a particular task, for example, particle analysis. Figure 2-1 overleaf illustrates the functions of Semper commands.

This chapter also lists commands that are installation-specific, that is, commands that apply only to a particular hardware combination. These commands are mainly concerned with framestore control and image capture.

For a list of commands in *alphabetic* order and full details of each command and syntax, refer to *Chapter 3: Semper 6 Commands*.

A list of commands by function is given below.

Variables and terminal information

name=value	sets the named key/option/variable to a specified value
ask	asks for a value from the terminal with a user-defined prompt
beep/bell/buzz	sounds bell on the terminal
cls	clears the terminal screen and places the cursor at home position
echo	directs logical output streams to the terminal and/or log files
help	provides on-line documentation during a session
inkey	waits for a terminal key to be pressed and returns key value
local	declares a variable to be restored after program execution
log	prints text/values in the log file
page	controls formatting and paging of terminal output
p(ixel)	resets an individual pixel to a user-defined value
pointer	sets gearing and sensitivity of pointing device (ie. mouse)
report	prints message for last reported/trapped error
show	prints information on variables set, devices assigned, time etc.
syntax	prints keys and options for a given command
time	prints elapsed time in seconds
type	prints text/values on terminal
unset	deletes a variable

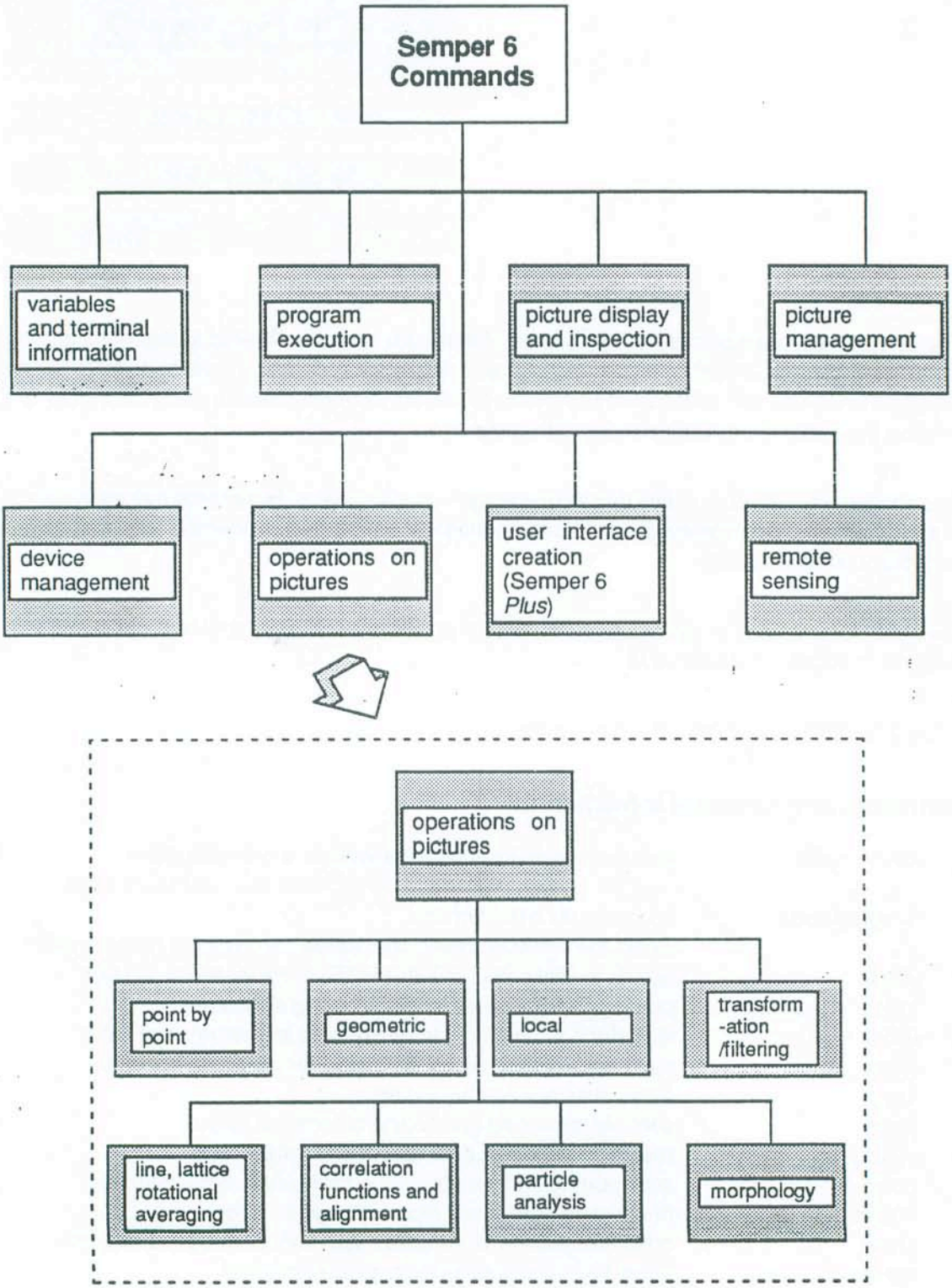


Figure 2-1. Semper 6 Commands By Function

Variables and terminal dialogue

name=val	Sets named key/option/variable to given value
ask	Seeks value(s) from terminal with user-defined prompt
beep	Sounds bell on terminal
bell	Sounds bell on terminal
buzzer	Sounds bell on terminal
cls	Clears terminal screen and places cursor at home position
diagnostic	Prints text (values to diagnostic output stream
echo	Directs logical output streams to terminal and/or log files
event	Provides access to Semper event queues
help	Provides on-line documentation during session
Inkey	Waits for terminal keys to be pressed and returns key values
local	Declares variables to be restored after program execution
log	Prints text/values to log output stream
page	Controls formatting and paging of terminal output
p(ixel)	Resets individual pixel(s) in picture to user-defined value(s)
pointer	Defines pointer (i.e. mouse) gearing and sensitivity
report	Prints message for last reported/trapped error
show	Prints information on variables set, devices assigned, time, etc.
syntax	Prints keys and options for given command
time	Prints elapsed time in seconds
type	Prints text/values to console output stream
unset	Unsets variable(s) permanently

Program execution

@name	Inserts named macro text in command line
@number	Inserts numbered macro text in command line (executes macro)
add	Adds new program(s) to program library
break	Resumes execution after end of **for** loop
edit	Edits contents of numbered macro
end	Last command in program or run file
exit	Halts session
for	Repeats subsequent commands (up to corresponding **loop**)
if, unless	Makes command execution conditional
jump	Resumes execution at given label
library	Executes program from program library
list	Prints program text or numbered macro text
loop	Terminating command **for** loop
macro	Causes rest of input line to be stored as numbered macro
next	Resumes execution at start of next cycle of **for** loop
order	Defines/prints search order for program libraries
quit	Halts session
rename	Renames program in program library
return	Returns from program or run file
run	Executes commands from named file (run file)
stop	Halts session
wait	Suspends execution until key pressed or specified time elapsed

Picture display/inspection

contour	Draws contour map on display (line graphical form)
display	Displays picture/region in various modes
drag	Drags line/arc/circle/sub-region/curve on the display with the cursor
examine	Prints picture size, class, data form, title, etc.
histogram	Generates/draws grey-level histogram of picture/region
ladjust	Provides mouse-controlled adjustment of look-up tables
lset	Generates/modifies look-up tables (colour scales, highlighting)
lut	Generates and manipulates monochrome/colour look-up tables
mark	Marks position(s), frame, circle or text on a display
ovread	Reads image stored in the overlay into a picture
ovwrite	Writes a binary image into the overlay plane
pfilter	Reduces the data in a position list picture after **sketch** operation
postscript	Writes picture/display region to file in PostScript format
print	Prints small picture region in numerical form
rgb	Converts a full colour image into monochrome/false colour form
sheet	Displays 2-D picture as height of solid sheet
solid	Displays thresholded 3-D picture as solid body
sketch	Records curve drawn free-hand on the display as a position list picture
spc	SuperPoses Contour levels on picture
survey	Scans picture/region and sets variables *min*, *max*, *mean*, *me2* and *sd*
view	Selects viewing field, zoom factor and (mono, false or colour) lut
xwires	Records position/direction/frame indicated via display cursor
ymod	Displays picture/region as surface height (line graphical form)

Miscellaneous

null	Default command, displays picture/region
pack	Prints packed integer values for given names
unpack	Prints three-character names for given packed integer values
user	Provides skeleton routine for user to adapt

Picture management

copy	Copies picture or program
create	Allocates/initialises picture storage (not normally needed)
delete	Deletes picture or program
letter	Generates lettering on a 5 by 6 pixel matrix
origin	Moves or resets picture coordinate origin
pcb	Stores picture coordinate range, class, data form in variables
picture	Returns picture parameters in variables
read	Reads picture from file outside Semper
reclass	Changes recorded picture class without altering data
renumber	Changes picture number without altering data
save	Saves picture in binary file outside Semper
select	Selects new current picture
title	Sets or alters picture titles
wp	Sets or unsets picture write-protect flag
write	Writes picture to file outside Semper

Device management

assign	Assigns memory, disc file, tape, display, log file, program library or help library as device
close	In some installations, outputs contents of display to hardware
compress	Compresses disc device, collecting empty space together
deassign	Deassigns device
directory	Prints directory state for disc device
erase	Erases image or overlay memory for display region
flush	Forces disc memory buffers to be copied to physical disc
partition	Assigns frames and positions for display partitions
ramps	Fills display region with repeated grey-level ramps
reinitialise	Re-initialises device, destroying any existing data
rewind	Rewinds tape device

Point by point operations

calculate	Calculates arbitrary arithmetical/relational/logical expression involving pixels, constants, variables and functions
correct	Forces picture modulus to supplied reference value
fit	Fits linear ramp to picture and subtracts/divides to remove ramp
gausslan	Generates single Gaussian-profile peak
lorentzlan	Generates single Lorentzian-profile peak
map	Processes picture with user-defined intensity mapping
mask	Resets pixels inside/outside circular or polygonal region
negate	Rescales picture interchanging *min* and *max*
noise	Generates (Gaussian or Poisson) noise-limited pictures
scale	Rescales picture, linearly or with histogram equalisation
threshold	Produce binary image given one or more intensity threshold

Geometrical operations

cut	Cuts region out of picture
expand	Horizontally resamples a picture
extract	Extracts sub-/super-region (magnified/rotated/skewed/warped)
find	Finds lowest/highest pixel or centre-of-mass of picture/region
fullplane	Converts half-plane Fourier picture to full-plane form
halfplane	Converts full-plane Fourier picture to half-plane form
lpd	Finds local maxima in a 1-D picture
magnify	Magnifies picture/region by integer factor
paste	Inserts one picture into another
peaks	Locates local peaks in picture and records positions in *Plist*
rotate	Rotates picture by large angles relatively efficiently
separate	Separates layers of multi-layer (3-D) picture as 2-D pictures
stack	Combines 2-D pictures as multi-layer (3-D) picture
transpose	Transposes picture (i.e. interchanges rows and columns)
turn	Performs simple picture reflections/rotations

Local operations

differentiate	Calculates picture derivatives via 3-point operator
edge	Applies magnitude or Roberts edge operator
fir	Applies arbitrary small block (FIR) filter
hp	Applies a square-block high-pass operator
lmean	Calculates local block mean (smoothes picture)
lsd	Calculates local standard deviation
lvariance	Calculates local block variance
rank	Applies local ranking (median, eroding or dilating) filter
rf	Applies two point recursive (IIR) filter for smoothing/sharpening
sharpen	Applies square-block edge-enhancing operator

Transformation/filtering

backproject	Back-projects 1-D picture into 2-D picture
ctf	Generates/applies electron-optical transfer function
fourier	Calculates Fourier transform (FTs)
hilbert	Calculates Hilbert transform
image	Re-calculates image from Fourier or Walsh transform
phr	Resets Fourier transform phases to random values
ps	Calculates a power spectrum (i.e. FT intensity)
walsh	Calculates Walsh transform
weight	Applies radial filter defined in 1-D picture
window	Applies window array filter (for lattice averaging)

Line, lattice and rotational averaging

base	Least-squares fits lattice parameters to list of lattice sites
flc	Fits Lattice Components to FT peaks (for lattice averaging)
lattice	Generates/marks positions of perfect lattice sites
motif	Calculates real-space average over given list of positions
project	Calculates 1-D projections or averages
section	Extracts radial sections averaged over arbitrary sectors
strain	Deduces local strain levels for distorted lattice

Correlation functions and alignment

acf	Calculates auto-correlation function
ocf	Calculates angular correlation function (for rotational alignment)
rcf	Calculates radial correlation function or phase residuals between two Fourier transforms
xcf	Calculates (spatial) cross-correlation function

Particle Analysis

analyse	Finds particles in picture according to given intensity thresholds
pcalculate	Calculates further parameters for given particle
pcurve	Calculates equivalent particle parameters for area inside curve
pdraw	Edits picture by drawing cursor-defined lines
pedit	Edits contents of particle parameter list and/or segmentation map
pextract	Extracts image of single particle from source picture
pferet	Calculates up to 9 feret diameters for given particle
phistogram	Generates histogram of given parameter for selected particles
pid	Returns particle id corresponding to given picture position
pmark	Marks selection of displayed particles
pset	Stores specified particle parameters in variables
pshow	Highlights selection of displayed particles
ptype	Prints parameters for selected particle(s)

Morphology

bclose	Closes regions in a binary image with a given structuring element
bdilate	Dilates regions in a binary image with a given structuring element
berode	Erodes regions in a binary image with a given structuring element
bhmt	Applies binary Hit or Miss transforms defined by structuring element
bmlut	Generates 3 x 3 neighbourhood mapping table for use with **bmmap**
bmmap	Applies 3 x 3 neighbourhood mapping to a binary image
bopen	Opens regions in a binary image with a given structuring element
chull	Generates the convex hull of the regions in a binary image
dclose	Closes regions in a binary image with a circular disc
ddilate	Dilates regions in a binary image with a circular disc
derode	Erodes regions in a binary image with a circular disc
dilate	Dilates simply or selectively, and enlarges objects without joining
dopen	Opens regions in a binary image with a circular disc
dt	Generates the distance transform for regions in a binary image
erode	Erodes simply or selectively, and generates outlines or skeletons
flood	Floods/marks regions in a binary image which overlap the seed image or seed points
hfill	Fills holes in regions in a binary image
label	Labels separate regions in a binary image
median	Smoothes binary picture
skiz	Generates the skeleton by zones of influence (skiz) of the regions in a binary or labelled image
zone	Generates the zones of influence of the regions in a binary or labelled image

Remote Sensing

box	Classifies picture using box or parallelepiped method
covariance	Calculates covariance of multi-spectral picture
destripe	Corrects picture for differences in line sensor characteristics
learn	Calculates statistics of training areas prior to classification
likelihood	Classifies picture using *maximum-likelihood* method
mindistance	Classifies picture using *minimum-distance-to-mean* method
pct	Calculates principal components transform of multi-spectral picture
rhistogram	Calculates 2-D histogram between bands of multi-spectral picture
warp	Corrects picture for geometric distortions

User Interface Commands

cell	Controls creation and behaviour of cell element
device	Returns limits of display, redraws screen, stacks cursor positions
execute	Defines actions that are executed before/after each Semper command sequence
justification	Controls positioning of displayed objects
menu	Controls creation and behaviour of menu element
mouse	Defines actions/positioning for mouse (pointing device)
panel	Controls creation and behaviour of panels
textfield	Controls creation and behaviour of textfields
uif	Controls loading, saving and execution of user interfaces

For further information please contact:

Synoptics Limited
271 Cambridge Science Park
Milton Road
Cambridge
CB4 4WE, UK
Tel: (0223) 423223. Telex:81417 INNCEN G.
Fax: (0223) 420020
Int: +44 223 423223.
Int Fax +44 223 420020

Synoptics Limited
164 Chief Justice Cushing Highway # 5
Cohasset
MA 02025
U.S.A
Tel: 0101 617 383 2289
Fax: 0101 617 383 2727

Chapter 3

SEMPER 6

COMMANDS

a

acf
add
analyse
ask
assign

b

backproject
base
bclose
bdilate
bell, beep, buzz
berode
bhmt
bmlut
bmmap
bopen
box
break

c

calculate
cell
chull
close
cls
compress
contour
copy
correct
covariance
create
ctf
cut

d

dclose
ddilate
deassign
delete
derode
destripe
device
diagnostic
differentiate
dilate
directory
display
dopen
drag
dt

e

echo
edge
edit
erase
erode
event
examine
execute
exit
expand
extract

f

find
fir
fit
flc
flood
flush
for
fourier
fullplane

g

gaussian

h

halfplane
help
hfil
hilbert
histogram
hp

i

if, unless
image
inkey

j

jump
justification

l

label
ladjust
lattice
learn
letter
library
likelihood
list
lmean
local
log
loop
lorentzian
lpd
lsd
lset
lut
lvariance

m

macro
magnify
map
mark
mask
median
menu
mindistance
monitor
motif
mouse

n

negate
next
noise

o

ocf
order
origin
ovread
ovwrite

p

pack
page
panel
partition
paste
pcalculate
pcb
pct
pcurve
pdraw
peaks
pedit
pextract
pferet
pfilter
phistogram
phr
picture
pid
p (pixel)
pmark
postscript
print
project
ps
pset
pshow
ptype

q

quit

r

ramps
rank
rcf
read
reclass
reinitialise
rename
renumber
report
return
rewind
rf
rgb
rhistogram
rotate
run

s

save
scale
section
select
separate
sharpen
sheet
show
sketch
skiz
solid
spc
stack
stop
strain
survey
syntax

t

textfield
threshold
time
title
transpose
turn
type

u

uif
unpack
unset
user

v

view

w

wait
waish
warp
weight
window
wp
write

x

xcf
xwires

y

ymod

z

zone

acf

keys:	[from]	<*number*>	source picture
	[to]	<*number*>	output picture

acf calculates *auto-correlation functions* (*Patterson* functions). An *acf* is the inverse transform of the intensity in the Fourier transform of a picture. It can be used as a resolution or directionality criterion, or may help reveal faint periodicities.

Examples

```
acf 1 2
acf from 1 to 2
```

These commands produce the *acf* of *Image* picture 1 in *Correlation* picture 2.

```
fourier 1 2; acf to 3
```

This command obtains the *acf* of picture 1 in picture 3, leaving the transform in 2.

Description

The command **acf** performs the following sequence of functions:

- transforms the source picture
- takes the squared modulus
- zeroes the central transform pixel
- inverse transforms
- divides by the final central value

The resulting *acf*, which has a zero mean and a maximum of 1 at the centre, is filed as a *Correlation* picture. You can also supply pictures that are already transformed (*Fourier* or *Spectrum*), in which case **acf** simply picks up the calculation at the appropriate point in the above sequence, so as to calculate the *acf* of the original image picture.

For real source data (that is if the source is a non-*Complex* image or a half-plane *Fourier* or *Spectrum*), the output is floating point and centro-symmetric. For complex source data (complex *Image* or full-plane *Fourier* or *Spectrum*), the output is complex and conjugate-symmetric. For details of picture classes such as *Fourier* and *Spectrum*, refer to *Appendix A: Picture Types*.

acf

Notes

restrictions:	image size must be a power of two (for example, size 128,128) unsuitable for direct output to display
multi-layer pictures:	faulted
forms used internally:	fp, complex

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	source picture	valid picture number

add

keys:	**name**	*'<text>'*	add text from the specified file
	macro	*<number>*	add text from the specified macro
	program	*'<text>'*	program name to be used if adding from a macro, single program from a file or interactive input
	device	*<number>*	device number of program library to be updated
options:	**again**		if **name** is omitted and was used in a command to open a file for reading, open the same file again
	verify		verify the command processing on the console

Use **add** to add a program to a program library (or replace it with a new version, if it already exists). You can also use **add** to convert a numbered macro into a program.

Examples

```
add
```

This command prompts at the terminal for a program name and then accepts a single program terminated by **end**.

```
add program 'quick'
```

This command accepts a single program from the terminal and names it quick.

```
add name 'newprogs'
```

This command reads one or more programs in turn from the file *newprogs.spl*.

```
add name 'newprogs' device 3 noverify
```

This command specifies the device number of the program library to which the program is to be added. By default, Semper uses the first writeable library that it finds in the current search order. The **noverify** option suppresses information about the process at the console.

```
add name 'programs' program 'work'
```

This command reads only the program named *work* from the file *programs.spl*.

```
add macro 2:12 program 'mac12'
```

This converts the macro in picture 12 on device 2 into a program called *mac12* on the first writeable program library that is found.

add

Description

The **add** command adds program text to the end of the program library. It deletes any older version of a given program, but the space is not released immediately. Note that repeated replacement of a program leads to the error messages; *No free directory slots on device no or insufficient space on deivce n.* Use the Semper command **compress,** to recover the full space on your device.

By specifying both the **name** and **program** keys you can include a single program from a file of programs. This is useful for replacing or recovering a single program without having to reload all of the programs in the file.

The **again** option allows you to re-open a file without having to specify the file name again with the **name** key. If the **name** key was not used before to open a file for reading, the **add** command will prompt for the file name.

Notes

restrictions:	**add** cannot be used within a program
see also:	**compress, end**

Defaults and Ranges

keys/options	defaults	range
name	*none*	valid filename
size	*none*	valid picture number
program	*none*	valid program name
device	first library in current search order	integer in range 1 to system limits (type ***show system***)
verify	verification on	

analyse

keys:	[from]	<*number*>	source picture
	[to]	<*number*>	output picture (particle parameter list)
	ge	<*number*>	upper intensity threshold for particles
	le	<*number*>	lower intensity threshold for particles
	size	<*x*>, <*y*>	dimensions of subregion to be analysed
	position	<*x*>, <*y*>	position/offset of subregion
	area	<*n1*>, <*n2*>	minimum and maximum area for retained particles
	segment	<*number*>	output picture for segmented image
	mark	<*number*>	mark particle positions with symbols
		<yes *or* no>	
	mkmode	<*number*>	mark mode
	mksize	<*number*>	mark size
options:	left/right, top/bottom		position of subregion
	id		mark particle with identifier instead of symbol
	cm		mark centre of particle area instead of reference point
	verify		verify results at the console

Semper's facilities for particle analysis (object analysis, object feature measurement etc.) are centred on the **analyse** command. Briefly, you choose intensity thresholds to delineate particles, use **analyse** to count and measure them, and other commands to type, display or process the measured values. The **analyse** command scans a picture or subregion, identifying and measuring all connected regions satisfying the threshold conditions that you specify, and records the resulting *ppl* (*particle parameter list*) as a *Plist* picture. You then use other commands such as **ptype** to print, display or process the results.

Examples

```
analyse ge 100; phistogram area
```

This command counts and measures particles in the current picture with pixel values 100 or more, storing the results as *ppl* 998, after which the **phistogram** command displays a histogram of the area measurements

```
analyse 50 to 51 le 6.35 segment 52
```

The **analyse** command counts and measures particles with pixel values 6.35 or less, storing the results as *ppl* 51, and creating a segmented version 52 of the original picture 50 for use by later commands such as **pshow**.

analyse

```
analyse 1 ge 0 size 400 area 10,50 mark display id
```

This command analyses positive particles in picture 1, considering only particles whose centre of area lies within a central region 400 points square, ignoring particles with areas less than 10 and greater than 50. It marks identifiers for each particle on the current display.

```
analyse 3 ge 100 mark mkmode 2 mksize 3
```

This command analyses particles in picture 3 with pixel values of 100 or more, marking each particle with a diagonal cross (mark mode 2) with a symbol size of 3 pixels (mark size 3).

```
analyse 1 size 300 position 100, 50 noverify
```

This command analyses particles in picture 1 with intensity values which are greater than or equal to the middle of the picture range. Only particles that lie within a region 300 points square centred at the position *x*=100, *y*=50 are considered. The **noverify** option suppresses the listing of results at the console.

Description

The command **analyse** records 25 parameters for each particle found in a particle parameter list (*ppl*). These parameters are described in *Appendix D: Particle Parameters*.

Particles are connected regions of pixels (see *Appendix G, Pixel Connectivity*) with values that satisfy the threshold conditions specified by the minimum and maximum keys **ge** and **le**. If you omit both keys, a default value equal to the middle of the source picture range is assumed for the key **ge.**

The largest particle identifier is equal to the number of particles found. The **analyse** command also sets the variable **n** to repeat the number of particles found.

Particles that touch the picture border may be truncated, and lead to misleading measurements (for example, suggesting some undersized particles in a picture of entirely identical spheres). You can overcome this problem by specifying a subregion for analysis using the **size** key. Using this key, particles whose centres of area lie outside the subregion are then ignored, but other particles are recorded correctly even if part of their area lies outside the subregion. The **position** key allows you to specify the exact position of the subregion on the display. Alternatively, you can use the options **top, right,** etc. to indicate the position of the subregion. Refer to *Appendix C, Semper Keys* and *Options* for further details of the standard subregion keys and options.

If you want to exclude particles less than or greater than a certain area (to eliminate large numbers of spurious measurements as a result of noise in the image), use the key **area** to indicate the minimum and maximum area of interest to you.

If you supply a suitable output picture number via the key **segment**, a version of the source picture is produced in which background pixels (belonging to no particle) are set to zero, and pixels belonging to each particle are set to the corresponding particle identifier (1,2...) assigned by **analyse**.

If you indicate a display with the key **mark**, **analyse** marks the reference point of each particle found in the style/size specified by **mkmode/mksize**. For detail of the keys **mark**, **mkmode** and **mksize**

analyse

refer to *Appendix C, Semper Keys and Options.* If you use option **id**, the particle identifier assigned is marked instead, and if you use **cm**, the mark is placed at the centre of area rather than at the reference point.

The **verify** option displays information the **analyse** command at the console. Use the **noverify** option to suppress this information.

Note that **analyse** creates a small temporary picture to start with, and replaces it with successively larger ones as the analysis progresses. This means that a small number of particles requires little picture disc space but that a large number of particles requires much more space. **analyse** also creates a second, fixed-size temporary picture. If you see one of the following error messages when using **analyse**; *Insufficient free space on device n* or *Disc fragmented – can't open n*, try assigning a larger scratch disc or try **compress**ing the disc, to release free space.

Further information

The following Semper commands can be used following the **analyse** command. For a full description of each command refer to the information given under its command heading in this manual.

The following commands can be used to type, display or recover the measured values in the *ppl* (particle parameter list) produced by **analyse**:

phistogram	generates a histogram based on one of the particle parameters, for a selection of particles
pmark	marks selected particles or parameter values on the display
pset	returns selected particle parameter values in variables
ptype	types selected particle parameter values

If you specify the **segment** option with **analyse**, Semper generates a segmented version of the scanned picture, in which all pixels belonging to a given particle are set to a particle identifier (1,2,3...). This information is used by the following commands:

pcalculate	calculates further parameters for a single particle
pedit	edits contents of particle parameter list and/or segmented picture
pextract	generates picture containing image of a single particle
pferet	calculates up to 9 feret diameters for a single particle
pid	identifies the particle containing a given pixel
pshow	highlights selected particles on the display

analyse

Notes

display marking:	particle reference points or centres	
multi-layer pictures:	layer 1 processed only	
forms used internally:	fp, integer	
variables set:	*pimage*	(source picture analysed)
	pplist	(output *ppl* number)
	psegment	(segmented output picture, if any)
	n	(number of particles found)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	998	valid picture number
ge	middle of source picture range	real number
le	*none*	real number
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0,0	within bounds of the picture (integers)
area	0, picture area	within bounds of the picture (real numbers)
segment	no segmented picture	positive integer
mark	mark off	see *Appendix C*
mkmode	mark mode 1	integer in range 1 to 5
mksize	mark size 2	positive integer
verify	verification on	

acquisition

This syntax is specific to Sprynt systems, using the Synapse as the image grabber

keys:	**time**	select recursive filter time constant
options:	**on**	turn acquisition on
	off	turn acquisition off
	enquire	set variables **acamera**(2) to the dimensions of the acquisition scan

The **acquisition** command allows you to control the Synapse framestore.

Examples

```
acquisition on time 2
```

This command turns on acquisition and selects filter time constant 2.

```
acquisition off
```

This command turns off acquisition.

```
acquisition enquire
```

This command sets variables **acamera**(2) to the dimensions of the acquisition scan.

Description

The **acquisition** command provides a facility for controlling the Synapse framestore. Acquisition can be turned on or off, the recursive filter time constant selected and information on the size of the acquisition scan obtained.

The key **time** selects the time constant for recursive filter. Value 0 corresponds to no recursive filtering; value 1,2 and 3 correspond to time constants of 0.125, 0.5 and 1.0 seconds for the CCIR video standard and 0.10, 0.42 and 0.83 seconds for RS–170.

acquisition

Defaults and Ranges

keys/options	defaults	range
time	*none*	integer in range 0 to 3
on	*no*	
off	*no*	
enquire	*no*	

ask

<variable name(s)> *'<text string>' <variable name(s)>*	***ask*** uses a special syntax. The text in the text string is typed out, as a prompt; at the terminal and the variable name(s) specify the values to be asked for at the terminal.

Use **ask** in macros or programs to accept values from the terminal. You can also specify a prompt string.

Examples

```
ask radius
```

This command asks for the value of the variable *radius* at the terminal.

```
ask 'Centre, radius: ' x, y, radius
```

This command prompts at the terminal for the values of *x*, *y* and *radius*, prompting with the text string *'Centre, radius:'*.

```
ask 'Defocus for picture ',n defocus
```

This command prompts for the defocus value for picture *n* and places the typed value in the variable called *defocus*.

```
n=yes; ask 'Remove background? (return/no) ' n; if n=yes remove...
```

This command prompts at the terminal for an answer to the question *'Remove background?'* and then directs program flow accordingly.

Description

You can specify a prompt with **ask** and mix expression values with the text, as in the third command example. Use a space (not a comma) to mark the end of the prompt. If you do not specify a prompt, as in the first command example, Semper uses the variable name(s) as a prompt.

If you are asked for more than one variable, as in the second example, enter the appropriate number of values separated by commas. If you omit some or all of the values, the corresponding variables are unchanged. The last example shows how you can use this feature to supply default values.

Notes

see also: **type**

assign

keys:	**name**	*'<text>'*	name of file or tape to be assigned
	size	*<number>*	size of new disc file or memory in kilobytes
	slots	*<number>*	number of directory slots in new picture storage device, program library or help library
	device	*<number>*	new device number
	width	*<number>*	maximum record length for log file
options:	**scratch/memory/ display/tape/file**		type of device to assign (the default is to assign a permanent disc file)
	scratch		assign a temporary disc file which is automatically deleted when deassigned
	memory		assign a memory-based storage device
	display		assign the display device (see **assign display** for precise details)
	tape		mount and assign a magnetic tape
	file		assign a log file
	program/help		assign program or help library instead of picture storage device
	old/new		required device status
	old		if permanent disc file, open the file if tape, mount the tape to read existing data if log file, reuse an existing file
	new		if permanent disc file or log file, create a new file if tape, mount the tape and ignore any existing data
	wp		make the device write-protected (read-only)
	append		position log file so that output is appended
	verify		verify assignment at the console

Use the **assign** command to provide access to the various kinds of data storage devices supported by Semper: picture storage devices, program libraries, help libraries and log files. The physical storage which can be associated with a device depends on the type of data being stored and is one of the following: memory, file (permanent or temporary), magnetic tape and a display or framestore. Permanent files are created or located with a file name which you must specify. The file name may be a fully specified path name or the name of a file which appears on Semper's file search path. Temporary files are created with a file name generated by Semper and they are deleted automatically at the end of a Semper session. The display device, as well as storing picture data, can display images on a monitor and it may also allow you to capture images from a camera. Each

assign

device you assign is given a unique device number which you use subsequently to refer to the device. Use the command **show devices** to list the devices currently assigned and the **deassign** command to deassign them.

Examples

```
assign name 'v6disc'
```

This command assigns an existing picture file *v6disc* to the first free device number.

```
assign name 'emsa.bak' new size 5000

assign new size 5000
File/tape name (as textstring):'emsa.bak'
```

These identical commands assign a new file called *emsa.bak*, with 5000 kilobytes of available disc space. Note that if you do not specify a filename, Semper prompts for a name to be entered at the terminal. The filename must be enclosed in quotes.

```
assign memory size 3000
```

This command assigns 3 megabytes of memory for storing pictures.

```
assign display
```

This command assigns the display device (currently device number 1 is always assigned for the display).

```
assign scratch size 2000
```

This command assigns a 2 megabyte temporary disc file which will be deleted using **deassign**.

```
assign file name 'results.out'
```

This command assigns the log file called *results.out* to log processing results.

```
assign help name 'semper'
```

This assigns Semper's help library called *semper.hlb*.

```
assign program name 'myfile.lib'
```

This command assigns a program library called *myfile.lib*.

assign

```
assign wp name 'replace' device 8
```

This command assigns a write-protected file called *replace.dsk* as device 8.

```
assign new tape name 'mrc132'
```

This command mounts/assigns a blank tape with the serial name *mrc132*.

Description

The **assign** command assigns a unique device number which you subsequently use to refer to the device. The device number is returned in the variable *n*. You can specify the device number to be assigned (provided it is not already in use) with the **device** key. Failing that, **assign** chooses the lowest free device number. Note that the display device is always assigned with device number 1. There is a limit to the number of devices. The command **show system** will list how many devices Semper can support. Use the command **show devices** to list the details of all the devices currently assigned. To deassign or close the device, use the **deassign** command. You specify which device to deassign with the **device** key (you may use the **display** option instead to deassign the display device). All devices are automatically deassigned at the end of a Semper session.

Semper supports four different kinds of data storage devices according to the type of data being stored:

picture storage device	–	Semper pictures
program library	–	Semper programs
help library	–	on-line help
log files	–	ASCII text file (output only)

A picture storage device (or picture library/archive) can store up to 999 Semper pictures of arbitrary size (within the limits set by the capacity of the device and the limits imposed by Semper itself). Pictures are primarily used to encode image data but they can also be used to store other kinds of data like display look-up tables, histograms, position lists, particle parameter lists, etc. Use the **examine** command to list the contents of a picture storage device and the commands **create, copy, renumber, delete, directory** and **compress** to manage its contents. When you refer to Semper pictures you must specify the device number either explicity (for example, **5:233, 2001, n:4**, etc.) or implicitly when the picture resides on the current picture storage device (for example, **:12, :pic, 100**). The variable *cd* returns the device number for the current picture storage device (so the previous examples are interpreted as **cd:12, cd:pic, cd:100**).

A program library contains the text for Semper programs together with some extra information to speed up the invocation of programs and the exectuion of **for** loops and **jump** commands. You invoke a program with the **library** command by giving the name of the program. Unlike Semper pictures, you do not have to specify a device number. If programs with the same name appear in different program libraries, the program library search order determines which one is invoked. Use the command **order** to set or modify the program library search order. Use the command **show programs** to list the contents of a program library and the commands **add, list, copy, rename, delete, directory** and **compress** to manage its contents.

assign

A help library contains the text for Semper's interactive **help** command. This allows you to obtain detailed and up-to-date information about all of Semper's facilities and commands. Help libraries are created and managed by a separate program **helpman** which you invoke outside of a Semper session.

You use log files to record textual data generated by Semper. There are six categories of output text: console, log, diagnostic, monitor, command and input text, any combination of which can be output to a log file. You use the **echo** command to specify what kind of text is to be output to a log file. The command **show echo** lists the current echo settings.

The physical storage associated with each type of data storage device can take a number of different forms:

picture device	–	memory, permanent or temporary binary file, display, tape
program library	–	memory, permanent or temporary binary file
help library	–	memory, permanent or temporary binary file
log files	–	permanent text file

The data stored in a memory-based device is permanently stored in memory. This provides the most efficient form of access to data, provided that your system can support large memory arrays (this is not the case for PC systems based purely on MS-DOS).

Files provide the most important form of data storage. They can reside on any physical medium which can be accessed via your machine's file system using the normal file naming conventions (principally hard disc, but could also be floppy disc, networked files, etc.). Permanent files (in the sense that they are intended to have a visible presence in the file system) must be given a file name. Temporary files are created with a Semper-generated name and they are automatically deleted when deassigned.

The display device, at the same time as functioning as a picture storage device (possibly providing access as efficient as with a memory-based device), can display and manipulate picture data. The resolution of the storage is limited to 8 bits or less per pixel, so there is the facility to scale data values into the range supported by the display memory. When data is read back the inverse scaling transformation is applied so that the data is returned scaled to the original source data range, but with possible loss of precision. The display device also supports the overlaying of graphical data on top of an image, cursor driven data input and the manipulation of look-up tables. You use the following commands to control these facilities:

general	–	**erase, partition, view**
look-up tables	–	**ladjust, lset, lut, pshow**
interactive input	–	**draw, pdraw, sketch, xwires**
display output	–	**display, ramps, rgb**
graphical i/o	–	**contour, mark, ovread, ovwrite, ymod**

assign

Display look-up tables define a mapping from stored data values to displayed intensities and colours. The mapping does not change the stored data and can usually be applied instantaneously. If the display device is a framestore, it will also allow you to capture images with a camera, with the camera image displayed live on the monitor. Once a camera image has been captured it can be accessed like any other display picture.

Some Semper systems support direct tape access for the retrieval of picture data from tape archives. Magnetic tapes are intrinsically serial in nature: writing data at a given position renders data further down the tape inaccessible. Picture numbers correspond directly to the relative position of the picture on the tape (picture 3 is the third recorded picture, etc.) but only the **copy** command is allowed to write to the tape. The **rewind** command initiates a rewind without waiting for the operation to complete.

The **assign** command can be used with the following combinations of keys and options:

assign old permanent file	**assign [old] [program/help] name '.....' [wp]**
assign new permanent file	**assign new [program/help] name '.....' size ... [slots ...]**
assign temporary file	**assign scratch [program/help] size ... [slots ...]**
assign memory	**assign memory [program/help] size ... [slots ...]**
assign display	**assign display [..... extra keys/options .....]**
assign log file	**assign file name '.....' [new/old] [width ...] [append]**
assign magnetic tape	**assign tape name '.....' [new/old] [wp]**

In the first four examples above, the default is to assign a picture storage device. If however you specify the option **program** or **help**, a program library or help library is assigned instead.

In the absence of the options **scratch, memory, display, tape** or **file,** the **assign** command assigns a permanent file. If the option **old** is specified, an existing file is assigned. If the option **new** is specified a new file is created. If during an interactive session a newly created file would replace an existing file, you will be asked for confirmation. Option **old** is assumed if neither **old** or **new** is specified.

If the **scratch** option is given, a temporary file is assigned. The file will be created in the current directory unless one of the following enviornment variables is set to a valid directory pathname:

semper$temp, tempdir, tempfiles, temp

in which case, the file will be created in the named directory. If more than one of these environment variables is set, the first one in the list which is set is used. Semper generates a unique 8 character file name beginning with "zzzz" and followed by the appropriate file extension. The contents of a temporary file are deleted automatically when the file is deassigned. On some systems, the file's directory entry is deleted as soon as the file is created, so the file will not appear in the file system but space is still allocated to the file. If Semper terminates abnormally, the operating system will automatically free this space.

assign

The **memory** option allows you to create memory-based devices where all the data is stored in memory. As long as your system can support large memory arrays, this approach provides the most efficient way to store and process data because it avoids the delays inherent in disc i/o. For commands which require more random access to data, storing all the data in memory can bring very significant performance improvements over and above the improvements obtained from Semper's disc caching scheme.

You can achieve similar results for accessing data in existing disc files with the **cache memory** command. This reads the entire contents of a file (which you have previously assigned as a Semper device) into a memory array, converting the device into a memory-based device. The data is written back to disc when the **flush, cache free** or **deassign** commands are used on the same device.

If the option **display** is specified, the **assign** command assigns the display device and returns the frame size in variables *fsize* and *fs2*, the monitor size in variables *msize* and *ms2* and the number of frames in variable *nframe*. You may find that your version of the **assign display** command accepts additional keys or options, controlling details of the hardware configuration, such as the number of display frames to be made available to you. Consult the documentaton for your installation for more details.

If the option **file** or **tape** is specified instead, a log file or tape is assigned.

When assigning a permanent file, magnetic tape or log file, a file or tape name must be given. This can be specified with the **name** key, but if you omit the key and you are running an interactive session, Semper will prompt for the file or tape name, for example,

```
assign
file name (as textstring): 'my_disc'
```

where "my_disc" is the file name (the character string must be enclosed in quotes).

Default file name extensions are provided according to the type of device being assigned:

picture file	*.dsk*
program library	*.plb*
help library	*.hlb*
log file	*.log*

If no file extension is given, the default extension is used. If you are assigning an existing file, and a file with the default extension does not exist, a further search for the file name without the extension is made. When creating a new file, the default file extension is always used. When referring to an existing file you can specify just a file name instead of a fully specified path name, in which case the file must lie somewhere on Semper's file search path to be found. The current directory is always the first directory in the search path. Use the command **show path** to list Semper's file search path. When creating a new file, you should give a fully specified pathname unless you want the file to be created in the current directory. The command **show devices** always lists the full path name.

assign

When creating a new permanent file, a temporary file or a memory-based device, you must specify a data storage size in kilobytes with the **size** key. The total size of the device also includes space for a header block and a directory, so the overall size of the device will be larger than the size you specify with the **size** key. You can control the size of the directory with the **slots** key. For picture storage devices, the default directory size is the maximum possible (each slot requires 8 bytes and the absolute maximum number of slots required is 2002, giving a maximum size of 16KB). Here you would use the **slots** key to create a picture device with a smaller directory. Each program in a program library takes up one slot of 64 bytes and the default is to create a program libary with 64 slots. Directory entries in a help library vary in size, but the slot size is assumed to be 64 bytes, with a default directory size of 400 slots (a total size of 25KB).

When assigning files or magnetic tapes, you can specify the **wp** option to protect all the data from accidental deletion or overwriting, irrespective of any other protection status. In multi-user installations the **wp** option is usually required when assigning files belonging to other users (for which you may not have write access privileges). It also makes it possible for more than one Semper session to share access to the same file. The **wp** option is ignored when option **new, scratch, memory** or **file** is used.

You use the **file** option to open/create log files for recording text output by Semper. The **echo** command controls what type of text is output to each file. When a file is assigned, all output to that file is disabled. Use the command **show echo** to list the current echo settings for each file. A file, when it is opened, is positioned at the beginning, unless option **append** is used, in which case it will be position at the end and text output to the file will be added to any text already in the file. Text is truncated to the output width for the file. This is specified by the **width** key and defaults to 132.

The operation of the **assign** command is confirmed on the console. You can suppress this output by setting the option **noverify.**

Notes

variables set:	*n*	(number of assigned device)
	fsize, fs2	(display frame size)
	nframe	(number of display frames)
	msize, ms2	(display monitor size)
see also: general:		**deassign, programs, scratch, show compress, copy, create, delete, directory, examine, renumber, flush**
programs:		**add, compress, copy, delete, for, jump, library, list, order, rename, show**
help:		**help**
memory:		**cache**
display:		**contour, display, drag, erase, ladjust, lset, lut, mark, ovread, ovwrite, partition, pdraw, pshow, ramps, rgb, sketch, view, xwires, ymod**
log file:		**echo, show**
tape:		**copy, rewind**

assign

Defaults and Ranges

keys/options	defaults	range
name	*none*, prompts for filename if interactive	valid file name
size	*none*	positive integer
slots	maximum possible for a picture device, 64 slots for a program library 400 slots for a help libary	positive integer
device	first free device number	integer in range 2 to system limits (type **show system**)
width	132	integer in range 1 to 200
scratch/memory/ display/tape/file	assign permanent disc file	
program/help	assign picture storage device	
old/new	if a permanent disc file or tape, **old** is assumed if a log file and the file does not exist, create it, othewise, output an error message	
wp	if existing help library, write-protection on otherwise, write-protection off	
append	position file at the beginning	
verify	verification on	

assign display

This syntax is specific to...
Silicon Graphics workstations with a GL display window

keys:	**size**	*<x>, <y>*	frame size in pixels
	frames	*<n>*	number of frames
	position	*<x>, <y>*	initial position of the bottom left-hand corner of the display window with respect to the bottom left-hand corner of the screen
options:	**display**		assign the display device

You use the **assign display** command to create a display window for displaying images. The display window emulates the functioning of a framestore: storing image data in the framestore causes the image to be displayed on the screen. The colours and intensities used to display images are determined by a display look-up table. There is also support for a number of overlays for displaying graphical data, a cursor and rubberband lines on top of images. The size of the framestore is specified with the **size** and **frames** keys. Use the command **deassign display** to destroy the display window.

Examples

```
assign display
```

This command creates a false colour display window with the default size of 768 by 512 pixels.

```
assign display size 400 position 0,0
```

This command creates a display window with size 400 by 400 pixels, positioned at the bottom left-hand corner of the screen.

```
assign display frames 3
```

This command creates a display window with 3 frames for displaying full colour images (if the graphics hardware supports this).

Description

The **assign display** command assigns the display device to emulate a framestore with a frame size given by the **size** key and the number of frames given by the **frames** key. If the **size** key is omitted, the frame size defaults to 768 by 512 pixels. If the Y dimension is omitted, it defaults to the X dimension. The frame size may not exceed the dimensions of the screen. The display window may be repositioned and resized at any time. If the frame size exceeds the current size of the display window, the bottom left-hand corner of the frame is displayed at the bottom left-hand corner of the window. The initial position of the display window may be specified with the **position** key (the default is for the window to be manually positioned).

assign display

Image data associated with each display frame and overlay data are separately stored in memory buffers so that the information can always be accessed, regardless of the visibility of the displayed image. The image data is stored with a maximum resolution of 8 bits (256 levels): the actual resolution depends on the graphics hardware installed in your workstation. If the hardware supports more than 8 bit-planes, the full 256 levels of image data can be stored and displayed. With 8-bit displays, 192 levels can be stored and displayed. The number of display levels (which is equal to the look-up table size) is listed by the **show system** command and returned in the variable *lsize* by the command **lut enquire**.

The **frames** key determines the number of display frames. The only acceptable values are 1 (the default) and 3. You can display monochrome and false colour images with a single frame. In order to display full colour images (RGB pictures with three layers), you must assign three display frames. The graphics hardware must support 24 bit-planes for this to succeed. Full colour look-up tables are supported by repainting the the display window whenever changes are made to the current look-up table (the variable *clut* records the current look-up table number). You will also have to create a partition with three frames in order to store and display RGB images, for example,

```
assign display frames 3
partition 1 frames 1,3
display 2:1 to dis:1
```

The resolution for each component of a full colour image (the number of display levels and the look-up table size) is the same as for a false colour image.

A total of eight overlays are supported. You can control the visibility and colour of overlays with the **overlay** command. The overlays are numbered from 1 to 8 and where overlay data overlaps, the highest numbered overlay is displayed. Cursor and rubberbanding functions are carried out using one overlay each. Different overlays must be used for displaying graphics, cursor and rubberband output. The display window is repainted whenever an overlay is turned on or off and when the colour of an overlay for a full colour display changes.

Interactive input is possible when the display window has input focus. During commands which track the pointer (for example, **ladjust**), the pointer is repositioned at the centre of the display window whenever it reaches the edges of the window. This allows you to generate continuous pointer movement regardless of the constraints imposed by the display window.

Notes

see also: **deassign, ladjust, lut, overlay, show**

assign display

Defaults and Ranges

keys/options	defaults	range
size	768, 512	any size not exceeding the screen size
frames	1	1 or 3
position	0, 0	any position not exceeding the limits of the screen

assign display

This syntax is specific to...
workstations with an X Windows display window

keys:	**size**	*<x>, <y>*	frame size in pixels
	frames	*<n>*	number of frames
	position	*<x>, <y>*	initial position of the top left-hand corner of the display window with respect to the top left-hand corner of the screen
options:	**display**		assign the display device

You use the **assign display** command to create a display window for displaying images. The display window emulates the functioning of a framestore: storing image data in the framestore causes the image to be displayed on the screen. The colours and intensities used to display images are determined by a display look-up table. There is also support for a number of overlays for displaying graphical data, a cursor and rubberband lines on top of images. The size of the framestore is specified with the **size** and **frames** keys. Use the command **deassign display** to destroy the display window.

Examples

```
assign display
```

This command creates a false colour display window with the default size of 768 by 512 pixels.

```
assign display size 400 position 0,0
```

This command creates a display window with size 400 by 400 pixels, positioned at the top left-hand corner of the screen.

```
assign display frames 3
```

This command creates a display window with 3 frames for displaying full colour images (if the X server supports this).

Description

The **assign display** command assigns the display device to emulate a framestore with a frame size given by the **size** key and the number of frames given by the **frames** key. If the **size** key is omitted, the frame size defaults to 768 by 512 pixels. If the Y dimension is omitted, it defaults to the X dimension. The largest frame size allowed is 16384 by 16384 pixels. The initial size of the display window is limited to three-quarters of the screen dimensions. The display window may be repositioned at any time and resized within the limits imposed by the screen and the window manager. The initial position of the display window may be specified with the **position** key (the default is to position the window at the top left-hand corner of the screen). If the window manager allows it, the window will be positioned as specified. Failing that, the window will have to be manually positioned.

assign display

If the display window is not large enough to display the entire frame, the view position can be altered with the **view** command. The view position is the frame position which appears at the centre of the display window. The default for the view position is the centre of the frame. The **xwires** command also provides the facility (enabled by pressing the *V* key) to make the view position track the movement of the cursor. This allows you to roam freely around a very large frame.

Image data associated with each display frame and overlay data are separately stored in memory buffers so that the information can always be accessed, regardless of the visibility of the displayed image. The image data is stored with a maximum resolution of 8 bits (256 levels): the actual resolution depends on the depth of any PseudoColor and GrayScale visuals supported by the X server (use the X command *xdpyinfo* to list the supported visuals). The minimum suitable depth is 6 bits. The PseudoColor visual with the greatest depth is used in preference to the GrayScale visual with the greatest depth. If no suitable visuals are supported, the **assign** command will fail. If the depth of the selected visual is greater than 8 bits, the full 256 levels of image data can be stored and displayed. The number of display levels (which is equal to the look-up table size) is listed by the **show system** command and returned in the variable *lsize* by the command **lut enquire**.

The **frames** key determines the number of display frames. The only acceptable values are 1 (the default) and 3. You can display monochrome and false colour images with a single frame. In order to display full colour images (RGB pictures with three layers), you must assign three display frames. The X server must support a 24-bit TrueColor visual for this to succeed. Full colour look-up tables are supported by repainting the the display window whenever changes are made to the current look-up table (the variable *clut* records the current look-up table number). You will also have to create a partition with three frames in order to store and display RGB images, for example,

```
assign display frames 3
partition 1 frames 1,3
display 2:1 to dis:1
```

The resolution for each component of a full colour image (the number of display levels and the look-up table size) is the same as for a false colour image.

A total of eight overlays are supported. You can control the visibility and colour of overlays with the **overlay** command. The overlays are numbered from 1 to 8 and where overlay data overlaps, the highest numbered overlay is displayed. Cursor and rubberbanding functions are carried out using one overlay each. Different overlays must be used for displaying graphics, cursor and rubberband output. The display window is repainted whenever an overlay is turned on or off and when the colour of an overlay for a full colour display changes.

Interactive input is possible when the display window (or any other window created by Semper) has input focus. If during a command which tracks the pointer (for example, **ladjust**), the pointer leaves the window, no further tracking will take place until it rejoins the window. Since pointer movement is handled incrementally, you can obtain more movement than the window allows by circling the pointer around the outside of the window and dragging it across the window in the same direction as before.

assign display

When image or graphics data is ouput by a Semper command, it is immediately stored in the memory buffers associated with the display window. By default, the display window is updated less often than the memory buffers in order to reduce the overhead incurred in communicating with the X server. The rate at which the display window is updated can be controlled with the **x11** command.

Notes

see also: **deassign, ladjust, lut, overlay, show, view, x11**

Defaults and Ranges

keys/options	defaults	range
size	768, 512	any size up to 16384 by 16384
frames	1	1 or 3
position	0, 0	any position not exceeding the limits of the screen

assign...display

This command is specific to...
PC + framestore/greystore
Silicon Graphics IRIS 4D workstation
Sun 3 workstation

This syntax is specific to...
PC + Data Translation DT2861 framestore

keys:	**iobase**	*<number>*	inform Semper of the framestore's i/o port base address
	membase	*<number>*	inform Semper of the framestore's memory base address
options:	**ntsc/cclr**		inform Semper of the video standard (USA or European)

This syntax is specific to...
PC + Imaging Technology PCVISIONplus framestore

keys:	**iobase**	*<number>*	inform Semper of the framestore's i/o port base address
		<n1>, <n2>, <n3>	if full colour, inform Semper of the i/o base address for each frame
	membase	*<number>*	inform Semper of the framestore's memory base address
	rate	*<number>*	inform Semper of the framestore's sampling rate
	frame	*<number>*	inform Semper of the number of frames that are present
options:	**ntsc/cclr**		inform Semper of the video standard (USA or European)

This syntax is specific to...
PC + Matrox PIP512/PIP1024 framestore

keys:	**iobase**	*<number>*	inform Semper of the framestore's i/o port base address
	vsyn	*<number>*	set the vertical position of the framestore display, relative to the monitor frame
	hsyn	*<number>*	set the horizontal position of the framestore display, relative to the monitor frame
options:	**ntsc/cclr**		set USA or European video standard

assign...display

This syntax is specific to...
PC + MRC500 framestore
PC + Synoptics Synergy framestore

keys:	**iobase**	*<number>*	inform Semper of the framestore's i/o port base address
options:	**ntsc/ccir**		set USA or European video standard

This syntax is specific to...
PC + Synoptics Synapse framestore

keys:	**iobase**	*<number>*	inform Semper of the framestore's i/o port base address
		<n1>, <n2>, <n3>	if full colour, inform Semper of the i/o base address for each frame
	frame	*<number>*	inform Semper of the number of frames present
options:	**ntsc/ccir**		set USA or European video standard

These keys and options are specific to...
PC + Quantimet 520 greystore

keys:	**iobase**	*<number>*	inform Semper of the greystore's i/o port base address
options:	**ntsc/ccir**		set USA or European video standard
	tv1/tv2/int		set the Quantimet 520 sync standard to serrated, block or internal sync

These keys and options are specific to...
PC + Metrabyte Corporation MV1 framestore

keys:	**iobase**	*<number>*	inform Semper of the framestore's i/o port base address
	frame	*<number>*	inform Semper of the number of frames present
options:	**ntsc/ccir**		inform Semper of the video standard (USA or European)

assign...display

This syntax is specific to...
Silicon Graphics IRIS 4D workstation

keys:	**size**	*<n1>,<n2>*	define the size of the window to be used for the display in pixels
	position	*<n1>,<n2>*	define the position of the top left of the window to be used for display in pixels relative to the top left of the screen
	frames	*<number>*	number of frames to be used for the display (1 or 3 frames)

This syntax is specific to...
Sun 3 workstation

keys:	**size**	*<n1>,<n2>*	define the size of the window to be used for the display in pixels
	position	*<n1>,<n2>*	define the position of the top left of the window to be used for display in pixels relative to the top left of the screen

Use the **assign...display** command to inform Semper of the configuration of your framestore or workstation and to configure any aspects of the framestore or workstation that are software selectable.

Examples

PC + Data Translation DT2861 framestore only

```
assign display iobase 528 ntsc
```

This command informs Semper that the i/o base address is 528 (210 hexadecimal) and that the framestore uses USA video standard giving a monitor size of 480 lines by 512 pixels.

```
assign display membase 168
```

This command informs Semper that the memory base address is168*64k (A80000 hexadecimal).

---

assign...display

PC + Imaging Technology PCVISIONplus framestore only

```
assign display iobase 640 membase 12 rate 12.5 ntsc frames 1
```

This command informs Semper that:

- the i/o base address is 640 (280 hexadecimal)
- the memory base address is 12*64k (C0000 hexadecimal)
- the sampling rate is 12.5 MHz
- the video standard is the USA standard
- the number of frames is 1

PC + *Matrox PIP512/PIP1024* framestore only

```
assign display iobase 128 vsyn -2 ntsc
```

This command informs Semper that the i/o base address offset is 128 (80 hexadecimal) and moves the framestore display two units downwards on the monitor (relative to the default position). The **ntsc** option sets the framestore video standard to the USA standard.

PC + MRC500 framestore and Synoptics Synergy framestore only

```
assign display iobase 640 ntsc
```

This command informs Semper that the i/o base address is 640 (280 hexadecimal) and sets the framestore to the USA video standard.

assign...display

PC + Synoptics Synapse framestore only

```
assign display iobase 704 ntsc frames 1
```

This command:

- informs Semper that the i/o base address is 704 (2C0 hexadecimal)
- sets the framestore video standard to USA standard
- informs Semper that the number of frames is 1

PC + Quantimet 520 greystore only

```
assign display iobase 704 ntsc tv1
```

This command informs Semper that the greystore i/o base address is 704 (2C0 hexadecimal). It causes the greystore to use the USA video standard and the Quantimet 520 to use the TV1 sync standard.

PC + Metrabyte Corporation MV1 framestore only

```
assign display iobase 640 ntsc frames 1
```

This command informs Semper that:

- the i/o base address is 640 (280 hexadecimal)
- the video standard is the USA standard
- the number of frames is 1 (single framestore)

Silicon Graphics IRIS 4D workstation only

```
assign display size 512, 512 position 20, 20 frames 3
```

This command assigns a window of size 512 by 512 pixels as a full colour display, with the top left of the window at 20, 20 on the screen.

assign...display

Sun 3 workstation only

```
assign display size 512, 512 position 20, 20
```

This command assigns a window of size 512 by 512 pixels as a false colour display, with the top left of the window at 20, 20 on the screen.

Description

The following description is applicable only to *PC + framestore/greystore* systems. If you have a *Sun* or *Silicon Graphics* workstation refer to the separate sections that follow this description.

The **assign...display** command is principally used to inform Semper of the configuration of your framestore. On some framestores part or all of the configuration is software selectable and it allows you to select the particular configuration you require.

Use the **iobase** key to inform Semper of the i/o port base address of your framestore/greystore. By default, Semper uses the standard factory-set base address.

Use the options **ntsc/ccir** to inform Semper of and/or set the video standard used by your framestore:

- **ntsc** is the USA video standard (525 lines, 60 Hz field – typically 450 lines are visible)
- **ccir** is the European standard (625 lines, 50Hz)

Unless you specify a video standard, Semper examines the DOS country code which is usually set in the CONFIG.SYS file. If it is 1 (North America) or 81 (Japan) Semper assumes **ntsc**, otherwise it assumes **ccir**.

PC + Data Translation DT2861 only

The additional option **membase** allows you to inform Semper of the memory base address of your framestore. This is useful if the default address of the framestore conflicts with another address used by your PC.

assign...display

PC + Imaging Technology PCVISIONplus framestore only

The PCVISIONplus framestore can be used in a number of different configurations. Use the **frame** key to specify the type of configuration in use, that is, the number of frames that are present:

- 1 frame if single framestore in the single store memory configuration
- 2 frames if single framestore in the dual store memory configuration
- 3 frames if true colour system (3 framestores) and each framestore is in the single store memory configuration
- 6 frames if true colour system (3 framestores) and each framestore is in the dual store memory configuration

You need 3 framestores to display full colour images. Semper uses the three framestores as a red, green and blue board. These drive the red, green and blue guns respectively of a colour monitor. Semper numbers the frames from 1 to 3 from the red to the blue boards for the single memory configuration. For the dual store memory configuration the frames are numbered from 1 to 3 from the red to blue boards using the left hand frames and from 4 to 6 from the red to blue boards using the right hand frames.

The three boards should be located at three different i/o base addresses but should share a common memory base address. Use the **iobase** key to inform Semper of the i/o base addresses, for example, **iobase 832, 768, 864**. Use the **membase** key to inform Semper of the memory block base address. You can specify the framestore sampling rate using the **rate** key (10 MHz or 12.5 MHz).

PC + Matrox PIP512/PIP1024 framestore only

The Matrox version of the **assign** command has the additional keys **vsyn** and **hsyn** which enables you to move the position of the framestore display relative to the monitor frame. Specify the horizontal and vertical movement in integer units. (Each unit corresponds to 8 pixels).

assign...display

PC + Synoptics Synapse framestore only

The Synapse framestore can be used in a number of different configurations. Use the **frame** key to inform Semper of the type of configuration, that is, the number of frames that are present:

- 1 frame if single framestore
- 3 frames if true colour system (3 framestores)

Note that for the full colour system the **iobase** key requires three values corresponding to the i/o base addresses of the red, green and blue boards respectively.

PC + Quantimet 520 greystore only

The Quantimet 520 greystore has the additional options **tv1/tv2/Int** that allow you to set the video standard of the Quantimet 520 to serrated, block or internal sync.

PC + Metrabyte Corporation MV1 framestore only

You can use the Metrabyte MV1 framestore in a number of different configurations. Use the **frame** key to describe the type of configuration in use, that is the number of frames that are present:

- 1 frame if single framestore
- 3 frames if true colour system (3 framestores)

You need three framestores to display full colour images. Semper treats the three framestores as red, green and blue boards and numbers them as frames 1 to 3. They drive the red, green and blue guns respectively, of a colour monitor.

Note that the three boards in a full colour system should be located at the same i/o base address but with board select jumpers DB0 to DB2 set to correspond to frames 1 to 3.

assign...display

Silicon Graphics IRIS 4D workstation only

Use the **assign...display** command to:

- assign the size of a display window in pixels (**size** key)
- define the position of the window on the display (**position** key). Note that if you do not specify a position, you must place the window interactively.
- define the number of frames (1 or 3) to be used for the display (**frames** key). Note that your hardware must have at least 24 bit plane graphics if you are to assign 3 frames.

Sun 3 workstation only

Use the **assign...display** command:

- to define the size of a display window in pixels (**size** key)
- to define the position of the window on the display (**position** key)

Notes

see also:	**assign**	
variables set:	*fsize, fs2*	(display frame size)
	nframe	(number of display frames)
	msize, ms2	(display monitor size)

assign...display

Defaults and Ranges

PC + Data Translation DT2861 framestore only

keys/options	defaults	range
iobase	560 decimal (230 hexadecimal)	hexadecimal value in range 210 to 3F0, in steps of 20 hex
membase	160*64k (A00000 hex)	hexadecimal value in range 10 to F8, in steps of 08 hex
ntsc/ccir	uses DOS country code	

PC + Imaging Technology PCVISIONplus framestore only

keys/options	defaults	range
iobase	if single framestore: 768 decimal (300 hex) if full colour: 832, 768, 864 decimal (340, 300, 360 hex)	hexadecimal value in range 100 to 3F0, in steps of 10 hex
membase	10 (A0000 hex)	9, 10, 12, or 13 (9, A, C or D hex)
rate	10 MHz sampling rate	10 or 12.5
frame	2 (single framestore, dual-store memory configuration)	1,2,3 or 6
ntsc/ccir	uses DOS country code	

PC + Matrox PIP512/PIP1024 framestore only

keys/options	defaults	range
iobase	0 decimal	0 or 80 hexadecimal
vsyn	0	integer
hsyn	0	integer
ntsc/ccir	uses DOS country code	

assign...display

PC + MRC500 framestore and Synoptics Synergy framestore only

keys/options	defaults	range
iobase	736 decimal (2E0 hex)	hexadecimal value in range 100 to 3F0, in steps of 8 hex
ntsc/ccir	uses DOS country code	

PC + Synoptics Synapse framestore only

keys/options	defaults	range
iobase	if single framestore: 640 decimal (280 hexadecimal) if full colour: 640, 704, 832 decimal (280, 2C0, 340 hexadecimal)	hexadecimal value in range 100 to 3F0, in steps of 40 hex
frame	1 (single framestore)	1 or 3
ntsc/ccir	uses DOS country code	

PC + Quantimet 520 greystore only

keys/options	defaults	range
iobase	832 decimal (340 hexadecimal)	hexadecimal value in range 100 to 3F0, in steps of 40 hex
ntsc/ccir	uses DOS country code	
tv1/tv2/int	*none*; you must specify a TV standard	

assign...display

PC + Metrabyte Corporation MV1 framestore only

keys/options	defaults	range
iobase	768 decimal (300 hexadecimal)	hexadecimal value in range 100 to 3F0, in steps of 10 hex
frame	1 (single framestore)	1 or 3
ntsc/ccir	uses DOS country code	

Silicon Graphics IRIS 4D workstation only

keys/options	defaults	range
size	768, 512	positive integer
position	*none*	positive integer
frames	1	1 or 3

Sun 3 workstation only

keys/options	defaults	range
size	768, 512	positive integers
position	window centred on the screen in *x*, with the top of the window 20 pixels from the top of the screen	positive integers

backproject

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	with	*<number>*	1-D picture to be back-projected
	angle	*<number>*	back-projection direction in radians, anti-clockwise from the positive *x* axis
options:	**[add]/multiply**		add the back-projected data to the source or multiply the data onto the source

The **backproject** command alters a source picture by sweeping a 1-D picture across it. It adds or multiplies pixels from the 1-D picture to the source picture. **backproject** performs the opposite of a projection operation (see **project**).

Examples

```
backproject 1 with 10 angle pi/4
```

This command sweeps the 1-D picture 10 across picture 1 from bottom left to top right, adding pixels to picture 1 as it does so.

```
backproject multiply 1 with 10
```

This command backprojects the 1-D picture 10 'on its end' horizontally across picture 1, multiplying pixels rather than adding.

Description

The key **with** is used with the command **backproject** to specify the 1-D picture to be backprojected. Choose one of the following operating options:

- **backproject add**
- **backproject multiply**

backproject add is the default (so there is no need to type **add** in the command). This adds the projection picture to the source. **backproject multiply** multiplies the projection onto the source.

Use the **angle** key to specify the projection in radians. The angle is measured anti-clockwise from the positive *x* axis. The projection is positioned so that the source picture origin projects into its own coordinate origin. Source pixels projecting outside the 1-D picture are treated as though they are projected onto the nearer end of the picture.

backproject

Notes

display marking:	projection direction and width of output
multi-layer pictures:	faulted
forms used internally:	complex
see also:	**project**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture number, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none*	valid picture number
angle	angle 0	real number in range 0 to 2π
[add]/multiply	add the pixels to the source	

base

keys:	**[from]**	<*number*>	*Plist* picture
	[to]	<*number*>	output picture
	position	<*x*>, <*y*>	centre of region to which fit is restricted
	radius	<*number*>	radius of region to which fit is restricted
	numbers	<*n1*>, <*n2*>	serial number of positions to which fit is restricted
	tolerance	<*number*>	fractional deviation from initial (estimated) lattice sites beyond which sites are excluded from fit
	mark	<*number*> <yes *or* no>	mark the final deviations of all positions from the lattice on the display
	times	<*number*>	if ***mark***, magnification of display marking
options:	**uvonly**		fit lattice base vectors *u*, *u2* and *v*, *v2* only, preserving lattice origin *w*, *w2*
	verify		verify details of fitting process at the console

Use the **base** command to fit a lattice to a list of positions in a *Plist* picture. This command refines initial estimates that you provide of the lattice base vectors *u* and *v* and the lattice origin *w*, using a least-squares criterion and reports the quality of the resulting fit. For further detail of *Plist* pictures, refer to *Appendix A: Picture Types*.

Note that the commands **peaks** and **xwires list** produce a list of positions that can be used by **base** to fit a lattice.

Examples

```
u=10,0 v=0,10 w=0,0; base 51
```

This command fits a lattice, roughly 10 pixels square, to sites listed in picture 51.

```
base 51 radius 100 position w,w2 mark display
```

This command fits positions within 100 pixels of the lattice origin *w*, and marks the final deviations on the display.

```
base numbers 100,150 uvonly verify
```

This command fits positions 100 to 150 only, with the lattice origin fixed at the picture origin, printing

base

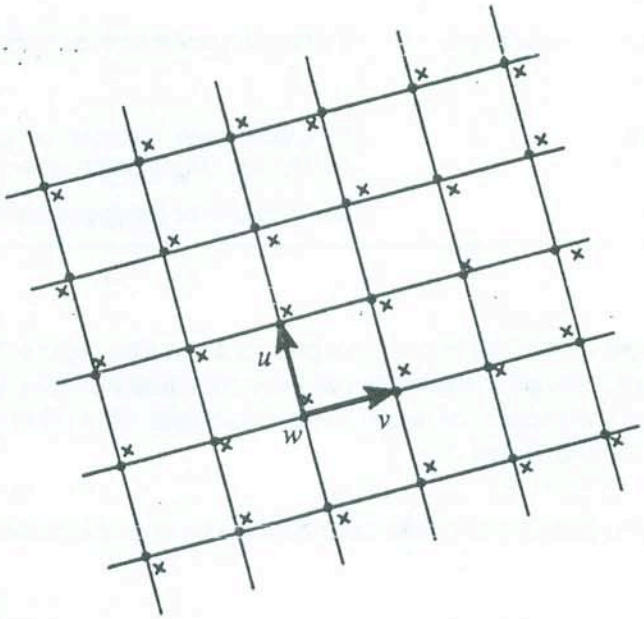
details of the fitting process at the console.

```
base 51 to 52 tolerance .1
```

This command produces a new *Plist* 52 containing only those positions from 51 that lie within 0.1 lattice vectors of a fitted lattice site.

Description

base refines the values that you supply for the base vectors *u* and *v* and the lattice origin *w*. If you can supply an accurate value for the lattice origin, *w*, (for example, because you are fitting a power spectrum for which *w* must be 0,0), you can suppress the fitting of *w* using the **uvonly** option. The diagram below illustrates the **base** command.



w = lattice origin

u, *v* = lattice base vectors

× marks position originally supplied in *Plist* picture

The initial estimates that you provide of *u*, *v* and *w* (in picture coordinates) should be accurate enough to permit correct indexing of most positions. If you cannot achieve such accuracy, restrict the fit initially to a limited region and then use **base** a second time without the restriction. To restrict a fit to a region use one of the keys: **position** and **radius**.

To prevent a few spurious positions from biasing the fitted lattice, positions are ignored if they are not reasonably near a site of the estimated lattice – specifically, if their indices (coordinates in terms of the lattice base vectors) are not within 0.3 of an integer. The key **tolerance** allows you to alter this threshold value; a value of 0.5 or greater eliminates selection on this basis.

base

If you specify the key **to** explicitly, as in the last command example, an output *Plist* is produced which includes only those positions included in the fitting process (that is, those within any defined subregion and within the relevant tolerance of the lattice).

base sets the variables *n* and *r* to the number of positions fitted, and their *r.m.s.* deviation respectively. If **mark** is set, the final deviations of all positions from the lattice are marked via small lines drawn from the positions in the directions of the corresponding lattice site, but 5 times larger than the actual deviation for visibility. You can alter the magnification factor of these marking lines using the **times** key.

Notes

restrictions:	*Plist* type pictures only
display marking:	final position deviations
form used internally:	complex
variables used:	*u(2)*, *v(2)* (initial estimates for lattice base vectors) *w(2)* (initial estimate for lattice origin)
variables set:	*u(2)*, *v(2)* (base vectors of fitted lattice) *w(2)* (origin of fitted lattice) *n* (number of sites included in fit) *r* (r.m.s deviation in source pixels between included sites and final fitted lattice)
see also:	**lattice, peaks, strain, xwires list**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture number, held in the variable *select*	valid picture number
[to]	no output	valid picture number
position	position 0,0	within bounds of picture (real numbers)
radius	infinite	real number
numbers	all positions	1 to total number of points in the *Plist* picture
tolerance	tolerance 0.3	real number in range 0 to 0.5
mark	mark off	see *Appendix C*
times	times 5	positive integer
verify	verification off	

bclose

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	with	*<number>*	picture specifying structuring element data
	mask	*<number>*	mask picture to restrict processing to specified region
	times	*<number>*	number of times to apply structuring element data
	edge	*<number>*	edge value
options:	**source/output**		fix value of source/output edge pixels according to the value of the **edge** key

You use **bclose** to carry out generalised binary closing of foreground regions. The way in which the closing is carried out is determined by the structuring element specified by means of the **with** key. The output picture will contain the binary, closed result, which represents the complement of the area swept out by the reflection of the structuring element about the origin, while it is contained entirely within the background regions of the source image. Closing smoothes the boundary of foreground regions by filling in small concavities, it fills in narrow gaps between regions and it removes holes that are smaller than the structuring element. Closing is idempotent – closing an already closed image does not change the image.

Bclose has exactly the same keys and options as the commands **bdilate** and **berode** and it produces exactly the same result as using the command **bdilate** followed by **berode** with the same settings for the controlling keys and options, but in less time.

Examples

```
create 99 size 3 value 1; bclose 1 with 99
```

This example closes picture 1 with square, 3 by 3 structuring element.

```
bclose 1 2 with 99 times 3
```

This command is equivalent to the following.

```
bdilate 1 2 with 99 times 3; berode 2 with 99 times 3
```

bclose

Description

Closing a binary image with a structuring element is equivalent to dilating and then eroding the image with the same structuring element. For example,

```
0 0 0 0 0 0 0 0 0 0                                  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0                                  0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 1 0 0                                    0 0 0 1 1 1 1 0 0
0 0 1 0 1 1 1 0 0          1 1 1                     0 0 1 1 1 1 1 0 0
0 0 1 1[1]1 1 0 0   (.)    1[1]1   =                 0 0 1 1[1]1 1 0 0
0 0 1 0 0 0 0 0 0          1 1 1                     0 0 1 1 1 1 1 0 0
0 0 0 0 1 1 1 0 0                                    0 0 0 0 1 1 1 0 0
0 0 0 0 0 1 0 0 0                                    0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0                                    0 0 0 0 0 0 0 0 0
```

where (.) denotes closing and [ ] marks the origin of the picture and the structuring element. Compare this with the result obtained by erosion followed by dilation,

```
0 0 0 0 0 0 0 0 0 0                                  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0                                  0 0 1 1 1 1 1 1 0
0 0 0 1 0 1 1 0 0                                    0 1 1 1 1 1 1 1 0
0 0 1 0 1 1 1 0 0          1 1 1                     0 1 1 1 1 1 1 1 0
0 0 1 1[1]1 1 0 0   (+)    1[1]1   =                 0 1 1 1[1]1 1 1 0
0 0 1 0 0 0 0 0 0          1 1 1                     0 1 1 1 1 1 1 1 0
0 0 0 0 1 1 1 0 0                                    0 1 1 1 1 1 1 1 0
0 0 0 0 0 1 0 0 0                                    0 0 0 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0                                    0 0 0 0 1 1 1 0 0

0 0 0 0 0 0 0 0 0 0                                  0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0                                    0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 0                                    0 0 0 1 1 1 1 0 0
0 1 1 1 1 1 1 1 0          1 1 1                     0 0 1 1 1 1 1 0 0
0 1 1 1[1]1 1 1 0   (-)    1[1]1   =                 0 0 1 1[1]1 1 0 0
0 1 1 1 1 1 1 1 0          1 1 1                     0 0 1 1 1 1 1 0 0
0 1 1 1 1 1 1 1 0                                    0 0 0 0 1 1 1 0 0
0 0 0 1 1 1 1 1 0                                    0 0 0 0 0 1 0 0 0
0 0 0 0 1 1 1 0 0                                    0 0 0 0 0 0 0 0 0
```

where (–) denotes erosion, (+) denotes dilation.

bclose

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

Each layer of the picture specified by means of the **with** key defines a structuring element to be applied to the source picture. Non-zero pixels define the components of the structuring element. Any zero pixels are ignored. The origin of the picture defines the origin of the structuring element.

For example,

```
0 0 0 0 0                    0 0 1 0 0              1
0 0 1 0 0          1         0 0 1 0 0              1
0 1[2]1 0    => 1[1]1        0 0 1 0 0      =>     [1]
0 0 1 0 0          1         0 0 1 0 0              1
0 0 0 0 0                    0 0 1 0 0              1
```

If there is more than one layer, the structuring elements thus defined are applied in sequence, starting with layer 1. This allows you to specify large structuring elements in decomposed form. You can also use the **times** key to apply the sequence of structuring elements more than once. The structuring element represented by a decomposed set is obtained by combining the decomposed set using dilation.

For example,

```
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1          1 1 1      1 1 1      1 1 1      1 1 1
1 1 1 1[1]1 1 1 1    =     1[1]1 (+)  1[1]1 (+)  1[1]1 (+)  1[1]1
1 1 1 1 1 1 1 1 1          1 1 1      1 1 1      1 1 1      1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1

1 1 1                          1
1[1]1        =     1[1]1 (+)   [1]
1 1 1                          1
```

As you can see, the 9 by 9 structuring element can be decomposed into four 3 by 1 structuring elements and four 1 by 3 structuring elements. As dilation is a commutative operation, the structuring elements can be applied in any order. The simplest way to apply this example is to create a two-layer picture with the 3 by 1 structuring element in one layer and the 1 by 3 structuring element in the other layer, specify this picture by means of the **with** key and set the **times** key to 4.

bclose

The **mask** key can be used to restrict processing to specified regions of the image. The options **source** and **output** and the **edge** key allow you to control edge effects. For more detailed information about these and related topics, consult the documentation for the commands **berode** and **bdilate**.

Notes

see also: **berode, bdilate**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	none	valid picture number
mask	none	valid picture number
times	1	postive integer
edge	0	real number
source/output	do not process undefined pixels	

bdilate

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	with	*<number>*	picture specifying structuring element data
	mask	*<number>*	mask picture to restrict processing to specified region
	times	*<number>*	number of times to apply structuring element data
	edge	*<number>*	edge value
options:	**source/output**		fix value of source/output edge pixels according to the value of the **edge** key

You use **bdilate** to carry out generalised binary dilation of foreground regions. The way in which the dilation is carried out is determined by the structuring element specified by means of the **with** key. The result of applying each structuring element is obtained by sweeping it over the source image and combining all those instances where its origin coincides with a non-zero pixel in the source image. The technique is quite often used to add layers of pixels round the boundaries of foreground regions, which is why it is called **dilation**.

Examples

```
create 99 size 3 value 1; bdilate 1 with 99 times 1
```

This example adds one layer of pixels to all foreground objects in picture 1 (same result as **dilate** 1 **times** 1)

```
create 99 size 3 value 1; bdilate 1 with 99 times 4
create 99 size 9 value 1; bdilate 1 with 99 times 1
create 99 size 3,3,4 value 1; bdilate 1 with 99 times 1
```

Each of the above examples adds four layers of pixels to all foreground objects in picture 1 (same result as **dilate**1 **times** 4)

```
create 99 size 9,9,2 value 0
for i=-4,4; pixel i,0,-1=1; pixel 0,i,0=1; loop,
bdilate 1 with 99 times 1
```

This example dilates foreground objects in picture 1 with 9 by 9 square structuring element which has been decomposed into two much reduced structuring elements (gives same result as previous example but in less time).

bdilate

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

Each layer of the picture specified by means of the **with** key defines a structuring element to be applied to the source picture. If there is more than one layer, the structuring elements thus defined are applied in sequence, starting with layer 1. Non-zero pixels define the components of a structuring element. Any zero pixels are ignored. The origin of the picture defines the origin of the structuring element.

For example,

```
0 0 0 0 0                          0 0 1 0 0          1
0 0 1 0 0          1               0 0 1 0 0          1
0 1[2]1 0    =>  1[1]1             0 0[1]0 0    =>  [1]
0 0 1 0 0          1               0 0 1 0 0          1
0 0 0 0 0                          0 0 1 0 0          1
```

where [ ] marks the origin of the picture and the structuring element.

The process of dilating an object with a structuring element is very similar to a "paintbrush" operation if you think of the structuring element as the brush and the foreground pixels in the source image as defining the brush positions.

For example,

```
0 0 0 0 0                          0 1 1 1 0
0 0 1 0 0          1 1 1           1 1 1 1 1
0 1[1]1 0   (+)    1[1]1     =     1 1[1]1 1
0 0 1 0 0          1 1 1           1 1 1 1 1
0 0 0 0 0                          0 1 1 1 0
```

where (+) denotes diiation.

Dilation is a commutative operation so it is equally valid to think of the source image as the brush and the structuring element as defining the brush offsets. The result is the union (logical OR) of the translates of the source image where the position of each non-zero pixel in the structuring element with respect to the origin defines an offset to be applied to the source image.

bdilate

For example,

```
0 0 0 0 0                    0 0 0 0 0          0 0 0 0 0    0 0 0 0 0
0 0 1 0 0                    0 0 1 0 0          0 0 0 1 0    0 0 1 1 0
0 1[1]1 0   (+)  [1]1   =    0 1[1]1 0   (|)    0 0[1]1 1  = 0 1[1]1 1
0 0 1 0 0                    0 0 1 0 0          0 0 0 1 0    0 0 1 1 0
0 0 0 0 0                    0 0 0 0 0          0 0 0 0 0    0 0 0 0 0
```

where (|) denotes union.

With a multi-layer picture you can specify a series of structuring elements to be applied in turn. For example, the following result

```
0 0 0 0 0 0 0                          0 0 1 1 1 0 0
0 0 0 1 0 0 0                          0 0 1 1 1 1 0
0 0 0 1 1 0 0          1 1 1           0 0 1 1 1 1 0
0 0 0[0]1 0 0    (+)   1[1]1    =      0 1 1[1]1 1 0
0 0 1 1 0 0 0          1 1 1           0 1 1 1 1 1 0
0 0 0 0 0 0 0                          0 1 1 1 1 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
```

can be obtained with fewer operations by applying two simplier structuring elements

```
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
0 0 0 1 0 0 0                          0 0 1 1 1 0 0
0 0 0 1 1 0 0          0 0 0           0 0 1 1 1 1 0
0 0 0[0]1 0 0    (+)   1[1]1    =      0 0 0[1]1 1 0
0 0 1 1 0 0 0          0 0 0           0 1 1 1 1 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0                          0 0 1 1 1 0 0
0 0 1 1 1 0 0                          0 0 1 1 1 1 0
0 0 1 1 1 1 0          0 1 0           0 0 1 1 1 1 0
0 0 0[1]1 1 0    (+)   0[1]0    =      0 1 1[1]1 1 0
0 1 1 1 1 0 0          0 1 0           0 1 1 1 1 1 0
0 0 0 0 0 0 0                          0 1 1 1 1 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
```

Decomposing large structuring elements in this way can significantly reduce the processing time. A structuring element can be decomposed in this way if a set of structuring elements can be found which

bdilate

result in the original structuring element when they are combined by dilation. The above example illustrates this, that is,

```
0 0 0        0 1 0        1 1 1
1[1]1  (+)   0[1]0   =    1[1]1
0 0 0        0 1 0        1 1 1
```

where (+) denotes dilation.

The processing time depends in part on the number of non-zero pixels in the structuring element. In the example above, there are nine in the original version and six altogether in the decomposed version, a reduction of two-thirds in the number of source image translations. Set against this is the fact that two iterations are required instead of one. For such a small example decomposing the structuring element has no advantage. For a 21 by 21 structuring element decomposed in the same way, however, the time taken is reduced by a factor or 4 or 5.

Decomposing structuring elements can be used to good advantage as the following examples illustrate

```
0 0 1 1 1 0 0
0 1 1 1 1 1 0
1 1 1 1 1 1 1      0 0 0      0 0 1      0 1 0      1 0 0      0 0 0
1 1 1[1]1 1 1  =   0[1]1 (+)  0[1]0 (+)  0[1]0 (+)  0[1]0 (+)  1[1]0
1 1 1 1 1 1 1      0 0 0      0 0 0      0 0 0      0 0 0      0 0 0
0 1 1 1 1 1 0
0 0 1 1 1 0 0                 0 0 0      0 0 0      0 0 0
                        (+)   0[1]0 (+)  0[1]0 (+)  0[1]0
                              1 0 0      0 1 0      0 0 1
```

```
0 0 0 0 0 0 1 1 1
0 0 0 0 0 1 1 1 1      0 0 0 0 0 0 1      0 0 0 0 0 0 0      0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1      0 0 0 0 0 1 0      0 0 0 0 0 0 0      0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 0      0 0 0 0 1 0 0      0 0 0 0 0 0 0      0 0 0 1 0 0 0
0 0 1 1[1]1 1 0 0  =   0 0 0[1]0 0 0  (+) 0 0 1[1]1 0 0  (+) 0 0 0[1]0 0 0
0 1 1 1 1 1 0 0 0      0 0 1 0 0 0 0      0 0 0 0 0 0 0      0 0 0 1 0 0 0
1 1 1 1 1 0 0 0 0      0 1 0 0 0 0 0      0 0 0 0 0 0 0      0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0      1 0 0 0 0 0 0      0 0 0 0 0 0 0      0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0
```

The whole process can be repeated by specifying the number of iterations with the **times** key. The default is to carry out one iteration. Processing is stopped if no change is detected after applying one complete sequence of structuring elements.

The **mask** key can be used to restrict processing to specified regions of the image, that is, wherever

bdilate

mask pixels are zero, the source pixel value is output and elsewhere, the dilated result is substituted. Of course, the **mask** picture must have the same dimensions as the source picture. For example, the **mask** image

```
0 0 1 0 0 0 0
0 1 1 1 0 0 0
0 1 1 1 0 1 0
0 0 0 1 1 1 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 0 0 0 0 0
```

when applied to one of the previous examples

```
0 0 0 0 0 0 0                          0 0 1 1 1 0 0
0 0 0 1 0 0 0                          0 0 1 1 1 1 0
0 0 0 1 1 0 0          1 1 1           0 0 1 1 1 1 0
0 0 0[0]1 0 0    (+)   1[1]1    =      0 1 1[1]1 1 0
0 0 1 1 0 0 0          1 1 1           0 1 1 1 1 1 0
0 0 0 0 0 0 0                          0 1 1 1 1 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
```

gives the modified result

```
0 0 . 0 0 0 0          . . 1 . . . .          0 0 1 0 0 0 0
0 . . . 0 0 0          . 0 1 1 . . .          0 0 1 1 0 0 0
0 . . . 1 . 0          . 0 1 1 . 1 .          0 0 1 1 1 1 0
0 0 0 . . . 0    +     . . . 1 1 1 .     =    0 0 0 1 1 1 0
0 0 1 . . . 0          . . . 1 1 1 .          0 0 1 1 1 1 0
0 0 0 . . 0 0          . . . 1 1 . .          0 0 0 1 1 0 0
0 0 0 0 0 0 0          . . . . . . .          0 0 0 0 0 0 0
```

So far, the result for dilation has not been defined round the edges of an image where any part of the structuring element falls outside the limits of the source picture. By default, the process of calculating the union of the translates of the source image simply omits any undefined source pixels . If all the source pixels which contribute to the result for a particular output pixel are undefined, the output pixel defaults to 0.

As an alternative, the **source** option allows you to specify a value for any pixel positions outside the source image which contribute to a translate of the source image. You specify the actual value with the **edge** key. All non-zero values for the **edge** key imply an edge value of 1. The combination **source edge 0** in fact produces the same result as the default case.

bdilate

For example,

```
0 0 1 0 0                    0 0 1 0 0          1 0 0 0 E          E E E E E
0 1 0 0 0       1 0          0 1 0 0 0          1 1 1 0 E          0 0 1 0 0
0 1[1]1 0  (+) 0[1]    =     0 1[1]1 0    (|)   0 0[0]0 E    (|)   0 1[0]0 0
0 0 0 0 0         0 1        0 0 0 0 0          0 0 0 0 E          0 1 1 1 0
0 0 0 0 0                    0 0 0 0 0          E E E E E          0 0 0 0 0

                             1 E 1 E E
                             1 1 1 0 E
                         =   0 1[1]1 E
                             0 1 1 1 E
                             E E E E E
```

where E represents the source edge value.

Instead of using the **source** option, you can, with the **output** option and the **edge** key, specify a value to assign to output pixels wherever the result depends on undefined source pixels.

For example,

```
0 0 1 0 0                    E E E E E
0 1 0 0 0       1 0          1 1 1 0 E
0 1[1]1 0  (+) 0[1]    =     0 1[1]1 E
0 0 0 0 0         0 1        0 1 1 1 E
0 0 0 0 0                    E E E E E
```

where E represents the output edge value.

Notes

see also: **dilate**

bdilate

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	none	valid picture number
mask	none	valid picture number
times	1	positive integer
edge	0	real number
source/output	do not process undefined pixels	

beep, bell, buzz

beep, bell *and* ***buzz*** *take no arguments*

Use **beep**, **bell** or **buzz** to sound the keyboard bell (for example, to indicate the end of a long process).

berode

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	with	*<number>*	picture specifying structuring element data
	mask	*<number>*	mask picture to restrict processing to specified region
	times	*<number>*	number of times to apply structuring element data
	edge	*<number>*	edge value
options:	**source/output**		fix value of source/output edge pixels according to the value of the **edge** key

You use **berode** to carry out generalised binary erosion of foreground regions. The way in which the erosion is carried out is determined by the structuring element specified by means of the **with** key. The result of applying each structuring element is obtained by positioning the origin of the structuring element over each source pixel and outputting a 1 if the structuring element matches the source image in that position. The technique is most often used to remove layers of pixels round the boundaries of foreground regions, which is why it is called **erosion**.

Examples

```
create 99 size 3 value 1; berode 1 with 99 times 1
```

This example removes one layer of pixels from all foreground objects in picture 1 (same result as **erode 1 times 1**)

```
create 99 size 3 value 1; berode 1 with 99 times 4
create 99 size 9 value 1; berode 1 with 99 times 1
create 99 size 3,3,4 value 1; berode 1 with 99 times 1
```

Each of the above examples removes four layers of pixels from all foreground objects in picture 1 (same result as **erode 1 times 4**)

```
create 99 size 9,9,2 value 0
for i=-4,4; pixel i,0,-1=1; pixel 0,i,0=1; loop
berode 1 with 99 times 1
```

This example erodes foreground objects in picture 1 with 9 by 9 square structuring element which has been decomposed into two much reduced structuring elements (gives same result as previous example but in less time).

berode

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

Each layer of the picture specified by means of the **with** key defines a structuring element to be applied to the source picture. If there is more than one layer, the structuring elements thus defined are applied in sequence, starting with layer 1. Non-zero pixels define the components of a structuring element. Any zero pixels are ignored. The origin of the picture defines the origin of the structuring element.

For example,

```
0 0 0 0 0                                0 0 1 0 0          1
0 0 1 0 0          1                     0 0 1 0 0          1
0 1[2]1 0    =>  1[1]1                   0 0[1]0 0    =>   [1]
0 0 1 0 0          1                     0 0 1 0 0          1
0 0 0 0 0                                0 0 1 0 0          1
```

where [ ] marks the origin of the picture and the structuring element.

The process of eroding an object with a structuring element can be explained in simple terms if you think of the structuring element as a binary template. The origin of the structuring element is positioned over each source pixel in turn and a 1 is output if the structuring element matches the source image, that is, if every non-zero component of the structuring element coincides with a non-zero pixel in the source image. Otherwise a 0 is output. Erosion functions just like a Hit or Miss transform where the structuring element defines only *hits*. For more details about Hit or Miss transforms consult the documentation for the **bhmt** command.

For example,

```
0 0 0 1 0                                0 0 0 0 0
0 1 1 0 0                                0 0 1 0 0
0 0[1]1 1       (-)  1[1]     =          0 0[0]1 1
0 1 1 0 0                                0 0 1 0 0
0 1 0 0 0                                0 0 0 0 0
```

where (–) denotes erosion.

An equivalent definition for erosion can be obtained by combining translates of the source image.

The result is the intersection (logical AND) of the translates of the source image where the position of the origin of the structuring element with respect to each non-zero pixel in the structuring element defines an offset to be applied to the source image.

berode

For example

```
0 0 0 0 0          0 0 0 0 0          0 0 0 0 0    0 0 0 0 0
0 0 1 0 0            0 1              0 0 1 0 0    0 0 0 0 0
0 1[1]1 0   (-)  [1]0     =   0 1[1]1 0   (&)    0 1[0]0 0  =  0 1[0]0 0
0 0 1 0 0                     0 0 1 0 0          1 1 1 0 0    0 0 1 0 0
0 0 0 0 0                     0 0 0 0 0          0 1 0 0 0    0 0 0 0 0
```

where (&) denotes intersection.

With a multi-layer picture you can specify a series of structuring elements to be applied in turn. For example, the following result

```
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
0 0 0 1 0 0 0                          0 0 0 0 0 0 0
0 0 1 1 1 0 0          1 1 1           0 0 0 0 0 0 0
0 1 1[1]1 1 0    (-)  1[1]1     =      0 0 0[1]0 0 0
0 1 1 1 1 0 0          1 1 1           0 0 1 1 0 0 0
0 1 1 1 1 0 0                          0 0 0 0 0 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
```

can be obtained with fewer operations by applying two simpler structuring elements

```
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
0 0 0 1 0 0 0                          0 0 0 0 0 0 0
0 0 1 1 1 0 0          0 0 0           0 0 0 1 0 0 0
0 1 1[1]1 1 0    (-)  1[1]1     =      0 0 1[1]1 0 0
0 1 1 1 1 0 0          0 0 0           0 0 1 1 0 0 0
0 1 1 1 1 0 0                          0 0 1 1 0 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0


0 0 0 0 0 0 0                          0 0 0 0 0 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
0 0 0 1 0 0 0          0 1 0           0 0 0 0 0 0 0
0 0 1[1]1 0 0    (-)  0[1]0     =      0 0 0[1]0 0 0
0 0 1 1 0 0 0          0 1 0           0 0 1 1 0 0 0
0 0 1 1 0 0 0                          0 0 0 0 0 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
```

berode

Decomposing large structuring elements in this way can significantly reduce the processing time. The structuring element can be decomposed if a set of structuring elements can be found which result in the original structuring element when combined by dilation. The above example illustrates this.

```
0 0 0        0 1 0        1 1 1
1[1]1  (+)   0[1]0   =    1[1]1
0 0 0        0 1 0        1 1 1
```

Where (+) denotes dilation.

For a precise definition of dilation and more information about decomposing structuring elements, consult the documentation for the **bdilate** command.

The whole process can be repeated by specifying the number of iterations with the **times** key. The default is to carry out one iteration. Processing is stopped if no change is detected after applying one complete sequence of structuring elements.

The **mask** key can be used to restrict processing to specified regions of the image, that is, wherever mask pixels are zero, the source pixel value is output and elsewhere, the eroded result is substituted. Of course, the **mask** picture must have the same dimensions as the source picture. For example, the **mask** image

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 0 0 0 0 0 0
```

when applied to one of the previous examples

```
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
0 0 0 1 0 0 0                          0 0 0 0 0 0 0
0 0 1 1 1 0 0          1 1 1           0 0 0 0 0 0 0
0 1 1[1]1 1 0    (-)   1[1]1    =      0 0 0[1]0 0 0
0 1 1 1 1 0 0          1 1 1           0 0 1 1 0 0 0
0 1 1 1 1 0 0                          0 0 0 0 0 0 0
0 0 0 0 0 0 0                          0 0 0 0 0 0 0
```

berode

gives the following result

```
0 0 0 0 0 0 0        . . . . . .        0 0 0 0 0 0 0
0 0 0 1 0 0 0        . . . . . .        0 0 0 1 0 0 0
0 0 1 1 1 0 0        . . . . . .        0 0 1 1 1 0 0
0 . . . . . .   +    . 0 0 1 0 0 0  =   0 0 0 1 0 0 0
0 . . . . . .        . 0 1 1 0 0 0      0 0 1 1 0 0 0
0 . . . . . .        . 0 0 0 0 0 0      0 0 0 0 0 0 0
0 0 0 0 0 0 0        . . . . . .        0 0 0 0 0 0 0
```

So far, the result for erosion has not been defined round the edges of an image where any part of the structuring element falls outside the limits of the source picture. By default, the process of calculating the intersection of the translates of the source image simply omits any undefined source pixels. If all the source pixels which contribute to the result for a particular output pixel are undefined, the output pixel defaults to 1.

As an alternative, the **source** option allows you to specify a value for any pixel positions outside the **source** image which contribute to a translate of the source image. You specify the actual value with the **edge** key. All non-zero values for the **edge** key imply an edge value of 1. For erosion, an edge value of zero effectively closes off all foreground regions which touch the edge of the image. A non-zero edge value leaves regions "open" – it is as if regions extend indefinitely at right angles to the edges of the image. The combination **source edge 1** in fact produces the same result as the default case.

For example,

```
0 1 0 0 0                    0 1 0 0 0        E E E E E        0 1 1 0 0
0 1 1 0 0        1 0         0 1 1 0 0        E 0 1 0 0        0 1 1 1 0
0 1 1 1 0  (-)   0[1]   =     0 1 1 1 0  (&)   E 0 1 1 0  (&)   0 1 1 0 1
0 1 1 0 1        0 1         0 1 1 0 1        E 0 1 1 1        0 0 0 1 1
0 0 0 1 1                    0 0 0 1 1        E 0 1 1 0        E E E E E

                             0 E 0 0 0
                             0 0 1 0 0
                         =   0 0 1 0 0
                             0 0 0 0 1
                             0 0 0 E 0
```

where E represents the source edge value.

Instead of using the **source** option, you can, with the **output** option and the **edge** key, specify the value to assign to output pixels wherever the result depends on undefined source pixels. A zero output edge value is particularly useful when using erosion for templating operations. It eliminates possible edge effects because output pixels can only be set if the structuring element lies fully inside the source image. A non-zero pixel in the output image means that the structuring element exactly macthed the source image at that position.

berode

For example,

```
0 1 0 0 0                E E E E E
0 1 1 0 0      1 0       E 0 1 0 0
0 1 1 1 0  (-) 0[1]  =   E 0 1 0 0
0 1 1 0 1      0 1       E 0 0 0 1
0 0 0 1 1                E E E E E
```

where E represents the output edge value.

Notes

see also: **bdilate, bhmt, erode**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	none	valid picture number
mask	none	valid picture number
times	1	postive integer
edge	0	real number
source/output	do not process undefined pixels	

bhmt

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	with	*<number>*	picture specifying structuring element data
	mask	*<number>*	mask picture to restrict processing to specified region
	times	*<number>*	number of times to apply structuring element data (zero for infinite number of iterations)
	edge	*<number>*	edge value
options:	**add/subtract**		add or subtract result of **hit** or **miss** transform to source image to produce output result
	source/output		fix value of source/output edge pixels according to the value of the **edge** key

You use **bhmt** to calculate generalised binary Hit or Miss transforms. The details of a particular transform are determined by the structuring element specified by means of the **with** key. The result of applying each structuring element is obtained by positioning the origin of the structuring element over each source pixel and outputing a 1 if the structuring element matches the source image in that position. Thickening and thinning transformations can be achieved by specifying the **add** and **subtract** options respectively. An example of this is obtaining the binary skeleton of the source image. Hit or Miss transforms are so called because the matching process can involve background pixels (miss) as well as foreground pixels (hit).

Examples

```
create 99 size 2, 1 value 0; p 0,0=1; bhmt 1 with 99
```

This example detects all pixels that lie on the lefthand edges of foreground regions of the binary image in picture 1.

```
bhmt 1 2 with 20 times 0 add
```

This command carries out on picture 1 the infinite thickening transformation defined by the structuring element in picture 20 and outputs the result to picture 2.

```
create 99 size 3 value 1; bhmt 10 with 99 subtract
```

This example leaves in picture 10 the four-connected outline of any foreground regions.

bhmt

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

Each layer of the picture specified by means of the **with** key defines a structuring element to be applied to the source picture. If there is more than one layer, the structuring elements thus defined are applied in sequence, starting with layer 1. A zero pixel value defines a Miss and a pixel value of 1 defines a Hit. Any other values are ignored. The origin of the picture defines the origin of the structuring elment.

For example,

```
2 2 2 2 2                       2 2 0 2 2          0
2 1 1 1 2       1 1 1           3 3 1 3 3          1
2 2[1]2 2   =>    [1]           4 4[1]4 4    =>  [1]
2 0 0 0 2       0 0 0           5 5 1 5 5          1
2 2 2 2 2                       6 6 0 6 6          0
```

where [ ] marks the origin of the picture and the structuring element.

The Hit or Miss transform can be explained in simple terms if you think of the structuring element as a binary template. The origin of the structuring element is positioned over each source pixel in turn and a 1 is output if all the components of the structuring element match the source image, that is, if misses in the structuring element match zeros in the source image and hits in the structuring element match non-zero pixels in the source image. Otherwise a 0 is output.

For example,

```
0 0 0 1 1                       0 0 0 1 0
0 1 1 0 0                       0 1 0 0 0
0 1[1]1 1   (*)   0[1]    =    0 1[0]0 0
0 1 1 0 1                       0 1 0 0 1
0 0 1 1 0                       0 0 1 0 0
```

where (*) denotes a Hit or Miss transform.

An equivalent definition for Hit or Miss transforms can be obtained by combining translates of the source image and its complement. A Hit specifies a translate of the source image itself. A Miss specifies a translate of the complement of the source image. The result is the intersection (logical AND) of the translates where the position of the origin with respect to each Hit or Miss in the structuring element defines an offset to be applied to the source image or its complement.

bhmt

For example,

```
0 0 0 0 0                   0 0 0 0 0       0 0 0 0 0       1 1 1 1 1
0 0 1 0 0         0 1       0 0 1 0 0       0 0 0 0 0       1 1 1 1 1
0 1[1]1 0   (*)  [1]   =    0 1[1]1 0   (&)  0 1[0]0 0  (&)  1 1[0]1 1
0 0 1 0 0                   0 0 1 0 0       1 1 1 0 0       1 0 0 0 1
0 0 0 0 0                   0 0 0 0 0       0 1 0 0 0       1 1 0 1 1

                            0 0 0 0 0
                            0 0 0 0 0
                       =    0 1[0]0 0
                            0 0 0 0 0
                            0 0 0 0 0
```

where (&) denotes intersection.

Apart from their use as template operators, hit or miss transforms are also used to implement thickening and thinning transformations. Here the result obtained from the hit or miss transform is added to or subtracted from the source image. You specify a thickening or thinning transformation by setting the option **add** or **subtract**. More precisely, **add** causes the union (logical OR) of the source and the hit or miss transform to be output, and **subtract** causes the difference between the source and the hit or miss transform (logical AND of source and complement of hit or miss transform) to be output.

With a multi-layer picture you can specify a series of structuring elements to be applied in turn. For example, here is a series of structuring elements that you might specify for a thinning transform to obtain an eight-connected skeleton.

```
0 0 0    0 0      0   1      1      1 1 1      1      1   0      0 0
 [1]     0[1]1    0[1]1    0[1]1     [1]     1[1]0    1[1]0    1[1]0
1 1 1      1      0   1    0 0      0 0 0      0 0      1   0      1
```

Applying these eight structuring elements in turn represents one complete iteration of the thinning transform, removing a layer of pixels from foreground regions and leaving unchanged any boundary pixels which belong to the skeleton. To obtain the complete skeleton, several iterations of the transform are required.

The whole process can be repeated by specifying the number of iterations with the **times** key. The default is to carry out one iteration. A zero value for the **times** key specifies an infinite number of iterations. Processing is stopped if no change is detected after applying one complete sequence of structuring elements. In the example above, you would specify a zero value for the **times** key to obtain the binary skeleton of the source image. After a certain number of iterations, the output image would no longer change and processing would be stopped automatically.

bhmt

The **mask** key can be used to restrict processing to specified regions of the image, that is, wherever mask pixels are zero, the source pixel value is output and elsewhere, the result of the Hit or Miss transform is substituted. Of course, the mask picture must have the same dimensions as the source picture. For example, the mask image

```
0 0 0 0 0
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 0 0 0 0
```

when applied to one of the previous examples

```
0 0 0 1 1                                   0 0 0 1 0
0 1 1 0 0                                   0 1 0 0 0
0 1[1]1 1   (*)   0[1]      =               0 1[0]0 0
0 1 1 0 1                                   0 1 0 0 1
0 0 1 1 0                                   0 0 1 0 0
```

gives the following result

```
0 0 0 1 1          . . . . .          0 0 0 1 1
0 . . . .          . 1 0 0 0          0 1 0 0 0
0 . . . .    +     . 1 0 0 0    =     0 1 0 0 0
0 . . . .          . 1 0 0 1          0 1 0 0 1
0 0 1 1 0          . . . . .          0 0 1 1 0
```

So far, the result for the Hit or Miss transform has not been defined round the edges of an image where any part of the structuring element falls outside the limits of the source picture. By default, the process of calculating the intersection of the translates of the source image or its complement simply omits any undefined source pixels. If all the source pixels which contribute to the result for a particular output pixel are undefined, the output pixel defaults to 1.

For example,

```
0 1 0 0 0                                   1 0 1 1 1
0 1 1 0 1                    1 0            0 0 0 0 0
0 1 1 1 0    (*)     [0]     =              0 0 0 0 1
0 1 1 0 1                                   0 0 0 1 0
0 0 0 1 1                                   0 0 1 0 0
```

bhmt

Notice how the structuring element in this example matches any zeros in the top row of the source image because here the top row of the structuring element falls outside the source image and is ignored.

As an alternative, the **source** option allows you to specify a value for any pixel positions outside the source image which contribute to a translate of the source image or its complement. You specify the actual value with the **edge** key. All non-zero values for the **edge** key imply an edge value of 1. An edge value of zero effectively closes off all foreground regions which touch the edge of the image. A non-zero edge value leaves regions "open" – it is as if regions extend indefinitely at right angles to the edges of the image.

For example,

```
0 1 0 0 0                1 0 1 1 1        E E E E E        C C C C C
0 1 1 0 1      1 0       1 0 0 1 0        0 1 0 0 0        0 1 1 1 C
0 1 1 1 0  (*) [0]   =    1 0 0 0 1  (&)   0 1 1 0 1  (&)   0 0 1 0 C
0 1 1 0 1                1 0 0 1 0        0 1 1 1 0        0 0 0 1 C
0 0 0 1 1                1 1 1 0 0        0 1 1 0 1        0 0 1 0 C

                         0 0 0 0 0
                         0 0 0 0 0
                     =   0 0 0 0 C
                         0 0 0 1 0
                         0 0 1 0 0
```

where E represents the source edge value and C its complement.

Instead of using the **source** option, you can, with the **output** option and the **edge** key, specify the value to assign to output pixels wherever the result depends on undefined source pixels. A zero output edge value is particularly useful when using Hit or Miss transforms for templating operations. It eliminates possible edge effects because output pixels can only be set if the structuring element lies fully inside the source image. A non-zero pixel in the output image means that the structuring element exactly matched the source image at that position.

For example,

```
0 1 0 0 0                E E E E E
0 1 1 0 1      1 0       0 0 0 0 E
0 1 1 1 0  (*) [0]   =    0 0 0 0 E
0 1 1 0 1                0 0 0 1 E
0 0 0 1 1                0 0 1 0 E
```

where E represents the output edge value.

bhmt

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	none	valid picture number
mask	none	valid picture number
times	1	zero or positive interger
edge	0	real number
source/output	do not process undefined pixels	

bmlut

keys:	**[to]**	*<number>*	destination picture to contain mapping tables
	if	*<expression>*	logical expression defining which neighbourhood configurations give a positive (non-zero) result
	unless	*<expression>*	logical expression defining which neighbourhood configurations give a negative (zero) result
	neighbours/with	*<number>*	use only with **erode** or **dilate** option:
	neighbours		minimum number of clear/set neighbours in order to erode/dilate centre pixel
	with		picture containing user supplied mapping table for use with **erode** or **dilate** command
options:	**erode/dilate/median**		generate mapping table for 3 x 3 neighbouhood transformations supported by **erode, dilate** or **median** commands
	skeletonise/ends/nodes/outline/ol4		use only with **erode** option:
	skeletonise		erode to 8-connected skeleton
	ends		erode ends of protruding branches/hairs
	nodes		erode skeleton node points, separating branches
	outline		generate 8-connected outline
	ol4		generate 4-connected outline
	separately		use only with **dilate** option: dilate, preserving separation between forground regions (reduces background to 4-connected skeleton)

You use **bmlut** to generate neighbourhood mapping tables. These can define any kind of 3 by 3 binary neighbourhood transformation. A mapping table is applied with the separate command **bmmap**. Once a mapping table has been created it can be used any number of times. With the option **erode, dilate** or **median**, you can generate mapping tables that correspond exactly to any of the morphological transformations supported by the commands **erode, dilate** and **median**. More general transformations can be defined by specifying logical expressions with the keys **if** and **unless** to define the outcome for the 512 possible configurations in a 3 by 3 neighbouring. These logical expressions should include at least one of the variables *p0* to *p8, n4, n8, c, c4* and *c8* to characterise the neighbourhood configurations as required for a particular morphological transformation.

bmlut

Examples

```
bmlut 99 erode outline; bmmap 1 2 with 99
```

This example outputs outlines of foreground regions in picture 1 to picture 2 (same result as **erode 1 2 outline**).

```
bmlut 99 if (p4+n8) > 4
```

This command outputs the mapping table for a binary median filter (same result as **bmlut 99 median**).

```
bmlut 99 if (p4 & n8~=0) | (~p4 & n8=8); bmmap 1 with 99
```

This example removes salt-and-pepper noise from picture 1.

```
bmlut 99 if p4 & n8=1
```

This command outputs a mapping table to detect line ends.

Description

Bmlut lets you set up any kind of transformation of a 3 by 3 binary neighbourhood. What you have to do is specify the outcome (a 0 or a 1) for each of the 512 possible neighbourhood configurations. **Bmlut** stores this neighbourhood mapping table in a Semper picture for later use with the **bmmap** command. Each configuration is represented by a 9-bit index value in the range 0 to 511. Each bit in the index value represents the state of one of the pixel locations in the 3 by 3 neighbourhood as follows.

```
0 3 6
1 4 7
2 5 8
```

where bit 0 is the least significant bit and bit 4 corresponds to the central pixel. The index value points to the entry in the mapping table where the result for that neighbourhood configuration is stored.

You may decide to create mapping tables directly by creating a new picture of the appropriate size and setting each mapping table entry with the **pixel** command. However, if the outcome for each neighbourhood configuration can be specified in the form of a common logical expression, **bmlut** can be directed to generate the mapping table for you.

bmlut

You specify the necessary logical conditions with the keys **If** and **unless**. Either key accepts a general Semper logical expression which **bmlut** evaluates for each of the 512 neighbourhood configurations. Each logical expression should normally refer to at least one of the variables *p0* to *p8, n4, n8, c, c4* and *c8* which **bmlut** sets itself before it evaluates the expression. The result obtained determines the value stored in the mapping table. If the **If** expression is true and the **unless** expression is false, a 1 is output, otherwise, a zero is output.

Two keys are provided for convenience in specifying some logical conditions but exactly the same result could be achieved with just one key, that is

```
bmlut if <if-expression> unless <unless-expression>
bmlut if <if-expression> & ~<unless-expression>
bmlut unless ~<if-expression> | <unless-expression>
```

are all equivalent. You may also omit either one of the two expressions

```
bmlut if <if-expression>
bmlut unless <unless-expression>
```

Bmlut sets the variables *po* to *p8* according to the value of the pixels in a particular neighbourhood configuration. For example a neighbourhood index value of 463

```
                    1 1 1
463 = 111001111  =>  1 0 1
                    1 0 1
```

causes variables *p4* and *p5* to be set to zero (false) and the variables *p0, p1, p2, p3, p6, p7* and *p8* to be set to 1 (true).

To create a mapping table which detects just this neighbourhood configuration (a **hit** or **miss** transform) you could specify the following logical expression with the **If** key

```
p0 & p1 & p2 & p3 & p4 & ~p4 & ~p5 & p6 & p7 & p8
```

This would cause a mapping table to be output with all entries set to zero except entry 463.

Of course, there are more useful neighbourhood transformations that can be defined besides these simple Hit or Miss transforms (which, anyway, are more efficiently carried out with the command **bhmt**). In particular, you should note that you are not restricted to pure logical expressions when using **bmlut.** For example, to create the mapping table to detect all neighbourhood configurations where the central pixel is surrounded by exactly 4 non-zero neighbours, you could specify the following logical expression

```
( p0 + p1 + p2 + p3 + p5 + p6 + p7 + p8 ) = 4
```

bmlut

Logical expressions which count pixels in a 3 by 3 neighbourhood are very common in defining the necessary conditions for many morphological transforms. Consequently, **bmlut** provides further variables *n4, n8, c, c4* and *c8* for added convenience. They are defined in the following way

```
n4 = p1 + p3 + p5 + p7

n8 = p0 + p1 + p2 + p3 + p5 + p6 + p7 + p8

c = (~p0 & p1) + (~p1 & p2) + (~p2 & p5) + (~p5 & p8) + (~p8 & p7)
                            + (~p7 & p6) + (~p6 & p3) + (~p3 & p0)

c4 = p1 + p3 + p5 + p7  - (p0 & p1 & p3) - (p2 & p1 & p5)
                        - (p6 & p3 & p7) - (p8 & p5 & p7)

c8 = ~p1 + ~p3 + ~p5 + ~p7 - (~p0 & ~p1 & ~p3) - (~p2 & ~p1 & ~p5)
                           - (~p6 & ~p3 & ~p7) - (~p8 & ~p5 & ~p7)
```

n4 and *n8* represent the number of 4-connected and 8-connected neighbours around the central pixel.

c represents the neighbourhood "crossing number" which is the number of 0–1 transitions in the central pixel's surround. This takes no account of connectivity between neighbouring pixels in the central pixel's surround, so the variables *c4* and *c8* are provided for this. They represent the number of separate 4-connected and 8-connected foreground regions which surround the central pixel. These two parameters are somethimes referred to as "connectivity numbers" in the morphological literature.

Further parameters mentioned in the literature include the bond number and the coefficients of curvature for 4-connected and 8-connected surrounds. These are not provided in variable form because they are less commonly used and are easily derived from the variables described above

bond number	`n4 + n8`
4-connected coefficient of curvature	`( 4 - n4 ) - c4`
8-connected coefficient of curvature	`( 8 - n8 ) - (c8 + c + c8)`

The 14 variables supported by **bmlut** in logical expressions specified with the keys **if** and **unless** provide a simple and elegant way to generate the mapping tables for many useful morphological transformations, as the following list of examples illustrate

bmlut

4-neighbour erode	`bmlut if p4 & n4=4`
8-neighbour erode	`bmlut if p4 & n8=8`
4-neighbour dilate	`bmlut if p4 | n4~=0`
8-neighbour dilate	`bmlut if p4 | n8~=0`
Hole detect	`bmlut if ~p4 & n4=4`
Interior pixel fill	`bmlut if p4 | n4=4`
End detect	`bmlut if p4 & n8=1`
End remove	`bmlut if p4 unless n8=1`
Isolated pixel detect	`bmlut if p4 & n8=0`
Isolated pixel remove	`bmlut if p4 unless n8=0`
Remove salt–and–pepper noise	`bmlut if (p4 & n8~=0) | (~p4 & n8=8)`
Spur detect	`bmlut if p4 & (n4 + n8) = 1`
Spur remove	`bmlut if p4 unless (n4 + n8) = 1`
4-connected outline	`bmlut if p4 unless n8=8`
8-connected outline	`bmlut if p4 unless n4=4`
H detect	`bmlut if p4 & n8=6 & c4=2 & c8=2`
H break	`bmlut if p4 unless n8=6 & c4=2 & c8=2`
Triple point detect	`bmlut if p4 & c>2`
Median filter	`bmlut if (p4 + n8) >4`

All of these examples have quite simple logical conditions. **Bmlut** supports general and quite lengthy logical expresssions as in the following example which generates the mapping table for obtaining the octagonal convex hull

```
bmlut if p4 | (p1 & p3 & p5)                              +
            | (p3 & p5 & p7)                              +
            | (p5 & p7 & p1)                              +
            | (p7 & p1 & p3)                              +
            | (p1 & p3 & (p2 | p6) & ~p5 & ~p7 & ~p8)     +
            | (p5 & p7 & (p2 | p6) & ~p1 & ~p3 & ~p0)     +
            | (p1 & p5 & (p0 | p8) & ~p3 & ~p7 & ~p6)     +
            | (p3 & p7 & (p0 | p8) & ~p1 & ~p5 & ~p2)
```

where "+" is Semper's line continuation character.

The first four lines of the logical expression can in fact be replaced with just "p4 | n4 >= 3".

Some morphological operations require a series of morphological transformations to be applied in turn (opening, for example, consists of erosion followed by dilation). **Bmlut** allows you to specify more than one logical expression with the **If** or **unless** keys. Each expression is used to generate a separate mapping table which is stored as a separate row in the output picture. You use commas "," to separate the expressions.

bmlut

For example, a more rounded result for dilation can be achieved by making alternate use of 4-connected and 8-connected dilation. The mapping table for this can be generated in the following way

```
bmlut if p4 | n4~=0, p4 | n8~=0
```

If the **If** and **unless** keys are used together and they specify a different number of expressions, **bmlut** will repeat the shorter set of expressions to make up the same number as the longer set. For example, quite a good thinning transformation can be obtained with four mapping tables as follows

```
bmlut if p4, p4, p4, p4 unless  c8=1 & n8~=1 & (~p1 & p7), +
                                c8=1 & n8~=1 & (~p7 & p1), +
                                c8=1 & n8~=1 & (~p3 & p5), +
                                c8=1 & n8~=1 & (~p5 & p3)
```

The string of expressions given by the **If** key can be replaced with just **p4** to give the same result.

```
bmlut if p4 unless  c8=1 & n8~=1 & (~p1 & p7),+
                    c8=1 & n8~=1 & (~p7 & p1),+
                    c8=1 & n8~=1 & (~p3 & p5),+
                    c8=1 & n8~=1 & (~p5 & 3)
```

The logical condition **unless c8=1 & n8~=1** is the basic condition for eroding pixels in order to thin objects down to an 8-connected skeleton (**c8=1** selects edge pixels that can be eroded without causing a break in the central pixel's surround and **n8~=1** protects line ends). Because neighbourhood transformations are applied in parallel to all the neighbourhoods in the source image, without reference to any changes that might occur in neighbouring pixel locations, this is not enough to maintain the connectivity of the result. This accounts for the extra condition on the end of each expression which restricts the erosion during each pass to one of the four cardinal directions and ensures that two-pixel-wide features are not completely eroded.

Substituting *variable c4* and *n4* for *c8* and *n8* will give a similar, but 4-connected result.

```
bmlut if p4 unless  c4=1 & n4~=1 & (~p1 & p7),+
                    c4=1 & n4~=1 & (~p7 & p1),+
                    c4=1 & n4~=1 & (~p3 & p5),+
                    c4=1 & n4~=1 & (~p5 & p3)
```

These thinning transformations can be thought of as erosions which preserve the connectivity of foreground regions and which do not erode line ends. One can easily modify these examples to model thickening transformations to produce background skeletons.

bmlut

Inverting the basic conditions for the 8-connected skeleton, you obtain the following command

```
bmlut if p4 | (c8=1 & n8~=7 & (~p1 & p7)), +
          p4 | (c8=1 & n8~=7 & (~p7 & p1)), +
          p4 | (c8=1 & n8~=7 & (~p3 & p5)), +
          p4 | (c8=1 & n8~=7 & (~p5 & p3))
```

This transformation preserves 8-connectivity of foreground regions – so the background skeleton will be 4-connected. For an 8-connected result, substitute *c4* and *n4* for *c8* and *n8*, as before, but notice that the inequality involving n4 must also be modified, that is

```
bmlut if p4 | (c4=1 & n4~=3 & (~p1 & p7)), +
          p4 | (c4=1 & n4~=3 & (~p7 & p1)), +
          p4 | (c4=1 & n4~=3 & (~p3 & p5)), +
          p4 | (c4=1 & n4~=3 & (~p5 & p3))
```

If you invert the result obtained by applying the mapping tables generated by this last example, you will obtain the 8-connected background skeleton, or exo-skeleton.

If you omit from the last example the logical condition **n4~=3** for protecting line ends , you will obtain, after inverting the processed result, the 8-connected skiz (skeleton by zones of influence) of the source image.

```
bmlut if p4 | (c4=1 & (~p1 & p7)), +
          p4 | (c4=1 & (~p7 & p1)), +
          p4 | (c4=1 & (~p3 & p5)), +
          p4 | (c4=1 & (~p5 & p3))
```

Bmlut also has keys and options (as an alternative to using the keys **if** or **uniess** which allow you to generate mapping tables for any of the neighbourhood transformations supported by the commands **erode, dilate** and **median (bmmap** is usually much faster to use than **erode, dilate** or **median**). You select which command by setting the corresponding option and you append exactly the same string of keys or options as you would use with the command itself. For example,

```
bmlut dilate separately
```

would generate the mapping tables that **bmmap** would require to carry out exactly the same morphological transformation as the command

```
dilate separately
```

bmlut

Note that the user-defined mapping tables supplied to **erode** and **dilate**, by specifying the key **with**, are 8-bit mapping tables which refer only to the central pixel's surround and have a different pattern of bits from that used by **bmlut**

```
7 6 5
0 * 4
1 2 3
```

Note also that when using the commands **erode skeletonise**, **erode ends** and **dilate separately**, the **times** key automatically defaults to 0 (infinite number of iterations). With **bmmap**, the default for the **times** key is always 1, so you must explicitly set it to zero in order to get the same result.

For more detailed information about these morpholgical transformations, consult the documentation for the commands **erode, dilate** and **median.**

Notes

see also: **bmmap, bhmt, erode, dilate, median,**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
if	true	valid expression
unless	false	valid expression
neighbours	none	1 to 8
with	none	valid picture number

bmmap

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	with	*<number>*	picture containing one or more neighbourhood mapping tables
	mask	*<number>*	mask picture to restrict processing to specified region
	times	*<number>*	number of times to apply neighbourhood mapping tables (zero for infinite number of iterations)
	edge	*<number>*	edge value
options:	**source/output**		fix value of source/output edge pixels according to the value of the **edge** key

You use **bmmap** to carry out generalised binary 3 by 3 neighbourhood transformations. The way in which the transformation is carried out is determined by one or more neighbourhood mapping tables contained in the picture specified by means of the **with** key. The result of applying each mapping table is obtained by calculating the neighbourhood index for each source pixel and using this index to look up the result in the mapping table. The mapping table defines the result for all the 512 possible configurations in a binary 3 by 3 neighbourhood. The command **bmlut** provides a convenient way to generate mapping tables for use with **bmmap.**

Examples

```
create 99 size 512, 1 value 0; origin left; pixel 511 = 1
bmmap 1 with 99 times 1
```

This example removes one layer of pixels from all foreground objects in picture 1 (same result as **erode** 1 times 1).

```
bmlut 99 erode outline; bmmap 1 2 with 99
```

This example outputs outlines of foreground regions in picture 1 to picture 2 (same result as **erode 1 2 outline**).

```
bmlut 99 if (p4 & n8~=0) | (~p4 & n8=8); bmmap 1 with 99
```

This example removes salt-and-pepper noise from picture 1.

```
bmlut 99 dilate separately; bmmap 1 with 99 times 0 mask 3
```

This example dilates foreground regions in picture 1, keeping unconnected regions separate, until they

bmmap

fill the foreground regions in picture 3.

```
bmlut 99 median; bmmap 1 with 99 times 5
```

This example applies a binary median filter to picture 1 and gives the same result as the following

```
for i=1,5; median 1; loop
```

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

Each row of the picture specified by means of the **with** key contains a mapping table which defines the result for each of the 512 possible binary 3 by 3 neighbourhood configurations. A 9-bit neighbourhood index value is constructed for each source pixel, where each bit in the index value is set according to the state of the corresponding pixel in the neighbourhood. The neighbourhood bits are arranged in the following way

```
0 3 6
1 4 7
2 5 8
```

Calculating the index value for a neighbourhood is equivalent to applying the following linear convolution kernel

```
2^0 2^3 2^6      1  8  64
2^1 2^4 2^7  =   2 16 128
2^2 2^5 2^8      4 32 256
```

provided that a pixel value in the neighbourhood is given a value of 0 or 1 according to whether the corresponding source pixel is zero or non-zero.

The index value is used to select the mapping table entry which defines the output result for the central pixel. If the mapping table entry is set to zero, a zero is output. Otherwise, a 1 is output. In this way, quite arbitrary 3 by 3 neighbourhood transformations can be caried out. It is important to note that the output values do not affect the result for neighbouring pixels during each pass through the image. In effect, the transformation is simultaneously applied to all the source pixels.

If the mapping table picture contains more than one row, a series of neighbourhood transformations will be applied in turn.

bmmap

For example, the morphological opening with a 3 by 3 structuring element can be specified if the first row of the picture contains the mapping table for erosion and the second row contains the mapping table for dilation

```
create 1 size 512, 1 value 0; origin left; pixel 511 =
create 2 size 512, 1 value 1; origin left; pixel   0 = 0
create 3 size 512, 2
paste 1 3 top
paste 2 3 bottom
bmmap ... with 3
```

The whole process can be repeated by specifying the number of iterations with the **times** key. The default is to carry out one iteration. A zero value for the **times** by specifies on on infinite number of iterations. Processing is stopped if no change is detected after applying one complete sequence of mapping tables.

The command **bmlut** provides a convenient way to generate mapping tables. For example the mapping tables for morphological opening, as given in the previous example, could be generated with the following command

```
bmult 3 if (p4 & n8 = 8), (p4 | n8 ~= 0)
```

The result for each neighbourhood configuration is defined as a logical expression which is evaluated for each of the 512 possible configurations. **Bmlut** also has keys and options which allow you to generate mapping tables for any of the neighbourhood tranformations supported by the commands **erode, dilate** and **median (bmmap** is usually much faster than **erode, dilate** or **median).**

The **mask** key can be used to restrict processing to specified regions of the image, that is, wherever mask pixels are zero, the source pixel value is output and elsewhere, the result of the 3 by 3 neighbourhood transformation is substituted. Of course, the mask picture must have the same dimensions as the source picture. For example, the mask image

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 1 1 1 1 1 1
0 0 0 0 0 0 0
```

when applied to the following example, where the neighbourhood transformation produces the eight-connected outline of the source image.

bmmap

source	result
0 0 0 0 0 0 0	0 0 0 0 0 0 0
0 0 0 1 1 0 0	0 0 0 1 1 0 0
0 0 1 1 1 1 0	0 0 1 0 0 1 0
0 1 1 1 1 1 0	0 1 0 0 0 1 0
0 1 1 1 1 0 0	0 1 0 0 1 0 0
0 1 1 1 1 0 0	0 1 1 1 1 0 0
0 0 0 0 0 0 0	0 0 0 0 0 0 0

gives the following result

0 0 0 0 0 0 0		. . . . .		0 0 0 0 0 0 0
0 0 0 1 1 0 0		. . . . .		0 0 0 1 1 0 0
0 0 1 1 1 1 0		. . . . .		0 0 1 1 1 1 0
0	+	. 1 0 0 0 1 0	=	0 1 0 0 0 1 0
0		. 1 0 0 1 0 0		0 1 0 0 1 0 0
0		. 1 1 1 1 0 0		0 1 1 1 1 0 0
0 0 0 0 0 0 0		. . . . .		0 0 0 0 0 0 0

So far, the result for a 3 by 3 neighbourhood transformation has not been defined round the edges of an image where any part of the neighbourhood falls outside the limits of the source picture. By default, source pixels round the edges of the image are replicated outwards.

As an alternative, the **source** option allows you to specify a value for any pixel positions outside the source image which contribute to the neighbourhood of a pixel. You specify the actual value with the **edge** key. All non-zero values for the **edge** key imply an edge value of 1. For example, if a zero edge value is specified, the neighbourhood transformation to produce the eight-connected outline of the source image would result in closed outlines for all regions. If a non-zero edge value is specified, outlines of regions which touch the edges of the picture are left open.

bmmap

source

```
0 1 1 1 0 0 0
1 1 1 1 0 0 0
1 1 1 1 1 1 0
0 1 1 1 1 1 1
0 0 0 1 1 1 1
0 0 0 0 1 1 1
0 0 0 0 1 1 1
```

result

```
0 1 1 1 0 0 0
1 0 0 1 0 0 0
1 0 0 0 1 1 0
0 1 1 0 0 0 1
0 0 0 1 0 0 1
0 0 0 0 1 0 1
0 0 0 0 1 1 1
```

source edge 0

```
0 1 0 1 0 0 0
1 0 0 1 0 0 0
1 0 0 0 1 1 0
0 1 1 0 0 0 1
0 0 0 1 0 0 0
0 0 0 0 1 0 0
0 0 0 0 1 0 0
```

source edge 1

Instead of using the **source** option, you can, with the **output** option and the **edge** key, specify the value to assign to output pixels wherever the result depends on undefined source pixels. A zero output edge value is particularly useful when using 3 by 3 neighbourhood transformations to detect particular neighbourhood configurations. It eliminates possible edge effects because output pixels can only be set if the entire neighbourhood region lies inside the source image.

Notes

see also: **bmlut, erode, dilate, median**

bmmap

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	none	valid picture number
mask	none	valid picture number
times	1	zero or positive integer
edge	0	real number
source/output	replicate edge pixels	

bopen

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	with	*<number>*	picture specifying structuring element data
	mask	*<number>*	mask picture to restrict processing to specified region
	times	*<number>*	number of times to apply structuring element data
	edge	*<number>*	edge value
options:	**source/output**		fix value of source/output edge pixels according to the value of the **edge** key

You use **bopen** to carry out generalised binary opening of foreground regions. The way in which the opening is carried out is determined by the structuring element specified by means of the **with** key. The output picture will contain the binary, opened result, which represents the area swept out by the structuring element while it is contained entirely within the foreground regions of the source image. Opening smoothes the boundary of foreground regions by removing small protrusions, it breaks narrow isthmuses between regions and it removes regions smaller than the structuring element. Opening is idempotent – opening an already opened image does not change the image.

Bopen has exactly the same keys and options as the commands **berode** and **bdilate** and it produces exactly the same result as using the command **berode** followed by **bdilate** with the same settings for the controlling keys and options, but in less time.

Examples

```
create 99 size 3 value 1; bopen 1 with 99
```

This example opens picture 1 with a square, 3 by 3 structuring element.

```
bopen 1 2 with 99 times 3
```

This command is equivalent to the following

```
berode 1 2 with 99 times 3; bdilate 2 with 99 times 3
```

Description

Opening a binary image with a structuring element is equivalent to eroding and then dilating the image with the same structuring element.

bopen

For example,

```
0 0 0 0 1 0 1                                  0 0 0 0 1 0 0
0 0 1 1 1 1 0                                  0 0 1 1 1 1 0
0 1 1 1 1 1 0                    1             0 1 1 1 1 1 0
0 1 1[1]1 1 0          (o)   1[1]1 =           0 0 1[1]1 1 0
0 1 0 0 1 1 0                    1             0 0 0 0 1 0 0
0 0 0 1 1 1 0                                  0 0 0 1 1 1 0
0 0 0 0 1 0 0                                  0 0 0 0 1 0 0
```

where (o) denotes opening and [ ] marks the origin of the picture and the structuring element. Compare this with the result obtained by erosion followed by dilation,

```
0 0 0 0 1 0 1                                  0 0 0 0 0 0 0
0 0 1 1 1 1 0                                  0 0 0 0 1 0 0
0 1 1 1 1 1 0                    1             0 0 1 1 1 0 0
0 1 1[1]1 1 0          (-)   1[1]1 =           0 0 0[0]1 0 0
0 1 0 0 1 1 0                    1             0 0 0 0 0 0 0
0 0 0 1 1 1 0                                  0 0 0 0 1 0 0
0 0 0 0 1 0 0                                  0 0 0 0 0 0 0

0 0 0 0 0 0 0                                  0 0 0 0 1 0 0
0 0 0 0 1 0 0                                  0 0 1 1 1 1 0
0 0 1 1 1 0 0                    1             0 1 1 1 1 1 0
0 0 0[0]1 0 0          (+)   1[1]1 =           0 0 1[1]1 0 0
0 0 0 0 0 0 0                    1             0 0 0 0 1 0 0
0 0 0 0 1 0 0                                  0 0 0 1 1 1 0
0 0 0 0 0 0 0                                  0 0 0 0 1 0 0
```

where (–) denotes erosion, (+) denotes dilation.

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

Each layer of the picture specified by means of the **with** key defines a structuring element to be applied to the source picture. Non-zero pixels define the components of the structuring element. Any zero pixels are ignored. The origin of the picture defines the origin of the structuring element.

bopen

For example,

```
0 0 0 0 0                          0 0 1 0 0              1
0 0 1 0 0            1             0 0 1 0 0              1
0 1[2]1 0      => 1[1]1            0 0 1 0 0      =>     [1]
0 0 1 0 0            1             0 0 1 0 0              1
0 0 0 0 0                          0 0 1 0 0              1
```

If there is more than one layer, the structuring elements thus defined are applied in sequence, starting with layer 1. This allows you to specify large structuring elements in decomposed form. You can also use the **times** key to apply the sequence of structuring elements more than once. The structuring element represented by a decomposed set is obtained by combining the decomposed set using dilation.

For example,

```
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1          1 1 1      1 1 1      1 1 1      1 1 1
1 1 1 1[1]1 1 1 1    =     1[1]1 (+)  1[1]1  (+)  1[1]1  (+)  1[1]1
1 1 1 1 1 1 1 1 1          1 1 1      1 1 1      1 1 1      1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1

1 1 1                          1
1[1]1        =      1[1]1 (+)  [1]
1 1 1                          1
```

As you can see, the 9 by 9 structuring element can be decomposed into four 3 by 1 structuring elements and four 1 by 3 structuring elements. As dilation is a commutative operation, the structuring elements can be applied in any order. The simplest way to apply this example is to create a two-layer picture with the 3 by 1 structuring element in one layer and the 1 by 3 structuring element in the other layer, specify this picture by means of the **with** key and set the **times** key to 4.

The **mask** key can be used to restrict processing to specified regions of the image. The options **source** and **output** and the **edge** key allow you to control edge effects. For more detailed information about these and related topics, consult the documentation for the commands **berode** and **bdilate**.

bopen

Notes

see also: **berode, bdilate**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	none	valid picture number
mask	none	valid picture number
times	1	postive integer
edge	0	real number
source/output	do not process undefined pixels	

box

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	with	*<number>*	picture containing training statistics
	stride	*<number>*	sampling interval across and down the picture
	position	*<x>, <y>*	subregion position
	size	*<x>, <y>*	size of subregion
options:	**left/right, top/bottom**		subregion position

The **box** command classifies a picture using a box classifier (*parallelepiped*). This command is used for *Remote Sensing* applications.

Examples

```
box 1 2 with 3
```

Perform a box classification of picture 1 into picture 2 using the training data from picture 3.

```
box 2:37 to 3:15 with 2:38 stride 4
```

Perform a box classification of picture 2037 into picture 3015, using the training data from picture 2038. The output picture, 3015, is a quarter the size of the original image.

Description

A box classification of the source picture is carried out based on the ranges of data described in the **with** picture. (Use the **learn** command to create this picture, containing training statistics of polygonal regions).

The classification process sorts the pixels of a multi-spectral image into separate classes on the basis of the pixels spectral characteristics. In the case of supervised classification, as provided by **box, mindistance** and **likelihood**, the spectral characteristics of each class are determined from sets of training data. The training data are formed from known areas of an image to provide the statistics of each class that is required.

The command gives a warning message if it is possible for *multiple* classifications to arise, although this does not mean that such classifications will occur. The classification process takes the first class that matches a given pixel.

box

The number of classes that the **box** command can process depends on the size of a Semper row buffer and on the number of layers in the source picture. Note that classes run from zero, indicating unclassified, to *n*, where *n* is the number of layers in the **with** picture. The *Remote Sensing* commands, **likelihood** and **mindistance** also perform picture classification.

The **stride** key gives the stride across and down the picture allowing a fast classification of the entire picture. By default **stride** has a value of 1 and the whole picture is classified. You can also indicate a subregion of the source picture, in which case only the subregion is processed, and the output is correspondingly smaller.

You can specify a subregion using the standard keys and options **size**, **position**, **left/right** etc. For further details of defining a subregion refer to *Appendix C, Semper Keys and Options.*

Notes

see also: **covariance**, **learn**, **mindistance**, **likelihood**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none*	valid picture number
stride	sampling interval 1	positive integer
position	position 0,0	within bounds of the picture (integers)
size	whole picture	less than or equal to the size of the picture (integers)

break

[*<variable name>*]	resume execution after the named **for** loop. The variable name is optional.

The **break** command causes Semper to resume execution at the end of the named **for** loop.

Examples

```
for n=1, 5; select n; for y 5,-5; for x -5,5; if p(x,y)=0 break n
loop x; loop y; display; loop
```

This sequence of commands only displays pictures which do not have zero pixels near the origin.

Description

If you do not specify a loop variable name, the innermost active loop is assumed by Semper. For example:

```
for n=1,10; for m=n,10; for x=-5,5; .. ; loop; break; loop; loop
```

in the above sequence the **break** command jumps beyond the *m* loop, to the next cycle of the *n* loop.

Notes

see also: **for, loop, next**

cache

This syntax is specific to... Windows systems, Sprynt systems and workstations running Unix

keys:	**number**	*<number>*	number of cache buffers
	size	*<number>*	cache buffer size (bytes)
	device	*<number>*	if **flush**, **memory** or **free**, disc device number
options:	**flush**		flush contents of cache buffer or specified device to disc
	memory		allocate permanent memory buffer for specified device
	free		free memory buffer assocated with specified device
	show		list current settings for disc cache perameters

Use the **cache** command to manage the way in which disc input/output is buffered in memory. This can lead to signifcant reductions in the amout of data traffic to and from the hard disc and it can reduce the total number of input/output requests. Data from a disc device can be buffered in a separate memory buffer allocated to that device or it can be buffered in Semper's disc cache. The disc cache is a collection of memory buffers which are shared between all the disc devices which are not separately buffered. With the **cache** command you can allocate and free separate memory buffers, display/change the number and size of cache buffers and you can force buffered data to be flushed out to disc.

Examples

```
cache show
```

This command lists the current disc cache parameter settings: number of cache buffers, buffer size and total cache size.

```
cache number 200
```

This command changes the number of cache buffers to 200, keeping the buffer size the same.

```
cache number 20 size 65536
```

This command reconfigures the cache to consist of 20 buffers of 65536 bytes each giving a total cache size of 1310720 bytes.

```
cache number 0
```

This command frees all cache buffers, effectively turning off disc caching.

cache

```
cache flush
```

This command causes all data modified in the cache to be written to disc.

```
cache device 2 flush
```

This command causes all modified data associated with device 2 to be written to disc.

```
cache device 3 memory
```

This command causes all data in device 3 to be buffered in a separate dynamically allocated memory buffer.

```
cache device 3 free
```

This command flushes and frees the separate memory buffer associated with device 3.

Description

Data can be buffered, either in one or more cache buffers which are drawn from a pool of cache buffers (the disc cache), or else in a buffer which is permanently allocated to the file with which the data is associated. The first approach allows you to make optimum use of limited memory resources by only caching data which needs to be accessed amongst all the disc files that are currently open. The second approach reduces actual disc input/output to an absolute minimum – data is read in once only, and written out when the buffer is flushed or freed. On systems which support virtual memory, you must always keep in mind the possibility that buffered data will be paged onto disc when the amount of allocated virtual memory exceeds the amount of real physical memory.

With the **cache** command you can, at any time, allocate and then free a separate memory buffer for any disc device which is currently assigned. You specify the device number with the **device** key and you allocate or free the memory buffer by specifying the **memory** or **free** options respectively. The device number is determined by the **assign** command when you open the disc file. When such a buffer is allocated, all the data on disc is read just once into memory. All subsequent access to the data is made via the memory buffer, thereby avoiding all disc input/output, until the buffer is freed or flushed.

There is no point in using the **memory** option with a temporary or scratched disc file unless you specifically intend to use the **free** option later during the same Semper session. If you do not intend to free the memory buffer before closing and deleting the file, you may as well use the **assign memory** command to open a memory-based device which will not tie up space on the hard disc.

The **cache** command allows you to control the number and size of cache buffers which make up the disc cache. The buffer size must be at least a multiple of 8 bytes and, on some systems, it may also have to be multiple of the operating system's page size or some other unit of size. The **cache show** command will list any constraints on the buffer size.

cache

Semper will automatically create a certain size of disc cache at the start of a session. The default size of the cache will vary according to the host system, but it should at least give reasonably good performance when processing medium sized images. Even when the cache is not large enough to hold all the data being processed by a particular Semper command, the disc cache can still provide some performance benefits because it reduces the number of input/output requests. On some systems, each input/output request can incur large operating system overheads.

If the number of buffers or the buffer size is set to zero, disc caching is disabled and disc data will be accessed directly from the hard disc. On systems which support virtual memory, the size of the disc cache needs to be managed so that it is not so large that it puts too heavy a burden on the operating system. If the size of the cache exceeds the amount of free physical memory, data will be swapped out of memory to make room. If this leads to the point where the code for active processes has to be swapped out, the performance of the whole system will suffer. Consult the on-line help entry for the **cache** command to find out how to get some indication of the demands being made for your system's memory. On systems which do not support virtual memory, the size of the disc cache will be limited by the amount of available memory. Also, allocating all of the available memory to the disc cache may cause problems elsewhere when further requests to allocate memory are refused.

With the **flush** option you can cause all modified data in the cache to be written to disc. With the **device** key, you can restrict the operation of the **flush** option to a specified disc device. Reconfiguring the disc cache causes its contents to be written to disc, as if the **flush** option had been specified. If the disc cache is large, this could take a noticeable length of time. Likewise, when a disc device is deassigned, its contents are flushed out to disc, except when the file is to be deleted (either because the **delete** option was specified in the **deassign** command or because the device is a temporary or scratch disc).

The **flush** command has the same effect as using the **cache flush** command.

The **show** option causes the current disc cache parameters to be listed on the console output stream.

Notes

see also: **assign, deassign, flush**

Defaults and Ranges

keys/options	defaults	range
number	*current value*	zero or positive integer
size	*current value*	zero or positive integer
device	*none*	integer in range 2 to system limit (type **show system**)

calculate

keys:	**[ ]**	*<numeric expression>*	picture expression to be calculated
	to	*<number>*	output picture

Use **calculate** to calculate more or less arbitrary mathematical combinations/functions of pictures.

Examples

```
calculate (:10+:11)/2 to display
```

This command displays the average of pictures 10 and 11.

```
calculate exp(-rr/2/25^2)
```

This command fills the current picture with gaussian having a sectional *r.m.s.* of 25 pixels.

```
fourier; calculate :sel*cc(:t)/(msq(:t)+.2); image
```

This command applies a *Wiener inverse filter* to compensate the current picture for degradation with a spatial frequency response in picture *t*.

Description

You define the expression to be calculated with the first assumed key. If you omit **to**, output goes to the first picture mentioned in the expression, or to the picture number held in the variable *select*, if the expression does not refer to a picture.

Note that the first picture mentioned in the expression determines:

- the default output class and form
- the coordinate origin

If you do not specify a picture in the expression, the output picture is used to determine the above. If the expression involves no picture number, for example:

```
calculate sin(kx*x+ky*y)
```

the output picture must already exist so as to define a coordinate system and picture dimensions.

calculate

Semper interprets the expression used with **calculate** once per row, not once per pixel, giving reasonable efficiency. It has the same syntax as all Semper expressions, except in the following respects. (For details of expression syntax, refer to *Appendix B: Semper Expression Syntax*).

- items of the form *:n* or *d:n* are taken to denote pixels from the indicated picture, rather than constants (that is, :5/2 means picture 5 halved, while *t*/:2 means *t* times the reciprocal of picture 2)
- the names *x, y, z* and *rr* are interpreted as x, y, z coordinates and the squared distance from the picture origin respectively, rather than as constants (see second example given above)
- the arithmetic operators + − * / accept complex as well as real operands
- the following additional or different functions are recognized:

c(x,y)	complex number *x+iy*
re(z), im(z)	real, imaginary part of complex *z*
cc(z)	complex conjugate of complex *z*
phase, modulus, msq	take a single argument which may be complex, rather than up to two real arguments

Although **calculate** can cope with most expressions, there is a limit to the complexity expressions which it can process. First of all expressions are limited by the need to stack intermediate results when evaluating nested expressions. If you encounter this limit, **calculate** will return with error 18: "Expression stack overflow". The size of the expression stack is equal to the number of row buffers. The second limitation is to do with the number of references to pictures which may be contained in an expression. The number of pictures is limited to the size of Semper's logical picture table less one entry for the destination picture. If there are too many pictures referred to in an expression, **calculate** returns with error 56: "Too many pictures open simultaneously". You can find out how many row buffers and picture opens are supported on your system by typing the command **show system.**

Notes

see also:	**show system,** *Appendix B: Semper Expression Syntax*
multi-layer pictures:	fully supported
forms used internally:	fp, complex

calculate

Defaults and Ranges

keys/options	defaults	range
[ ]	*none; supply expression*	see *Appendix B: Semper Expression Syntax*
to	first picture number in expression; otherwise number held in the variable *select*	valid picture number

cell

keys:	**add**	*<number>*	add a cell to the specified menu
	id	*<number>*	define which cell to use
	in	*<number>*	place cell in specified panel
	begins	*'<text>'*	specify the *interaction begins* action for a cell
	changes	*'<text>'*	specify the *state change* action for a cell
	ends	*'<text>'*	specify the *interaction ends* action for a cell
	cycle	*<number>*	cycle a cell through a specified number of states
	column	*<number>*	position a cell at the column number on its menu
	row	*<number>*	position a cell at the row number on its menu
	offset	*<x>, <y>*	position a cell at the specified offset on its menu
	position	*<x>, <y>*	position a cell on its panel at the given coordinates
	name	*'<text>'*	specify a name for the cell
	text	*'<text>'*	the cell is to contain the specified text
	title	*<number>*	the cell is to contain the specified picture number title
	size	*<x>, <y>*	define a minimum *x* and *y* size for a cell
	background	*<number>*	number of cell background
	foreground	*<number>*	number of cell foreground
options:	**create**		create a cell
	destroy		destroy a cell
	check/flash/invert/tick		use check marks/flashing/inverse video/tick marks to highlight a cell
	box		draw a box around a cell
	drop		deactivate a cell's menu on selecting the cell

The command **cell** controls the creation and operations of the Semper 6 *Plus* cell element. A cell is like a labelled box. It contains Semper commands that are performed when the cell is selected. For a description of Semper 6 *Plus* elements, refer to the *Semper 6 Plus User Interface Guide* contained in the *Semper 6 Guide*.

cell

Examples

```
cell create text 'circle' invert
```

This command creates a new text cell with the contents '*circle*' on the current panel, with inverse video highlighting.

```
cell create text 'move' add m5
```

This command creates a cell with the contents '*move*' and adds the cell onto the menu number with its identifier held in the variable *m5*. By default, the cell has flash highlighting.

```
cell id c00 box
```

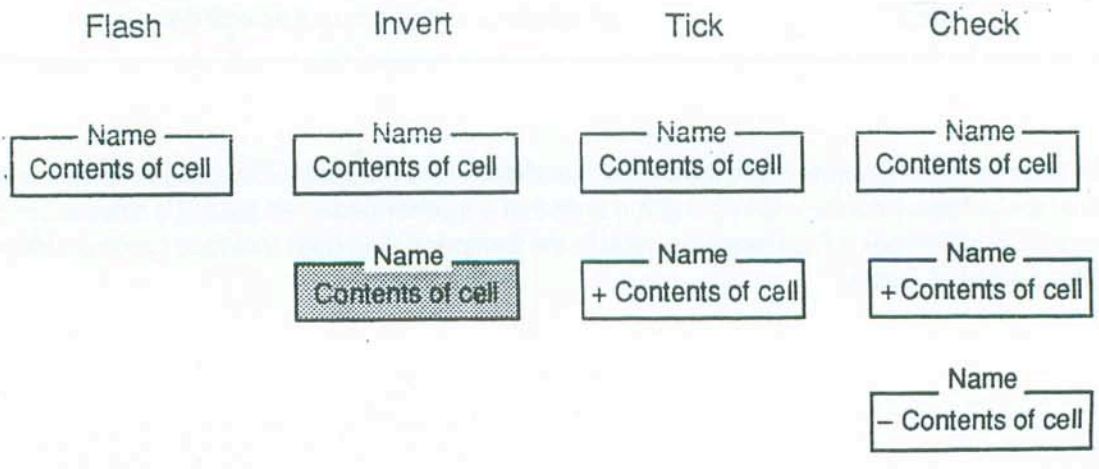
This command draws a box around the text of an existing cell. The cell's identifier is stored in the variable *c*00.

```
cell id c00 begins 'type "START"' ends 'type "END"'
```

This command sets the action to be performed when the cursor enters the cell whose identifier is stored in the variable *c*00. When the cursor enters the cell the string START is displayed and on leaving the string END is displayed.

Description

A Semper 6 *Plus* cell can be created using the **create** option and removed using **destroy**. Use the **id** key to define a cell to act upon. There are four ways of highlighting a cell using flashing (**flash**), inverse video (**invert**), tick marks (**tick**) and check marks (**check**). The default is *flashing* highlighting. The diagram below illustrates the different types of Semper 6 *Plus* cell.



cell

A cell is more than a box containing a name. You can specify that a cell contains Semper commands that are acted on when that cell is selected. A cell can go through the following transitional states:

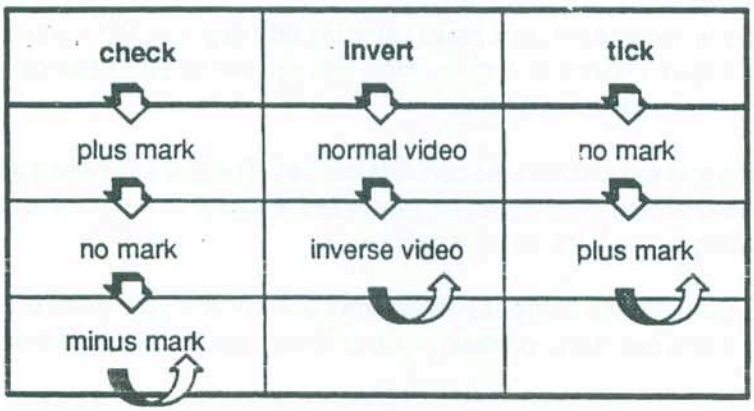
- *interaction begins*
- *state changes*
- *interaction ends*

You define the action taken for each of these states using the **begins**, **changes** and **ends** keys. These keys allow you to specify as text the commands, keys and options to be executed by Semper when interaction with a cell begins or ends, or a cell changes state.

There are two ways to interact or change the state of a cell:

- use the **cycle** key to cycle through the states of a cell
- use the mouse to enter or select a cell

The **cycle** key cycles a cell through its highlight states. The table given below details the cycles of each style of cell. For example, if a *check* cell currently has no mark, the command **cell cycle 2** changes the highlight to a *plus* mark. Note that for **flash** highlighting there is only one state, that is *not flashing*, but you can use the **cycle** key to cause a cell to flash a specified number of times.



cell

The actions taken when the mouse enters a cell, the cell is selected and the mouse leaves a cell are detailed in the table below and are dependent on the highlight style of the cell. Note that this table only applies to a cell placed on a panel and not on a menu.

style	mouse enters	cell selected	mouse leaves
check	no change	cycle by 1	no change
flash	no change	cycle by 2	no change
invert	cycle	no change	cycle
tick	no change	cycle by 1	no change

By default, a cell is created on the current panel but you can use the **in** key to specify a different panel. Use the **position** key to define the position of a cell on its panel.

You can move a cell from a panel to a menu using the **add** key. Use the keys **row** and **column** or **offset** to specify a cell's location on a menu. Note that you can use the **drop** option to deactivate a cell's menu. This saves processing time.

Use the **title** or the **text** key to define the contents of a cell. The **title** key copies up to the first twenty characters of the specified picture title or substitutes '*Picture: n*' if no title exists. The **text** key accepts a text string as the contents of a cell.

You can name a cell using the **name** key. This allows you to identify the purpose of a cell on a panel or menu. If you name a cell, a box is drawn around the cell and the cell name is displayed on the top line of the box.

You can change the attributes of a cell using the keys **size**, **foreground** and **background** and the option **box**. The **size** key determines the minimum size of a cell. Use **foreground** and **background** to change the foreground and background colours of a cell. The **box** option draws a box around a cell, similar to that drawn when a cell is named.

Notes

see also: **menu**, **panel**, **textfield**
Semper 6 Plus User Interface Guide

variables used: *eno (cell number,* if **id** not specified*)*
*pno (*panel number, if **in** not specified)

variables set: *eno (*set by the **create** option to the cell identifier number)

cell

Defaults and Ranges

keys/options	defaults	range
add	*none*	valid menu number
id	current cell, held in variable *eno*	valid cell number
in	current panel, held in variable *pno*	valid panel number
begins	*none*	text string containing valid Semper commands
changes	*none*	text string containing valid Semper commands
ends	*none*	text string containing valid Semper commands
cycle	*none*	positive integer
column	column 1	positive integer
row	row 1	positive integer
offset	offset 0,0	integer
position	position 0,0	integer
name	*none*	text string
text	*none*	text string
title	*none*	valid picture number
size	Semper 6 *Plus* determines appropriate size of cell	positive integer
background	background colour of panel/menu	integer: range is machine dependent
foreground	foreground colour of panel/menu	integer: range is machine dependent
check/flash /invert/tick	flash highlighting	

cget

This command is specific to... PC + MRC500 framestore

keys:	**[to]**	*<number>*	specify Semper picture number to contain the new image
	name	*'<text>'*	read from the specified file

You use **cget** to read a DOS file which contains an image written by the *MRC500 microscope control software*.

Examples

```
cget 23 name 'two'
```

This command reads the file called *two.pic*, written by the MRC500 microscope control software, into Semper picture 23.

Description

Pictures that are written by the MRC500 microscope control software can be two or three-dimensional, and in either *integer* or *byte* form. **cget** creates a Semper image to match the specifications of the file.

cget searches for files in the current directory and then throughout the DOS PATH. You can avoid a time-consuming path scan by specifying a full path name with the filename.

Notes

multi-layer pictures:	fully supported
forms used internally:	byte, integer
see also:	**cput**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid Semper picture number
name	*none*	valid filename

chull

keys:	[from]	<number>	source picture
	[to]	<number>	output picture

Use the **chull** command to generate the convex hull for the regions defined in the binary source image. If you imagine an elastic band stretched around all foreground regions in the source picture, the convex hull is the area enclosed by the elastic band. The convex hull therefore includes in its boundary all the extreme points of the source regions, and it has no concavities. You can detect the presence of concavities in binary objects by looking at the difference between the objects and their convex hulls. If there is no difference between an object and its convex hull, the object must be convex. **Chull** also returns the convex perimeter and area in the variables *p* and *a*.

Examples

```
chull; type 'convex perimeter and area: ',p,a'
```

This example replaces the current picture with its convex hull and types the perimeter and area of the convex hull.

```
chull 1 2; calculate :2 & ~:1
```

This example calculates the convex deficiency of regions in picture 1 and puts the result in picture 2.

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels. The output picture will contain the convex hull of the foreground regions. Note that the source image may contain any number of separate foreground regions. **Chull** always produces a single convex hull which includes all the foreground regions in the source picture.

Chull returns the perimeter and area of the convex hull in Semper variables *p* and *a*. The perimeter and area correspond exactly to the values which the **analyse** command would return for the convex hull.

The range of the final result is stored in the output picture label.

Notes

variables set:	*p*	(perimeter of convex hull)
	a	(area of convex hull)
see also:	**analyse**	

chull

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

close

***close** takes no arguments*

You can use **close** (in some installations) to release material to the display device that is currently retained in main memory buffers. This is usually used if the display is a hardcopy device.

cls

***cls** takes no arguments*

Use **cls** to clear the terminal screen only (to clear the framestore use the **erase** command).

compress

keys:	**device**	*<number>*	device to be compressed
options:	**verify**		verify the command processing at the console

The **compress** command compresses the fragmented free space available on a disc.

Examples

```
compress
```

This command compresses the current device.

```
compress device 4
```

This command compresses device 4.

```
compress noverify
```

This command compresses the current device, without sending information about the process to the console.

Description

If a disc device contains a useful amount of free space, but is too fragmented for Semper to use it, use **compress** to collect all the data at the beginning and all the free space at the end. Also, if a program library is full because of repeated addition of new versions to the end, **compress** recovers the space occupied by the deleted versions. By default, the details of the **compress** operation are verified at the console. Use the **noverify** option to suppress this information.

If you cancel this command while it is running, Semper responds as soon as possible. However, the result is a partially compressed disc in which some, but not all, the data are at the beginning of the disc.

Defaults and Ranges

keys/options	defaults	range
device	current device, held in the variable *cd*	integer in range 1 to system limits (type ***show system***)
verify	verification on	

contour

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	times	*<number>*	magnification factor
	levels	*<number>*	number of contour levels
	size	*<x>, <y>, <z>*	dimensions of subregion to be contoured
	position	*<x>, <y>, <z>*	displacement of subregion position
	layer	*<n1>, <n2>*	range of layers in subregion
	mark	*<number>*	mark source subregion
		<yes *or* no>	
options:	**preset**		set contour range from current *min, max*
	letter		letter top of partition with picture number and title
	border		mark picture border
	left/right, top/bottom, far/near		subregion position

Use the **contour** command to draw picture intensity contours on a display overlay.

Examples

```
contour 52 times 3
```

This command draws a contour map on the display overlay of picture 52 at three times the normal size.

```
contour 52 times 3 to display:4
```

This command performs the same functions but in display:4.

```
contour display:3 levels 9
```

This command adds 9 contours to display:3.

Description

You can contour a subregion using the standard subregion keys (**size**, **partition**, **layer**, **left/right**, **top/bottom**, **far/near**). Refer to *Appendix C, Semper Keys and Options* for further detail. Successive layers are output to successive display frames. You can mark the subregion on the display using the **mark** key.

contour

Use the **times** key to specify a magnification factor for contouring. Set the variables *min, max* and the key **level** to set your own contour heights and specify the option **preset** to set the range. If you do not specify *min, max* and **preset**, Semper uses the lowest and highest pixel values found in the picture. Note that contour levels do not include *min* and *max* themselves.

The **contour** command is closely related to the **display** command. The display partition is lettered and bordered in the same way, depending on the options **letter** and **border**. The size and position of the display is identical, so that superposition, as in the second command example, is possible. Graphical annotation and cursor measurement are made in the same coordinate system. Use the command **spc** if you want a permanent record of contours in a picture, by alteration of its pixels.

Notes

restrictions:	not 1-D
display marking:	region, if subregion
multi-layer pictures:	fully supported
forms used internally:	fp,complex
variables used:	*min*, *max* (if **preset**, supplies pixel range spanned by contours)
variables set:	*min*, *max* (unless **preset** is specified, pixel range for contouring)
see also:	**display, spc**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current display picture, held in the variable *display*	valid picture number
times	times 1	positive integer
levels	5	positive integer
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0,0,0	within bounds of picture (integer)
layer	all layers unless variables *si3/po3* set	1 to total number of layers in the picture (integers)
mark	mark off	see *Appendix C*
preset	lowest/highest pixel values used	
letter	lettering on	
border	bordering on	

copy

keys:	**[from]**	*<number>*	source picture
		<n1>, <n2>	range of source pictures
	[to]	*<number>*	output picture, or first of a range of output pictures, if you copy a range of source pictures
	program	*'<text>'*	program name to be copied
	as	*'<text>'*	new name for copy of a program
	device	*<number>*	program library to which program is copied
options:	**verify**		verify copying process at the console

Use **copy** to copy a picture or set of pictures to another device, for example, a tape. **copy** can also be used to copy a program or to change the format of a picture.

Examples

```
copy display to 3
```

This command copies picture *display* to picture 3.

```
copy 52 to 4:0 noverify
```

This command copies picture 52 to the end of a tape assigned as device 4, without sending information about the copying process to the console.

```
copy 100,200 to 5:1
```

This command copies any pictures numbered between 100 and 200 on the current device to 5:1, 5:2, etc. in succession.

```
copy 2 integer
```

This converts picture 2 to an integer form.

```
copy program 'myprog' as 'newprog'
```

This makes a copy of program *myprog.spl* called *newprog.spl*, provided that a program called *newprog.spl* does not already exist.

copy

Description

copy works in two modes; *picture mode* (by default) and *program mode* (using the **program** key).

In *picture mode*, each characteristic of a picture is copied to the output picture, except for the write-protection status. (You can protect the new picture using the command **wp**). If you change the form of a picture during copying (for example if picture 1 is in floating point form and you change to byte representation using **copy 1 to...byte**) the range of information is deleted from the output picture.

In *program mode*, the output is copied within a device. If you omit a new name for a copied program Semper asks for one at the terminal. To copy to a new device use the **device** key, in which case the output name defaults to the source name. The program copied is the first one found in the current device search order, if more than one version exists. The keys **from** and **program** are mutually exclusive.

The **verify** option sends details of the **copy** operation to the console; the **noverify** option suppresses this information. Note that you can use the general keys **byte**, **integer**, **fp** and **complex** with the **copy** command to change the picture format, for example, **copy 3 fp**. Refer to *Appendix C: Semper Keys and Options* for more detail.

Notes

multi-layer pictures: fully supported
forms used internally: all
see also: **wp**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
program	*none:* ***copy*** defaults to copying pictures	valid program name
as	original program name, if you specify a ***device***	valid program name
device	first device in current search order	integer in range 1 to system limits (type ***show system***)
verify	verification on	

correct

keys:	[from]	<*number*>	source picture
	[to]	<*number*>	output picture
	with	<*number*>	picture containing reference modulus
	verify		verify the processing at the console

correct resets the modulus of a *Complex* picture to that of a given reference picture, without altering the phase. This command is commonly used in iterative phase-determination algorithms of the *Gerchberg* type. For information on *Complex* pictures, refer to *Appendix A: Picture Types*.

Examples

```
correct with 40
```

This command alters the current picture to have the modulus given in picture 40.

```
correct 1 to 2 with 4 verify
```

This command applies the alteration to picture 1, storing the result as picture 2. The **verify** option is used to send information about the processing to the console.

Description

correct resets the modulus of each pixel of the source picture (**from**) to the reference modulus given in the reference picture (**with**). Note that it copies zero-valued source pixels unchanged. **correct** calculates a mismatch (error) criterion, prints this to the console if you specify the **verify** option and returns a variable *e*. This variable consists of the summed squared difference between the source modulus and the reference, expressed as a fraction of the summed squared reference.

Notes

multi-layer pictures:	faulted
forms used internally:	complex
variables set:	*e* (mismatch criterion fractional summed squared difference)

correct

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none:* specify a picture number	valid picture number
verify	verification off	

covariance

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	position	*<x>, <y>*	position/offset of subregion
	size	*<x>, <y>*	dimensions of subregion
options:	**left/right, top/bottom**		subregion position

The **covariance** command forms the covariance matrix of a multi-layer picture. Use this command in *Remote Sensing* applications.

Examples

```
covariance 1 2
```

Calculate the covariance matrix of picture 1, storing the result in picture 2

```
covariance 1 2 top left size 256,256
```

Calculate the covariance matrix of picture 1, storing the result in picture 2. Only the top left 256 by 256 subregion is used in calculating the covariance information.

Description

The **covariance** command calculates and outputs the covariance matrix of the input picture. It also calculates the mean, standard deviation and range of each layer. The number of layers which this command can process is both picture and system dependent; large (wide) pictures have to have fewer layers.

Use the standard subregion keys and options, **position**, **size**, **left/right**, **top/bottom** to define a subregion. For further detail, refer to *Appendix C, Semper Keys and Options.*

It is possible to use the **covariance** command to identify rectangular training regions for use in supervised classification, and stack the resulting (covariance) pictures together.

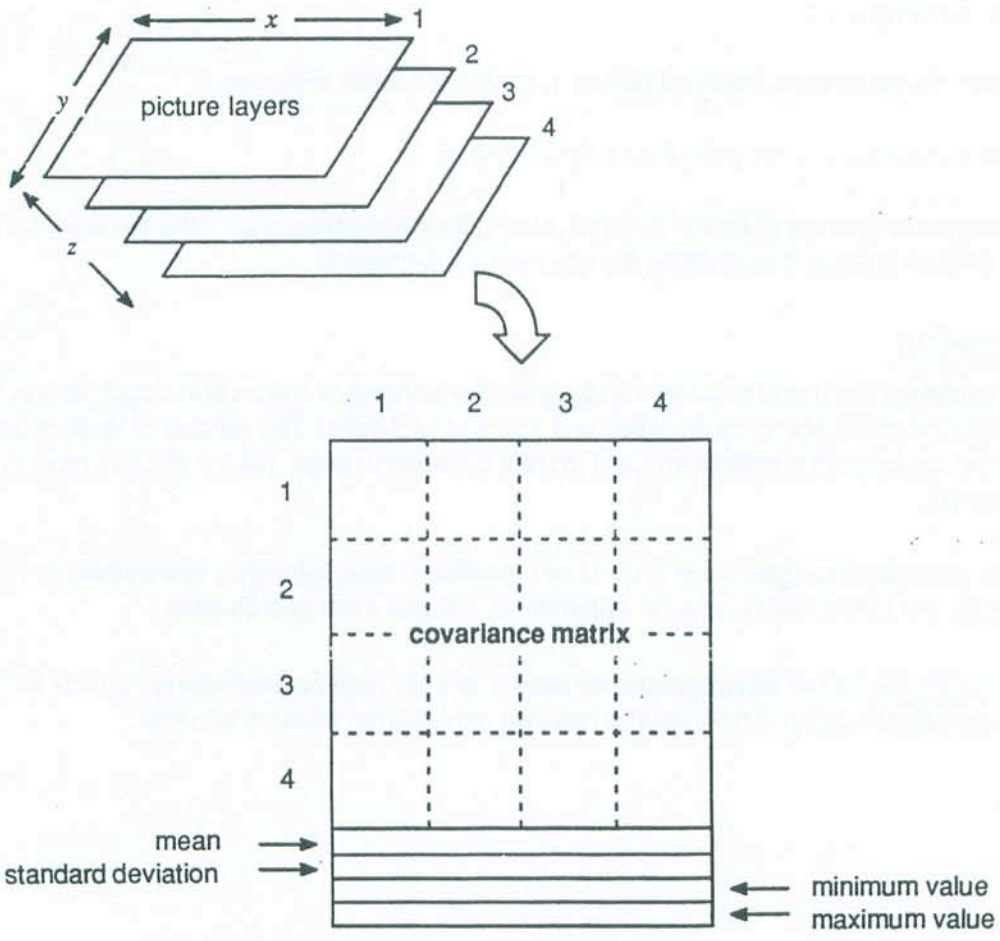
covariance

The covariance matrix is a measure of the degree of correlation of pixel values between different layers of a picture. The matrix is defined as:

$$C_{ij} = (x_i - m_i)\ (x_j - m_j)$$

where: C = covariance
x = pixel value in layer
m = mean value of layer
i = 1 to number of layers
j = 1 to number of layers

The following diagram shows how the covariance matrix is stored as a picture.



covariance

Notes

see also: **box, learn, likelihood, mindistance**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
position	position 0,0	within the bounds of the picture (integer)
size	whole picture	less than or equal to the size of the picture (integer)

cput

This command is specific to... PC + MRC500 framestore

keys:	**[from]**	*<number>*	Semper picture to be written
	name	*'<text>'*	write picture to the specified file
options:	**new**		allow any existing file of the same name to be overwritten

You use **cput** to write a picture to a DOS file that can subsequently be read by the *MRC500 microscope control software.*

Examples

```
cput.23 name 'one'
```

This command writes picture 23 to a file called *one.pic.*

```
cput 24 new name 'two'
```

This command writes picture 24 to a file called *two.pic.* If the file *two.pic* already exists then it is overwritten.

Description

Pictures produced by **cput** can be recovered by **cget.**

cput converts all pictures to *byte* format before writing the output file. The dynamic range of a picture should therefore not exceed 0...255.

Notes

multi-layer pictures:	fully supported
forms used internally:	byte
see also:	**cget**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid Semper picture number
name	*none*	valid filename

create

keys:	**[ ]**	*<number>*	picture to be created
	size	*<x>,<y>,<z>*	dimensions of picture
	value	*<number>*	value to which pixels are initialised
		<n1>,<n2>	real and imaginary value for complex picture
options:	**image/fourier/spectrum/ correlation/undefined/ walsh/plist/lut**		class of picture
	list/curve		type of *Plist* picture, if **plist**
	open/closed		type of curve, if ***plist* curve**
	byte/integer/fp/complex		data form of picture

Use the **create** command to:

- initialise disc pictures to constant values, possibly for calculation or pasting operations
- test if there is enough space for a picture
- redeclare displays to use the display memory in a different way

Examples

```
create 1 size 256 byte; calculate 100+.13*x-.096*y
```

This example generates a *Byte* picture 1 filled with a planar intensity ramp.

```
min=0 max=255 create display size 512
```

This command allows you to treat a 512x512 display, initially created with a different black-white range, as if its range were 0-255.

```
create 52 fourier complex size 129, 256 value 0,0; origin left
p 2,3=p,q
```

This sequence of commands creates a half-plane *Fourier* picture containing a single non-zero Fourier coefficient at (2,3).

Description

By default, the **create** command creates an *Image* picture in floating point form with the same dimensions as the current picture. You can change these defaults using the **size** key, the class option (**fourier/spectrum** etc) and the form option (**byte/integrer/fp/complex**). Note that when creating a display picture, **create** does not erase the associated partition when the **erase** option is set (unlike most Semper commands).

create

For a *Plist* picture, you can use the additional options **list/curve, open/closed** to specify the type of *Plist*:

...plist or **...plist list**	(list)
...plist curve or **...plist open curve**	(open curve)
...plist closed curve	(closed curve)

create can be used to access byte data on magnetic tape in files that do not have the standard 256 byte labels used by Semper. Use **create** to specify the picture size; Semper assumes class *Image* and byte form. A picture opened in this way is not open permanently, so you need to use **copy** to copy it to a device.

Note that usually you do *not* need to create a picture before using it as output for a Semper command, as commands create their own output pictures as necessary, using the (assumed) key **to.**

Notes

multi-layer pictures:	fully supported
forms used internally	all
see also:	**copy**

Defaults and Ranges

keys/options	defaults	range
[]	current picture, held in the variable *select*	valid picture number
size	dimensions of current picture	positive integers
value	no pixel initialisation	real numbers
image/fourier/ spectrum/correlation/ undefined/walsh/ plist/lut	image	
list/curve	list	
open/closed	open	
byte/integer/fp complex	fp	

ctf

keys:	**[from]**	*<number>*	source picture used by options ***add*** or ***multiply***
	[to]	*<number>*	output picture
	size	*<x>, <y>*	if no source picture, dimensions of picture
	defocus	*<number>*	defocus value (underfocus positive)
options:	**add/multiply**		add *ctf* to source/multiply source by *ctf*
	halfplane		if no source picture, generate right half-plane output
	phase		produce weak-phase object *ctf*
	amplitude		produce weak-amplitude object *ctf*
	wave		produce 'wave' form of *ctf*, that is, exp-i (gamma)
	squared		produce squared *ctf* (not necessarily modulus squared)
	cc		produce complex conjugate of *ctf*
	physical		interpret variables describing imaging conditions in physical units (*nm*, *mrad*) rather than reduced. Need to set variables *cs* and *kv*.
	verify		verify processing at the console

The command **ctf** produces or applies *contrast transfer functions* (Fourier transform filters), that characterise imaging in high resolution electron microscopy. Use **ctf** for image simulation or restoration.

Examples

```
library em
```

This command runs a library program that prompts for all relevant instrumental conditions (see *variables used*).

```
min=0 max=4; ctf squared to display size 128
```

This command displays the intensity in the weak-phase object contrast *ctf* for the current imaging conditions, possibly for comparison with an experimental diffractogram.

ctf

```
ctf multiply 30 to 31 verify
```

This command multiplies picture 30 by the transfer functions, placing the result in picture 31. The **verify** option sends information about the command process to the console.

Description

The following options and keys are used with **ctf** to define:

- how the contrast transfer function is stored as output
- the kind of *ctf* that is generated
- whether the imaging conditions are defined as *reduced* units

The **ctf** command itself stores the *contrast transfer function* as a *Fourier* full-plane output picture, unless you specify otherwise using the option **halfplane** and the key **size** to specify dimensions. The command **ctf add** adds the *ctf* to the output picture. **ctf multiply** multiplies the source picture by the *ctf*.

ctf itself generates a weak-phase object *ctf* (the default kind of *ctf* is **ctf phase**). Other kinds of *contrast transfer function* that you can specify are:

- **ctf amp** (generates a weak amplitude *ctf* 2cos(gamma)
- **ctf wave** (generates a complex form exp(i-gamma)
- **ctf phase amp** (generates the product of the two *ctfs*)

With any of the above options **ctf squared** causes the squared modulus of the *contrast transfer function* to be used, and **ctf cc** its complex conjugate.

ctf itself assumes reduced units, axial distances in scherxzer units $Cs^{0.5}$ $lambda^{0.5}$ transverse in glaser units $Cs^{0.25}$ $lambda^{0.75}$ and angles in G1/Sch.

ctf physical assumes all distances in nanometres and all angles in millradians.

The imaging conditions, for example, the image sampling interval, are determined by variables rather than by the above options and keys. These variables are listed and described in the *Notes* given below. To set these variables run the library program *em*, as shown in the first command example.

Notes

multi-layer pictures:	faulted
forms used internally:	complex

ctf

variables used:

step	Image sampling interval (half minimum period in transform). A value is always required for this variable.
cs	Spherical aberration coefficient/mm (required if **physical** option used).
kv	Accelerating voltage in kv (required if **physical** option used).
astigmatism	Astigmatism (maximum minus minimum underfocus). Default is 0.
aphi	Azimuth of maximum underfocus/*mrad*; clockwise from positive *x* axis. Default is 0.
swidth	Illumination divergence (*rms* of equivalent gaussian profile). Default is 0.
ewidth	Focus spread (*rms* of equivalent gaussian profile). Default is 0.
tilt	Magnitude of primary beam tilt (*wrt* optical axis). Default is 0.
tphi	Azimuth../mrad. Default is 0.
oardius	Objective aperture radius (*wrt* optical axis). Default is infinity.
oadispl	Magnitude of objective aperture displacement. Default is 0.
oaphi	Azimuth../mrad. Default is 0.
drift	Magnitude of linear specimen drift during exposure. Default is 0.
dphi	Azimuth../mrad. Default is 0.
vibration	Lateral specimen vibration level (*rms*). Default is 0.

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	source picture	valid picture number
size	dimensions of current picture	positive integers
defocus	defocus off	
add/multiply	generate *ctf* in new output picture	
verify	verification off	

cut

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	size	*<x>, <y>, <z>*	dimensions of subregion to be cut
	position	*<x>, <y>, <z>*	position of subregion to be cut
	layer	*<n1>, <n2>*	range of layers in subregion
	value	*<number>*	intensity value for any part of the subregion that lies outside the source image
	mark	*<number>*	mark source subregion
		<yes or *no>*	
options:	**left/right, bottom/top, far/near**		position of subregion in relation to picture border

Use **cut** to extract a subregion from a picture, to cut a rectangular region with one or more layers. Use **paste** to insert a cut region into a picture.

Examples

```
cut 1 2 size 200 top left
```

This command cuts out the top left 200 pixels square of picture 1 as picture 2.

```
xwires region; cut @region to 2
```

This command cuts a region marked with the cursor from the current picture.

```
cut layer 1 to 2
```

This command cuts out the first (back) layer only.

```
cut 1 display size 200 value (max-min/2)
```

This command cuts a subregion of size 200 from picture 1 and gives a grey intensity to any part of the cut subregion that lies outside the image.

Description

Define the region to be cut using the standard subregion keys/options (**size**, **position**, **layers**, **left/right** etc). See *Appendix C, Semper Keys and Options* for further detail of defining a subregion. Note that the **xwires** command can also be used to define a subregion from cursor movements.

cut

If a subregion exceeds the size of the source picture it is padded with zeros rather than truncated. If you include the source picture origin in the region to be cut, it is recorded as the output picture origin also.

Use the **value** key to define an intensity for part of a cut subregion that does not contain an image. The diagrams below illustrates this concept. *Diagram 1* shows a subregion that is to be cut. The shaded part of the subregion is the area that does not contain an image and can be given an intensity value. *Diagram 2* shows the resulting cut image.

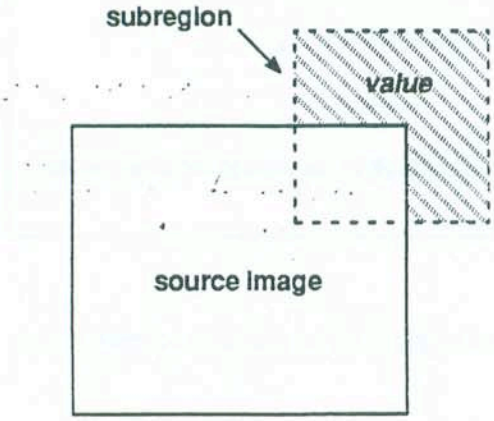


Diagram 1

Diagram 2

Notes

display marking:	region cut
multi-layer pictures:	fully supported
forms used internally:	integer, fp, complex
see also:	**extract** (for rotation and shear operations)
	magnify (for scaling)
	paste (to paste a cut region into another picture)

cut

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
size	whole picture	less than or equal to size of picture (integers)
position	position 0,0,0	within bounds of picture (integers)
layer	all layers unless *si3/po3* set	integers in range 1 to number of layers
value	intensity 0	real number
mark	mark off	see *Appendix C*

deassign

keys:	**device**	*<number>*	device to be deassigned
options:	**delete**		delete disc device after deassigning
	display		deassign the current display device
	verify		verify information about the process at the console

Use **deassign** to deassign a device, making the device available to other users. Semper imposes a limit on the number of devices that can be assigned at any one time. You can nevertheless have access to any number of devices *in turn*, by deassigning one device before assigning the next one.

Examples

```
deassign
```

This command deassigns the current device. (The variable *cd* holds the current device number).

```
deassign device fs
```

This command deassigns the device number held in the variable *fs*, that is, the display device.

```
deassign display
```

This command deassigns the display device.

```
deassign delete
```

This command deassigns the current device *cd* and deletes the file associated with it.

Description

Use **deassign** to deassign a file, tape or the display. You can also deassign program and help libraries, unless a library stores a program that is currently active. Note that the command **show devices** lists currently assigned devices. For information on devices refer to the *Semper 6 Guide*; *Chapter 4: Devices and Storage* in the *Advanced Users' Guide*.

If you use the option **delete**, you are asked to confirm a file deletion, unless the device is a scratch workspace or a log file. Scratch workspace devices are deleted automatically. Note that you cannot delete a device that has been write-protected (see the command **assign wp**). A message confirming that a device has been deassigned or deleted is output on the console.

Notes

see also: **assign, show devices**

deassign

Defaults and Ranges

keys/options	defaults	range
device	current device, held in variable *cd*	integer in range 1 to system limits (type ***show system***)
verify	verification on	

dclose

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	radius:	*<number>*	size of structuring element
options:	**diamond/square/octagon/circle**		shape of structuring element

You use the **dclose** command to close a binary source image with a circular disc of arbitrary radius. The output picture will contain the binary, closed result, which represents the complement of the area swept out by the structuring element while it is contained entirely within the background regions of the source image. You can select a different shape for the structuring element by specifying one of the options **diamond, square** or **octagon**. In all cases, the size of the structuring element is specified with the **radius** key.

Examples

```
dclose 1 2 radius 11.2
```

This command closes the binary image in picture 1 with a circular disc of radius 11.2 pixel units and outputs the result to picture 2

```
dclose 1 square radius 10
```

This command closes the binary image in picture 1 with 21 by 21 square structuring element

```
dclose octagon radius 20
```

This command closes the binary image in the current picture with an octagon of radius 20

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

The **dclose** command works by thresholding the distance transform of the source image (see the **dt** command for more information about distance transforms). The distance transform can be obtained with just two or four passes through the image which means that the time it takes to close the source image does not depend on the size of the structuring element. By default, the Euclidean distance transform is evaluated, which means that the source image can be closed with an exact circular disc with the specified radius.

dclose

In fact the **dclose** command is exactly equivalent to using a combination of commands **dt** and **threshold.** That is, the result obtained from the command

```
dclose <shape> radius r
```

can be obtained also in the following way:

```
dt <shape> bg
threshold le r
dt <shape> fg
threshold gt r
```

where <shape> is the option **diamond, square, octagon** or **circle.** However, the **dclose** command is faster and easier to use.

When calculating the distance transform, the **dclose** command may have to open a temporary picture (see the **dt** command for details).

The **dclose** command will fault any source picture which has a zero range. On successful completion, the range of the final result is stored in the output picture label.

Notes

see also: **dt, threshold**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
radius	*none*	real positive number
diamond/square/ octagon/circle	close with a circular disc	

ddilate

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	radius	*<number>*	size of structuring element
options:	**diamond/square/octagon/circle**		shape of structuring element

You use the **ddilate** command to dilate a binary source image with a circular disc of arbitrary radius. The output picture will contain the binary, dilated result, which represents the area swept out by the structuring element as its centre traverses all the foreground regions in the source image. You can select a different shape for the structuring element by specifying one of the options **diamond, square** or **octagon**. In all cases, the size of the structuring element is specified with the **radius** key.

Examples

```
ddilate 1 2 radius 11.2
```

This command dilates the binary image in picture 1 with a circular disc of radius 11.2 pixel units and outputs the result to picture 2.

```
ddilate 1 square radius 10
```

This command dilates the binary image in picture 1 with a 21 by 21 square structuring element.

```
ddilate octagon radius 20
```

This command dilates the binary image in the current picture with an octagon of radius 20.

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

The **ddilate** command works by thresholding the distance transform of the source image (see the **dt** command for more information about distance transforms). The distance transform can be obtained with just two or four passes through the image which means that the time it takes to dilate the source image does not depend on the size of the structuring element. By default,the Euclidean distance transform is evaluated, which means that the source image is dilated with an exact circular disc with the specified radius.

ddilate

In fact, the **ddilate** command is exactly equivalent to using the commands **dt** and **threshold** in turn. That is, the result obtained from the command

```
ddilate <shape> radius r
```

can be obtained also in the following way

```
dt <shape> bg
threshold le r
```

where <shape> is the option **diamond, square, octagon** or **circle**. However, the **ddilate** command is faster and easier to use.

When calculating the distance transform, the **ddilate** command may have to open a temporary picture (see the **dt** command for details).

The **ddilate** command will fault any source picture which has a zero range. On successful completion, the range of the final result is stored in the output picture label.

Notes

see also: **dt, threshold**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
radius	*none*	real positive number
diamond/square/ octagon/circle	dilate with a circular disc	

delete

keys:	[]	*<number>* *<n1>, <n2>*	picture to be deleted range of pictures to be deleted
	program	*'<text>'*	program to be deleted
options:	**malformed**		delete picture even if the label is corrupt
	verify		verify information about the process at the console

Use **delete** to delete unwanted pictures to create free space for new pictures. You can also use **delete** to delete a program from a program library.

Examples

```
delete
```

This command deletes the current picture.

```
delete 100, 199
```

This command deletes every picture in the range 100 to 199.

```
delete 4:9
```

This command deletes all records of pictures beyond picture 8, assuming device 4 is a tape device.

```
delete program 'combine'
```

This deletes a program called *combine* from the first program library found to contain the program called *combine*.

Description

delete can be used in two modes:

- picture mode
- program mode

In *picture mode*, you can delete one or a range of pictures within a single device. This releases free space, but it may be necessary to compress the device if the space is fragmented (see the command **compress**). Note that you cannot delete a write-protected picture, so you can use the command **wp** to protect your pictures from accidental deletion.

delete

In *program mode*, the named program is deleted from its program library using the **program** key. If the named program is found in more than one program library the copy that is deleted is the first in the current search order. You cannot delete a currently active program. Note that free space is not available immediately after using the **delete program** command; you need to use the **compress** command to free device space.

Use the **malformed** option to delete a picture that has a corrupted label. (Semper cannot process a picture with a malformed label, it is only possible to delete the picture and recover the space it occupies on a device).

Notes

see also:

compress	
deassign delete	(to delete entire disc files)
lut delete	(to delete look-up tables)
partition delete	(to delete partitions)
reinitialise	(to delete the contents of a picture disc)
wp	

Defaults and Ranges

keys/options	defaults	range
[ ]	current picture, held in variable *select*	valid picture number
program	*none:* deletes picture(s)	valid program name
verify	verification on	

derode

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	radius	*<number>*	size of structuring element
options:	**diamond/square/octagon/circle**		shape of structuring element

You use the **derode** command to erode a binary source image with a circular disc of arbitrary radius. The output picture will contain the binary, eroded result, which represents the area swept out by the centre of the structuring element whilst the structuring element is contained entirely within the foreground regions of the source image. You can select a different shape for the structuring element by specifying one of the options **diamond**, **square** or **octagon**. In all cases, the size of the structuring element is specified with the **radius** key.

Examples

```
derode 1 2 radius 11.2
```

This command erodes the binary image in picture 1 with a circular disc of radius 11.2 pixel units and outputs the result to picture 2.

```
derode 1 square radius 10
```

This command erodes the binary image in picture 1 with a 21 by 21 square structuring element.

```
derode octagon radius 20
```

This command erodes the binary image in the current picture with an octagon of radius 20.

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

The **derode** command works by thresholding the distance transform of the source image (see the **dt** command for more information about distance transforms). The distance transform can be obtained with just two or four passes through the image which means that the time it takes to erode the source image does not depend on the size of the structuring element. By default, the Euclidean distance transform is evaluated, which means that the source image is eroded with an exact circular disc with the specified radius.

derode

In fact, the **derode** command is exactly equivalent to using the commands **dt** and **threshold** in turn. That is, the result obtained from the command

```
derode <shape> radius r
```

can be obtained also in the following way

```
dt <shape> fg
threshold gt r
```

where <shape> is the option **diamond, square, octagon** or **circle**. However, the **derode** command is faster and more convenient to use.

When calculating the distance transform, the **derode** command may have to open a temporary picture (see the **dt** command for details).

The **derode** command will fault any source picture which has a zero range. On successful completion, the range of the final result is stored in the output picture label.

Notes

see also: **dt, threshold**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
radius	*none*	real positive number
diamond/square/ octagon/circle	erode with a circular disc	

destripe

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	lines	*<number>*	destripe over the specified number of lines
	rsd	*<number>*	the reference standard deviation required for all sensors
	rmean	*<number>*	the reference mean required for all sensors
	layer	*<number>*	destripe the specified layer
options:	**mode**		destripe, taking into account the standard deviation of the sensors

The **destripe** command corrects an image for instrumentation errors, notably in images from the Landsat satellite, where the receiver has different characteristics on different lines. It destripes a source picture. Use this command in *Remote Sensing* applications.

Examples

```
destripe 1 2 line 6
```

This command destripes all bands (layers) of picture 1 using 6 line sensors and only correcting by the average value.

```
destripe 1 2 line 6 mode
```

As above but correct using the standard deviation also.

Description

You can destripe the source picture either by subtracting the mean value, or by using the standard deviation and mean of a reference line. The **lines** key specifies the number of lines over which destriping is to occur, for example, a *Landsat* image would require **lines 6** because of the way that its sensor is made. The **lines** key must have a value of at least 2. If you do not specify the **rsd** or **rmean** keys, the output picture is adjusted by the mean of a reference line (sensor):

$$x_{ni} = x_{oi} - (m_i - m_r)$$

where m_i is the mean of the current sensor and m_r is the mean of the reference sensor.

destripe

If you specify the **mode** option the output picture is corrected by:

$$xni = \left(\frac{\sigma r}{\sigma i}\right) (xoi - mi) + mr$$

where:

- mi is the mean of the current sensor
- mr is the mean of the reference sensor
- σi is the standard deviation of the current sensor
- σr is the standard deviation of the reference sensor

If you assign values to the **rmean** key or to the **rsd** and **rmean** keys, these values are used as the reference values, rather than the values from one of the sensors.

If you do not specify a layer using the **layer** key, all layers of the picture are destriped and the output picture contains the same number of layers as the source picture.

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
lines	*none*	integer in range 2 to half size of the picture
rsd	derived from source picture	real number
rmean	derived from source picture	real number
layer	all layers	integer in range 1 to number of layers

device

keys:	active	<*number*>	element selection is only allowed on the specified device
	query	<*number*>	determines the size of the device and the number of supported colours
	refresh	<*number*>	redraws the user interface on the specified device
	restore	<*number*>	restores the previously saved cursor position of the given device
	save	<*number*>	saves the current cursor position of the specified device

Use **device** to define the devices on which interaction takes place with your defined Semper 6 *Plus* interfaces. For further detail refer to the *Semper 6 Plus User Interface Guide*.

Examples

```
device query 2
```

This command determines the size of device 2 (the framestore) and the number of colours it supports. It stores these values in the variables *uix*, *uiy* and *nco* respectively.

```
device active 1
```

This command defines that interaction with a user-defined interface can only take place on device 1 (the host display).

Description

Devices are referred to by number. The following device numbers are currently in use:

- device 0: all devices
- device 1: the display on a host computer
- device 2: the framestore

If the Semper variable *cdi* is not changed, the default device is the display. Set the variable *cdi* to 2 if panels etc. are to appear on the framestore. If you do not specify a device using **device active**, element selection is allowed on both displays. Note that if the framestore is used to display panels, there may be hardware restrictions on the colours available for display.

The **query** key allows a program to find out about its environment, returning the size of the display and the number of colours supported by the display. This is useful when a program starts to define a user interface, so that it can position and colour all the objects correctly.

device

The default setting for the colours are as follows:

Number	Colour
0	Black
1	White
2	Red
3	Green
4	Blue
5	Cyan
6	Magenta
7	Yellow

Note that some devices may not support colour or only support a restricted range, or use different defaults (for example, 0 = white, 1 = black).

The **save/restore** keys provide an alternative method of storing the cursor position to the **mouse** command (when using the **query** option and **position** key). The **device save/restore** keys have the advantage that calls may be stacked, easing programming requirements.

The **refresh** key redraws the screen, after clearing it.

Notes

variables set: *uix, uiy* (set by **device query** to the size of the requested device)
nco (set by **device query** to the number of colours on the display)

see also: **uif, mouse**

Defaults and Ranges

keys/options	defaults	range
active	*none*: element selection on both displays	integer in range 1 to 2
query	*none*	integer in range 1 to 2
refresh	*none*	integer in range 1 to 2
restore	*none*	integer in range 1 to 2
save	*none*	integer in range 1 to 2

diagnostic

<variable name(s)> *'<text string>'<variable name(s)>*	**diagnostic** uses a special syntax. It prints the value of the specified variable(s) and any given text string to the diagnostic output stream.

Use the **diagnostic** command to print text and/or numerical values to the diagnostic output stream.

Examples

```
diagnostic 'unexpected error ',x,' reading data'
```

This command outputs a user-generated error message to the diagnostic output stream.

Description

The output takes exactly the same form as in the **type** command. The **diagnostic** command allows you to mix expression values and text. Specify text within single quotes and use a comma to separate values or expressions.

For further detail of the output streams that Semper uses, refer to the **echo** command or to the section *Controlling output in Semper in Chapter 2* of the *Advanced Users' Guide*, contained in the *Semper 6 Guide*.

Notes

see also: **type, echo**

differentiate

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	angle	*<number>*	differentiation direction in radians, anti-clockwise from the positive *x* axis

Use **differentiate** to calculate picture derivatives, that is the rate of change in a picture, in any direction. This command uses a three-point local operator on *Images*, with a special treatment of *Fourier* transforms.

Examples

```
differentiate 1 to 2
```

This command differentiates along the *x* axis in picture 1, replacing each pixel by the difference between its right-hand neighbour and itself. It places the output in picture 2.

```
xwires line; min=-40 max=40; differentiate to display angle theta
```

This command differentiates the display in a direction marked with the cursor.

Description

Use the **angle** key to specify the differentiation direction. The direction is measured anti-clockwise in radians from the positive *x* axis. The default angle is 0, if the angle is not set or if the source picture is 1-D.

On most classes of picture, **differentiate** uses a 3-point local operator of the form:

$$cos(\mathbf{angle}).p(x+1,y) + sin(\mathbf{angle}).p(x,y+1) - (cos(\mathbf{angle})+sin(\mathbf{angle})).p(x,y)$$

The exception is a picture of type *Fourier*, in which the source is multiplied by the phase shifting factor:

$$e^{2\pi ik}$$

where $i = \sqrt{-1}$ and with *k* measured in the appropriate direction. This is equivalent to differentiating the *Image* from which the *Fourier* transform was derived, but rather differently approximated.

Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex

differentiate

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
angle	angle 0	real number in range 0 to 2π

dilate

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	times	*<number>*	force repeat count for dilation process
	neighbours	*<number>*	only dilate points with specified number of clear neighbours
	with	*<number>*	picture containing user-supplied map functions
options:	**separately**		dilate objects whilst preserving separation (via 4-connected skeleton background)

Use **dilate** to process a binary picture, by adding a one-pixel border to objects made up of non-zero pixels. **dilate** can also be used to clean up an object by filling holes and points in an object without affecting its real edge. Note that zero value pixels are referred to as *clear* or *background* pixels and non-zero value pixels as *set* or *object* pixels.

Examples

```
dilate display
```

This command adds a one-pixel wide border to all objects (by setting clear pixels that have a set neighbour).

```
dilate 50 to 51 times 3
```

This command adds a three-pixel border to all objects (that is, it repeats the **dilate** command three times).

```
dilate neighbours 4
```

This command adds pixels at sites that have at least four set neighbours and is used to fill points and line holes in objects without affecting their 'real' edge.

```
dilate neighbours 8
```

This command fills isolated point holes only.

```
dilate separately
```

This command dilates objects but preserves the 4-connectivity of the background, so that objects and object points that are originally separated remain so.

dilate

```
stack 3, 6 to 7, dilate 50 with 7
```

This command applies the maps in pictures 3 to 6 in turn, to dilate objects in picture 50.

Description

The **dilate** command is part of the *morphology* group of commands that perform morphological (shape) operations on binary pictures. Other morphology commands include **analyse**, **erode** and **median**. You use **dilate** on binary pictures, that is, pictures that have only two classes of value:

- zero for background pixels
- non-zero for pixels belonging to the shape (object) of interest

Note that as Semper does not actually store pixels in single bit form, **dilate** distinguishes between zero (*clear*) and non-zero (*set*) pixels instead. You can translate a picture into binary form using the **calculate** command, for example, **calculate :51>20** or **calculate :51<thr**.

By default, **dilate** adds a one-pixel wide border around pixels that are set. You can use the **times** key to specify the number of times the **dilate** command is repeated and so determine the width of the border. If you set **times** to zero, **dilate** repeats itself until the picture stabilizes (or until you *abandon* the operation). This is faster than putting **dilate** in a **for** loop, as the command is able to omit picture rows that have already stabilized in previous passes.

By default, **dilate** sets pixels with at least one set neighbour. You can change this minimum number using the **neighbours** key and perform more and more selective dilation as you increase it towards the maximum of 8. The default mode of **dilate** is in fact **dilate neighbours 1**. The command **dilate neighbours** is exactly equivalent to **erode neighbours**, with the object and background interchanged. You may find it helpful to look at the **erode neighbours** examples given in this manual.

The **dilate separately** command dilates objects but preserves the 4-connectivity of background, so that objects and object points that are initially separated remain so. (Refer to *Appendix G, Pixel Connectivity* for an explanation of 4 and 8 connectivity). The diagram given below illustrates how the command **dilate separately** works.

dilate

The **dilate separately** command dilates objects in sequences of four passes, in which pixels are removed, without affecting the 4-connectivity of the background, from the top, the bottom, the left and the right in turn. By default, the sequence is repeated until the picture stabilizes, but you can limit the number of sequences if you wish, using the **times** key.

Notes

see also: **analyse, calculate, erode, median**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
times	1	zero or positive integer
neighbours	1 pixel	integer in range 1 to 8
with	*none*	valid picture number

directory

keys:	**device** *<number>*	gives directory of specified device

The **directory** command prints directory information about a picture disc device or a program library. It details the free and used space on a device.

Examples

```
directory
```

This command prints information about the current device.

```
directory device 4
```

This command prints directory information about device 4.

Description

For a *picture disc*, the **directory** command prints information about the number of used and free directory slots and disc space. It also details the size of the largest free segment and the first unused picture number.

For a *program library*, **directory** prints information about directory slot, directory space and disc space usage under the headings *active*, *deleted* and *free*. Note that you can use the **compress** command to free deleted space.

Use the **device** key to specify a device number. The default device is the current device.

Notes

see also: **compress, examine**

Defaults and Ranges

keys/options	defaults	range
device	current device, held in the variable *cd*	integer in range 1 to system limits (type ***show system***)

display

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output display picture
	times	*<number>*	magnification factor
	size	*<x>, <y>, <z>*	dimensions of subregion to be displayed
	position	*<x>, <y>, <z>*	position/offset of subregion
	layer	<n1>, <n2>	range of layers in subregion
	height	*<number>*	if 1-D picture, height of graph in framestore pixels
	aspect	*<number>*	terminal aspect ratio for character-form displays (number of columns per inch divided by number of lines per inch)
	width	*<number>*	number of characters per line, for character form displays
	mark	*<number>*	mark source subregion
		<yes *or* no>	
options:	**preset**		set display black/white levels using values held in variables *min*, *max*
	scale		if ***noscale***, set black/white levels for unit scaling
	negated		negate black/white levels on display
	repeating		repeat pixels when magnifying, instead of interpolating
	letter		display picture number and title etc. at top of the partition
	border		mark partition border
	left/right, top/bottom, far/near		subregion positions
	type/log		picture is output in character form on console or to log output stream

Use **display** to display pictures. The command has more features than the **copy...to display** command, for example, automatic scaling, contrast reversal, lettering, borders, subregion display and magnification, 1-D graphs, histograms and character-form displays on your terminal.

display

Examples

```
display negated
```

This command displays the current picture with reverse contrast.

```
xwires region; display dis @region times 5 to dis:2
```

This command magnifies a region on the current display by a factor of five. The region is marked using the cursor (**xwires** command). The magnified region is displayed on dis:2.

```
display layer 1 type
```

This command displays the first (back) layer on the terminal.

```
extract 50 51 size 256,1 angle pi/4; display
```

This extracts and displays a diagonal line scan from picture 50.

```
min=0 max=1E7; display preset
```

This command displays a picture using the black/white levels determined by *min*, *max*.

Description

The **display** command displays the source picture within the limits of the display partition that you specify as the output display picture number. If necessary, the output is scaled down by a suitable integer sampling factor so that it just fits inside the partition. The end result is a display picture that can be processed like any other Semper picture (unless it is undersampled or displayed in graph or character form).

The **display** command has the following features:

- multi-layer subregions
- magnification facilities
- automatic and user-defined grey-scaling
- bordering of the display
- line graphical display of 1-D pictures
- character form displays

You can define a subregion using the keys **size**, **position** and the options **left/right** etc. See *Appendix C, Semper Keys and Options* for further detail. Note that subregions are displayed as independent pictures as far as their coordinate system is concerned (as if using **cut**) but retain their source picture coordinates for later graphical purposes. Use the **layer** key to specify the range of layers to be displayed.

display

Use the **times** key to magnify an image and the **repeating** option to replicate the pixels in an image, instead of interpolating.

The **display** command finds the grey-scale range of a picture or region and returns it in the variables *min, max*. These variables are used to set the black and white levels of the display. You can specify the grey-scaling directly by using the option **preset**. **display preset** uses the current values of *min* and *max* rather than those found by surveying the picture. You can alternatively specify the **noscale** option, to force unit intensity scaling.

By default, **display** outlines the picture border and displays the picture number, size, title and black/white range. Use the option **noborder** to turn off the bordering and **noletter** to turn off the lettering.

display gives a line graphical display of 1-D pictures on the display overlay to the same lateral scale as a 2-D display. This is also true of histogram displays. You can use the **height** key to force graph height. (Use the **times** key to increase the width).

Use the options **type/log** to force a character-form display of an image to the console or to a log file. Use the **width** key to specify the number of characters per line.

By default **display** assumes all layers, unless you specify a particular range of layers with the **layer** or **size** key.

The partition into which the picture is written may have more frames allocated to it than there are output layers, in which case the output of picture layers resumes with the first and successive layers until all of the frames in the partition have been filled. This facility allows you to handle single layer pictures on a full colour framestore with the minimum of fuss, simply by creating partitions that are three frames deep.

Notes

display marking:	region, if subregion
multi-layer pictures:	fully supported
forms used internally:	all
variables used:	*min*, *max* (if **preset**, pixel range for black/white level scaling)
variables set:	*min*, *max* (unless **preset**, pixel range within displayed picture/ region)
see also:	**copy...to display**, **cut**, **xwires**, **partition**

display

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	display picture, held in the variable *display*	valid picture number
times	times 1	positive integer
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0,0	within bounds of the picture (integers)
layer	all layers unless variables *si3*/*po3* are set	integer in range 1 to number of layers
height	lesser of half graph width and half partition height	positive integer
aspect	default given by the ***page*** command	real number
width	default given by the ***page*** command	positive integer
mark	mark off	see *Appendix C*
scale	unit scaling off	
negated	negation off	
letter	lettering on	
border	bordering on	
type/log	display on display device	

dopen

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	radius	*<number>*	size of structuring element
options:	**diamond/square/octagon/circle**		shape of structuring element

You use the **dopen** command to open a binary source image with a circular disc of arbitrary radius. The output picture will contain the binary, opened result, which represents the area swept out by the structuring element while it is contained entirely within the foreground regions of the source image. You can select a different shape for the structuring element by specifying one of the options **diamond, square** or **octagon**. In all cases, the size of the structuring element is specified with the **radius** key.

Examples

```
dopen 1 2 radius 11.2
```

This command opens the binary image in picture 1 with a circular disc of radius 11.2 pixel units and outputs the result to picture 2

```
dopen 1 square radius 10
```

This command opens the binary image in picture 1 with 21 by 21 square structuring element

```
dopen octagon radius 20
```

This command opens the binary image in the current picture with an octagon of radius 20

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

The **dopen** command works by thresholding the distance transform of the source image (see the **dt** command for more information about distance transforms). The distance transform can be obtained with just two or four passes through the image which means that the time it takes to open the source image does not depend on the size of the structuring element. By default, the Euclidean distance transform is evaluated, which means that the source image can be opened with an exact circular disc with the specified radius.

dopen

In fact, the **dopen** command is exactly equivalent to using the commands **dt** and **threshold.** That is, the result obtained from the command

```
dopen <shape> radius r
```

can be obtained also in the following way:

```
dt <shape> fg
threshold gt r
dt <shape> bg
threshold le r
```

where <shape> is the option **diamond, square, octagon** or **circle.** However, the **dopen** command is faster and easier to use.

When calculating the distance transform, the **dopen** command may have to open a temporary picture (see the **dt** command for details).

The **dopen** command will fault any source picture which has a zero range. On successful completion, the range of the final result is stored in the output picture label.

Notes

see also: **dt, threshold**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
radius	*none*	real positive number
diamond/square/ octagon/circle	open with a circular disc	

drag

keys:	[]	*<number>*	picture/partition/frame on which to drag object
	position	*<x>,<y>*	initial position of anchor point if **with** set, curve origin if **to** set, start point of line if **radius** set, circle/arc centre if **size** set, sub-region centre position
	with	*<number>*	drag open/closed curve defined in given plist picture
	to	*<number>* *<x>,<y>*	if **with** set, destination picture for plist otherwise, end point of line
	radius	*<number>*	drag circle/arc with given radius if **angle** set, arc otherwise, circle
	size	*<x>,<y>*	drag sub-region with given dimensions
	angle	*<x>,<y>* *<number>*	if **radius** set, start/end angles for anti-clockwise arc if **size** set, orientation angle of sub-region
	sampling	*<number>*	sampling (magnfication) factor for sub-region
	tolerance	*<number>*	arc-to-chord tolerance for arcs, circles and curves
options:	**picture/partition/frame**		use indicated coordinate system
	line/circle/arc/region/curve		drag specified object
	open/closed		if **closed** set, curve displayed with line joing first and last point in plist
	left/right, bottom/top		if **size** set, sub-region justified on indicated border
	uv		sub-region has sampling lattice defined by variables **u,u2** and **v,v2**
	re/im		drag object with respect ot real or imaginary part of complex display picture if **verify** set, mark final dragged position on real or imaginary part
	view		switch view to make display region visible
	verify		mark dragged object at its final position

You use the **drag** command to position a variety of graphical objects on the display. Graphical objects have an origin or anchor position whose initial position is determined by the **position** key. The final position of the origin is returned in the variables *x* and *y*. The following graphical objects are supported: lines, arcs, circles, all types of sub-regions and open or closed curves.

drag

Examples

```
drag line position 20,30 to 100,170
```

This command drags a line with initial start point at (20,30) and end point at (100,170), returning the final start position in variables *x* and *y*.

```
drag partition dis:2 circle radius 50
```

This command drags a circle with radius 50 from the centre of display partition 2.

```
drag dis:2 size 200 top left
```

This command drags a 200 by 200 sub-region initially positioned at the top left of display picture dis:2.

```
drag frame curve with 2:90 @xy closed to 2:91
```

This command drags the closed curve specified by picture 2:90 with the initial position of the curve's origin at (*x*,*y*), and outputs the shifted result as another position list in picture 2:91.

Description

The **drag** command supports three sets of display coordinates: *picture, partition* or *frame* coordinates. You select which one to use by specifying one of the options **picture, partition** or **frame.** The **picture** option is assumed if no option is given. You can also specify which display picture, partition or frame to use. For display pictures and partitions the *display* variable supplies the default picture or partition number. For frame coordinates, the *cframe* variable supplies the default frame number.

The graphical object to be dragged first appears exactly as if the **mark** command had been used to draw it using the same keys and options as specified in the **drag** command (with the one exception of the **position** key specifying an offset for a curve, a facility which the **mark** command does not support). The **position** key defines the initial position of the object's origin or anchor position, except for justified sub-regions (options **left/right** and **top/bottom**), where the **position** key defines an offset from the initial justified position. The anchor position for each type of graphical object is as follows:

Line	Start point of line
Arc or circle	Centre point of arc or circle
Sub-region	Centre of sub-region
Curve	Origin of curve

The type of graphical object is determined by the presence of the keys **with** (open or closed curve), **to** (line), **radius** (arc or circle) and **size** (sub-region), in that order. The options **line, arc, circle** and **curve** are ignored and have only been included to allow you to improve the readability of the **drag** command.

drag

Note that the **with** key takes precedence over the **to** key because the **to** key may be used in conjunction with the **with** key to specify an output picture number.

For curves, the option **open** or **closed** may also be specified. If the option **closed** is specified, a line is drawn from the end of the curve to the start point to ensure that the curve always appears closed. The keys **sampling** and **angle** may be used to specify scaled and angled sub-regions. If the option **uv** is given in conjunction with the **size** key, it specifies a skewed sub-region where the sampling lattice vectors are obtained from the variables *u,u2* and *v,v2*.

Once the object appears on the display you should be able to move it around interactively by moving the mouse or by pressing any of the cursor keys. When you have finally positioned the object, press a mouse button or a key on the keyboard to terminate the **drag** command. The final position of the object's anchor point will be returned in variables *x* and *y*. The final position of the end point of a line is also returned in variables *xn* and *yn*. When dragging curves, you have the option to output a new position list which describes the curve in its final position. You specify the picture number for the new position list with the **to** key. By default, the object is redrawn in its final position. You can prevent this from happening by specifying the option **noverify**.

Although the **drag** command allows you to drag literally any shape of object (with a position list), the speed of response of the dragging process will depend on the complexity of the object (number, length and orientation of line segments). A reasonable approximation of an *arc, circle* or *curve* can be obtained by specifying the **tolerance** key. This reduces an object to the smallest set of line segments which departs from the original object by no more than the distance given by the **tolerance** key. By default, a tolerance value of 1.0 is assumed.

On complex display pictures, where the real and imaginary parts of the image are displayed side by side, dragging normally takes place with respect to the real part of the image. If, however, you specify the option **im**, dragging will be carried out with respect to the imaginary part of the image. By default, the final position of the dragged object is verified by marking it on both the real and imaginary parts. If you specify the option **re** or **im**, verification is restricted to just the real or imaginary part.

The **drag** command is designed to work with other commands such as **xwires, sketch** and **mark**, which operate with the different types of graphical objects supported here. The key, option and variable names used are also compatible with the named macros **@xy, @line, @arc, @circle** and **@region.** For example, you can define and then drag a circular arc on the current display like this:

```
xwires frame are noverify; drag frame @arc
```

As a final example, you could use the **sketch** and **drag** commands to obtain the distance between two similar objects in an image like this:

```
display 3:10            display source image
sketch to 3:11 closed   sketch around first object
drag with 3:11          drag outline and align with second object
type 'object separation =', root (x^2+y^2)
```

drag

Notes

variables used:	*u,u2,v,v2*	sampling lattice vectors (**uv** option)
variables set:	*x,y*	final position of anchor point
	xn,yn	final position of end point of line

see also: **xwires, sketch, mark**

Defaults and Ranges

keys/options	defaults	range
[ ]	*display* if picture or partition; current frame if frame, held in variable *frame*	valid picture/partition/frame number
position	0,0	real numbers
with	none	valid picture number
to	none	valid picture number
radius	none	positive real number
size	none	positive integer
angle	none	real number in range 0 to 2π
sampling	1	positive real number
tolerance	1	positive real number
picture/partition/ frame	picture coordinates	
line/circle/arc/ region/curve	do nothing	
open/closed	closed curve of Plist defines a closed curve, otherwise open curve	
left/right, top/ bottom	position centre of sub-region at initial position of anchor point	
re/im	drag object with respect to real part of complex display picture if verification is on, mark both parts of complex display picture	
verify	verification on	

dt

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
options:	**fg/bg**		evaluate distance transform for foreground/background pixels with respect to background/foreground regions
	circle/diamond/square/octagon		evaluate true Euclidean, 4-connected, 8-connected or octagonal distance transform
	open/closed		treat edges of image as being open or closed

You use the **dt** command to find the shortest distance between each foreground pixel from the nearest background pixel. With the **bg** option, the same can be done for background pixels with respect to foreground regions. The option **diamond, square** or **octagon** may be specified to obtain distances based on 4-connected and 8-connected metrics in less time than for the exact Euclidean distance.

Examples

```
dt 1 2
```

This command finds the shortest distance between foreground pixels and the nearest background pixel in picture 1 and outputs the result to picture 2.

```
dt 1 2 square
```

This command finds the shortest 8-connected distance between foreground pixels and the nearest background pixel in picture 1.

```
dt 1 2 octagon bg
```

This command finds the shortest octagonal distance for background pixels with respect to foreground regions in picture 1.

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels.

By default the **dt** command calculates the Euclidean distance transform. The distances for most pixels

dt

are the exact diagonal distances to the nearest pixel of opposite polarity. In a few very rare cases, the algorithm used may pick the wrong pixel as the nearest pixel, but the distance to this pixel is guaranteed not to differ from the correct value by more than 0.1 pixel unit.

When calculating Euclidean distances, the **dt** command has to open a temporary, two layer, integer picture with the same *x* and *y* dimensions as the source picture. The final result is output in floating-point form.

The **bg** option may be specified to cause the distance transform to be evaluated for background pixels. In this case, all foreground pixels will return a zero result. The options **fg** and **bg** are mutually exclusive.

By default, the distance transform is calculated as if regions touching the edges of the image continue to infinity. This corresponds to the option **open**. With the **closed** option, regions can be closed off round the edges of the image, with the result that all distance contours are closed.

The **dt** command will fault any source picture which has a zero range. On successful completion, the range of the final result is stored in the output picture label.

In addition to Euclidean distances, the **dt** command also provides options to evaluate the distance transform using more simple 2D metrics. A 2D metric is defined in terms of a formula for calculating the distance between two points (xy, y1) and (x2, y2). The Euclidean metric gives the shortest distance between two points with the well-known formula for calculating the distance

$$d_{euclidean} = \text{root}\ ((x2-x1)^2 + (y2-y1)^2)$$

The simplest non-euclidean metric supported by the **dt** command is selected with the **diamond** option. This causes distances to be measured using the 4-connected or "city block" metric which has the following formula

$$d_{diamond} = \text{mod}\ (x2-x1) + \text{mod}\ (y2-y1)$$

The distance values represent the length of the shortest 4-connected path between two points.

With the **square** option, the 8-connected metric is selected instead. This measures the length of the shortest 8-connected path between two points where each horizontal, vertical or diagonal step along the path adds one unit to the length of the path. The formula for this metric is as follows:

$$d_{square} = \text{max}\ (\text{mod}\ (x2-x1), \text{mod}\ (y2-y1))$$

Note that the 8-connected metric always gives a result which is less than or equal to the Euclidean metric and the 4-connected metric always gives a result which is greater than or equal to the Euclidean metric

$$d_{square} <= d_{euclidean} <= d_{diamond}$$

dt

The last non-Euclidean metric supported by the **dt** command is selected with the **octagon** option. This combines the results of the 4-connected and 8-connected metrics to give a closer approximation to the Euclidean metric

$$d_{octagon} = \max(\text{fix}(2*(d_{diamond}+1)/3)), d_{square})$$

All of these distance transforms are faster to evaluate than the Euclidean distance transform. They do not require a temporary picture to be opened unless the output picture is a display picture, in which case a temporary, single layer, integer form picture is opened. All the distance values will be exact integers.

The options **diamond, square, octagon** and **circle** (the default) describe the shape of the distance contours obtained from an image with a single background pixel, for example

```
create,size 11 value 1
pixel 0,0=0
dt <shape>
print
```

where <shape> is the appropriate option.

dt diamond

```
8 7 6 5 4 5 6 7 8
7 6 5 4 3 4 5 6 7
6 5 4 3 2 3 4 5 6
5 4 3 2 1 2 3 4 5
4 3 2 1 0 1 2 3 4
5 4 3 2 1 2 3 4 5
6 5 4 3 2 3 4 5 6
7 6 5 4 3 4 5 6 7
8 7 6 5 4 5 6 7 8
```

dt square

```
4 4 4 4 4 4 4 4 4
4 3 3 3 3 3 3 3 4
4 3 2 2 2 2 2 3 4
4 3 2 1 1 1 2 3 4
4 3 2 1 0 1 2 3 4
4 3 2 1 1 1 2 3 4
4 3 2 2 2 2 2 3 4
4 3 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4 4
```

dt octagon

```
6 5 4 4 4 4 4 5 6
5 4 4 3 3 3 4 4 5
4 4 3 2 2 2 3 4 4
4 3 2 2 1 2 2 3 4
4 3 2 1 0 1 2 3 4
4 3 2 2 1 2 2 3 4
4 4 3 2 2 2 3 4 4
5 4 4 3 3 3 4 4 5
6 5 4 4 4 4 4 5 6
```

dt

dt circle

```
5.657 5.000 4.472 4.123 4.000 4.123 4.472 5.000 5.567
5.000 4.243 3.606 3.162 3.000 3.162 3.606 4.243 5.000
4.472 3.606 2.828 2.236 2.000 2.236 2.828 3.606 4.472
4.123 3.162 2.236 1.414 1.000 1.414 2.236 3.162 4.123
4.000 3.000 2.000 1.000 0.000 1.000 2.000 3.000 4.000
4.123 3.162 2.236 1.414 1.000 1.414 2.236 3.162 4.123
4.472 3.606 2.828 2.236 2.000 2.236 2.828 3.606 4.472
5.000 4.243 3.606 3.162 3.000 3.162 3.606 4.243 5.000
5.657 5.000 4.472 4.123 4.000 4.123 4.472 5.000 5.567
```

The shape of the distance contours can be more clearly seen if you blank out all values greater than 4 in the example given.

dt diamond

```
. . . . 4 . . . .
. . . 4 3 4 . . .
. . 4 3 2 3 4 . .
. 4 3 2 1 2 3 4 .
4 3 2 1 0 1 2 3 4
. 4 3 2 1 2 3 4 .
. . 4 3 2 3 4 . .
. . . 4 3 4 . . .
. . . . 4 . . . .
```

dt square

```
4 4 4 4 4 4 4 4 4
4 3 3 3 3 3 3 3 4
4 3 2 2 2 2 2 3 4
4 3 2 1 1 1 2 3 4
4 3 2 1 0 1 2 3 4
4 3 2 1 1 1 2 3 4
4 3 2 2 2 2 2 3 4
4 3 3 3 3 3 3 3 4
4 4 4 4 4 4 4 4 4
```

dt octagon

```
. . 4 4 4 4 4 . .
. 4 4 3 3 3 4 4 .
4 4 3 2 2 2 3 4 4
4 3 2 2 1 2 2 3 4
4 3 2 1 0 1 2 3 4
4 3 2 2 1 2 2 3 4
4 4 3 2 2 2 3 4 4
. 4 4 3 3 3 4 4 .
. . 4 4 4 4 4 . .
```

dt circle

```
..... ..... ..... ..... 4.000 ..... ..... ..... .....
..... ..... 3.606 3.162 3.000 3.162 3.606 ..... .....
..... 3.606 2.828 2.236 2.000 2.236 2.828 3.606 .....
..... 3.162 2.236 1.414 1.000 1.414 2.236 3.162 .....
4.000 3.000 2.000 1.000 0.000 1.000 2.000 3.000 4.000
..... 3.162 2.236 1.414 1.000 1.414 2.236 3.162 .....
..... 3.606 2.828 2.236 2.000 2.236 2.828 3.606 .....
..... ..... 3.606 3.162 3.000 3.162 3.606 ..... .....
..... ..... ..... ..... 4.000 ..... ..... ..... .....
```

dt

Using option **square** generates a result that is equivalent to applying the command **erode** or **dilate** as many times as it takes to empty or fill the source image. Thresholding the result obtained with the **fg** option, to retain all pixels greater than *n* gives the result for *n* erosions. Thresholding the result obtained with the **bg** option, to retain all pixels less than or equal to *n* gives the result for *n* dilations. Thresholding the Euclidean distance transform makes it possible to erode or dilate objects by non-integer distances.

In fact, thresholding the distance transform is equivalent to eroding or dilating with a structuring element whose shape corresponds to the option specified for the distance transform and whose radius is the threshold value used. Combining erosion with dilation (or vice versa) allows you to obtain the opening and closing of a binary image with the same structuring element. The commands **derode, ddilate, dopen** and **dclose** can be used to do this directly and in less time than it would take with the **dt** and **threshold** commands.

Notes

see also: **derode, ddilate, dopen, dclose, erode, dilate, threshold**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
fg/bg	evaluated distance transform for foreground pixels	
circle/diamond/ square/octagon	evaluate Euclidean distance transform	
open/closed	treat edges of source image as being open	

echo

keys:	**device**	*<number>*	change echo settings for assigned text file
options:	**[standard] output**		change echo settings for standard output stream
	[standard] error		change echo settings for standard error stream
	terminal		same as **standard output**
	console		turn console echo on
	diagnostic		turn diagnostic echo on
	log		turn log echo on
	monitor		turn monitor echo on
	command		turn command echo on
	input		turn non-command input echo on
	all/none		turn all echoing on/off

Use **echo** to specify the output destination for the different classes of output text generated by Semper.

Examples

```
echo commands terminal; library lattice
```

This example echoes the library commands from the lattice program on the terminal.

```
assign file name 'errors'; echo diagnostics device n
```

This example command opens a file to log all error messages in the file *errors.log*.

```
assign file name 'results'; echo console device n
echo noconsole terminal; library measure; echo console terminal
deassign device n
```

This sequence of commands captures the results listed by the library program called *measure* in a file called *results.log*. It suppresses output of the results to the terminal.

echo

Description

Semper defines six different classes of output text and each one of these is directed to a separate logical output stream as follows:

• console	normal terminal output, results etc.
• diagnostic	unsolicited output, that is, error and warning messages
• log	results etc, that would not normally appear on the terminal
• monitor	diagnostic information for debugging purposes
• command	command reflection
• input	non-command input, for example, response to **ask** etc.

The **echo** command specifies where output to each of these streams will end up. Semper supports output to the following:

- standard output stream (the terminal)
- standard error stream
- log files

Some operating systems support the concept of standard output and error streams and they allow you to redirect output from these. This is particularly useful when running Semper in batch mode. When Semper starts up, console, diagnostic and monitor output is sent to the standard output stream.

In order to switch output to the standard output stream, you specify the **standard output,** option, or the **terminal** option, which is a synonym for **standard output**. To switch output to the standard error stream, use the option **standard error**. Use the **device** key if you want to switch output to a log file. The log file must have been opened already with the **assign file** command.

You can then specify which logical output streams to turn on or off using a combination of the options **console, diagnostic, log, monitor, command** and **input**, together with the options **all** or **none.**

You can turn off echoing by prefixing an option with **no**, for example, **noconsole nolog** turns off the echoing of the console and log output streams. The blanket options **all** or **none** turn on or off all echoing and you can combine them with a specific option, for example, **all nolog** echoes everything except log output, **none console** echoes only console output.

Use the **show echo** command to list the current echo settings.

Notes

see also: **assign file,show echo**

echo

Defaults and Ranges

keys/options	defaults	range
device	*none*	valid device number

edge

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
options:	**roberts**		applies the *Roberts* edge detector to the current picture

Use **edge** to apply one of two forms of edge-detecting operator: a 3-point gradient magnitude, or the 4-point *Roberts* larger absolute diagonal difference.

Examples

```
edge 1 to 2
```

This command places the edge magnitude of picture 1 in picture 2.

```
edge roberts
```

This command applies the *Roberts* edge operator to the current picture.

Description

The **edge** command by default uses the the following 3-point gradient magnitude vector:

$$|(p(x+1,y)-p(x) , p(x,y+1)-p(x))|$$

where */numeric expression/* represents an absolute value.

You can specify the following 4-point *Roberts* operator using the **roberts** option:

$$max\{ |p(x+1,y+1)-p(x,y)| , |p(x+1,y)-p(x,y+1)| \}$$

In both cases, the top row and the right hand column are simply set to zero.

Note that thresholding (for example, using the command **calculate :sel>10**) may be useful for making yes/no edge decisions on the basis of the output produced by **edge.**

Notes

multi-layer pictures:	layers processed independently
forms used internally:	fp
see also:	**calculate**

edge

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
roberts	gradient magnitude	

edit

keys:	**[from]**	*<number>*	source macro
	[to]	*<number>*	output macro
options:	**verify**		print information about the process at the console

Use **edit** inside a Semper session to alter the text of an existing numbered macro, or to create a new macro.

Example

```
edit 4                                          enter editor
Editor ready:

for n=n1,n2; examine n; display a2              initial macro text
:>>>>>>>>>>>>>>>>>>####>>>>>>>>>####            move pointer and erase characters
...n=n1,n2; exa n; dis a2                       ...Indicates erased text

:e/a2/n/ c/; wait                               change a2 to n; add new text
...n=n1,n2; exa n; dis n; wait                  ...Indicates result of exchange

:w                                              write text to file and exit
Editing complete
```

Note that computer output is shown in **bold**.

Description

The **edit** command enters a simple editor within Semper. On entering the editor you see a window of the macro text with quotation marks '...' at the beginning or end of the window to indicate there is additional text that you cannot see. A colon (:) marks the current position of the pointer. To leave the editor type **w** (to save changes) or **q** (to quit without saving).

If you specify a source picture only, for example, **edit to 901**, you enter the editor with no existing text and can create a new macro.

The following list describes the commands that are available within the editor:

?	help – list editor commands
n	move to **n**ext command (**0n** moves to end of macro)
p	move to **p**revious command (**0p** moves to start of macro)
d	**d**elete (rest of) command
l	**l**ist full macro text

edit

w	**w**rite text to file and quit
q	**q**uit without saving changes
f/*s*/	**f**ind (move to next occurrence of) string *s*
s	repeat last **s**earch (**f**ind command)
c/*s*/	**c**ontinue macro with string *s* (i.e. add to end)
a/*s1*/*s2*/	**a**fter string *s1* insert string *s2*
b/*s1*/*s2*/	**b**efore string *s1* insert string *s2*
e/*s1*/*s2*/	**e**xchange: change string *s1* to string *s2*
x	repeat last e**x**change (**a**,**b**,**e** commands)
g/*s1*/*s2*/	perform exchange **g**lobally (throughout macro text)
>	move one character to the right
#	delete next character
_	replace character by space
%	change character to upper case
$	change character to lower case

The commands **a**, **b** and **e** operate from the prompt to the visible window, without moving the window. All string matching is case independent. You can use empty string matching to insert text at the current position, for example:

```
b//text/
```

inserts *text* at the current pointer position.

You can enter several commands to a line, if required, with no special separator and in upper or lower case. Commands involving strings use the character immediately following the command itself as the delimiting character (/ is used in the above examples), even if this is a space character. A number before a command indicates the number of repetitions of a command (or sequence of commands if they are enclosed in brackets), for example:

```
3n 12(f/jump/5>3$)
```

a zero placed before a command gives indefinite repetition, for example, **0p** moves to the start of the macro.

edit

Note that currently you cannot edit a *program* inside a Semper session. To edit a program you must write it to file, leave Semper and use a text editor provided by your operating system. For example:

```
list program 'progname' name 'filename'      write program to file
stop                                         leave Semper

   ...edit the file                          use an editor provided by
                                             your operating system

semper                                       restart Semper
add name 'filename'                          reload the program
```

Notes

forms used internally: integer
see also: **add, list program, spawn**

Defaults and Ranges

keys/options	defaults	range
[from]	*none*	valid picture number
[to]	source macro if defined, otherwise *none*	valid picture number
verify	verification on	

eget

This syntax is specific to...
Silicon Graphics workstations

keys:	[to]	*<number>*	output picture
	name	*'<text>'*	source file name
options:	**again**		if **name** is omitted and was used in a command to open a file for reading, open the same file again

You use **eget** to read images from a Type 2 Eikonix picture file.

Examples

```
eget 20 name 'eikpic'
```

This command reads data from the file *eikpic* into picture 20 on the current device.

Description

Files are searched for in the current directory and then throughout the search path. You can specify a full path name to avoid the path scan.

Notes

see also: **eput**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

eput

This syntax is specific to...
Silicon Graphics workstations

keys:	[from]	*<number>*	source picture
	name	*'<text>'*	output file name
options:	new		overwrite the output file if it already exists

You use **eput** to write a picture to a Type 2 Eikonix picture file.

Examples

```
eput'20 name 'eikpic'
```

This command writes data to the file *eikpic* from picture 20 on the current device.

```
eput 20 name 'eikpic' new
```

As for the above example but also allows an existing file named *eikpic* to be overwritten.

Description

Files are created in the current directory, unless you specify a pathname.

Notes

see also: **eget**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

erase

keys:	[]	*<number>*	picture/partition/frame to be erased
	size	*<x>, <y>*	dimensions of subregion to be erased
	position	*<x>, <y>*	position/offset of subregion
options:	**partition/frame/picture**		specify coordinate system to be used
	image/overlay		erase image/overlay memory only
	left/right, bottom/top		subregion position
	re/im		erase real or imaginary part of a complex display picture
	view		switch view to make the display region visible

Use **erase** to erase a display or a subregion of a display. You can also use **erase** to erase either an image or an overlay memory.

Examples

```
erase
```

This command erases the current display picture.

```
xwires dis:3 region; erase @region
```

This command erases the region indicated by the cursor in picture *dis:3*.

```
erase partition dis:5 image size 100 top left
```

This command erases the image memory only of the top left 100 pixels square subregion of partition *dis:5*.

```
erase frame 2 overlay view
```

This command erases the overlay memory only of frame 2 and displays the blank frame on the screen.

Description

By default, **erase** erases the current display picture. You can also use **erase** to erase a partition or frame. You can specify a subregion using the standard 2-D subregion keys and options, **size**, **position** etc. All frames of a multi-frame partition and all layers of a multi-layer picture are affected by **erase**. For further detail see *Appendix C: Semper Keys and Options.*

erase

For complex display pictures, where the real and imaginary parts of the image are displayed side by side, **erase** would normally erase both parts. If, however, you specify the option **re** or **im**, only the real or imaginary part will be erased.

Defaults and Ranges

keys/options	defaults	range
[]	current display if a picture or partition (held in the variable *display*), current frame (held in the variable *cframe*) if a frame	valid picture, partition or frame number
size	whole picture/partition/frame	less than or equal to the size of picture, partition or frame (integers)
position	position 0,0	within bounds of the picture, partition or frame (integers)
partition/frame/ picture	picture	
image/overlay	both, if you do not specify an option	
re/im	erase both parts of complex display picture	
view	view unchanged	

erode

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	times	*<number>*	force repeat count for erosion process
	neighbours	*<number>*	erode points with specified minimum number of clear neighbours
	with	*<number>*	picture containing user-supplied map functions
options:	**skeletonise**		erode object down to 8-connected skeleton
	ends		erode ends of protruding branches/hairs
	nodes		erode intersections from skeleton, separating branches
	outline		erode interior of object, leaving (8-connected) outline
	ol4		erode interior of objects, leaving 4-connected outline

Use **erode** to process a binary picture by removing a one-pixel border from an object, made up of non-zero pixels. This erosion process sometimes helps to clarify the relationship between objects or particles. Note that zero-value pixels are referred to as *clear* or *background* pixels and non-zero value pixels as *set* or *object* pixels.

Examples

```
erode display
```

This command strips a one-pixel wide border from all objects (by clearing pixels which have a clear neighbour).

```
erode 50 to 51 times 3
```

This command strips a three-pixel border from all objects (that is, it repeats the **erode** command three times).

```
erode neighbours 5
```

This command strips off border pixels which have at least five clear neighbours. This removes isolated points and lines without affecting the edge of extended objects.

erode

```
erode neighbours 8
```

This command clears isolated set pixels only.

```
erode skeleton
```

This command thins down objects to (8-connected) skeleton lines or curves.

```
erode ends
```

This command strips away all free ends (branches, hairs) from a skeleton or extended object.

```
erode display with 21
```

This command applies a user-defined operation defined in picture 21.

Description

The **erode** command is part of the *morphology* group of commands that perform morphological (shape) operations on binary pictures. Other morphology commands include **analyse**, **dilate** and **median**. You use **erode** on binary pictures, that is, pictures that have only two classes of value:

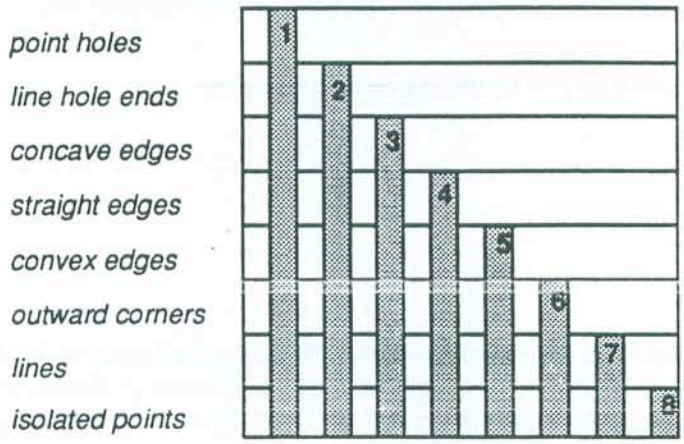
- zero for background pixels
- non-zero pixels belonging to the shape (object) of interest

Note that as Semper does not actually store pixels in single bit form, **erode** distinguishes between zero (*clear*) and non-zero (*set*) pixels instead. You can translate a picture into binary form using the **calculate** command, for example, **calculate :51>20** or **calculate :51<thr**.

By default, **erode** clears pixels that have at least one clear neighbour. You can use the **times** key to specify the number of times the **erode** command is repeated and so determine the depth of the erosion. If you set **times** to zero, **erode** repeats itself until the picture stabilizes (or until you *abandon* the operation). This is faster than putting **erode** in a **for** loop, as the command is able to omit picture rows that have already stabilized in previous passes. Note that the sequence **erode; dilate** is a useful way of smoothing object borders.

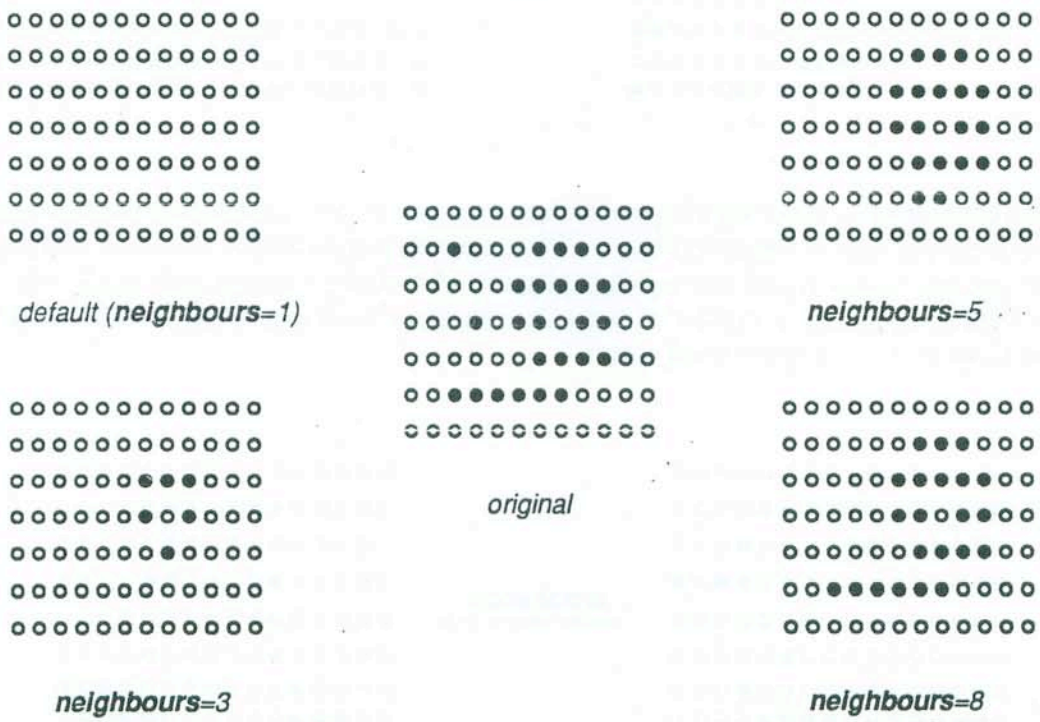
By default, **erode** clears pixels with at least one clear neighbour. You can use the **neighbours** key to perform more selective erosion as you increase this number towards the maximum of 8. The diagram given opposite shows the erosion of different features of an object according to the value of the **neighbours** key.

erode



Erosion process according to value of ***neighbours*** *key*

The following diagram shows the result of giving different values to the **neighbours** key for the same object configuration:



erode

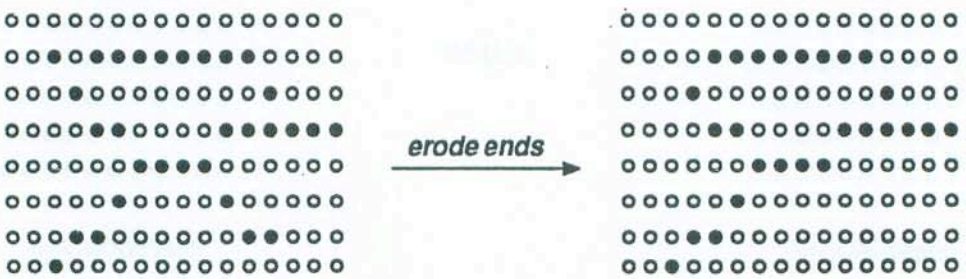
You can use the **with** key to supply your own mapping function for erosion.

The options that you can use with **erode** are described below:

- **skeletonise**
- **ends**
- **nodes**
- **outline, ol4**

Use the **skeletonise** option to thin down an object to (8-connected) skeleton lines or curves. (Refer to *Appendix G: Pixel Connectivity* for an explanation of 8-connectivity). Semper thins down objects in four passes, removing pixels from the top and left, the bottom and right, the bottom and left and the top and right in turn, without affecting the 8-connectivity of objects. By default, Semper repeats this sequence until the picture stabilises but you can limit the number of sequences using the **times** key. The diagram below illustrates the effect of the **skeleton** option.

Use the **ends** option to strip away all free ends (branches, hairs) from a skeleton or extended object. Note that Semper does not erode any branches that reach the picture border. By default, end pixels are eroded one by one in successive picture passes until no further change takes place. You can use the **times** key to limit the number of passes and therefore the amount of erosion. The diagram below illustrates the use of the **ends** option.



erode

Use the **nodes** option to delete pixels from a skeleton at intersections, and so separate branches in a picture so that you can measure them separately, for example. The diagram below illustrates the effect of the **nodes** option.

Use the **outline** option to erode interior points of objects (in a single pass) leaving minimal 8-connected outlines. Use the **ol4** option to leave outlines that are 4-connected rather than 8-connected.

Notes

see also: **analyse, calculate, dilate, median**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
times	1	zero or positive integer
neighbours	1 pixel	integer in range 1 to 8
with	*none*	valid picture number

event

options:	**keyboard**	select the keyboard event queue
	buttons	select the buttons/switches event queue
	pointer	select the pointer event queue
	break	select the break event queue (only with ***open*** and ***close***)
	open/close	open or close the selected queue
	count	return the number of entries in the selected queue(s)
	read	return the first entry from the selected queue(s)
	flush	empty the selected queue
	status	obtain status information about the selected queue(s)
	verify	(with ***status***) print the queue status information

The **event** command provides command level access to the internal Semper event queues. This command allows Semper programs to open, close, check, read and flush the keyboard, pointer and button queues and to open and close the break/abandon queue.

Examples

```
event open pointer
```

This command opens the pointer queue.

```
event count keyboard
```

This command returns the number of outstanding key presses in the variable *nk*.

```
event read flush keyboard
```

This command reads the first key press into the variable *n* and flushes the rest.

event

Description

The **event** command provides access to most of the facilities available in the Semper event queue mechanism. There are four internal event queues:

- keyboard — records key presses from the keyboard
- buttons — records mouse button presses
- pointer — records the movements of the mouse
- break — records any abandon or break requests

Use the **keyboard**, **buttons**, **pointer** and **break** options to select these queues. Note that operations on the break queue are limited to using the **open** and **close** options to switch on and off respectively the event handling on the queue. For example, an **event break close** command can be used to protect sensitive program sections from abandon requests.

If you do not specify a queue, by default all queues are selected, but if you use the **open**, **read** and **close** options you need to specify one or more queues explicitly.

The options **open** and **close** are used to switch queues on and off.

The **count** option returns the number of entries in the specified queue(s). The value is returned in the variables *nk* (keyboard queue), *np* (pointer queue) and *nb* (button queue).

The **read** option is used to extract an entry from the head of the specified queue(s). The values returned depend on the queue:

- the **keyboard** queue returns the key code in the variable *n*. If the queue is empty the value returned is –1. Refer to *Appendix J, ASCII Key Codes* for a translation of the key codes.

- The **pointer** queue returns the incremental *x* and *y* changes in *dx* and *dy*. If the queue is empty the values returned are 0,0.

- The **buttons** queue returns the number of the button closed (or zero if it is a button open event) in the variable *nbc*, the number of the button opened (or zero if it is a button close event) in the variable *nbo* and the bit packed button state in the variable *nbb*. If the queue is empty *nbc* and *nbo* are both returned as zero.

The **flush** option removes all the entries from the specified queue.

The **status** option examines the state of the specified queues, returning the value 0 if the queue is closed, 1 otherwise. The values are returned in the variables *nkq* (keyboard queue), *npq* (pointer queue) and *nbq* (button queue). In addition, if you specify the **verify** option with the **status** option, the current queue length and maximum length are written to the console output.

event

An example program, using the **event** command, is given below. In this program, Semper commands are used to invert the current look-up table every half second until the user presses a mouse button or a key on the keyboard.

```
! Read the current status of the queues
event status keyboard buttons

! Open those queues that are required
unless nkq event open keyboard; unless nbq event open buttons

! Loop with test
m: lut invert; wait 0.5; event count keyboard buttons;if (nb+nk)=0 jump m

! Flush the relevant queue
if nk>0 event flush keyboard; if nb>0 event flush buttons

! Reset the queue states
unless nkq event close keyboard; unless nbq event close buttons
```

Note that opening and closing queues should be done with care, as it is possible to hang a session if all input queues are closed. To be safe, the above script should either use **trap** statements to catch abandon requests, or should be bracketed with **event break close...event break open**.

Notes

variables set:

n (keycode with ***event read keyboard***)
nbo, nbc, nbb (button status with ***event read buttons***)
dx, dy (pointer moves with ***event read pointer***)
nkq, nbq, npq (queue open/close state with ***event status...***)
nk, nb, np (current queue length with ***event count...***)

examine

keys:	[ ]	*<number>*	picture to be examined
		<n1>, <n2>	range of pictures to be examined
	device	*<number>*	examine pictures on the specified device
	text	*'<text>'*	examine pictures whose titles contain the specified text
options:	**full/brief**		print details in full/abbreviated form
	all		examine all the pictures on the current device
	image/macro/fourier/ spectrum/correlation /undefined/...walsh/ histogram/plist/lut		examine pictures of a specified type only
	byte/integer/fp/complex		examine pictures of a specified form only
	list/curve		examine *Plist* pictures of a specified type only
	open/closed		examine *Plist* pictures of a specified curve type only

Use **examine** to list details of pictures stored on a device.

Examples

```
examine
```

This command types the standard set of details for the current picture

```
examine display full
```

This command displays full details of the current display picture

```
examine all brief
```

This command displays abbreviated details for all pictures on the current device.

```
examine device 4
```

This command displays details of all pictures on device 4.

examine

```
examine 50,60
```

This command types details for any pictures in the range 50 to 60.

```
examine fourier
```

This gives details of any *Fourier* pictures on the current device.

```
examine complex device 4
```

This command types details of any *Complex* pictures on device 4.

```
examine text 'eels'
```

This command types details of any pictures on the current device whose title includes the string 'eels'.

Description

examine prints details of the picture number, size, class, form and title, and the letters *wp* if a picture is write-protected. The **brief** form omits the title; the **full** form includes the data range, the creation date, the coordinate origin (as column, row and layer numbers relative to the left, top and back), and device-specific details such as the disc blocks occupied, the display black-white levels etc.

The option **all** is assumed if you quote a class name, form name, or **text** without an explicit range, such as 50, 60 and also if you use the **device** key.

Defaults and Ranges

keys/options	defaults	range
[]	current picture, held in the variable *select*	valid picture number
device	current device, held in the variable *cd*	1 to system limits (type ***show system***)
text	*none*	text string; maximum length is installation dependent

execute

keys:	**after**	*'<text>'*	execute the specified string after every Semper command
	before	*'<text>'*	execute the specified string before every Semper command
options:	**off/on**		disable/enable 'before and after' command execution

The **execute** command defines the actions to be carried out immediately before and after a Semper command is executed.

Examples

```
execute before 'type "About to execute command"'
```

This command types out the message *About to execute command* each time Semper goes to the user-interface for input. Note that this will *not* occur during the execution of library files.

```
execute off
```

This command disables the defined **execute** actions.

Description

If **execute** is used more than once, the actions are accumulated, not overwritten. If you enter several commands on one line using a semi-colon';', the action is executed before and after the entire sequence of Semper commands, not before each element in the string.

Defaults and Ranges

keys/options	defaults	range
after	*none*	text string; length is installation dependent
before	*none*	text string; length is installation dependent
off/on	execution on	

expand

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	ratio	*<n1>, <n2>*	specify resampling ratio
options:	**nneighbour**		use nearest neighbour instead of interpolating

Use the **expand** command to resample pictures horizontally, allowing you to compensate for the distortion that is introduced by framestores with non-square pixels. Typically these framestores are North American systems that are used with European camera signals.

Examples

```
expand display to 2 ratio 2,3
```

This command expands the current display to picture 2, resampling the picture on the basis that the ratio of the original to the required sampling rate is 2:3.

```
expand 1 to display ratio 6,5 nneighbour
```

This command expands picture 1 directly to the current display picture with a ratio of 6:5, using the nearest neighbour pixel value instead of interpolating.

Description

The **expand** command performs horizontal resampling for any pair of positive integer ratio values that you specify using the **ratio** key. Note that the processing of this command is most efficient for small ratio values. It is also possible to resample a picture using the **extract...uv** command, but this will tend to be slower for the ratios most commonly used.

For a standard (CCIR) camera input a sampling rate of 15MHz is required. Most North American (EIA or NTSC) framegrabbers only sample at 10MHz or 12.5MHz, requiring ratios of 2:3 (10MHz :15MHz) and 5:6 (12.5MHz:15MHz) in order to resample images.

By default, the new pixel values are generated by interpolation, unless you specify the **nneighbour** option, in which case the values of the nearest pixels in the source image are used.

The **expand** command is intended for use on pictures that you have just captured. It only processes the real part of *Complex* pictures.

expand

Notes

see also: **extract...uv**
multi-layer pictures: all layers processed
forms used internally: integer, fp

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
ratio	*none*	positive integers

extract

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	size	*<x>, <y>*	dimensions of extracted subregion
	position	*<x>, <y>*	position/offset of subregion
	sampling	*<number>*	sampling interval within subregion
	angle	*<number>*	rotate a subregion through the specified angle in an anticlockwise direction
	with	*<number>*	*Plist* picture listing source positions at which output samples are required
	mark	*<number>* *<yes or no>*	mark source subregion, if ***with***, mark sample points
	mkmode	*<number>*	if ***with***, mark mode
	mksize	*<number>*	if ***with***, mark size
options:	**left/right, top/bottom**		subregion position
	uv		extract samples on a lattice defined by the variables *u*, *u2* and *v*, *v2*
	average		average source samples over undersampled regions
	verify		print information about the process at the console
	nneighbour/bicubic		use nearest neighbour extraction or bicubic interpolation instead of bilinear interpolation

Use **extract** to extract rotated, (de)magnified or skewed regions of a picture. You can also use **extract** to rotate pictures, to repeat pictures in periodic arrays, and to average pixels in small blocks while undersampling.

Examples

```
extract size 100 top left to 2
```

This command extracts a 100 square region top left to picture 2 (like **cut**).

extract

```
extract size 512 display
```

This command repeats a small current picture periodically on the display.

```
xwires curve sampling 1; extract 1 2 with 999 nneighbour
```

This command evenly extracts spaced samples from picture 1, along an arbitrary curve drawn with the cursor, using nearest neighbour extraction.

```
extract 1 2 size 128 sampling 2 average
```

This command averages (the central 256 square of) picture 1 in 2 by 2 blocks to form a 128 square picture 2.

```
extract to display size 100 sampling .75 position x,y
```

This command interpolates a region 75 points square centred at x,y to form a 100 square display.

```
extract to 50 size 300,1 angle pi/4 position x,y; display
```

This command displays a diagonal line scan through x,y.

```
xwires curve sampling 1; extract 1 2 with 999
```

This command extracts evenly spaced samples from picture 1, along an arbitrary curve drawn with the cursor.

```
u=u/64,u2/64 v=v/64,v2/64; extract size 64 uv
```

This command extracts a single unit cell of a periodic image with lattice base vectors *u,v*.

Description

Use the **extract** command to perform the following basic actions:

- extract a region (using bilinear interpolation if necessary)
- expand the extracted region by **sampling**
- rotate the region using **angle**

Note that if the specified region overflows the source picture boundary, the source picture is treated as continuing periodically in both directions.

extract

Use the **with** key to perform an arbitrary geometric transformation. Supply a *Plist* picture number for **with** and Semper generates an output picture with the same column and row length as the *Plist*, but with each position replaced by a value interpolated from the source picture at that position. For example, the following sequence of commands resamples picture 50 on a radial polar grid (radius 0–49, angle 0–359 degrees).

```
create 3 plist size 360,50,2
origin bottom left
calculate ifelse(z,y*sin(rad(x)),y*cos(rad(x)))
extract 50 to 51 with 3
```

See *Appendix A: Picture Types* for details of *Plist* pictures.

Use the **mark** key to mark the source subregion or, if you specify **with**, the sample points. Use the keys **mkmode** and **mksize** to specify the style of marking. Refer to *Appendix C, Semper Keys and Options* for details of the keys **mark**, **mkmode** and **mksize**.

The **uv** option causes **extract** to distort the extracted region so that the vector (*u,u2*) defined by the current value of the variables becomes one point to the right in the output and (*v,v2*) one point upwards. **sampling** and **angle** are ignored in these circumstances.

Apply the **average** option only when undersampling without interpolation, as it may be useful for preserving the original signal-to-noise ratio. The averaged blocks, exceptionally, have the top left rather than centre at the nominal sampling position, which allows you to reduce a *2n* square picture to a *n* square without an additional **position** adjustment.

The **extract** command uses bilinear interpolation between the four neighbouring source pixels to produce the output values at each sample position but you can also use the **nneighbour** option to specify nearest neighbour extraction or the **bicubic** option to specify bicubic interpolation:

- If you specify the **nneighbour** option, the value of the nearest neighbouring source pixel is taken as the output value.
- If you specify the **bicubic** option, the output pixel values are generated using bicubic interpolation between the 4 by 4 array of source pixels that surrounds each sample position.

Roughly speaking, nearest neighbour extraction is twice as fast as bilinear interpolation and bicubic interpolation is twice as slow. Nearest neighbour extraction is useful when it is necessary to avoid the artefacts caused by the interpolation process. Bicubic interpolation avoids the sharp discontinuities of intensity gradient which becomes more apparent when using bilinear interpolation in conjunction with very small sampling intervals (high magnification).

extract

Note that you may find **extract** slow in operation for large pictures, if the rows to be extracted lie at large angles to those of the source. In this case, use **rotate** to perform large angle rotations (even if initially you have to extract a larger region than you want).

If the source picture origin is included in the extracted region, it is recorded as the origin of the output (rounded to the nearest pixel), unless **extract with** is used.

Notes

display marking:	extracted region
multi-layer pictures:	layers processed independently
forms used internally:	integer, fp, complex
variables used:	*u, u2, v, v2* (if ***uv***, base vectors defining the size and shape of a skewed region)
see also:	**cut**, **rotate**, **warp**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0,0	real numbers
sampling	1 sampling interval	positive real number
angle	angle 0	real number in range 0 to 2π
mark	mark off	see *Appendix C*
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
verify	verification off	
nneighbour/bicubic	bilinear	

find

keys:	**[from]**	*<number>*	picture to be searched
	position	*<x>, <y>*	centre of circular region to be searched
	radius	*<number>*	radius of circular region to be searched
	mark	*<number>*	mark resulting positions and circular region
		<yes *or* no>	
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**highest/lowest/cm**		find highest point, lowest point or centre of mass
	negative		if ***cm***, include negative pixels in search
	squared		if ***cm***, find centre of mass of squared pixels
	iterated		if ***cm***, repeat the centre of mass location once, with a region shifted so as to be centred at the centre of mass found initially
	verify		print information about the process at the console

Use **find** to find the highest point, lowest point or centre of mass of a picture or circular subregion, returning its coordinates as *x, y* and the pixel values or mass as *t*.

Examples

```
find 51 verify
```

This command finds the highest point in picture 51 and sets *x, y* to its coordinates. It prints information about the processing at the console.

```
find lowest radius 35 position x,y
```

This command finds the lowest point of the current picture within 35 pixels of *x, y*.

```
xwires circle; find cm @circle nonegative
```

This command finds the centre of mass of a circular region that you indicate with the cursor (using three perimeter points), ignoring negative source pixels.

Description

Use the **mark** key to mark the resulting positions from **find** or the circular region. Use the **mkmode** and **mksize** keys to determine the type of display marking. Refer to *Appendix C, Semper Keys and Options* for details of the keys **mark**, **mkmode** and **mksize**.

find

If you use **find...cm** to find the centre of mass of a picture, three additional options can be specified: **negative**, **squared**, **iterated**:

- Use the **negative** option to include negative pixels in the search.
- Use the **squared** option to square pixels before finding the centre of mass.
- Use the **iterated** option to repeat the centre of mass determination once, with the second region centred at the centre of mass of the first. This can be useful for locating the centre of mass of a signal on a large background which dominates the centre of mass completely.

Notes

display marking:	subregion scanned, position found
multi-layer pictures:	faulted
forms used internally:	fp
variables set:	*x, y* (coordinates of highest/lowest/centre of mass point)
	t (picture value at selected point, or mass if ***cm***)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
position	position 0,0	within bounds of picture (real numbers)
radius	infinite radius	positive real number
mark	mark off	see *Appendix C*
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
highest/lowest/cm	highest	
negative	include negative pixels in search	
verify	verification off	

fir

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	with	*<number>*	picture containing filter kernel
	radius	*<number>*	if ***gaussian***, rms kernel radius
options:	**add/subtract**		produces filter output only, added to or subtracted from source
	modulus		if ***add/subtract***, produce modulus of output
	separable		apply 1-D kernel separately in *x* and *y* directions
	gaussian/laplacean		use gaussian profile smoothing kernel, or 3x3 point laplacean kernel

Use **fir** to apply *fir* filters (small block convolution operators), which replace each pixel by a weighted average of its neighbours. You can define the kernel weights in a small subsidiary picture, or use the standard forms generated internally by **fir**.

Examples

```
fir 1 to 2 with 51
```

This command produces a picture 2 by applying the kernel in picture 51 to picture 1.

```
fir laplacean subtract to display
```

This command displays a sharpened version of the current picture.

```
fir laplacean modulus
```

This command obtains the magnitude of the laplacean of the current picture. This is a rather noise sensitive form of edge operator.

```
fir display gaussian radius 2
```

This command smooths the picture display.

```
create 90 size 5,1; p -2=1,2,3,2,1; fir 1 with 90 separable
```

This command defines a 1-D smoothing kernel and applies it to picture 1 in *x* and *y* directions independently.

fir

Description

If you supply a kernel for the **fir** command, note that kernels need not be square, and may be up to 21 points wide. For large kernels you may well find it faster to effect the convolution by multiplying the *Fourier* transform of the source picture by that of the kernel (embedded in a zero picture of the same size).

There are three ways of defining a kernel:

- use the **with** key and supply your own kernel.
- you can use the **gaussian** option, which generates a square kernel of the form:

$$e^{-\left(\frac{d^2}{2r^2}\right)}$$

  for distance d from the centre, truncated beyond e^{-1} with a minimum size of 3, and normalised to a unit sum (which preserves the source mean). The kernel is in fact separated for speed.

- you can use the **laplacean** option, which invokes a fixed kernel of:

$$\frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{or} \quad [\,1 \;\; -2 \;\; 1\,] \quad \text{for 1-D source pictures}$$

If your kernel has the form:

$k(x,y)=p(x)p(y)$

that is, it separates into identical x and y factors, you will find execution is faster if you supply the separated form only and use the **separable** option. The last command given in the **Examples** section, uses a 1-D kernel (1 2 3 2 1) and applies an equivalent kernel:

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 3 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

With each kernel, you can specify the **add** or **subtract** options, which adds or subtracts the filter output to or from the original picture. This is done in most cases simply by adjustment of the supplied kernel without additional computation.

fir

Note that while a kernel of a size up to 21 square is acceptable, the **fir** command processes square, horizontal or vertical 3- and 5- point kernels more efficiently (that is, 5x5, 5x1, 1x5, 3x3, 3x1 and 1x3 kernels). Edge pixels are treated as if the edge pixels of the source continued indefinitely outwards.

Notes

multi-layer pictures: layers processed independently
forms used internally: fp

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none*	valid picture number
radius	radius 1	positive real number
add/subtract	apply the kernel to the source picture	

fit

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
options:	**subtract**		subtract fitted ramp from source
	divide		divide source by fitted ramp (after ***subtract***, if set)
	constant		include constant from subtracted/divided ramp

Use **fit** to fit a linear ramp function to a picture. You can subtract and divide by the ramp in order to level background variations from one side of a picture to another.

Examples

```
fit
```

This command fits a ramp *Ax+By+C* to the current picture.

```
fit display noconstant subtract
```

This command subtracts a fitted ramp *Ax+By* from the display (omitting the **constant** term C keeps the data range within that accommodated by the display).

```
fit 50 divide to 51 fp
```

This command divides picture 50 by a fitted ramp *Ax+By+C*, forcing the output class to be floating point since the value will be spread either side of 1.

Note that in the above examples the fitted ramp coefficients are returned in variables *a*, *b* and *c*.

Description

fit uses a *least squares* fit criterion to fit the coefficients of a linear function:

Ax+By+C

to the source picture, where *x* and *y* are the picture coordinates. (Note that for a 1-D picture, **fit** uses the linear function *Ax+C*).

If you use the **subtract** option, **fit** produces an output picture by subtracting the fitted ramp from the source. If you use the **divide** option, **fit** divides by the fitted ramp. If you specify both **subtract** and **divide** the subtraction is performed first.

If you set the **noconstant** option the constant term *C* is omitted from the subtracted and/or divided ramp, and is useful for levelling up a background without shifting the entire data range, as is shown in the second command example.

fit

The most appropriate way of correcting an image for background variations depends on how the variations have arisen and what is required. Some ground rules for using **fit** are given below:

- You **divide** when your image has the form *l.t* (for example, uneven illumination *l* on a scene with reflectivity or transmissivity *t*).
- You **subtract** and **divide** when it has the form *l.(1+t)* (for example, similar illumination with small variations *t* about 1 in reflectivity or transmissivity; or an image whose local contrast *t*, fractional intensity deviation, you need to extract).
- You **subtract** when an image has the form *l+t* (for example, additional stray light *l* entering a camera).

Note that **subtracting** is a useful *ad hoc* expedient when an exact model for the imaging is not available or is too complicated.

If you need to compensate for more than the linear part of the variation, try using the command **lmean** over a large block size to obtain the background function itself and then use **calculate** to subtract/divide the background function.

Notes

multi-layer pictures:	faulted	
forms used internally:	fp	
variables set:	*a, b, c*	(coefficients of ramp *Ax+By+C*. *C* is not set if you use the **noconstant** option).
see also:		**lmean, calculate**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
constant	constant set	

flc

keys:	**[from]**	*<number>*	source picture
	to	*<number>*	output picture, fitting *all peaks*
	size	*<x>, <y>*	dimensions of output picture
	radius	*<number>*	maximum radius considered in *all peaks* mode (radius given in pixels)
	position	*<x>, <y>*	position of single peak to be fitted in picture coordinates
	line	*<n1>, <n2>*	position of single peak to be fitted in terms of base vectors *u, u2* and *v, v2*
	snr	*<number>*	threshold intensity signal-to-noise ratio below which peaks are discounted
	mark	*<number>*	mark fitted sites
		<yes *or* no>	
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**wiener**		perform *Wiener*-optimised least squares peak fit
	verify		print information about fit at the console

Use **flc** (*fourier lattice component*) to perform lattice averaging by the transform peak profile fitting method. **flc** fits lattice (fourier) components to the isolated peaks found in the transform of a periodic image, discommensurate with the sampling lattice.

Examples

```
flc position 52.6,20.9
```

This command, given a half-plane *Fourier* current picture, fits a transform peak at the indicated position, returning the component value as *t, t2.*

```
u=20.3,2.5 v=1.2,16.8; flc line 3,-1
```

This command fits the peak with indices 3,–1 on the lattice defined by *u* and *v.*

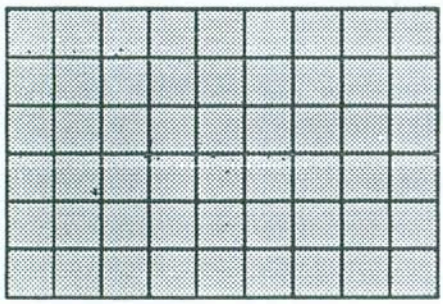
```
fourier 1; u=..,.. v=..,..; flc to 2 size 65,128; image
```

This command transforms picture 1, fits all sites on the lattice defined by *u* and *v*, and recovers a single unit cell of the image. *u* and *v* might be fitted more accurately using **xwires list; library lattice; base 999** after displaying the power spectrum (*ps*).

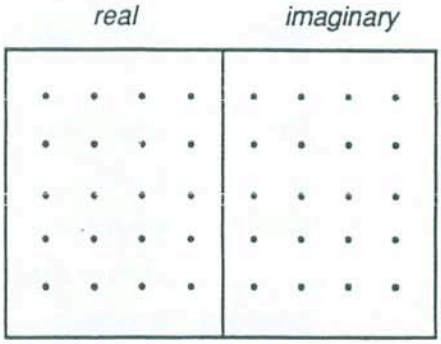
flc

Description

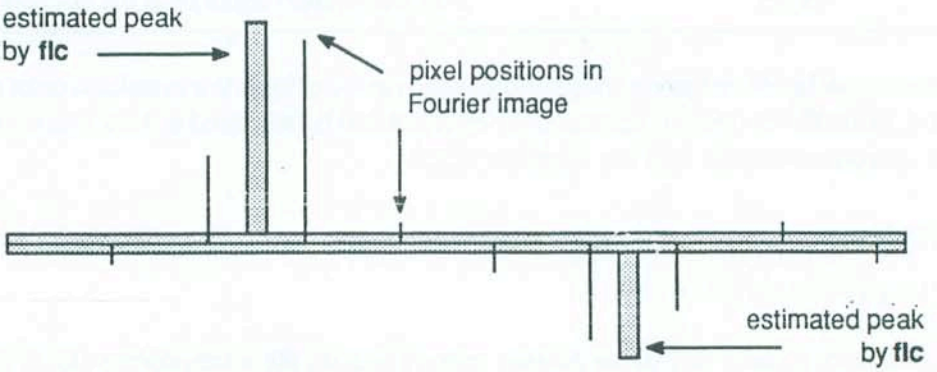
The *fourier* transform of a 2-D periodic image consists of an array of isolated peaks at positions forming a lattice that is reciprocal to the real space lattice. Unless the signal periodicity is an exact multiple of the sampling lattice in both directions, the peaks fall between the transform sampling values and (ideally) have a crossed sinc profile. **flc** allows you to recover the peak values by fitting the samples. The diagram below illustrates the stages of the **flc** command.



Stage 1. 2-D periodic image



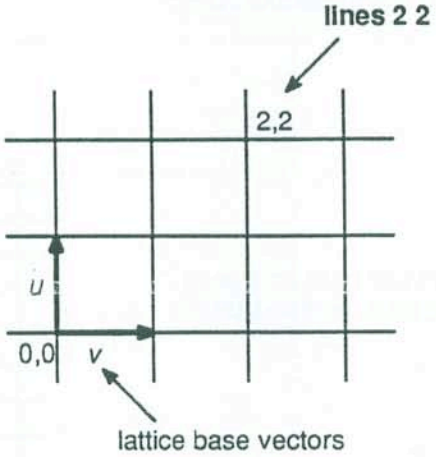
Stage 2. Fourier transform – regular lattice of points



Stage 3. Peaks extrapolated by **flc**

If you do not specify an output picture using **to**, only one peak is fitted and its values are returned in the variables *t* and *t2*. You can determine that the position is fitted directly using **position** or indirectly using lattice indices specified by **line**, as is shown in the following diagram. The source (and output) must be a half-plane *Fourier* picture.

flc



Reference by **lines** key using the lattice index

If you specify an output picture, all sites on the lattice defined by *u* and *v* are fitted and the resulting values are stored in an output picture. The output picture is a half-plane *Fourier*, size 17 by 32, unless you specify different dimensions using **size**. Note that you can exclude peaks beyond a cut-off radius from the origin using the **radius** key.

Pixel (*h,k*) receives the value for the site with indices (*h,k*) etc. so that on inverse transformation the output yields one unit cell of the structure, sampled on a lattice exactly subdividing the unit cell. You can restore normal cartesian (square) sampling using the command **extract uv.**. (Use **library reciprocal** and **library invert** in turn to convert the *fourier* plane base vectors *u and v* to suitable values for **extract**).

The **verify** option causes details of the fitting to be reported at the console.

The following steps are performed for each peak that is fitted:

Step 1 The mean noise intensity is estimated from the peripheral values of an 8 point square region around the peak.

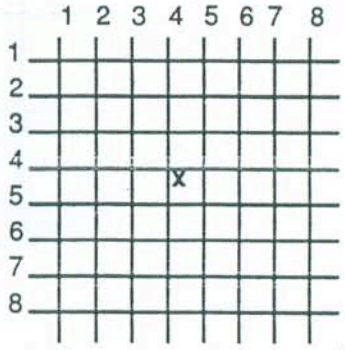
Step 2 The signal intensity is estimated via the integrated intensity in a 2 point square region, subtracting the estimated noise intensity.

Step 3 If the resulting signal-to-noise intensity ratio is less than 3 (or the value of the key **snr**), the peak is discarded (with a warning message in single peak mode).

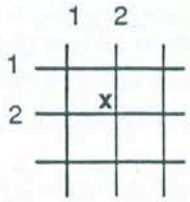
Step 4 A simple estimate of the lattice *fourier* component is made by combining a modulus given by the square root of the estimated signal intensity with a phase given by the phase of the average of the 4 points around the peak position. (2 or 1 points as necessary if one or both of the peak coordinates is in fact integral).

The following diagrams illustrate these four steps:

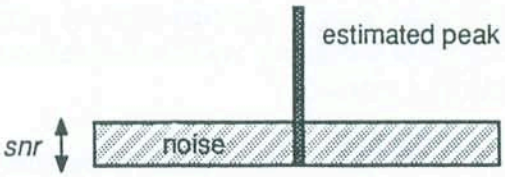
Step 1. Mean noise estimated from an 8 point square region around the peak (**x** marks peak)



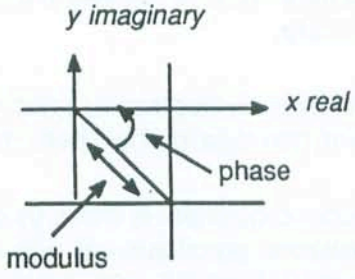
Step 2. ***snr*** = *integrated intensity from a 2 point square region minus estimated noise intensity* (**x** marks peak)

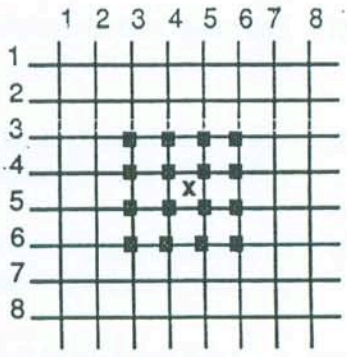


Step 3. If signal-to-noise intensity is less than 3 or value of ***snr***, discard the peak



Step 4. Estimate of fourier lattice component by combining modulus and phase – phase from the average of the 4 points around the peak position. (**x** marks the peak)





flc

If you specify the **wiener** option, **flc** performs two further steps:

- The ideal crossed sinc profile values are generated for a 4 point square region and a better value for the lattice component is obtained from the corresponding data samples *d* using an expression of the form:
  $t = sum(s.d) / [sum(s.s) + nu / < |t|^2 >]$
  where *nu* denotes the mean noise intensity as estimated earlier, and $< |t|^2 >$ the expected signal intensity, estimated using an initial evaluation with *nu* set to zero.

- The extent to which the local data conform to the expected crossed sinc profile is assessed using a *Chi–squared* difference between predicted and actual transform values, for report on the console.

Do *not* use the **wiener** option under the following conditions:

- the periodic signal does not fill the entire original image field
- the periodic signal is imperfect so that the peak profiles are distorted
- you do not accurately know the reciprocal lattice

If these conditions do not apply, **wiener** provides a more reliable estimate of the lattice fourier component than the usual mode.

If you indicate a display using the **mark** key, **flc** marks the fitted sites in the style and size specified by **mkmode** and **mksize**. Refer to *Appendix C, Semper Keys and Options* for details of the keys **mkmode** and **mksize**.

Notes

display marking:	sites fitted
multi-layer pictures:	faulted
forms used internally:	complex
variables used:	*u, u2, v, v2* (fourier space lattice base vectors when peak position is specified using line)
variables set:	*t, t2* (real, imaginary parts of fitted fourier component (single peak modes)
see also:	**base, extract, lattice**

flc

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
to	*none*	valid picture number
size	picture size 17, 32	positive integers
radius	*none*	positive real number
position	*none*	within bounds of picture (real numbers)
line	*none*	within terms of lattice index
snr	ratio 3	positive real number
mark	mark off	see *Appendix C*
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
verify	verification off	

flood

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	position	*<x>,<y>*	single seed point position
	with	*<number>*	seed picture (binary image or position list)
options:	**fg**		flood foreground regions
	bg		flood background regions

You use the **flood** command to flood foreground and/or background regions defined in the binary source image. A flooded region is any region which contains one or more seed points. The output picture will contain a binary image in which pixels corresponding to flooded regions are set to 1 and everything else is set to 0. You can specify a single seed point with the **position** key. Multiple seed points can also be supplied in a Semper picture by means of the **with** key, either in the form of a position list or as a binary image in which the foreground pixels define the seed points. The **fg** and **bg** options specify whether to flood foreground or background regions (or both). By default, only foreground regions are flooded.

Examples

```
flood position 20,30
```

This command removes all foreground regions in the current picture except the one (if any) which includes the pixel at position (20,30).

```
flood 1 with 2 to 3 bg
```

This command floods the background regions of picture 1 which overlap foreground regions of picture 2 and outputs the result to picture 3.

```
display 2:4; xwires list noverify
flood 2:4 with 999 to 2:5 fg bg
calculate ifelse(2:5,~2:4,2:4) to 2:5; display 2:5
```

This example allows you to point with the cursor to any set of connected regions (foreground and/or background) in picture 2:4 and invert the selected regions.

flood

Description

The **flood** command has a number of uses:

- It provides a general seed point filling facility.
- It allows you to add or remove connected regions in a binary image simply by pointing to them (see last example above).
- It allows you to explore the connectivity relationships between regions.

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels. Background pixels are always treated as being 4-connected, that is, two background pixels are 4-connected only if they are either horizontally or vertically adjacent to each other. Foreground pixels are treated as being 8-connected, that is, two foreground pixels are 8-connected if they are in any way adjacent.

Because of the difference in the way foreground and background pixels are connected, the result of labelling the foreground regions of an image can be quite different from the result of labelling the background regions of the negated version of the same image.

For example,

Source image

```
0 0 0 1 1 1 1 1 1
0 1 1 0 0 0 1 0 1
0 1 1[0]1 1 0 0 1
0 0 0 0[1]1 0 1 1
0 0 0 1 1[0]0 1 1
0 0 1 1 0 0 1 0 0
0 0 1 1 1 1 0 0 0
```

flood fg

```
0 0 0 1 1 1 1 1 1
0 1 1 0 0 0 1 0 1
0 1 1 0 1 1 0 0 1
0 0 0 0 1 1 0 1 1
0 0 0 1 1 0 0 1 1
0 0 1 1 0 0 1 0 0
0 0 1 1 1 1 0 0 0
```

negate

```
1 1 1 0 0 0 0 0 0
1 0 0 1 1 1 0 1 0
1 0 0[1]0 0 1 1 0
1 1 1 1[0]0 1 0 0
1 1 1 0 0[1]1 0 0
1 1 0 0 1 1 0 1 1
1 1 0 0 0 0 1 1 1
```

flood bg

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0
0 0 0 0 1 1 0 0 0
0 0 0 1 1 0 0 0 0
0 0 1 1 0 0 0 0 0
0 0 1 1 1 1 0 0 0
```

where [ ] = seed points.

flood

The seed picture is specified by means of the **with** key. If the seed picture is a binary image, it must have the same size as the source picture. If the seed picture is a position list, any positions that fall outside the limits of the source picture are ignored. The positions are rounded to the nearest source pixel position. The seed positions are used to generate a temporary, byte seed image with the same size as the source picture.

Likewise, if the single seed point position specified with the **position** key falls outside the source picture, it is ignored, otherwise it is rounded to the nearest source pixel position.

If no valid seed points have been specified or if none of the seed points are contained within any of the regions that are allowed to be flooded, a blank image is output (all pixels are set to zero).

The **flood** command produces the image containing the flooded regions in two or more passes through the data (two passes except for very detailed images).

The results of the first pass are output to a temporary, integer form, disc picture with the same dimensions as the source picture. The final result is a binary image which is always output to a picture of byte form.

The range of the final result is stored in the output picture label.

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none*	valid picture number
position	*none*	real numbers
fg/bg	flood foreground regions	

flush

***flush** takes no arguments*

Use **flush** to force an actual disc file update from buffer in main memory. This guarantees that a disc file contains your latest changes in case of computer failure.

Description

Most Semper installations buffer part of their disc files in main memory for speed and only update actual files periodically (whenever their directories are altered, for example). There is a certain risk of data loss attached to this, as some installations do not guarantee to return the contents of the buffers in the event of a session abort (with the exception of VAX/VMS installations). Use **flush** to minimise this risk.

for

for <*variable name*> [=] <*n1*>, <*n2*>, [<*step size*>]...	***for*** uses a special syntax. Specify a variable that is to take the values starting with *n1* and ending with *n2* in an incremented loop. You can specify the amount by which the variable is incremented using *step size* (this is optional).

You can make Semper repeat a group of commands by surrounding them with the commands **for...loop**.

Examples

```
for n=11,14; ps n to n+10
section; survey; loop
```

This sequence of commands produces in pictures 21 to 24 the rotational averages of the power spectra of pictures 11 to 14, and types their ranges.

```
for x 1,0, -0.25; type x; loop
```

This sequence of commands types 1, 0.75, 0.5, 0.25 and 0 in turn on successive lines.

Description

You end a **for** loop by specifying the **loop** command. If you omit a step size for the loop the default step is 1 or –1 according to the loop direction. Note that you cannot specify a zero step size.

You can extend a **for...loop** over several lines, as in the first command example. If you are working interactively, the terminal prompt reminds you about active loops. There is also an installation dependent maximum loop length, typically 1000 characters. This limit applies in *run* files, but not in *library* programs.

The loop variable is local to its **for** loop, that is, its original state or value is restored when the loop ends. (See the **local** command for further detail).

You can *nest* **for** loops up to an installation dependent maximum depth, which is typically 6. Each **for** command in a nested loop requires a corresponding **loop** command. Note that you can specify a variable name with **loop** to clarify your structure. For example:

```
for s=1,2
for n=5,8; type n; loop n
type 'who do we appreciate?...Semper 6!'
loop s
```

for

You can also re-use the same loop variable in each nested loop as each variable is *local* to its loop.

To break out of a **for...loop** use the command **break** or **next**.

Notes

restrictions: a special restriction applies to **for** loops with respect to numbered macros (which will not be supported in later increases of Semper). Loops are not executed correctly in a numbered macro when the macro itself is called from within a library program.

see also: **break, loop, next**

fourier

keys:	[from]	<*number*>	source picture
	[to]	<*number*>	output picture

Use **fourier** to calculate a 1-D or 2-D discrete *fourier* transform of a picture.

Examples

```
fourier 1 to 4
```

This command transforms picture 1 to picture 4.

Description

Note that a source picture need not be square, but that its size must be a power of two.

For *byte*, *integer* and *fp* source pictures, the output of the **fourier** command is a half-plane transform (*Fourier*, *Complex*). For a *Complex* source, it is full-plane; you cannot force a different output form.

The output cannot be sent directly to the display (**fourier to display**) because of incompatible forms and ranges used during intermediate stages of the transform calculation.

To invert a transform and recover the original image, use the **image** command.

Notes

restrictions:	image size must be a power of two unsuitable for direct output to display
multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**image, fullplane, halfplane**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

fscontrol

This command is specific to...
PC + MRC500 framestore
PC + Synoptics Synergy framestore

fscontrol *takes no arguments*

fscontrol is an interactive framestore control command for the MRC500 and Synergy processor. This allows you to execute framestore commands, read and write framestore program memory, reset/halt/run-up the board, download command files, etc. All this is available from a completely separate command interpreter, which you enter by typing the command **fscontrol**. Note that this command can have far-reaching effects and is intended as a debugging tool only for the experienced user.

fscontrol provides basic help information on use; type *H* for help after typing **fscontrol**.

Notes:

see also: **fsxecute** (this command provides control of the framestore directly from Semper itself).

fsxecute

This command is specific to...
PC + MRC500 framestore
PC + Synoptics Synergy framestore

keys:	**cmd**	*<number>*	specify a command number to execute
	arg, ar2....ar8	*<number>*	specify command argument(s)
	wait	*<number>*	specify number of seconds to wait for command to finish
	put	*<number>*	write the specified picture to framestore data-buffer
	get	*<number>*	read back the specified picture from framestore data-buffer
	gnumber	*<number>*	specify picture width to ***get***
	gfrom	*<number>*	start address of information in 2100 program memory to ***get***
	name	*'<text>'*	specify a filename, containing a command file for the ***load*** option
	port	*<number>*	specify an i/o port base address
options:	**reset**		reset the board
	halt		halt the board
	run		run the board
	load		download the framestore program memory from disc

The **fsxecute** command provides direct control of the MRC500 or *Synoptics* Synergy framestore from Semper itself. Note that this command can have far-reaching effects and is intended only for the experienced user.

Examples

```
fsxecute reset
```

This command resets the board.

```
fsxecute put 3 cmd 9
```

This command loads data from picture 3 into the overlay look-up table.

fsxecute

```
fsxecute 1 0,0,512,1,6 get 7 gnumber 512
```

This command reads the first 512 pixels of row 0 into picture 7.

```
fsxecute load name 'mine' run
```

This command loads in a command file called *mine* and sets the board into the *run* state.

```
rep: trap=40 fsxecute; type frc; jump rep
```

This command displays the status on an interrupt (for example, during slow scan), held in the variable *frc* (framestore return code).

Description

The syntax and functions of the **fsxecute** command can be divided into the following three areas:

- framestore command execution (**cmd**, **arg**, **ar2...ar8**, **wait**)
- reading and writing picture data (**put**, **get**, **gnumber**, **gfrom**)
- hardware control/ program loading (**reset**, **halt**, **run**, **load**, **name**, **port**)

Executing framestore commands

You can execute a framestore command by setting the key **cmd** to the number of the required command and the keys **arg**, **ar2**, **ar3...ar8** to the value of any required parameters. Refer to *Chapter 2, Command Set* of the following manual:

Synergy Image Processing Subsystem Software Manual

for a detailed description of each command and its corresponding number and arguments.

For example, to turn on a single pixel in overlay 0, type the command:

```
fsxecute cmd 15 args x,y,0,0
```

This executes the required command, waits for it to finish, and sets the value of the variable *frc* (framestore return code) to the value of the returned status. Because this is the most common use of the **fsxecute** command, the default keywords **$1**, **$2**, **$22...$28** are used so that you can type:

```
fsxecute 15 x,y,0,0
```

which has the same effect as the previous command example.

fsxecute

You can also use the **wait** key with the following effect:

wait=0	wait until a command finishes or a user interrupt (*frc* = status)
wait<0	do not wait (*frc*=–1)
wait>0	wait for specified number of seconds, then return *frc*=status if the command is finished or *frc*=–1 if it is not

The variable *frc* holds the *framestore return code.* The default value for **wait** is 0. Note that if a command returns a non-zero status, **fsxecute** assumes that this indicates an error and aborts using error code 40. You may need to avoid this for a few commands by trapping the error using **trap=40**.

Reading and writing picture data

You can write data to the framestore data-buffer by setting the key **put** to a picture number. If the picture's storage class is *fp* or *complex*, the real part is converted to 2100 fixed-point format. The rows are sent in order from the top row, and layers from the back layer. The picture class is ignored, and Semper only faults the picture if too large (that is, if its total pixels exceed 768).

You can read data back from the framestore by setting the key **get** to a picture number. By default, Semper reads back the whole data-buffer into an *integer* form *Image* class picture of size 768*1*1. You can read smaller or larger amounts of data by setting the key **gnumber** to the desired picture width. You can read data from other places in 2100 program memory (for example, the status buffer) by setting **gfrom** to the start address. You can change the class of the new picture (*integer* by default) using the general options **byte**, **fp** and **complex**. (For further detail of general keys and options, refer to *Appendix C: Semper Keys and Options*). For *Fp* and *Complex* class pictures the data is treated as 2100 fixed-point, and converted from the raw integers accordingly. The resulting picture is always 1-D, and has its origin set at the left.

The 2100 fixed-point format is described in the following manual:

Synergy Image Processing Subsystem Software Manual

and is a fractional representation where 0x8000= –1, 0x7FFF= (almost) +1.

Hardware control and program loading

You can **reset**, **halt** and **run** the board. If you specify the **load** option the framestore program memory is downloaded from disc. By default, the program memory is loaded from the file *semper.cmd*, unless you use the **name** key to specify a different filename (you do not need to specify the extension *.cmd* as Semper does this for you).

You can set the key **port** to specify an i/o port base address. This overwrites the previous value for the system and, with care, you can use this to switch Semper output between framestores.

fsxecute

Further information

Semper performs the available keys and options in the following order:

port
halt
reset
load
run
put
(perform **cmd**)
wait
(set *frc*)
get

The **run** option is implicit if any command is to be executed, and **reset** is implicit if you specify the **load** option.

Notes

see also: **fscontrol**
variables set: *frc* (framestore return code)

Defaults and Ranges

keys/options	defaults	range
cmd	command –1	integer in range 0 to 41
arg, ar2...ar8	arg=0, ar2=0...ar8=0	valid command argument
wait	0	real number
put	*none*	valid picture number
get	*none*	valid picture number
gnumber	768	positive integer
gfrom	*none*	positive integer
name	*semper.cmd*	valid filename
port	*none*	positive integer

fullplane

keys:	[from]	<number>	source picture
	[to]	<number>	output picture

Use **fullplane** to convert a half-plane *Fourier* or *Spectrum* picture into its full-plane equivalent.

Examples

```
fullplane 2 to 3
```

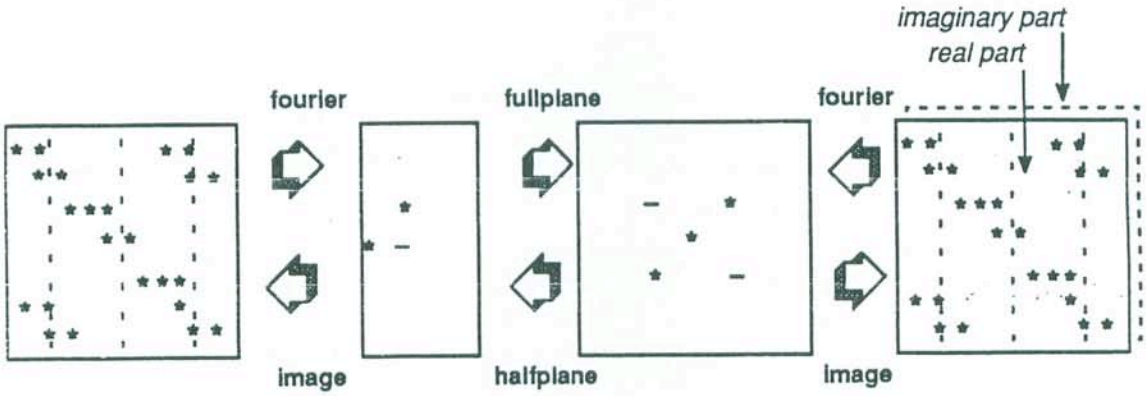
This command places the full-plane equivalent of a half-plane transform 2 in 3.

```
ps; fullplane; display
```

This command displays the power spectrum of the current picture in full-plane form.

Description

fullplane converts a half-plane picture using the (conjugate) symmetry of the pixels. Use the **halfplane** command to reverse this process. The following diagram illustrates the **fullplane** command.



Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**fourier, halfplane, ps**

fullplane

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

gaussian

keys:	**[to]**	*<number>*	output picture
	size	*<x>, <y>, <z>*	dimensions of output picture
	radius	*<number>*	*rms* width in section of generated function

Use **gaussian** to generate a picture containing a single gaussian-profile peak at the origin, for test or other purposes.

Examples

```
gaussian size 128,1
```

This command replaces the current picture with a 128 point 1-D picture (form *Fp*), with a gaussian peak (rms width 16 pixels) at its origin.

```
gaussian to 2 size 40,40,40 radius 5
```

This command generates a 3-D gaussian peak with a sectional rms width of 5 pixels in picture 2.

Description

gaussian generates the following function:

$$e^{-\left(\frac{x^2}{2r^2}\right)}$$

where *x* is the distance from the picture origin and *r* is the rms width. Values below $e^{-7.5}$ are set explicitly to zero. Note that the **lorentzian** command generates a similar lorentzian profile picture.

Notes

multi-layer pictures: fully supported
forms used internally: fp
see also: **lorentzian**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
size	32, *size*, 1	positive integers
radius	*size*/8	positive real number

grab

This syntax is specific to...
Silicon Graphics workstations with an Imaging Technology FG-100 frame grabber

keys:	[to]	*<number>*	output picture number
	wait	*<number>*	time in seconds for image acquisition
	size	*<x>, <y>*	size of subregion of image acquisition window
	position	*<x>, <y>*	centre position of subregion or offset
options:	**left/right/top/bottom**		position of subregion relative to the monitor limits

Use **grab** to capture live images using the FG-100 frame grabber.

Examples

```
grab
```

This command starts live input and reads back the entire digitised image into the current picture, when a key or mouse button is pressed.

```
grab 2:3 wait 0
```

This command captures the next whole frame and copies this to picture 2:3.

```
grab display:1 size 300
```

This command copies the central 300 square region of the captured image directly to display partition 1.

Description

You use **grab** to capture images using the FG-100 frame grabber board installed in your system. As soon as you type **grab**, the frame grabber starts to digitise image data from the signal arriving at the currently selected video input channel. The digitised data is then mapped according to the contents of the current input look-up table and stored in the frame grabber's memory, where it may be viewed if a suitable monitor is connected to the frame grabber's output leads. Live input continues until you press a key or mouse button. Finally, the digitised image is copied to the specified output picture.

You can specify a time, in seconds, for the live input to continue using the **wait** key. When the time limit is reached, input will continue until a complete frame has been digitised. Specify a zero wait time to digitise the next complete frame.

grab

You can use the standard 2-D subregion keys (**size**, **position**) and options (**left/right...**) to copy part of the captured image to the output picture. Otherwise, the entire captured image is copied.

Use the **video** command to configure the frame-grabber and the **ilut** command to specify the contents of the input look-up table.

Images are captured to a resolution of 8 bits. The input look-up tables are also 8 bits deep. Consequently, the resolution of captured images is restricted to 8 bits and by default, the data is stored in the output picture in byte form. The captured data values will always be in the range 0 to 255.

Notes

see also: **ilut, video**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
wait	system prompts for key or mouse button to be pressed	real number
size	entire image acquisition window	integers, less than or equal to the size of the window
position	0,0	integers, within the bounds of the window

halfplane

keys:	[from]	<*number*>	source picture
	[to]	<*number*>	output picture

Use **halfplane** to convert a full-plane *Fourier* or *Spectrum* picture into its half-plane equivalent.

Examples

```
halfplane 50 51
```

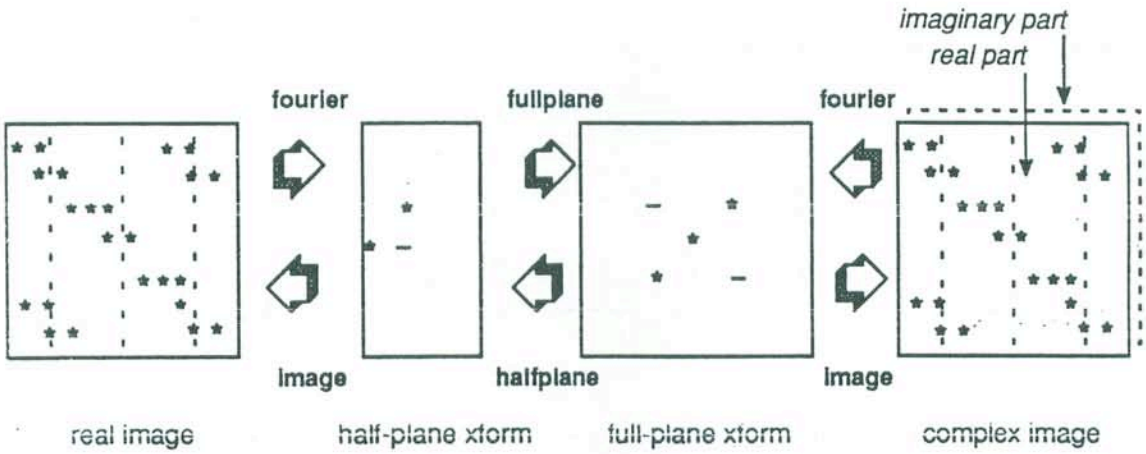
This command places the right half-plane of picture 50 in 51.

```
fourier 1; fullplane; display; halfplane; weight with...; image
```

This sequence of commands displays the transforms of picture 1 in full-plane form before filtering it.

Description

halfplane converts a full-plane picture by discarding the left hand half-plane. Use the **fullplane** command to perform the opposite operation. The following diagram illustrates these commands.



Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**fourier, fullplane, ps**

halfplane

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

help

<text> <text>...	display brief information about the specified topic(s)
<text>.**syntax**	display the syntax for the specified command(s)
<text>.**command**	display help for the specified command only
/full *<text>*	display help information about a topic in full
/header *<text>*	display the header line only for each entry
/topics *<text>*	list the topic names beginning with specified name only
/topics	list the topic names of all help library entries
/topics/full	list the topic names with alias (alternative) names

The **help** command displays Semper on-line help for a command or topic.

Examples

```
help
```

This command lists a summary of how to use the on-line help.

```
help mask
```

This command displays help information for the **mask** command.

```
help xwires mark
```

This command displays help information about the **xwires mark** command.

```
help x
```

This command displays help information for all topics beginning with *x*.

```
help dis
```

This command displays help information for all topics beginning with *dis*, that is, *disc* and **display**.

```
help mask.syntax
```

This command displays the syntax of the **mask** command.

help

```
help display

help display.command
```

The first command in this example would tell you about *display devices* as well as the **display** command. The second command would only list information about the **display** command.

```
help display.syntax
```

This command lists the syntax of the **display** command.

```
help/topics a b c
```

This command lists all topics beginning with *a*, *b* and *c*.

```
help/topics.command
```

This command lists all topics that describe commands.

```
help xwires/full
```

This command provides full information about the **xwires** command.

```
help x/header
```

This command displays the header only of each **x** entry in the help file.

Description

The **help** command allows you to ask for information about a command or topic that is concerned with running Semper or image processing in general (for example, about Semper picture classes or image transforms). It allows you to specify that you would like to see help on:

- command syntax (**.syntax**)
- command/topic description (**.command**)
- header information about a command/topic (**/header**)
- full help entry for a command/topic (**/full**)
- list of help titles only (**/topics**)

help

Note that you can ask for several topics at once, for example, **help acf backproject**, lists help information for the commands **acf** and **backproject**. You can also abbreviate topic names, for example:

```
help h
```

provides help on all help topics beginning with **h**, which would include the following:

- halfplane.command
- halfplane.syntax
- help.command
- hilbert.command
- hilbert.syntax
- histogram
- histogram.command
- histogram.syntax
- hp.command
- hp.syntax
- hue

Note that all letters are treated as significant by the **help** command (as opposed to the first three), for example, **help logical** does not match *logfile*. A full-stop causes matching to resume after the next full stop in the topic name, for example, **d.s** matches **display.scaling**. Alternative names (aliases) are recorded for most topics, which means that by typing, for example, **help classification** you will see help information for the *Remote Sensing* classification commands **box**, **learn**, **likelihood**, **mindistance** etc.

If you see the message *'No help library assigned'* this means that you need to use the **assign** command to assign one (and preferably include the command in your start-up file).

Note that if you do not have a help library assigned you can always use the **syntax** command to see the syntax of a Semper command. **syntax** lists the keys and options for each command in a form that is understood and used by Semper as the precise command definition.

Background commands

The **help** command not only provides information about commands but also provides information on a range of image processing topics and about using Semper. These topics are listed below. For example, to see help information on morphology type **help morphology**.

Abandon	Classes	Conditionals
Blocks	Commands	Connectivity
Cd	Comments	

help

Coordinates
Correlation
Cursor.Movement
Date
Debugging
Devices
Display
Display.Variable
Elements
Errors
Expressions
Fileopening
Filepath
Filetypes
Filters
Forms
Fourier
Fs
Functions
Graphics
Graphics.Coordinates
Histogram
Hue
Illumination
Images
Keyboard
Keys
Labels
Layout
Logicals
Luts
Macros
Mark
Minimum
Mkmode
Morphology
Options
Origin
Pa.Additional
Pa.Conditionals
Pa.Picturenumbers
Pa.Sort
Particle.Analysis
Particle.Parameters
Partitions
Picturenumbers
Pixels
Plists
Program.Libraries
Program.Structure
Programs
Protection
Ranges
Regions
Relinking
Select
Semper
Semper6Plus
Sizes
Startup
Subscripts
Tapes
Text
Textstrings
Titles
Transforms
Upgrade
Variables
Viewing

Notes:

see also: **assign, syntax**

hfill

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
options:	**fg/bg**		fill holes in foreground or background regions

You use the **hfill** command to fill the holes in the foreground or background regions defined in a binary source image. The output picture will contain a binary image in which any region which is classed as a hole in the source image is inverted. The **fg** or **bg** option specifies whether to fill in holes in foreground or background regions. By default, only the holes in foreground regions are filled.

Examples

```
hfill
```

This command fills the holes in the foreground regions of the current picture.

```
hfill bg 2:1 display
```

This command fills the holes in the background regions of picture 2:1 and outputs the final result to the current display picture or partition.

```
hfill bg 1 2; calculate :1 & ~:2 to 2
```

This example deletes any regions touching the edges of picture 1 and leaves the result in picture 2.

Description

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels. Background pixels are always treated as being 4-connected, that is, two background pixels are 4-connected if they are either horizontally or vertically adjacent to each other. Foreground pixels are treated as being 8-connected, that is, two foreground pixels are 8-connected if they are in any way adjacent.

Because of the difference in the way foreground and background pixels are connected, the result of filling the holes in the foreground regions of an image can be quite different from the result obtained by negating the image, filling the holes in the background regions and negating the end result.

hfill

For example,

source image

```
0 0 0 1 0 0 1 1 0
0 0 0 1 0 1 0 0 1
0 0 0 1 0 1 0 0 1
1 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 1 1 1
1 0 0 1 0 1 0 0 0
1 0 0 1 0 1 0 0 0
1 1 1 1 0 1 0 0 0
```

hfill fg

```
0 0 0 1 0 0 1 1 0
0 0 0 1 0 1 H H 1
0 0 0 1 0 1 H H 1
1 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 1 1 1
1 H H 1 0 1 0 0 0
1 H H 1 0 1 0 0 0
1 1 1 1 0 1 0 0 0
```

output image

```
0 0 0 1 0 0 1 1 0
0 0 0 1 0 1 1 1 1
0 0 0 1 0 1 1 1 1
1 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 1 1 1
1 1 1 1 0 1 0 0 0
1 1 1 1 0 1 0 0 0
1 1 1 1 0 1 0 0 0
```

negate

```
1 1 1 0 1 1 0 0 1
1 1 1 0 1 0 1 1 0
1 1 1 0 1 0 1 1 0
0 0 0 1 1 1 0 0 1
1 1 1 1 1 1 1 1 1
0 0 0 0 1 0 0 0 0
0 1 1 0 1 0 1 1 1
0 1 1 0 1 0 1 1 1
0 0 0 0 1 0 1 1 1
```

hfill bg

```
1 1 1 0 1 1 0 0 1
1 1 1 0 1 0 1 1 0
1 1 1 0 1 0 1 1 0
0 0 0 1 1 1 0 0 1
1 1 1 1 1 1 1 1 1
0 0 0 0 1 0 0 0 0
0 H H 0 1 0 1 1 1
0 H H 0 1 0 1 1 1
0 0 0 0 1 0 1 1 1
```

negate

```
0 0 0 1 0 0 1 1 0
0 0 0 1 0 1 0 0 1
0 0 0 1 0 1 0 0 1
1 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0
1 1 1 1 0 1 1 1 1
1 1 1 1 0 1 0 0 0
1 1 1 1 0 1 0 0 0
1 1 1 1 0 1 0 0 0
```

where H = region classed as a hole.

A hole is a background or foreground region in the source picture which is entirely surrounded by a foreground or background region (depending on the **fg** or **bg** option respectively), or, equivalently, a background region which is not 4-connected to, or a foreground region which is not 8-connected to, any of the edges of the source picture.

The **hfill** command produces the hole-filled image in two or more passes through the data (two passes except for very detailed images). The results of the first pass are output to a temporary, integer form, disc picture with the same dimensions as the source picture. The final result is a binary image which is always output to a picture of byte form.

The range of the final result is stored in the output picture label.

hfill

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
fg/bg	fill holes in foreground regions	

hilbert

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
options:	**left/right**		transform vanishes over left/right half-plane
	zero		set imaginary part of central transform column to zero rather than duplicate real part
	lp		apply low pass linear ramp filter to transform rows

The **hilbert** command calculates the discrete *hilbert* transform of a picture. It adds it as an imaginary part to the picture so that its *fourier* transform vanishes over one half-plane. You can use this operation to analyse images that are formed by one-sided imaging systems.

Examples

```
hilbert 1 2
```

This command copies the real part of picture 1 to *Complex* picture 2, and adds the row by row *hilbert* transform as an imaginary part. The *fourier* transform of picture 2 vanishes over the left half-plane.

```
hilbert right nozero
```

This command produces a picture whose *fourier* transform vanishes over the right half-plane, and whose imaginary row means match the real row means.

Description

The **hilbert** command adds a *hilbert* transform, as imaginary parts, row by row to the image, by suppressing one half of the row *fourier* transforms. You can specify the side using the **left/right** options, (by default the left side is used). The output is a *Complex* picture by default. The imaginary part of the row means (that is, of the central *fourier* component) are set to zero unless you specify option **nozero**, in which case the real part mean is copied across.

A *Hilbert* transformation is a convolution with $\frac{1}{\pi x}$ and is like differentiation in the x direction, with a high-frequency de-emphasis. The convolution uses alternate points only of the source. Specifying the **lp** option causes an additional linear ramp filter to be included in the row transform manipulation, falling from 1 at the centre to zero at the outside. This filter approximates the convolution differently and weights all source points equally.

hilbert

Notes

multi-layer pictures: faulted
forms used internally: complex

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
left/right	left	
zero	set to zero	

histogram

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output histogram
	channels	*<number>*	number of histogram channels
	size	*<x>, <y>, <z>*	dimensions of subregion to be processed
	position	*<x>, <y>, <z>*	position/offset of subregion
	layer	*<number>* *<n1>,<n2>*	layer to be processed range of layers to be processed
	height	*<number>*	histogram height in framestore pixels
	times	*<number>*	display magnification factor
	aspect	*<number>*	if **type/log**, text aspect ratio
	width	*<number>*	if **type/log**, number of characters per line
options:	**preset**		set histogram channel limits from *min*, *max*
	left/right, bottom/top, near/far		position of subregion
	repeating		repeat histogram counts when magnifying, instead of interpolating
	letter		mark top of display partition with picture number and title
	border		mark picture border
	type/log		output histogram in character form to console or log output stream

The **histogram** command creates a picture intensity histogram, showing the intensity ranges of pixels in a picture.

Examples

```
display 4:23; xwires region; histogram 4:23 @region
```

This command displays a histogram of a subregion of picture 4:23. The subregion is marked out using the cursor (**xwires** command).

```
histogram channels 100 to 51
```

This command places a 100 channel histogram of the current picture in picture 51.

histogram

```
min=10 max=20 histogram preset type
```

This command creates a histogram of pixels between an intensity range of 10 and 20. The histogram is output to the console in character form.

Description

If you direct a histogram to a display partition, it is displayed in graphical form. Otherwise it is stored as a class *Histogram* picture, which you can display when required or use it to perform histogram equalisation with the **map** command.

To generate a histogram of a (multi-layer) subregion of a picture use the standard subregion keys and options, **size**, **position** etc. See *Appendix C: Semper Keys and Options* for further detail of subregion keys and options.

By default, the **histogram** command finds the actual range of pixel intensity, returning this range in the variables *min* and *max*. If you *abandon* during a range-finding scan an estimated range is used. However, if you use the option **preset** the existing range of values, held in *min* and *max* are used. The histogram channels are then spread evenly over this range; pixel intensities outside this range are not counted. Semper does not distinguish between imaginary parts of complex pixels from real parts for counting purposes.

If the range *min,max* lies in the range 20,256, it is used as a default for the number of channels (otherwise the default is 256). You can specify a number of channels using the **channels** key (up to the maximum row length for floating point pictures).

Note that the keys **height**, **times** and the options **repeating**, **letter**, **border**, **type** and **log** are relevant only if the histogram is displayed in graphical form rather than stored as a *Histogram* picture. The **aspect** and **width** keys are only relevant if you are outputing your graph in character form to the console or to the log output stream.

Notes

display marking:	scanned region
multi-layer pictures:	fully supported
forms used internally:	fp, complex
variables used:	*min, max* (if **preset**, pixel range shown by histogram channels)
variables set:	*min, max* (unless **preset**, pixel range of histogram channels)
see also:	**display**, **map**

histogram

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current display picture, held in the variable *dispiay,* histogram shown in graphical form	valid picture number
channels	*max,min* if in range 20,256; otherwise 256	integer in range 1 to 256
size	whole picture	less than or equal to the size of the picture (integers)
position	0, 0, 0	within the bounds of the picture (integers)
layer	all unless variables *si3/po3* set	integers in range 1 to number of layers
height	lesser of half histogram width and half partition height	less than or equal to the height of the picture (integer)
times	1	positive integer
aspect	default given by the **page** command	positive integer
width	default given by the **page** command	positive real number
preset	lowest, highest pixel values	
repeating	interpolate between histogram counts when magnifying	
letter	lettering on	
border	bordering on	
type/log	histogram shown in graphical form on display	

hp

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	over	*<number>*	filter kernel size
options:	**horizontal**		apply 1-D horizontal filter only
	vertical		apply 1-D vertical filter only

hp provides a high-pass filter that levels the local background throughout a picture, retaining the high frequency (rapidly varying) detail only. The **hp** command subtracts from each pixel the average value over a block of neighbouring pixels.

Examples

```
hp to 999; display
```

This command displays the fine detail in the current picture (detail with a period no larger than 5 pixels).

```
hp over 100
```

This command subtracts a local mean calculated over 100 square neighbourhoods.

```
hp 50 51 over 30 horizontally
```

This command applies a horizontal 1-D form of the filter.

Description

Use the **over** key to specify the size of block over which the local average is to be taken (the default is 5) . You can use 1-D horizontal or vertical forms of filter as well as square forms, by using the **horizontal** and **vertical** options.

The general form of the filter kernel is illustrated here, for the particular case of **over 3**:

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

hp processes edge pixels of the source – where the block averaged overflows the source – as if the boundary values are repeated indefinitely outwards. If **over** is an even number, the replaced source pixel is rounded to the bottom right from the block centre.

hp

Note that the action of **hp** is closely related to that of **lmean**.

Notes

multi-layer pictures: faulted
forms used internally: fp, complex
see also: **lmean**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
over	filter kernel size 5	positive integer

hplj

This command is specific to...
IBM PC XT or AT compatible

keys:	**[ ]**	*<number>*	source picture number or display partition number
	name	*'<text>'*	output file name
	dpi	*<number>*	printer resolution in dots per inch
options:	**partition**		source is a display partition
	negated		reverse pixel intensity scaling
	preset		use values of *min* and *max* to define intensity levels for black and white
	old/new		output file status
	old		re-use an existing output file (must be used for output to a DOS device)
	new		create a new file, deleting the old file if it exists

Use the **hplj** command to output the contents of a picture or display partition (including the overlay in white) for printing using Hewlett-Packard's *LaserJet* format. The output from the command can be directed into a file if you specify the file name with the **name** key. In the absence of the **name** key, output is sent directly to the parallel port, which should be connected to a Hewlett-Packard *LaserJet+* or equivalent printer.

Examples

```
hplj 21
```

This command prints picture 21 centrally on an A4 page, with the title and creation date at the foot of the page

```
hplj partition 4
```

This command prints display partition 4 complete with any overlay data in white.

```
hplj partition negated
```

This command prints the current display partition with reversed contrast.

```
min=.2 max=2.8; hplj preset
```

This command prints the current picture with source levels 0.2 and 2.8 printed as black and white. Any pixel values outside this range are truncated.

hplj

```
hplj 4:3 dpi 75 name '75dpi'
```

This command outputs the contents of picture 3 on device 4 at a resolution of 75 dots per inch to the file *75dpi.hpl*.

Description

The size of the image being output must not exceed 512 by 512 pixels. 512 pixels is the maximum that can fit across an A4 page. Note also that some printers may not have enough memory to print a full page at the highest resolution. You specify the output resolution with the **dpi** key. Only resolutions of 75, 100, 150 and 300 dots per inch are accepted. If the **dpi** key is not set, the highest of these four values is chosen for which the output image still fits on the printer page. If you want to print images larger than 512 by 512, use the **extract** command first of all to sub-sample the image. Fifteen distinct grey levels are achieved by means of suitable dither patterns.

If the **partition** option is set, the contents of a display partition is output together with any overlay data contained in the partition. The image data will be printed just as it appears on the display. Overlay data is printed in white. If you specify the **negated** option, the image data will be printed with an inverted greyscale. You can output the contents of a display frame, complete with annotation, if you first create a full-frame partition, for example,

```
partition 9 frame 1
hplj partition 9
```

prints display frame 1.

If the **partition** option is not set, the **hplj** command outputs the contents of a picture. By default, the minimum and maximum source intensity levels are printed as black and white respectively. The **negated** options swaps the black and white levels. You can specify black and white levels yourself by setting the variables *min* and *max* and specifying the **preset** option.

You can direct the printer output to file by specifying the file name with the **name** key. If you do not specify a file extension, the default extension *.hpl* is used. Option **new** causes a new instance of the file to be created. Option **old** means that the file must already exist before it is reused. You must specify this option if the file name refers to a DOS device, for example, *lpt2*. With neither option, an error is returned if the file already exists, otherwise, a new file is created.

If the **name** key is omitted, the **hplj** command sends its output directly to the parallel port. Note that it may take two to three minutes to transmit the data to a printer via a DOS device.

Hewlett-Packard LaserJet+ is a registered trademark of Hewlett-Packard.

hplj

Notes

variables used:	*min, max*	if **preset** and not **partition**, intensity levels in the source picture to be printed as black and white
variables set:	*min, max*	if not **preset** and not **partition**, minimum and maximum intensity levels in the source picture

Defaults and Ranges

keys/options	defaults	range
[ ]	if **partition**, current partition, held in the variable *display* otherwise, current picture, held in the variable *select*	valid partition number valid picture number
name	send output directly to the printer port	valid file name
dpi	Largest value for which image still fits on the printer paper	75, 100, 150 or 300
preset	if not **partition**, print minimum and maximum intensity levels in the source picture as black and white	
old/new	if the output file already exists, output an error message	

hplj

This command is specific to...
Sprynt Systems and workstations running Unix

keys:	[]	*<number>*	source picture number or display partition number
	name	*'<text>'*	output file name
	dpi	*<number>*	printer resolution in dots per inch
options:	**partition**		source is a display partition
	negated		reverse pixel intensity scaling
	preset		use values of *min* and *max* to define intensity levels for black and white
	old/new		output file status
	old		re-use an existing output file (must be used for output to a device special file)
	new		create a new file, deleting the old file if it exists

Use the **hplj** command to output the contents of a picture or display partition (including the overlay in white) for printing using Hewlett-Packard's *LaserJet* format. The output from the command is directed into a file which should then be sent to a Hewlett-Packard *LaserJet* or equivalent printer. You specify the file name with the **name** key.

Examples

```
hplj partition 4 name 'part4'
```

This command outputs the contents of display partition 4, complete with any overlay data in white, to the file *part4.hpl*.

```
min=.2 max=2.8 hplj preset name 'image'
```

This command outputs the current picture to the file *image.hpl*, so that source levels 0.2 and 2.8 are printed as black and white. Any pixel values outside this range are truncated.

```
hplj 4:3 dpi 75 name '75dpi'
```

This command outputs the contents of picture 3 on device 4 at a resolution of 75 dots per inch to the file *75dpi.hpl*.

hplj

Description

The size of the image being output must not exceed 512 by 512 pixels. 512 pixels is the maximum that can fit across an A4 page. Note also that some printers may not have enough memory to print a full page at the highest resolution. You specify the output resolution with the **dpi** key. Only resolutions of 75, 100, 150 and 300 dots per inch are accepted. If the **dpi** key is not set, the highest of these four values is chosen for which the output image still fits on the printer page. If you want to print images larger than 512 by 512, use the **extract** command first of all to sub-sample the image. Fifteen distinct grey levels are achieved by means of suitable dither patterns.

If the **partition** option is set, the contents of a display partition is output together with any overlay data contained in the partition. The image data will be printed just as it appears on the display. Overlay data is printed in white. If you specify the **negated** option, the image data will be printed with an inverted greyscale. You can output the contents of a display frame, complete with annotation, if you first create a full-frame partition, for example,

```
partition 9 frame 1
hplj partition 9 name 'frame1'
```

outputs display frame 1 to the file *frame1.hpl*.

If the **partition** option is not set, the **hplj** command outputs the contents of a picture. By default, the minimum and maximum source intensity levels are printed as black and white respectively. The **negated** options swaps the black and white levels. You can specify black and white levels yourself by setting the variables *min* and *max* and specifying the **preset** option.

You direct the output of the **hplj** command to a file by specifying the file name with the **name** key. If you do not specify a file extension, the default extension *.hpl* is used. Option **new** causes a new instance of the file to be created. Option **old** means that the file must already exist before it is reused. You must specify this option if the file name refers to a device special file. With neither option, an error is returned if the file already exists, otherwise, a new file is created.

Hewlett-Packard LaserJet+ is a registered trademark of Hewlett-Packard.

Notes

variables used:	*min, max*	if **preset** and not **partition**, intensity levels in the source picture to be printed as black and white
variables set:	*min, max*	if not **preset** and not **partition**, minimum and maximum intensity levels in the source picture

hplj

Defaults and Ranges

keys/options	defaults	range
[]	if **partition**, current partition, held in the variable *display* otherwise, current picture, held in the variable *select*	valid partition number valid picture number
name	*none*, prompts for file name if interactive	valid file name
dpi	Largest value for which image still fits on the printer paper	75, 100, 150 or 300
preset	if not **partition**, print minimum and maximum intensity levels in the source picture as black and white	
old/new	if the output file already exists, output an error message	

if, unless

<condition>	***If, unless*** uses a special syntax. Specify a condition and then an action to be taken if that condition is true (***If***) or untrue (***unless***).

Use **If, unless** to make command execution conditional.

Examples

```
if sd<.15 jump noprint; type 'Standard deviation', sd; noprint: ...
```

This command jumps to the label called *noprint* if the standard deviation is less than 0.15.

```
unless newval=oldval type 'New value is ', newval
```

This command types out the new value if it is different from the value held in the variable *oldval*.

```
if n=0 if x<pi return
```

This command returns program execution from a library or run file if *n* is equal to zero and *x* is less than π.

```
if n=0 & x<pi return
```

This performs the same function as the above command, using the logical AND operator (*&*).

Description

Use **If, unless** to direct command execution interactively or program flow. You can use any valid Semper expression as a condition. Refer to *Appendix B, Semper Expression Syntax* for details of the components of a valid expression.

Note that you can specify multiple conditions by repeating **If** or **unless** as in the third command example, or by using the following logical operators as in the last example:

```
~ & |
```

which represent NOT, AND and OR respectively.

iget

This syntax is specific to...
Silicon Graphics workstations

keys:	[to]	*<number>*	output picture
	name	*'<text>'*	source file name
options:	**again**		if **name** is omitted and was used in a command to open a file for reading, open the same file again

You use **iget** to read an image stored in a file using the Silicon Graphics image file format. The image can be compressed (run-length encoded). You can check the file header information for these image files by using the Unix command *istat* (see the corresponding Unix manual page).

Examples

```
iget dis:1 name 'cat.rgb'
```

This command copies the image in file *cat.rgb* directly to display partition 1.

Description

Files are searched for in the current directory and then throughout the search path. You can specify a full path name to avoid the path scan.

Notes

see also: **iput**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

ilut

This syntax is specific to...
Silicon Graphics workstations with an Imaging Technology FG-100 frame grabber

keys:	**[number]**	*<number>*	input look-up table number
	from	*<number>*	read look-up table data from specified monochrome *Lut* class picture
	to	*<number>*	write look-up table data to *Lut* class picture
options:	**reset**		set look-up table to default linear ramp (unit mapping)
	invert		invert look-up table mapping

Use **ilut** to set or examine the contents of any of the FG-100 frame grabber's input look-up tables.

Examples

```
ilut 1 reset
```

This command restores look-up table 1 to a default linear ramp.

```
ilut 2 reset invert
```

This command sets look-up table 2 to invert all incoming data.

```
ilut 10 from 2:5
```

This command loads look-up table data from *Lut* class picture 2:5.

```
ilut 3 to 999; display 999 to display:5
```

This command displays contents of look-up table 3 in display partition 5.

Description

The **ilut** command operates on one of the FG-100 frame grabber's input look-up tables. There are a total of 16 look-up tables, numbered from 1 to 16. Each table has 256 entries with values that should lie in the range 0 to 255. The currently selected video input look-up table is used to map incoming data during image acquisition. At the start of a Semper session, all the look-up tables are initialised to linear ramps that leave the input data unmodified. Use the **video** command to select the current input look-up table.

ilut

When you load look-up table data from a *Lut* class picture, you must ensure that it is a monochrome *Lut* picture with dimensions 256,1,1. Only the least significant 8 bits of each look-up table entry are relevant when mapping incoming data during image acquisition.

Notes

see also: **grab, video**

Defaults and Ranges

keys/options	defaults	range
[number]	1	positive integer (1 to 16)
from	data is read from frame grabber	valid picture number
to	*none*	valid picture number

ilut

This command is specific to...
PC + Data Translation DT2861 framestore
PC + Imaging Technology PCVISIONplus framestore
PC + Metrabyte Corporation MV1 framestore
PC + Matrox PIP512/PIP1024 framestore

This syntax is specific to...
PC + Data Translation DT2861 framestore

keys:	**from**	*<number>*	source picture, of class *Lut*
	to	*<number>*	destination picture, of class *Lut*
	like	*<number>*	output lut used as a source for the input lut
options:	**input**		specify the source lut as the input lut currently stored in the framestore
	reset		specify a linear input lut
	invert		specify an inverted linear input lut

This syntax is specific to...
PC + Imaging Technology PCVISIONplus framestore
PC + Metrabyte Corporation MV1 framestore

keys:	**[number]**	*<number>*	look-up table number
	from	*<number>*	source picture, of class *Lut*
	to	*<number>*	destination picture, of class *Lut*
	like	*<number>*	output lut used as a source for the input lut
options:	**input**		specify the source lut as the input lut currently stored in the framestore
	reset		specify a linear input lut
	invert		specify an inverted linear input lut
	monochrome/false/colour		input lut mode

ilut

This syntax is specific to...
PC + Matrox PIP512/PIP1024 framestore

keys:	**from**	*<number>*	source picture, of class *Lut*
	to	*<number>*	destination picture, of class *Lut*
	like	*<number>*	output lut used as a source for the input lut
options:	**reset**		specify a linear input lut
	invert		specify an inverted linear input lut

Use the **ilut** command to manipulate the input look-up-table (*lut*). It allows you to copy a lut from a variety of different sources into one of the framestore input luts. Additionally it allows you to read back the framestore input luts and to store input luts as Semper pictures.

Examples

```
ilut reset
```

This command loads a linear lut into the framestore input look-up-table.

```
ilut like 1
```

This command loads the framestore input lut with the output lut 1.

```
ilut like 1 to 4:25
```

This command loads the framestore input lut with output lut 1 and also stores it on disc as a *Lut* class Semper picture 4:25.

```
ilut input to 4:10
```

This command reads the framestore input lut and stores it on disc as a *Lut* class Semper picture 4:10.

ilut

Description

When using the **Ilut** command you need to specify a framestore input look-up-table (*lut*) number, a source from which the lut is to be obtained and an (optional) destination picture number. The look-up-table that you specify as the source is automatically loaded into the framestore input lut.

PC + Data Translation DT2861 framestore only

You can use the **Input** key to specify the input lut as the source itself. Note that you must specify a destination, using the **to** key, with the **Input** option. You can specify an input look-up table using the **number** key.

PC + Imaging Technology PCVISIONplus and Metrabyte MV1 framestore only

You can use the **Input** key to specify the input lut as the source itself. If Semper does not know the mode of the input lut, you need to specify either **monochrome**, **false** or **colour**. Note that you must specify a destination, using the **to** key, with the **Input** option.

Notes

see also: **Iadjust, Iset, lut**

Defaults and Ranges

keys/options	defaults	range
[number]	*none*	valid lut number
from	*none*	valid picture number
to	*none*	valid picture number
like	*none*	valid output lut number

image

keys:	[from]	<number>	source picture
	[to]	<number>	output picture

Use **image** to perform inverse *fourier* or *walsh* transforms, that is to recreate an *Image* from its transform. **Image** includes normalisation so that its use after the **fourier** or **walsh** commands recovers the original values exactly.

Examples

```
image 50 51
```

This command recovers an *Image* picture 51 from a *Fourier* or *Walsh* picture 50.

```
image byte
```

This command inverse transforms the current picture, leaving the result in *Byte* form.

Description

When **image** performs an inverse *fourier* transform, the default form of output is *Floating point* if the transform is half-plane, and *Complex* if it is full-plane. For an explanation of general keys such as **byte** that can be used with **Image**, see *Appendix C: Semper Keys and Options*.

Notes

restrictions:	image sizes must be powers of two unsuitable for direct output to display
multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**fourier, walsh**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

inkey

<variable name(s)>	***inkey*** waits for a keystroke to enter the value of the specified variable(s)

Use **inkey** to read individual keystrokes from the terminal, without waiting for the user to press a return and without any 'echo' on the terminal screen.

Examples

```
inkey n
```

This command waits until the user presses a key and sets *n* to the decimal value of the ASCII code for the key (for example, pressing key A sets *n* to 65).

```
inkey c1, c2, c3
```

This command reads three keys in turn, returning their ASCII codes in variables *c1*, *c2* and *c3*.

```
inkey
```

This command waits until a key is pressed before continuing. The keystroke value is not returned.

input

This syntax is specific to...
IBM PC XT or AT compatible

keys:	**[to]**	*<number>*	output picture
	name	*'<text>'*	input file name
	size	*<x>,<y>,<z>*	if **raw**, size of raw binary data
	skip	*<number>*	if **raw**, number of bytes to skip before reading image data
options:	**cut/dump/paint/link/raw**		input format (the default is to read a Semper 6 data file
	cut		read a Media Cybernetics CUT file
	dump		read a Micro Semper 1 dump file
	paint		read a greyscale Paintbrush PCX file
	link		read a Link Analytical data file
	raw		read a raw binary data file
	swap		if **raw**, read data in *Motorala* instead of *Intel* byte ordering
	byte/integer/fp/complex		if **raw**, input data format
	byte		read image data as bytes (the default)
	integer		read image data as 16-bit integers
	fp		read image data as floating-point values
	complex		read image data as complex values
	again		if **name** is omitted and was used in a command to open a file for reading, open the same file again

Use **input** to read pictures from files created by Semper 6, (from the **output** command), from dump files created by MicroSemper 1, or from other selected programs, such as Media Cybernetics CUT file format.

Examples

```
input 5 name 'newdata'
```

This command reads picture 5 from the Semper 6 file called *newdata.pic.*

input

```
input 20 dump name 'olddata'
```

This command reads picture 20 from a MicroSemper1 dump file called *olddata.dum.*

```
input 104 cut name 'halo'
```

This command reads picture 104 from a Media Cybernetics CUT format file called *halo.cut* and obtains the picture label information from the LAB file *halo.lab.*

```
input 125 raw name 'binary' size 512,512
```

This command reads picture 125, of size 512 by 512 pixels, from the raw binary file called *binary.bin.*

```
input 126 raw again skip 1078 size 512,512
```

This command reads again the file in the previous example, skipping some header bytes.

```
input 11 link name 'import'
```

This command reads picture 11 from the Link Analytical data file *import.im.*

```
input 2:4 paint name 'cells'
```

This command reads picture 4 on device 2 from the Paintbrush PCX file *cells.pcx* (or *cells.pcc* if *cells.pcx* is not found).

Description

By default, **input** expects to read a Semper 6 data file unless you specify the **cut, dump, paint, link** or **raw** option. The default file format contains exactly the same information as files handled by the **read** and **write** commands, but in raw binary form.

If you specify the **cut** option, **input** reads a Media Cybernetics CUT file. Semper also looks for a Semper LAB file (produced by the **output** command) which contains the associated picture label. If no LAB file is found, the output picture is given the source file name as a title.

If you specify the the **dump** option, **input** reads a Micro Semper 1 dump file. Note that not all MicroSemper 1 picture classes are readable (*Macro, Object* and *Program* pictures cannot be read by **input**).

If you specify the **paint** option, **input** reads a greyscale Paintbrush PCX file. Not all Paintbrush formats are supported, in particular, new encoding methods and true colour formats will not be read.

input

If you specify the **link** option, **input** reads a Link Analytical AN10 or eXL file. Not all Link formats are supported (only 8-bit and 16-bit images can be read).

If you specify the **raw** option, Semper reads a binary file. You need to specify the size of the file using the **size** key, with pixels starting with the top left pixel of layer 1, reading along the row. By default, one byte is read for each pixel. You can specify the pixel data form with the options **byte, integer, fp** or **complex.** You can also use the **skip** key to specify the number of bytes to skip before reading data from a file. This key is useful if you have picture files that contain header information of a known size.

The bytes are always ordered in a Semper 6 data file so that the least significant byte appears at the first (lowest) address (*Intel* format). When reading a raw binary file (**raw** option), you can use the **swap** option to read the binary data assuming the *Motorola* packing format.

The following default file extensions are provided:

Semper 6 data file	*.pic*
raw binary file	*.bin*
MicroSemper 1 dump file	*.dum*
Media Cybernetics CUT file	*.cut*
Semper LAB file	*.lab*
Paintbrush PCX file	*.pcx, .pcc*
Link Analytica data file	*.im*

For Paintbrush PCX files, the default extension *.pcc* is used if a file with the extension *.pcx* is not found.

The **again** option allows you to re-open a file without having to specify the file name again with the **name** key. If the **name** key was not used before to open a file for reading, the **input** command will prompt for the file name. This option is most useful for unpicking formats in conjunction with the **raw** option and **skip** key. For example, you could read a file *mystyle.bin* consisting of two 16-bit values giving the image size, followed by the raw image data as bytes, with the following commands:

```
input 1 raw name 'mystyle' size 2,1 integer
origin left; dx=p(0); dy=p(1)
input 1 raw again skip 4 size dx,dy byte
```

Note that **input** searches for files in Semper's file search path (use the command **show path** to list the file search path). You can avoid a time-consuming path scan by specifying a full path in the filename.

Files written using the **output** command on a Unix worksttion can be read by the **input** command.

input

Notes

see also: **output, read, write**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts for file name if interactive	valid filename
size	*none*	valid picture size
skip	0	positive integer
map	*none*	valid picture number
cut/dump/paint/ link/raw	read a Semper 6 data file	
swap	if **raw,** read data assuming *Intel* byte ordering	
byte/integer/fp complex	if **raw,** read data as bytes	

input

This syntax is specific to...
Sprynt systems and workstations running Unix

keys:	**[to]**	*<number>*	output picture
	name	*'<text>'*	input file name
	size	*<x>,<y>,<z>*	if **raw**, size of raw binary image
	skip	*<number>*	if **raw**, number of bytes to skip before reading image data
	map	*<number>*	if **raster**, picture containing Sun Raster colormap
options:	**raw/raster**		input format (the default is to read a Semper 6 data file
	raw		read a raw binary data file
	raster		read a Sun Raster file
	swap		if **raw**, write data in *Motorala* instead of *Intel* byte ordering
	byte/integer/fp/complex		if **raw**, input data format
	byte		read image data as bytes (the default)
	integer		read image data as 16-bit integers
	fp		read image data as floating-point values
	complex		read image data as complex values
	again		if **name** is omitted and was used in a command to open a file for reading, open the same file again

Use **input** to read pictures from files created by Semper 6 on workstations or PCs (using the **output** command) or by other selected programs.

Examples

```
input 5 name 'newdata'
```

This command reads picture 5 from the Semper 6 file called *newdata.pic*.

```
input 125 raw name 'binary' size 512,512
```

This command reads picture 125, of size 512 by 512 byte pixels, from the raw binary file called *binary.bin*.

input

```
input 33 raw integer name 'mydata' size 640,480 swap
```

This command reads picture 33, of size 640 by 480 pixels, as 16 bit integers with *Motorola* byte ordering from the file *mydata.bin*.

```
input 1 raw name 'special.pic' size 128,128 skip 64
```

This command reads picture 1, of size 128 by 128 byte pixels, from the raw binary file *special.pic*, skipping the first 64 bytes of the file.

```
input 1 raster name 'dump.rff' map 2
```

This command reads a Sun raster file, called *dump.rff*, into picture 1, storing any colormap information in picture 2.

Description

By default, **Input** expects to read a Semper 6 data file unless you specify the **raw** or **raster** options. The default extension for Semper 6 files is *.pic*. The default file format contains exactly the same information as files handled by the **read** and **write** commands, but in raw binary form.

If you specify the **raw** option, Semper reads a raw binary file. The default extension for raw binary files is *.bin*. You also need to specify the size of the file using the **size** key, with pixels starting with the top left pixel of layer 1, reading along the row. By default, one byte is read for each pixel. You can specify the pixel data form with the options **byte, integer, fp** or **complex.** You can also use the **skip** key with the **raw** option to specify the number of bytes to skip before reading data from a file. This key is useful if you have picture files that contain header information of a known size.

The bytes are always ordered in a Semper 6 data file so that the least significant byte appears at the first (lowest) address (*Intel* format). When reading a raw binary file (**raw** option), you can use the **swap** option to read the binary data assuming the *Motorola* packing format.

The **again** option allows you to re-open a file without having to specify the file name again with the **name** key. If the **name** key was not used before to open a file for reading, the **Input** command will prompt for the file name. This option is most useful for unpicking foreign file formats in conjunction with the **raw** option and **skip** key. For example, you could read a file *mystyle.bin* consisting of two 16-bit values giving the image size, followed by the raw image data as bytes, with the following commands:

```
input 1 raw name 'mystyle' size 2,1 integer
origin left; dx=p(0); dy=p(1)
input 1 raw again skip 4 size dx,dy byte
```

input

If you specify the **raster** option, Semper reads a Sun Raster image file. The default extension for raster files is *.rff*. Currently only files with 8 bits per pixel are accepted. The file packing types support the old and new unpacked types 0 (RT_OLD) and 1 (RT_STANDARD), and also the byte encoded type 2 (RT_BYTE_ENCODED). If you use the **map** key with the **raster** option, Semper stores any colormap data that is present in the specified picture number. This information can be used to remap the image data that you read in using the Semper **map** command. If the map data has three rows (RGB colormap) you can remap the original image as follows:

```
input 1 raster name 'myfile.rff' map 2 ; ! read the data
project 2 vertically fp                ; ! sum columns
calculate :2/3 to :3                   ; ! average
copy 3 2 byte                          ; ! restore as byte map
map 1 to 3 with 2                      ; ! and map the data
```

Files written using the **output** command on a PC can be read into a UNIX workstation linked over a *PC-NFS* network. Note that **input** searches for files in the file search path (use the command **show path** to list the search path). You can avoid a time-consuming path scan by specifying a full path in the filename.

Notes

see also: **output, read, write, show path**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts for file name if interactive	valid file name
size	*none*	valid picture size
skip	0	positive integer
map	*none*	valid picture number
raw/raster	read a Semper 6 data file	
swap	if **raw**, read data assuming *Intel* byte ordering	
byte/integer/fp complex	if **raw**, read data as bytes	

interrupt

This command is specific to...
IBM PC XT or AT compatible

keys:	[ ]	*<number>*	specify the interrupt number
	ax	*<number>*	specify values for registers of the same name
	bx	*<number>*	
	cx	*<number>*	
	dx	*<number>*	
	ds	*<number>*	
	es	*<number>*	
	si	*<number>*	
	di	*<number>*	

Use the **Interrupt** command to access the PC interrupt mechanism directly from Semper.

The **Interrupt** command should only be used when absolutely necessary, and then with extreme care. Many of the available interrupts can hang up the system or cause data loss or even ***physical damage to the hardware***. Please note that *Synoptics* cannot be held responsible for any damage caused by the use of this command.

Examples

```
interrupt 5
```

This command performs a screen dump of the current screen to the printer.

```
interrupt 0x23
```

This command simulates a user <control-break>action (an abandon request).

Description

The **Interrupt** command allows you access to the underlying PC interrupt mechanism. It is useful for accessing features that are not otherwise available in Semper, but should be used with extreme caution.

You can specify a value for most registers when making the interrupt call using the keys **ax, bx, cx,** etc. Semper returns these values in corresponding variables after the call. This allows you to make calls to inquire about available hardware, for example. For further information, refer to your PC technical manual on BIOS, DOS, printer and other interrupts.

interrupt

Notes

variables set: *ax, bx, cx, dx, ds, es, si, di* (set to the returned values of the registers)
flags (set to the flag register contents on return)

see also: **loread**, **interrupt**

Defaults and Ranges

keys/options	defaults	range
[ ]	*none*	unsigned integer in the range 0 to 255 which is a valid interrupt code (see PC hardware manual)
ax, bx, cx, dx, ds, es, si, dl	*none*	unsigned integer expressing a 16 bit value

ioread

This command is specific to...
IBM PC XT or AT compatible

keys:	**address using**	*<number>/ <number>*	specify the port address to be read *or* specify a picture number containing port address(es)
	to	*<number>*	write the data to the specified destination picture
	size	*<x>, <y>, <z>*	size of destination picture if ***using*** key not specified
options:	**word**		perform 16 bit instead of 8 bit reads

Use **ioread** to read data from the input/output ports on a PC. For example, this command can be used to read status information on your PC or to set up devices.

Care should be taken when using the **ioread** command, as reading some ports on a PC can lock up the system or cause undesirable effects, such as, reformatting the hard disk. Refer to your PC hardware manual for details of the correct i/o port addresses. Please note that *Synoptics* cannot be held responsible for any damage caused by the use of this command.

Examples

```
ioread address 0x61
```

This command performs a byte read from input/output port hex 61 and stores the data in the Semper variable *n*.

```
ioread word address 0x300 to 12 size 256,1
```

This command reads 256 words (a word is sixteen bits) from i/o port hex 300 to picture 12, which has the dimensions 256,1,1.

```
ioread word using 99 to 1
```

This command reads words from the ports specified in picture 99 and writes the data to picture 1.

ioread

Description

You can use the **ioread** command in three ways:

- to perform a single read from an input/output port.
- to read a group of data from a port to a destination picture.
- to use a picture containing port addresses to read data.

In the simplest mode of operation, a single read is made from a specified i/o port and stored in the Semper variable *n*. If you use this mode, you only need to specify the **address** key to give the address of the port. By default, **ioread** reads eight bits (a byte) from the port. You can read sixteen bits (a word) by specifying the **word** option.

You can also read a group of data from a port using the **to** key to specify a destination picture and the **address** key to specify the address of the port. In this case, you also need to specify the **size** key, giving the number of rows, columns and layers of the destination picture. You can use the **word** option to read groups of sixteen bits of data.

The third method of reading data using **ioread**, is to supply a picture that contains port addresses by specifying the **using** key. In this case the **to** key specifies an output picture which will have the same dimensions as the **using** picture, with each pixel in the output picture being read from the corresponding address given in the picture specified by the **using** key. The data is read in order from the top left of the front layer, by column, row and layer to the bottom right of the back layer. This method does not require the **address** or **size** keys, but the **word** option can be used. Note that to create a picture containing the port addresses, use the Semper **create** command and add the port addresses to the picture using the **calculate** or **p** commands to define the value of individual pixels.

Notes

variables set: *n* (set when reading data in single read mode)
see also: **iowrite**, **interrupt**

Defaults and Ranges

keys/options	defaults	range
address/using	*none*	valid hexadecimal port address/ valid picture number
to	*none*	valid picture number
size	columns=1 layers=1	positive integers
word	byte form (8 bits)	

iowrite

This command is specific to...
IBM PC XT or AT compatible

keys:	**address using**	*<number>/ <number>*	specify the port address *or* the picture containing the port address(es) to write
	value from	*<number>/ <number>*	specify data value to write to port *or* picture containing data to write to port(s)
options:	**word**		perform 16 bit instead of 8 bit writes

Use the **iowrite** command to write data to input/output ports on the PC. For example, this command can be used to set up devices on a PC.

Extreme care should be used with the **iowrite** command, as writing to many of the ports on the PC can hang up the system or cause data loss or even ***physical damage to the hardware***. Refer to your PC hardware manual for details of the correct i/o port addresses. Please note that *Synoptics* cannot be held responsible for any damage caused by the use of this command.

Examples

```
iowrite address 0x61 value 51
```

This command writes the value 51 in byte form to the input/output port with the address hex 61.

```
iowrite word address 0x300 from 12
```

This command writes the contents of picture 12 as words (a word is sixteen bits) to i/o port hex 300.

```
iowrite word using 99 from 1
```

This command writes words to the ports specified in picture 99 using data from picture 1.

Description

You can use the **iowrite** command in three ways:

- to perform a single write to an i/o port.
- to write a group of data to a port from a source picture.
- to use a picture containing port addresses to write the data.

iowrite

In the simplest mode of operation, a single write is made to a specified port using the **value** key to supply data. In this mode, you only need to specify the port address using the **address** key. By default, **iowrite** writes eight bits (a byte) to the port. You can read sixteen bits (a word) by specifying the **word** option.

You can also write a group of data to the port using the **from** key to specify a source picture that contains data and the **address** key to specify the address of the port. You can use the **word** option to read groups of sixteen bits of data.

The third method of writing data using **iowrite**, is to supply a picture that contains port addresses by specifying the **using** key. In this case, use the **from** key to specify an input picture which must have the same dimensions as the **using** picture, as each pixel in the **from** picture is written to the corresponding address in the **using** picture. The data is written in order from the top left of the front layer, by column, row and layer to the bottom right of the back layer. This method does not require the **address** or **size** keys, but the **word** option can be used.

A simple example of using the **ioread** and **iowrite** commands is given below:

```
ioread address 0x61
a=and(n,0xfc) b=or(n,3)
for i 1,20
   iowrite address 0x61 value b; wait .1
   iowrite address 0x61 value a; wait .1
loop
iowrite address 0x61  value n
```

Notes

see also: **ioread**, **interrupt**

Defaults and Ranges

keys/options	defaults	range
address/using	*none*	valid hexadecimal port address/ valid picture number
value/from	*none*	unsigned integers (0 to 255 for byte data, 0 to 65535 for word values)/valid picture number
word	byte form (8 bits)	

iput

This syntax is specific to...
Silicon Graphics workstations

keys:	**[from]**	*<number>*	source picture
	name	*'<text>'*	output file name
options:	**new**		overwrite the output file if it already exists

You use **iput** to write a picture to a file using the Silicon Graphics image file format. The file can then be accessed by other facilities like *istat*, *icut*, *ipaste* and *snapshot* (see the corresponding Unix manual pages for these commands). The data written to file is run-length encoded.

Examples

```
iput 3:2 name 'picture.img'
```

This command writes picture 3:2 to the file *picture.img*.

Description

Files are created in the current directory, unless you specify a pathname.

Notes

see also: **iget**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

jump

<label name>	***jump*** jumps to the specified label

The **jump** command causes Semper to resume execution at the command with the specified label.

Examples

```
repeat: ...
if sd<.2 jump ok; type 'Trying again'; jump repeat; ok: type '...
```

This command tests for the value of the variable *sd* and if it is less than 0.2, program execution jumps to the label called *ok*.

Description

If you are running interactively, the specified label must be within the current command line. If the **jump** command is within a program, the label can be anywhere within the program. Note that, unlike Semper 5, you are not allowed to jump to labels in other programs.

You cannot jump into the middle of a **for** loop, although jumping out or within a loop is allowed. Note that Semper only uses the first three characters of a label for identification. Within a run file, Semper searches for the label within the current line first, and then from the start of the file if necessary. Ensure that all label names that are within a program are unique (the first three characters must be unique).

justification

options:	**left/right**	locate the object by the left/right point
	top/bottom	locate the object by the top/bottom point

The **justification** command defines which of the nine points on a Semper 6 *Plus* object is used to locate it. A Semper 6 *Plus* object includes cells, menus, panels and textfields. For further detail of Semper 6 *Plus* elements, refer to the manual:

Semper 6 Plus User Interface Guide

Examples

```
justification top right
```

This command locates an object by its top right point.

Description

By default, Semper 6 *Plus* justifies all objects (cells, panels etc.) by their centre point. The **justification** command allows you to alter this positioning method. If you omit either the **left/right** direction or the **top/bottom** direction, Semper assumes a *central* justification in the omitted direction. The diagram below illustrates the justification points of an object.

Top Left	**Top**	**Top Right**
Left	**Central**	**Right**
Bottom Left	**Bottom**	**Bottom Right**

Defaults and Ranges

keys/options	defaults	range
left/right	central justification	
top/bottom	central justification	

label

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
Options:	**fg**		label foreground regions
	bg		label background regions

You use the **label** command to label separate foreground and/or background regions defined in a binary source image. Each labelled region is assigned a unique, integer number counting upwards from 1. The output picture will contain an image in which each pixel is set to the number assigned to the underlying region, or zero, if the underlying region has not been labelled. The **fg** and **bg** options control whether foreground or background regions (or both) are labelled in this way. By default, only foreground regions are labelled.

Examples

```
label
```

This command labels all the foreground regions of the current picture.

```
label 2:1 bg; picture range; type 'Number of objects = ',i2
```

This example labels all the background regions of picture 2:1 and types on the console the number of labelled regions.

```
label 44 to display fg bg
```

This command labels all the connected regions of picture 44 and outputs the final result to the current display picture or partition.

Description

The **label** command has two particular uses:

- It can be used as a quick way to count individual foreground regions.
- It enables you to determine whether there exists a connected path between any two pixels in an image (if they are, they will have the same label).

label

The source picture is treated as a binary image: zero values denote background pixels and non-zero values denote foreground pixels. Background pixels are always treated as being 4-connected, that is, two background pixels are 4-connected only if they are either horizontally or vertically adjacent to each other. Foreground pixels are treated as being 8-connected, that is, two foreground pixels are 8-connected if they are in any way adjacent.

Because of the difference in the way foreground and background pixels are connected, the result of labelling the foreground regions of an image can be quite different from the result of labelling the background regions of the negated version of the same image. For example,

Source image

```
0 0 0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0 0
0 1 1 0 1 1 0 0 0
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0
```

label fg

```
0 0 0 0 0 0 0 0 0 0
0 1 1 0 2 2 0 0 0
0 1 1 0 2 2 0 0 0
0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 2 0
0 0 0 0 0 0 0 0 0
```

negate

```
1 1 1 1 1 1 1 1 1
1 0 0 1 0 0 1 1 1
1 0 0 1 0 0 1 1 1
1 1 1 1 1 1 0 1 1
1 1 1 1 1 1 1 0 1
1 1 1 1 1 1 1 1 1
```

label bg

```
0 0 0 0 0 0 0 0 0 0
0 1 1 0 2 2 0 0 0
0 1 1 0 2 2 0 0 0
0 0 0 0 0 0 3 0 0
0 0 0 0 0 0 0 4 0
0 0 0 0 0 0 0 0 0
```

The **label** command produces the labelled image in two or more passes through the data (two passes except for very detailed images). The results of the first pass are output to an integer form picture with the same dimensions as the source picture. This will be the output picture itself unless the output picture is a display picture, in which case a temporary disc picture is created to hold the results of the first pass.

The range of the final result is stored in the output picture label. This means that the total number of labelled regions can be obtained by means of the **survey** command with no additional processing. The range is limited by the range of 16-bit integers which means that a maximum of 32767 separate regions can be labelled (and therefore counted) in one image.

label

Notes

see also: **analyse**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
fg/bg	label foreground regions	

ladjust

keys:	**range**	*<n1>, <n2>*	adjust the display levels over the specified range
	scaled	*<number>*	scale the ***range*** values in terms of the black and white levels of the partition number specified by ***scaled***
	end	*<n1>, <n2>*	if ***tie***, constrain the lut modification range
	hsv	*<n1>...<n6>*	if ***initially***, set the lut to a linear ramp in hue, saturation and intensity over the specified end limits
options:	**tie**		use the ***end*** key to constrain the lut modifications
	all/red, green, blue		if full colour lut, specify the colour channel(s) to be modified
	brightness		vary the brightness of pixels
	dbrightness		vary the rate of change of brightness
	contrast		synonym for ***dbrightness***
	hue		vary the hue of pixels
	dhue		vary the rate of change of hue of the pixels
	saturation		vary the colour saturation of the pixels
	dsaturation		vary the rate of change of colour saturation
	position		vary the position of the centre of the modification range
	dposition		vary the width of the modification range
	width		synonym for ***dposition***
	lower		vary the lower limit of the modification range
	upper		vary the upper limit of the modification range
	initially		set the lut to the hue, saturation and intensity specified by ***hsv*** over the modification range

The **ladjust** command allows you to adjust the current display output look-up table interactively, using a mouse or cursor keys.

Examples

```
ladjust brightness contrast
```

This command adjusts the display brightness on vertical movement of the mouse/cursor and

ladjust

adjusts the contrast on horizontal movement.

```
ladjust range 64, 192 hue saturation
```

This command adjusts the hue on vertical movement of the mouse/cursor and saturation on horizontal movement, over the central range of a (false-colour) look-up table.

```
min=0 max=1; create display size 256 fp; ladjust range 0.5, 1 scaled
   display hue
```

This sequence of commands adjusts the hue of the upper half of the look-up table (the **range** values are scaled to match the *display* scaling).

```
ladjust range 120, 130 position width initially hsv 120,1,1
```

This command adjusts the position and width of a highlight band. The intensities of the band are initially set to bright green. Variables *r* and *r2* (range values) are set to the lower and upper limits of the adjustment range on exit.

```
ladjust brightness dbrightness blue
```

This command adjusts the brightness and contrast of the blue component only of a full-colour lut.

```
ladjust range 80, 120 tie end 40, 200 brightness contrast
```

This command adjusts the brightness and contrast over the range 80 to 120 but constrains the modification so that it does not affect the lut below 40 and above 200. These end values keep their original values despite the modification of the range.

Description

ladjust allows you to relate the parameters of a look-up table to the movements of a mouse or to the movements of the cursor using a cursor key. For example, you can associate the horizontal movements of a mouse to the **brightness** of a lut and the vertical movement of a mouse to the **contrast**. Then, by moving the mouse, you set the output look-up table to achieve the desired effect.

If you are using a *false-colour* look-up table, you can control the colour as well as the brightness of each display level, using the options **hue** and **saturation**.

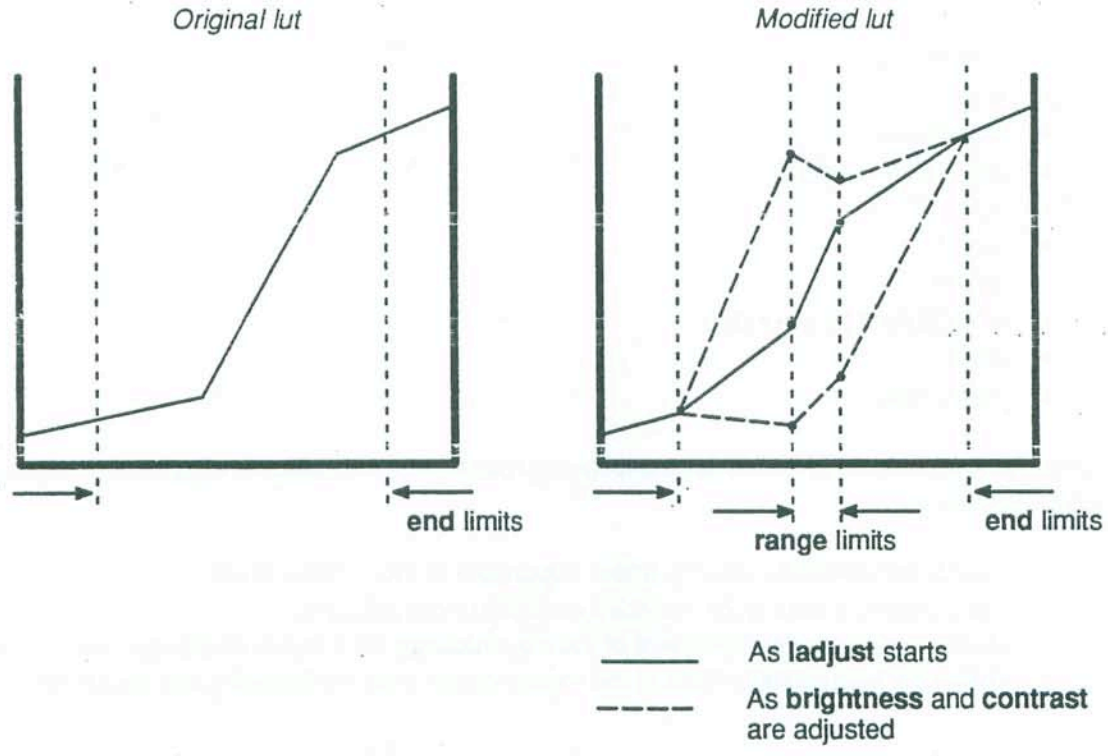
ladjust

You can specify a range of display levels to adjust using the **range** key. You can also vary the range itself, using the following options to perform thresholding or highlighting of an image:

- **position** (centre of range, *(n1+n2)/2)*
- **dposition** (width of range, *(n2–n1)*)
- **width** (synonym for **dposition**)
- **lower** (lower limit of range, *n1*)
- **upper** (upper limit of range, *n2*)

If you use the **scaled** key to specify a range relative to the black and white levels of an existing display picture then the range values (held in the variables *r* and *r2*) are converted back once you finish adjusting the lut interactively.

The **end** key allows you to adjust a parameter (**hue**, **brightness**, **saturation**) within the values specified by **range**, leaving the look-up table continuous over end values. For example, if you change the brightness over the range, then the brightness ramps linearly between the values held in **end** *n1* and **range** *n1* and the values **range** *n2* and **end** *n2*. Specify the **tie** option with the **end** key. The following diagram illustrates how the levels outside the range change to blend with the levels you are adjusting but that the **end** limits do not change.



ladjust

Use the **hsv** key with the **initially** option to set the lut over a range to a linear ramp in hue, intensity and saturation:

hsv *<hue>, <saturation>,<intensity>*

Note that **hue** and **saturation** only apply to false-colour luts. Hue is specified normally in the range 0 to 360, saturation and intensity in the range 0 to 1. The defaults for **hsv** are:

hsv	*hs2*	*hs3*	*hs4*	*hs5*	*hs6*
0 *(hue)*	1 *(saturation)*	1 *(intensity)*	*hsv (hue)*	*hs2 (saturation)*	*hs3 (intensity)*

(The initial colour is bright.red and the final colour is the same as the initial colour).

Further information

Note that if you only specify one option (for example, **hue**) this option is related to the vertical movement of the mouse and the horizontal movement has no effect. If you specify two or more options, the options are allocated first to vertical movement of the mouse and then to horizontal movement. Semper selects the options in a particular order to relate to mouse movements. The order of selection is as follows:

- **brightness**
- **hue**
- **saturation**
- **dposition** or **width**
- **position**
- **lower**
- **upper**
- **dbrightness** or **contrast**
- **dhue**
- **dsaturation**

Once you type **ladjust**, you can use the following keys to alter the variables attached to the mouse or cursor keys:

r resets the variables currently under adjustment to their initial values
? print a report showing the variables currently under adjustment
x define the horizontal movement of the mouse/cursor from the following key (see over)
y define the vertical movement of the mouse/cursor from the following key (see over)

ladjust

Permitted keys:

d the following character defines a *rate-of-change* parameter

b brightness
h hue
s saturation
c contrast
l lower limit of range
u upper limit of range
p position of centre of range
w width of range

o turn off the adjustment

For example, the key sequence **xbydb** would set horizontal movement to vary brightness and vertical movement to vary contrast.

Notes

see also: **lset**, **lut**
variables set: *r*, *r2* (lower and upper range limits)
variables used: *r*, *r2* (lower and upper range limits)

Defaults and Ranges

keys/options	defaults	range
range	total range of display levels supported in look-up table; often 0–255 or 0–127	range is hardware dependent
scaled	*none*	valid display picture number
end	total range of display levels (as ***range***)	range is hardware dependent
hsv	see table in ***Description***	range is hardware dependent
all/red, green, blue	all three channels	
position	centre of range, *(n1+n2)/2*	
dposition	width of range, *(n2–n1)*	
width	as ***dposition***	
lower	***range*** *n1*	
upper	***range*** *n2*	

lattice

keys:	**[ ]**	*<number>*	display picture, partition, frame or picture number
	[to]	*<number>*	output *Plist* listing lattice sites
	mark	*<number>*	display picture to be marked
		<yes *or* no>	
	size	*<x>, <y>*	dimensions of subregion
	position	*<x>, <y>*	position/offset of subregion
	radius	*<number>*	maximum radius for lattice sites
	spacing	*<number>*	specify lattice spacing
	mark	*<number>*	mark lattice lines or individual sites
	mkmode	*<number>*	mark mode, if ***sites***
	mksize	*<number>*	mark size, if ***sites***
options:	**picture/partition/frame/from**		interpret **[ ]** as display picture, partition, frame or picture number
	sites		mark lattice as individual sites rather than by lines
	left/right/bottom/top		subregion position

Use **lattice** to mark a grid or lattice on a display, or to create a list of lattice sites in a *Plist* picture.

Examples

```
lattice spacing 20 mark display
```

This command marks a lattice with a spacing of 20 pixels throughout the current display picture.

```
u=19.3,24.5 v=-2.3,29.5 w=2.4,-5.1; lattice to 1
```

This command lists in picture 1 the sites of a lattice covering the area of the current picture display, using the specified base vectors (*u, v*) and origin (*w*).

```
lattice from 6 to 3 size 256 bottom left radius 150
```

This command lists in *Plist* picture 3 those sites of the lattice defined by *u,v,w* that lie within a radius of 150 from *w,w2* and also in the indicated subregion of picture 6.

lattice

Description

You can either define a lattice by setting *u* and *v* to its base vectors, and *w* to the position of its origin (if this is not the picture origin) or use the **spacing** key as a quick way of defining a square lattice (*u*=**spacing**,0 *v*=0,**spacing** *w*=0,0), usually to mark a coordinate grid on a display.

The field of view of a lattice is determined with respect to a source region, in one of the two following ways:

(i) you can select a display picture, partition or frame by means of the **picture/partition/frame** option with the field of view determined with respect to the relevant coordinate system, for example, **lattice display**

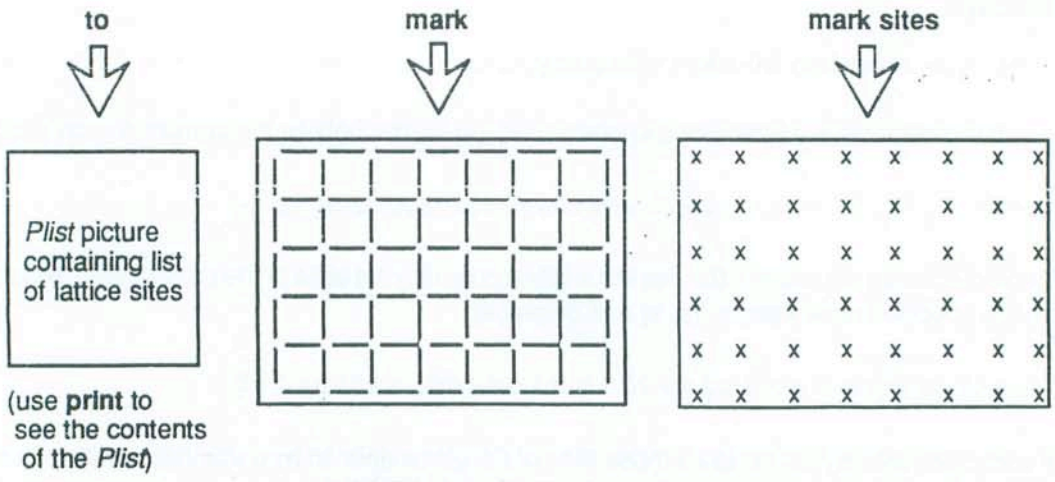
(ii) you can specify a source picture using the **from** option, for example, **lattice from 6**

In either case, you can use the standard 2-D subregion options/keys, for example, **size**, **position** to display the lattice over a subregion. You can also use the **radius** key to restrict the lattice to a given radius from the origin *w, w2* as in the last command example.

The form of lattice output is determined by two independent keys, **mark** and **to**. If you use **mark**, the lattice is marked on the indicated display picture. If you use **to**, it is output to the specified picture in the form of a *Plist*. You must specify either **mark** or **to** to obtain a result from the **lattice** command.

By default, lattice **mark**ing takes the form of intersecting lines with short gaps at the lattice sites. Use the **sites** option to mark the individual sites instead, and specify the form of marking using the general keys **mkmode** and/or **mksize**. For further detail of these keys, refer to *Appendix C: Semper Keys and Options*. The following diagram illustrates the output of the **lattice** command if **mark** is set.

lattice output



lattice

Notes

variables used:

u,u2	(first base vector of lattice)
v,v2	(second base vector of lattice)
w, w2	(position/offset of lattice origin)

Defaults and Ranges

keys/options	defaults	range
[ ]	*none*	valid display picture/partition/ frame number or, if ***from***, valid picture number
[to]	*none*	valid picture number
mark	mark off	see *Appendix C*
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0,0	within bounds of the picture (integers)
radius	infinite	positive real number
spacing	*none*	positive real number
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
picture/partition/ frame/from	picture	

learn

keys:	**[from]**	*<number>*	source picture (1 or more layers)
	to	*<number>*	output picture containing training statistics (multi-layer)
	data	*<n1>...<n9>*	*Plist* picture describing training area
	class	*<n1>...<n9>*	specify a class for a region
options:	**verify**		type the gathered statistics in detail

The **learn** command calculates the training statistics of polygonal regions. These statistics can be used in supervised classification. Use the **learn** command in *Remote Sensing* applications.

Examples

```
learn 1 to 20 data 10, 11, 12
```

This command calculates the statistics of areas described by the *Plists* 10, 11 and 12. The statistics are stored in picture 20, with the raw data taken from picture 1.

```
learn 2 to 20 data 10, 11, 12 verify
```

This command performs the same actions as the above command, but also details the number of pixels in each class, sending output to the terminal.

```
learn 2 to 20 data 10,11,12,13,14 class 1,1,1,2,2
```

This command calculates the statistics for two classes. The first class contains 3 regions and the second contains 2 regions.

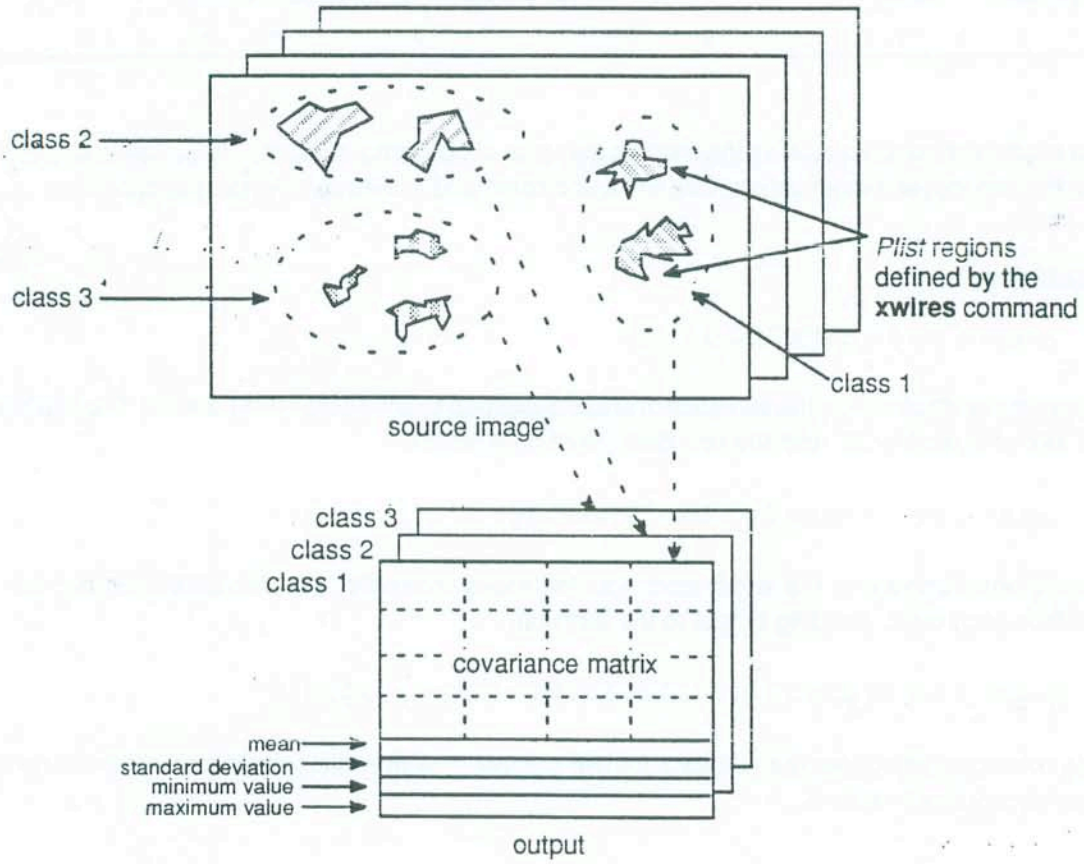
Description

The **learn** command calculates the statistics of arbitrary polygonal regions, the output of which is suitable for use by the supervised classification commands, such as **box**, **mindistance** and **likelihood**. The statistics calculated for each region are as follows:

- covariance matrix
- mean
- standard deviation
- minimum
- maximum

learn

learn calculates the mean, standard deviation, minimum and maximum for each layer of the source picture. The **data** key specifies the *Plists* which describe the polygon, or *training region*. (These describe the area for which the statistics are calculated). You can specify up to nine polygons in a comma separated list using the **data** key. If you require more than nine polygons, use the **stack** command to combine the result of several **learn** commands. The region's statistics are stored in a separate layer of the output picture specified by **to**. The following diagram illustrates the typical operation of the **learn** command.



By default, each training region belongs to a separate class. Use the **class** key to specify that a class contains more than one training region. There must be the same number of **class** keys as **data** keys, for example:

```
learn to 999 data 3,7
```

specifies 2 regions and therefore requires 2 classes. The command:

```
learn to 999 data 3, 4, 5, 6
```

learn

specifies 4 regions. Classes can be numbered from one up to the number of regions specified by the **data** key.

If you specify the **verify** option, **learn** outputs the number of pixels that it finds in each class to the console. **learn** displays an asterisk (*) beside a class which does not have enough data for the maximum likelihood classification (if the number of pixels in the training region is less than ten times the number of layers in the source picture). If statistics are calculated from too small a training region, the covariance matrix is likely to be singular (no inverse) and the maximum likelihood classification method cannot be used. See the **likelihood** command for details of the maximum likelihood classification.

The **learn** command estimates the maximum number of pixels in each region in order to create a temporary picture. This temporary picture contains the pixels of an area whilst its statistics are calculated. It is possible for **learn** to fail if there is not enough space for the temporary picture on disc. (Use the **compress** command to gather free disc fragments). **learn** may also fail in the unlikely event that the maximum number of pixels is estimated incorrectly. This situation is only likely to occur if you supply very long, thin horizontal training regions. Note that the **learn** command sets as many of the variables (*area*, *ar2*, *ar3*, *ar4*, *ar5*, *ar6*, *ar7*, *ar8* and *ar9*) as there are classes. The variables are set to the number of pixels found in each class; use the **verify** option to see these values.

Notes

variables set: *area, ar2, ar3...ar9* (number of pixels found in each class)
see also: **box, compress, likelihood, mindistance, stack**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
to	*none*, specify an output picture	valid picture number
data	*none*	positive integer
class	each region is associated with a separate class	positive integer
verify	verification off	

letter

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	text	*'<text>'*	text of lettering
	size	*<number>*	row length of output, if no source picture
	fg	*<number>*	foreground colour used in lettering
	bg	*<number>*	background colour used in lettering
options:	**title**		take text from source picture title

Use **letter** to add lettering at the bottom of a picture, or to create a new picture containing nothing but text (to **paste** into another picture).

Examples

```
letter 1 text 'Fish-eye view of woodland growth'
```

This command adds the lettering below the bottom of picture 1. Note that you cannot see the lettering until you display or re-display picture 1.

```
letter 1 to 2 title
```

This command takes the lettering from the title of picture 1, and puts the lettering in picture 2.

```
letter fg 0 bg 255; paste to 5 bottom right
Text (as textstring): 'Plate',n
```

Note that the terminal prompts for text in this example. This command places a small block of black on white lettering (*'Plate'* and the value of variable *n*) in picture 999, and pastes it into picture 5 at the bottom right.

Description

Use the **text** key to enter the text of the lettering or use the **title** option so that the lettering is taken from a specified picture title.

Use the keys **bg** and **fg** to define 'background' and 'foreground' values for the pixels that make up the lettering. By default, **bg** and **fg** take the values of the variables *min* and *max* respectively. If you do not indicate a source picture, as in the last command example, the default output is 999.

When you have no source picture, you can specify the output row length of the lettering using the **size** key. If the lettering is too wide, it is split into more than one line as necessary. By default, **size** is

letter

made just large enough to accommodate the lettering (with a border at least a pixel wide), up to the maximum row length for your installation.

Notes

multi-layer pictures: faulted
forms used internally: integer, fp, complex
see also: **paste**

Defaults and Ranges

keys/options	defaults	range
[from]	*none*	valid picture number
[to]	***from*** if set, otherwise 999	valid picture number
text	*none;* prompts at terminal for text	text string: length is installation dependent
size	minimum size necessary to fit lettering	maximum row length of your installation
fg	value held in *max*	positive integer
bg	value held in *min*	positive integer

library

<library program name>	run the specified library program

The **library** command causes Semper to execute a specified program from the beginning to a **return** command or to the end of the program.

Examples

```
library partitions
```

This command executes the standard library program called *partitions*, which defines partitions subdividing a display frame with arbitrary numbers of rows and columns.

Description

You can specify several library commands in a single line, include them in a **for** loop, nest calls freely etc. Note that Semper identifies a program name from all the letters in its name, and not just the first three letters (as is the case with commands). Type **show system** to see the maximum name length. Use the **list** command to list a library program or to copy it to a new file. Use **show programs** to see a list of all the library programs contained in the currently assigned program libraries.

Notes

see also: **list, return, show**

likelihood

keys:	**[from]**	*<number>*	source picture (1 or more layers)
	[to]	*<number>*	output picture (single layer)
	with	*<number>*	picture containing training statistics
	stride	*<number>*	sampling interval across and down the picture
	position	*<x>, <y>*	position, offset of subregion
	size	*<x>, <y>*	dimensions of subregion
	probability	*<n1>...<n9>*	the *a priori* probability of a class
	threshold	*<n1>...<n9>*	the threshold percentage to apply to each class
options:	**left/right, top/bottom**		position of subregion

The **likelihood** command classifies a picture using the *maximum likelihood* method. Use the **likelihood** command in *Remote Sensing* applications.

Examples

```
likelihood 1 2 with 3
```

This command classifies picture 1 using the training data contained in picture 3 and sends the output to picture 2.

```
likelihood 1 2 with 3 probability .1,.05,.50
```

This command performs the same actions as the above example and also assigns *a priori* probabilities to the three classes.

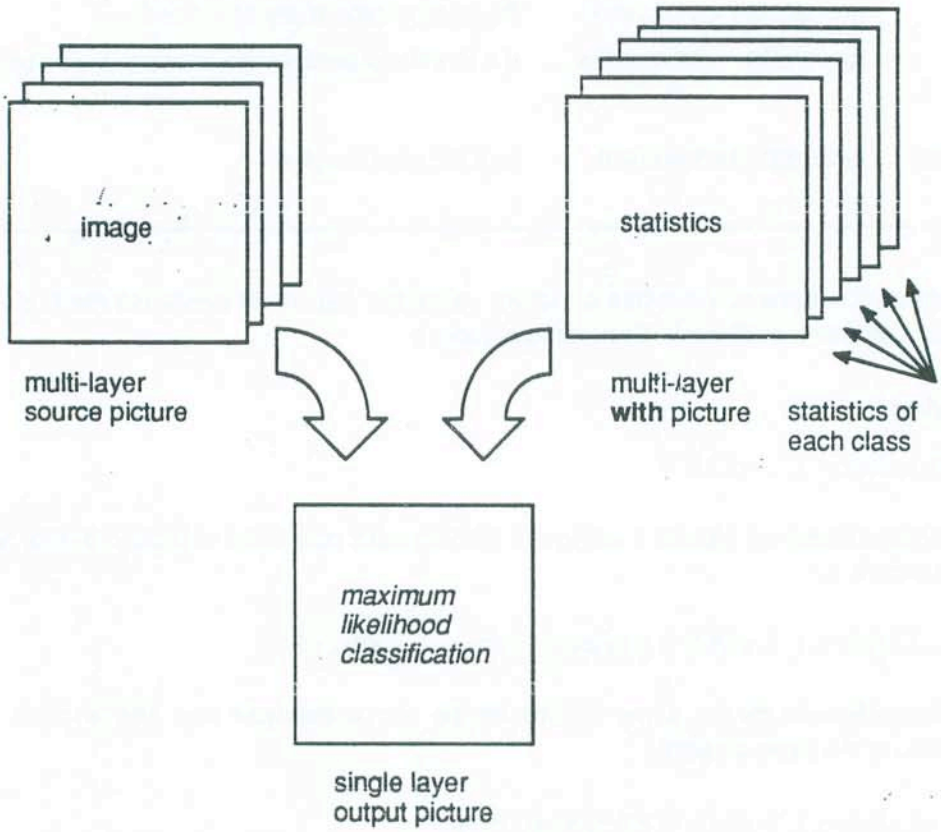
```
likelihood 1 2 with 3 threshold 90
```

This command performs the same action as the first command example and also assigns a threshold of 90% to each class. This means that 90% of the pixels within each class are classified.

likelihood

Description

This command performs a maximum likelihood classification of the source image based upon the statistics given in the **with** picture. The **with** picture, which is multi-layer, is the output of the **learn** or **covariance** commands. Each layer of the **with** picture contains the statistics of one class. Classes run from zero, indicating unclassified, to *n*, where *n* is the number of layers in the **with** picture. Note that the number of classes that can be processed depends upon the size of the Semper row buffer and on the number of layers in the source picture.



By default, all classes are assumed to have an equal probability, that is, are equiprobable. You can specify the *a priori* probability of a class using the **probability** key. The sum of the probabilities must be less than or equal to 1.0; allowing a probability of elements being unclassified. If you specify one probability, then you must give the probabilities of all classes.

Use the **threshold** key to specify the percentage of pixels within a class that are to be classified (**threshold** assumes that the classes are normally distributed). When you apply a threshold percentage (in the range 1 to 100) some pixels may remain unclassified and are indicated in the output picture by a zero value. If you omit the **threshold** key all pixels in the source picture are classified.

likelihood

Normally the number of thresholds should be the same as the number of layers (classes) in the **with** picture, but if only one threshold value is given then it is applied to all classes. In this way, you can perform classifications that involve more than nine regions with a threshold applied. If you specify more than one threshold value then the number of thresholds must agree with the number of classes, as is the case for probabilities.

By default, the output picture has the same size (number of rows and columns) as the source picture. You can alter this by using the **size** and **position** keys to specify a subregion. You can also use the **stride** key to specify the step across and down the picture, allowing a fast classification of a picture. The **stride** key defaults to a value of 1 so that the whole picture is classified.

You can define a subregion using the standard subregion keys and options, **size**, **position**, **left/right** etc. For further detail of these keys and options, refer to *Appendix C, Semper Keys and Options.*

It is possible for the **likelihood** command to fail before classification if any of the class covariance matrices are singular (that is, have no inverse). This is most likely to arise if insufficient pixels have been used in the training region (see **learn**).

Notes

see also: **covariance, learn**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none*	valid picture number
stride	stride 1	positive integer
position	position 0,0	within bounds of picture (integers)
size	whole picture	less than or equal to the size of the picture (integers)
probability	*none*	real number in range 0 to 1
threshold	all pictures are classified	integer in range 1 to 100

list

keys:	**[from]**	*<number>*	number of macro to list
	program	*'<text>'*	name of program to list
	device	*<number>*	list program in library on specified device number
	name	*'<text>'*	send listing to a named file rather than to the console
options:	**all**		list all programs in all assigned program libraries
	new		overwrite existing output file
	old		re-use existing output file

Use **list** to list the text of a program or a numbered macro at the console. You can also use **list** to write a program to a file.

Examples

```
list program 'combine'
```

This command lists the program called *combine* on the terminal.

```
list program 'combine' name 'progtext'
```

This command creates a file called *progtext.spl* containing the program text of program *combine*.

```
list all
```

This command lists all programs in any assigned program libraries.

```
list 30
```

This command lists macro number 30 at the console.

Description

Use the **device** key with **list** if two or more programs have the same name, to specify the device number of the library containing the program.

list

You can use the command **list program...name** to recover the source text of library programs, by listing a program from a Semper library to file.

Note that you can only send *program* text to a file outside Semper using **list**. You cannot send *macro* text to a file (to do this, refer to the **write** command). If you type **list name 'filename'** and the file already exists, Semper displays an error message. You need to use the option **new** to overwrite an existing output file.

Notes

see also: **write**

Defaults and Ranges

keys/options	defaults	range
[from]	macro number held in the variable *select*	valid macro number
program	*none*	valid program name
device	first library in current search order	integer in range 1 to system limits (type ***show system***)
name	*none*	valid filename

live

This syntax is specific to Sprynt systems, using SCIB as the image grabber

keys:	[to]	*<number>*	display picture to copy captured images to
	tod	*<number>*	picture on a memory device or picture disc device
	sigma	*<number>*	number of images to integrate over
	skip	*<number>*	number of images to skip between those integrated
	wait	*<number>*	the time in seconds for which to repeat copying
	timeout	*<number>*	approx maximum time in seconds to wait for an image to be captured before erroring
	size	*<n1>, <n2>*	if not capturing to the display then specifies the size of disc picture
	window	*<n1>, <n2>*	specifies the size of subregion of the SCIB scan to be captured
options:	exttrig		only to capture when external trigger received
	preset		record display picture black and white levels as current values of **min** and **max.**
	menu		not to issue 'Live – press any key or mouse button to grab' prompt if **wait** unset
	Initialise		if it is first invokation of the live command in current Semper session then a couple of dummy captures will be done before proper capture

Use **live** to grab frames continuously from a video_rate source for a fixed or variable period.

Examples

```
live
```

This command captures images from the video input into the current display picture until you press a key or the mouse button. At this point it freezes the result in the framestore.

```
partition 2 size 512,512; live to dis:2 wait 5
```

This command captures images from the video input into partition 2 for five seconds and then freezes the result in the framestore.

live

This syntax is specific to Sprynt systems, using SYNAPSE as the image grabber

keys:	**[to]**	*<number>*	display picture to copy captured image to
	subsample	*<number>*	subsample factor to use in copy
	window	*<n1>, <n2>*	Synapse acquisition scan position with which to align top left of copy region when copying
	wait	*<number>*	if set then the time in seconds for which to repeat copying
options:	**synchronous**		if acquisition turned on then temporarily halt acquisition while doing the copy
	preset		record picture black and white levels as current values of **min** and **max**
	menu		if **wait** key unset then not issue 'Live – press any key or mouse button to grab' prompt

---

PC + SCIP framestore only

```
live to dis:1 tod cd:1
```

This command captures images continuously to partition 1 and CD:1 and stops when any key or mouse button is pressed.

```
live to dis:1 tod cd:1 sigma 16 skip 1
```

This command captures integrated images to the display and disc simultaneously. The integrated images are the sum of 16 images. Every other image from the camera is ignored. ie. Image 0,2,4,6,...28, 30 are summed. CD:1 will be a1 layer integer picture. DIS:1 will be a1 layer byte picture. When a key or mouse button is pressed, capture stops.

---

live

The **timeout** key is used to set the maximum time in seconds to wait for image to be captured before giving a time out error. If a time out error occurs it is likely to be due either to the camera not being connected/turned on or that the way the sync is connected to the input board is not as selected in the mode being used.

Capture can be triggered by an external signal. If the option **exttrig** is specified capture will only be done when an external trigger is received.

The SCIB board requires either 1or 2 captures to initialise its memory after power up. When the **initialise** option is set and it is the first invokation of the live or snapshot command in the current Semper session, a couple of dummy captures from SCIB will be done before the proper capture.

PC + Sprynt using Synapse as the grabber only

The key **subsample** specifies a subsampling factor to use in the copy. The value of subsample affects the dimensions of the area of the acquisition scan of that is samples. eg if the display partition or the disc picture is specified to have size128,128 and subsample has value 2 then a 256,256 *area* of the Synapse acquisition scan is subsampled.

When the option **synchronous** is set and acquisition is on then it is temporarily halted at the beginning of the next first field while the copying from Synapse to Sprynt takes place. This ensures that the image copied consists only of a single complete first field, second field pair.

---

live

Description

The **live** command for SCIB is available for both the Synapse grabber and SCIB grabber. When SCIB is used you can capture colour images to the display, and capture sequences of images or preform image integration on monochrome images.The keys and options for SCIB and Synapse grabbers are not exactly the same.

The **to** key can used to specify both display and disc picture if Synapse is the grabber, but can only be used to specify display picture if SCIB is the grabber.

When the **wait** key set it is the time in seconds for which capturing continuous. If unset then capturing is repeated until any key or mouse button is pressed.

If you only want to capture a subregion of a camera scan use the **window** key to specifiy the position in the camera scan from which the top left pixel in the captured images is to come, where window 0,0 denotes the top left pixel in the camera scan.

If the **preset** option is set the black and white levels recorded with a Semper display picture are set to the current value of *min* and *max*. When it is not set and an image is captured into the display the black and white levels and variables *min* and *max* are set to 0.0 and 254.0 if the display is true colour, and 0.0 and 127.0 if the display is false colour if the SCIB is the grabber; the black and white levels and variables *min* and *max* are set to 0.0 and 255.0 if Synapse is used as the grabber.

The option **menu** is for UIF programs. When it is set and the key **wait** is unset then the 'Live – press any key or mouse button to grab' prompt is not issued.

PC + SCIB framestore only

When SCIB is used as the grabber capturing to display can be done with both colour and monochrome images. When capturing monochrome images display partitions should only have one frame.

Capturing to disc can only be done on monochrome images when Semper is running in a false colour display mode. The key **tod** is used to specify the disc picture. The disc picture must either be on a memory device or on a picture disc device with its own memory buffer. The size of disc picture is the size of display picture if capturing to display at the same time. When capturing to disc only, the **size** key can be used to specify the size of disc picture.

Integrating capture requires capture to disc. Simultaneous capture to the display is optional. Which and how many images to capture are controlled by the keys **sigma** and **skip**.

live

Defaults and Ranges

keys/options	defaults	range
[to]	*the current display picture*	valid picture number
wait	*none*	positive integer
window	*copy region positioned at centre of camera scan*	integer pair in range 0,0 to the camera scan size
tod	*none*	valid memory or disc picture number
size	*the size of SCIB scan (has no effect if capture to the display at the same time)*	integer pair in range 0,0 to the camera scan size
subsample	*1*	positive integer
sigma	*1*	positive integer
skip	*0*	positive integer
timeout	*4*	–1 or positive integer
preset	*no*	
menu	*no*	
synchronous	*yes*	
exttrig	*no*	
initialise	*on*	

live

This command is specific to...
PC + framestore/greystore

This syntax is specific to...
PC + Data Translation DT2861 framestore

keys:	**wait**	*<number>*	specify time in seconds to capture video input
	frame	*<number>*	grab input into specified framestore frame
options:	**snap**		grab the next frame from the video input into the framestore
	external		as ***snap***, but only grab the frame when a high-to-low transition is detected on the *Trigger In* line

This syntax is specific to...
PC + Imaging Technology PCVISIONplus framestore

keys:	**wait**	*<number>*	specify time in seconds to capture video input
	partition	*<number>*	capture the image into the specified display partition
	ilut	*<number>*	specify an input look-up table
	channel	*<number>*	specify video input channel (1 or 2)
	mask	*<number>*	set the video input write-protect mask
	lzoom	*<number>*	specify the sub-sampling factor
options:	**preset**		use the existing values of *min* and *max* for black and white scaling
	pll/crystal		the framestore uses its phase locked loop with an external sync source, or its own internal crystal, as the sync source

This syntax is specific to...
PC + Matrox PIP512/PIP1024 framestore

keys:	**wait**	*<number>*	specify time in seconds to capture video input
options:	**snap**		grab the next frame from the video input into the framestore

live

This syntax is specific to...
PC + MRC500 framestore
PC + Synoptics Synergy framestore

keys:	**wait**	*<number>*	specify time in seconds to capture video input
	partition	*<n1>, <n2>*	specify a source partition of the video signal to record and a destination partition for data output
	time	*<number>*	time constant for pseudo real-time recursive filters
options:	**preset**		use the existing values of *min* and *max* for black and white scaling

This syntax is specific to...
PC + Synoptics Synapse framestore

keys:	**wait**	*<number>*	specify time in seconds for video input
	partition	*<number>*	capture the image in the specified display partition
	time	*<number>*	selects a particular time constant for the real-time recursive filter
options:	**preset**		use the existing values of *min* and *max* for black and white scaling

This syntax is specific to...
PC + Quantimet 520 greystore

keys:	**wait**	*<number>*	specify time in seconds to capture video input
	time	*<number>*	selects a particular time constant for the real-time recursive filter
	partition	*<number>*	capture the image in the specified display partition
	gain	*<number>*	specify the gain setting to be applied to the video input signal
	offset	*<number>*	specify the offset to be applied to the video input signal
options:	**preset**		use the existing values of *min* and *max* for black and white scaling
	start/stop		start/stop video acquisition

live

This syntax is specific to...
PC + Metrabyte Corporation MV1 framestore

keys:	**wait**	*<number>*	specify time in seconds to capture video input
	partition	*<number>*	capture the image in the specified display partition
	ilut	*<number>*	specify an input look-up table
	channel	*<number>*	specify the video input channel (1 or 2)
	gain	*<number>*	specify the gain setting to be applied to the video input signal
	offset	*<number>*	specify the offset to be applied to the video input signal
	roffset	*<number>*	specify the offset to be applied to the video input of the red framestore (full colour systems only)
	goffset	*<number>*	specify the offset to be applied to the video input of the green framestore (full colour systems only)
	boffset	*<number>*	specify the offset to be applied to the video input of the blue framestore (full colour systems only)
options:	**preset**		use the existing values of *min* and *max* for black and white scaling
	pll/crystal		the framestore uses its phase locked loop with an external sync source, or its own internal crystal, as the sync source

Use **live** to grab frames continuously from a video-rate source for a fixed or variable period.

Examples

```
live
```

This command captures frames from the video input into the current frame or partition until you press a key or the mouse button. At this point it freezes the result in the framestore or greystore.

live

```
live wait 5
```

This command captures frames from the video input for five seconds and then freezes the result in the framestore/greystore.

PC + Data Translation DT2861 framestore only

```
live snap
```

This command grabs the next frame from the video input into the framestore.

```
live snap external
```

This command performs the same function as **live snap** except that it does not grab the frame until a high-to-low transition is detected on the *Trigger In* line.

```
live frame 3
```

This command captures frames from the video input into frame 3 of the framestore, until you press a key or the mouse button.

PC + Imaging Technology PCVISIONplus framestore only

```
live partition 1 izoom 2
```

This command captures frames from the video input and places them in partition 1, until you press a key or the mouse button. The live image is zoomed to half its size by specifying **izoom 2**.

```
live partition 2 channel 1 crystal
```

This command captures frames from video input channel 1 and places them in partition 2, until you press the a key or the mouse button. The framestore uses its own internal crystal as the sync source.

PC + *Matrox PIP512/PIP1024* framestore only

```
live snap
```

This command grabs the next frame from the video input into the framestore.

live

PC + MRC500 framestore and Synoptics Synergy framestore only

```
live partition 2,3 wait 2.0
```

This command captures frames from the video input over the area of partition 2 for two seconds, then re-samples the data in this region so that it fills partition 3.

```
min=-100 max=100; live partition 1 preset wait 1.0 fp
```

This command captures the video input for one second into partition 1. The black and white levels are recorded as having the values of *min* and *max*. The data is recorded as representing a floating point picture.

PC + Synoptics Synapse framestore only

```
live wait 2 partition 1; copy display:1 to :400
```

This command captures frames from the video input into the frame(s) of partition 1 for two seconds and then freezes the result in the framestore. The part of the image in partition 1 is copied to disc picture 400. By default the picture is scaled from 0 to 255.

```
live time 1 wait 5
```

This command captures frames from the video input into the current display partition for five seconds using the real-time recursive filter to reduce noise. It then freezes the result in the framestore.

PC + Quantimet 520 greystore only

```
live wait 2 partition 1; copy display :1 to :400
```

This command captures frames from the video input into the greystore for 2 seconds and then freezes the result. It then copies the part of the image in partition 1 to disc picture 400. By default, the picture is scaled from 0 to 63.

live

```
live time 1 gain 5 offset 2
```

This command captures frames from the video input into the current display partition, using the real-time recursive filter to reduce noise. When you press a key or the mouse button the result is frozen in the greystore. It applies a gain of 5 and an offset of 2 to the video input signal.

```
live time 2 start; live stop
```

This command starts and then stops video acquisition using a recursive filter to reduce noise.

PC + Metrabyte Corporation MV1 framestore only

```
live partition 1 roffset 10
```

This command captures frames from the video input into the framestore, until you press a key or mouse button – at which point it freezes the result in the framestore and creates Semper picture *display:1* in partition 1. This command also applies an offset of 10 to the red input channel (full colour systems only). Offsets can be integers in the range 0 to 255, where a zero value gives the lightest image.

Description

The **live** command captures frames from a video input for a period of time and then freezes a single frame. It destroys the previous contents of the image plane of the framestore or greystore frame receiving the video input. To keep a permanent record of the live image you need to store the grabbed image using the **create** and **copy** commands, for example:

```
live wait 1.5 frame 1; min=0 max=255
create display:1 size 512 frame 1 byte;copy dis:1 to :400
```

This sequence of commands places a grabbed image in frame 1 of the framestore into disc picture 400. The picture is scaled so that black is recorded as value 0 and white as value 255. Note that you do not need to create a display using the **create** command for the following framestores: *Imaging Technology* PCVISIONplus, MRC500, *Metrabyte Corporation* MV1, *Synoptics* Synapse and Synergy framestores.

Use the **wait** key to specify the time to freeze video input. If you specify a positive value for **wait** the framestore captures frames from the video input for the specified number of seconds before freezing. If you supply a negative value, Semper waits for you to press a key or the mouse button before freezing the input.

live

To capture an image in a particular display partition, use the **partition** key. Centring only occurs on a few framestores and the framestore creates a display picture at this partition. (This key is not available if you have a *Data Translation* DT2861 framestore or a *Matrox* framestore).

Use the **preset** option to determine the black and white scaling of a live image. By specifying **preset**, you force Semper to use the existing values held in the variables *min* and *max* for black/white scaling instead of the actual range of values in the image. (This key is not available if you have a *Data Translation* DT2861 framestore or a *Matrox* framestore).

PC + Data Translation DT2861 framestore only

Use the **frame** key to capture the live image in a frame other than the currently viewed frame in the framestore.

Use the **snap** option to capture the next frame from the video input into the framestore. Note that you can only specify the **external** option if you are using the **snap** option.

PC + Imaging Technology PCVISIONplus framestore only

A number of additional keys (**llut**, **channel**, **mask**, **lzoom**) and options (**pll/crystal**) are available with this framestore, to determine the form of live input.

The **llut** key allows you to specify which input look-up table maps the video input. The **channel** key allows you to specify which one of two input channels to use. Use the **mask** key to set the video input write-protect mask. For example, a value of 0 allows you to write to all bit planes, a value of 127 allows you to write to the most significant bit plane. The **partition** key specifies in which partition the image is to be captured. The framestore attempts to centre the grabbed image on the specified partition. The **lzoom** key allows you to specify the size of the captured image by controlling the sub-sampling factor. A value of 2 gives a half size image and a value of 1 gives a full size image.

The options **pll/crystal** allow you to specify an external or internal sync source. If you specify **pll**, the framestore uses its *phase locked loop* with an external sync source as the source. If you specify **crystal**, the framestore uses its own internal crystal as the sync source.

---

live

PC + Matrox PIP512/PIP1024 framestore only

The **snap** option supplied with the *Matrox* framestore, lets you capture the next frame from the video input into the framestore.

PC + Synoptics Synapse framestore only

If you have a *Synoptics* Synapse framestore you can grab live images with or without noise filtering. The **time** key allows you to specify a time constant for the real-time recursive noise filter. Values of 1,2 and 3 supplied with the **time** key correspond to time constants of 0.5, 0.125 and 1.0 seconds.

PC + Quantimet 520 greystore only

If you have a *Quantimet 520 Image Analysis System* you can also apply a noise filter using the **live** command. Use the **time** key to specify a time constant for the recursive filter. Values of 1, 2 and 3 correspond to the time constants of 0.5, 0.125 and 1.0 seconds (these figures may vary for different greystore installations).

Use the **gain** and **offset** keys to control the gain and offset of the Quantimet scanner. Use these keys to compensate for variations in the lighting conditions and camera sensitivity, so as to make best use of the available greyscale.

The **start** and **stop** options enable you to start live input and then execute other Semper commands with the greystore still *'live'*. Note, however, that certain types of host access to the greystore are faulted in this *'live'* start.

PC + Metrabyte Corporation MV1 framestore only

A number of additional keys (**ilut, channel, gain, offset, roffset, goffset, boffset**) and options (**pll/crystal**) are available with this framestore, to determine the form of live input.

The **ilut** key allows you to specify which input look-up table maps the video input. The **channel** key allows you to specify which one of the two input channels that the video input is connected to. Use the **gain** and **offset** keys to control the gain factor and offset applied to the video input of the

live

framestore. The keys **roffset**, **goffset** and **boffset** allow you to control the offset applied to the video input on the red, green and blue framestores respectively (full colour systems only).

The options **pll/crystal** allow you to specify an external or internal sync source. If you specify **pll**, the framestore uses its *phase locked loop* with an external sync source as the source. If you specify **crystal**, the framestore uses its own internal crystal as the sync source.

Notes

see also:	**create, copy, slowscan, sscan, snap, video**
variables used:	*min, max* (if **preset**)
variables set:	*min, max* (if not **preset**)

---

live

Defaults and Ranges

keys/options	defaults	range
wait	waits for you to press a key or mouse button	positive or negative integer
frame	currently viewed frame	valid frame number
partition	current display partition, held in the variable *display* or, if source and output partitions, partition 1	valid partition number
time	time=0	1, 2 or 3, except *Synergy*, which takes a positive real value
ilut	*none*	valid input look-up table number
channel	*none*	channel 1 or 2, if *Metrabyte MV1*, channel 1, 2, 3 or 4
mask	mask=0 (write to all bit planes)	integer in range 0 to 255
lzoom	*none*	positive integer
gain	*none*	integer in range 0 to 15, if *Metrabyte MV1* a gain factor of 0.5, 1.0, 1.5 or 2.0
offset	*none*	integer in range 0 to 15, if *Metrabyte MV1* an integer in the range 0 to 255
roffset	*none*	integer in range 0 to 255
goffset	*none*	integer in range 0 to 255
boffset	*none*	integer in range 0 to 255
preset	use actual black/white pixel values for scaling	
pll/crystal	use internal sync source (***crystal***)	

lmean

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	over	*<number>*	width of block/ strip averaged around each pixel
options:	**horizontally**		average horizontal strips around each pixel only
	vertically		average vertical strips around each pixel only

Use **lmean** to calculate the local mean over a square block of neighbouring pixels around each source pixel. In effect, **lmean** tells you the general brightness around each pixel, and the others measure the brightness variation.

Examples

```
lmean display over 50
```

This command smooths picture *display* over 50 point square blocks.

```
lmean 1 to 2 over 8 vertically
```

This command smooths vertically only, over 8 point columns

Description

Note that with **lmean**, execution time is more or less independent of the block size, so you can use very large block sizes if necessary. You use the **over** key for the size of block over which the local average is to be taken (default 5). You can average using 1-D **horizontal** or **vertical** forms as well as square forms. Note the possible clash of the **vertical** option with the general key **verify**.

Edge pixels of the source, where the block averaged overflows the source, are processed as if the boundary values are repeated indefinitely outwards. If you specify an even value for the **over** key, the replaced source pixel is rounded to the bottom right from the block centre.

Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**lsd, lvariance**

lmean

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
over	width/strip 5	positive integer

local

<variable name(s)>	specify a local variable

If you name one or more variables in a **local** command within a program or interactively, Semper notes their current state or value and restores this at the end of the program or command line.

Examples

```
n=.2; ...
local n, m; ... ; n=5; library anyprog; ...
```

This sequence of commands leaves *n* set to 0.2, in spite of the assignment and program call.

```
n=.2 m=100; ...
for n=1,5; ... ; for m=n,5; ... ; library anyprog; loop; loop
```

This sequence of commands leaves *m* and *n* set to 0.2 and 100 in spite of the loops and program calls (the loop variables *m* and *n* are automatically made local).

Description

If you use **local** interactively, the value of a local variable is restored interactively at the end of the command line, provided that any **for** loops that you specified after the **local** command are closed using **loop**. The **local** command allows you to use variable names in programs without any risk of their clashing with names used at higher or lower calling levels.

Note that variables that are named in **for** loops are automatically treated as local variables, so you do not need to take any special action to preserve them.

Notes

see also:	**for...loop**
restrictions:	**local** may not be used within a loop

log

<variable name(s)> *'<text string>' <variable name(s)>*	***log*** uses a special syntax. It prints the value of the specified variable(s) and any given text string to the log output stream.

Use the **log** command to print text and/or numerical values to the log output stream.

Examples

```
log min, max
```

This command prints the values of the variables *min* and *max* to the log output stream.

```
log 'Subregion mean and standard deviation are', mean, sd
```

This command prints the text, mean and standard deviation to the log output stream.

Description

The output takes exactly the same form as in the **type** command. **log** allows you to mix expression values and text. Specify text within single quotes and use a comma to separate values or expressions.

For further detail of the output streams that Semper uses, refer to the section *Controlling output in Semper* in *Chapter 2* of the *Advanced Users' Guide*, contained in the *Semper 6 Guide*.

Notes

see also: **type**

loop

loop [<*variable name*>]	You close a ***for*** loop with the ***loop*** command. A variable name with ***loop*** is optional.

You can make Semper repeat a group of commands by surrounding them with the commands **for...loop**.

Examples

```
for n=11,14; ps n to n+10
section; survey; loop
```

This sequence of commands produces in pictures 21 to 24 the rotational averages of the power spectra of pictures 11 to 14, and types their ranges.

```
for x 1,0, -0.25; type x; loop x
```

This sequence of commands types 1, 0.75, 0.5, 0.25 and 0 in turn on successive lines.

Description

The loop variable is local to its **for** loop, that is, its original state or value is restored when the loop ends. (See the **local** command for further detail). You can *nest* **for** loops up to an installation dependent maximum depth, which is typically 6. Each **for** command in a nested loop requires a corresponding **loop** command. Note that you can specify a variable name with **loop** to clarify your structure. For example:

```
for s=1,2
for n=5,8; type n; loop n
type 'who do we appreciate?...Semper 6!'
loop s
```

You can also re-use the same loop variable in each nested loop as each variable is *local* to its loop. To break out of a **for...loop** use the command **break** or **next**.

Notes

restrictions: a special restriction applies to **for** loops with respect to numbered macros (which will not be supported in later increases of Semper). Loops are not executed correctly in a numbered macro when the macro itself is called from within a library program.

see also: **break, for, local, next**

lorentzian

keys:	**[to]**	*<number>*	output picture
	size	*<x>, <y>, <z>*	dimensions of picture produced
	radius	*<number>*	width parameter for lorentzian function

Use **lorentzian** to generate a picture containing a single lorentzian-profile peak at the origin, for test or other purposes.

Examples

```
lorentzian size 120
```

This command replaces the current picture with a 120 square *Fp* picture with a lorentzian peak (half width at half height 16) at its origin.

```
lorentzian 2 size 40,40,40 radius 5
```

This command generates a 3-D lorentzian of *rms* width 5 pixels in picture 2.

Description

Use the **size** key to determine the size of the output picture generated by **lorentzian**. Use the **radius** key to specify the width parameter for the lorentzian function. For example, pixels distant *x* from the picture origin are set to:

$$\frac{r^2}{(r^2+x^2)}$$

where *r* is the radius specified with the **radius** key. If you specify a negative **radius**, the function $\frac{r^2}{x^2}$ (with central value zero) is stored instead - effectively a zero width lorentzian.

Note that the **gaussian** command generates a similar gaussian profile picture to that generated by **lorentzian**.

Notes

multi-layer pictures:	fully supported
forms used internally:	fp
see also:	**gaussian**

lorentzian

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in variable *select*	vaiid picture number
size	size 32, value held in *si2*, 1	less than or equal to the size of the picture (integers)
radius	size/8	real number

lpd

keys:	[from]	<*number*>	1-D source picture
	[to]	<*number*>	if **line**, output picture
	[plist]	<*number*>	if one or more peaks are detected, output picture containing list of peak parameters with same format as a particle parameter list
	baseline	<*number*>	threshold below which any source values are ignored
	height	<*number*>	threshold above which peak is accepted
	saturate	<*number*>	threshold above which minima are ignored
	area	<*number*>	threshold for integrated peak area
options:	**line**		output 1-D picture with peak maxima set to the integrated area with negative excursions outside the bounds of integration and zeros elsewhere

Use the **lpd** command to search for positive peaks in a 1-D picture and output a list of peak parameters. Optionally, you can also output a line spectrum with the peak maxima and bounds indicated. In all cases the Semper variable *n* is set to the number of peaks found.

Examples

```
lpd 1 plist 2
```

This command searches for peaks in picture 1 and outputs details as a Plist in picture 2. The variable *n* is set to number of peaks found.

```
lpd 1 line to 2 height 50 baseline 10
```

This command searches for peaks with a minimum height of 50 and outputs a line spectrum indicating their position and size. Values below the value 10 are assumed to be outside of any peak detected.

Description

The basic peak detection performed by **lpd** can be refined using several different criteria.

The key **baseline** is used to specify a minimum background level, below which value points are not considered to be part of a peak. This value should be set high enough to eliminate false peaks caused by baseline wobble, without eliminating too much of a detected peak. The default value used for **baseline** is 0 (which may be below the minimum value in the input spectrum!).

The key **height** is used to specify a minimum height which must be achieved by at least one point within the peak. The default value used is one third of (**baseline** – *minimum source value*). If this value is negative it is faulted.

lpd

The key **area** is used to specify a minimum area for peaks of interest. The default value is 0. This key can be used together with the **height** key to restrict detection to sharper peaks in the input spectrum.

The key **saturate** is used to determine a threshold for minima used to separate peaks. To be accepted as a minimium position, a value must lie below this threshold. The default value is the maximum value in the input spectrum (i.e. the feature is normally turned off). This is useful for saturated peaks that have small dips below the maximum value.

If the **line** option is given, the command will output a line spectrum consisting of a single point at the position of each peak maxima with a height equal to the integrated peak area. The start and end points of the peak are marked with a negative value.

The output Plist contains parameters for each peak. For consistency with the output of the **analyse** command the output is dimensioned for 25 parameters, but only 8 are supplied. The names and layer positions of the parameters set are shown below. The commands used for particle analysis after the **analyse** command, for example **ptype**, can also be used on the output of the **lpd** command.

Layer	Parameter	Parameter
1	xref	X position of maximum
3	id	peak number
7	contact	saturated peak flag (1 = saturated, 0 = not)
8	xmin	X position of left hand edge of peak
9	xmax	X position of right hand edge of peak
10	ymin	minimum value in peak
11	ymax	maximum value in peak
19	area	integrated peak area

Notes

variables set: *n*
see also: **analyse, peaks, ptype**

lpd

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
[plist]	999	valid picture number
baseline	0	positive real number
height	(baseline – min. source value)/3	positive real number
saturate	maximum source value	real number greater than **baseline** and **height**
area	0	postive real number

lsd

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	over	*<number>*	width of block/ strip evaluated around each pixel

Use **lsd** to calculate the local standard deviation (root mean square deviation) over a square block of neighbouring pixels around each source pixel. In effect, **lsd** tells you the brightness variation around each pixel.

Examples

```
lsd 1; display
```

This command presents bright pixels wherever the source varies strongly within a 5 square neighbourhood – this operation is quite a good smooth edge detector.

```
hp 1 to 2 over 20; lsd to 3 over 20; min=-2 max62; calc :2/:3 to dis
```

This command displays 1 with mean and standard deviation both standardised over 20 pixel blocks.

Description

Note that with **lsd**, execution time is more or less independent of the block size, so you can use very large block sizes if necessary. You use the **over** key for the size of block over which the local average is to be taken (default 5).

Edge pixels of the source, where the block averaged overflows the source, are processed as if the boundary values are repeated indefinitely outwards. If you specify an even value for the **over** key, the replaced source pixel is rounded to the bottom right from the block centre.

Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**imean, ivariance**

lsd

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
over	width/strip 5	positive integer

lset

keys:	**number**	*<number>*	look-up table number
	range	*<n1>, <n2>*	set the appearance of a specified range of display levels
	scaled	*<number>*	scale the ***range*** values using the black and white levels of the display picture specified by **scaled**
	brightness	*<n1>, <n2>*	specify the pixel brightness range
	hue	*<n1>, <n2>*	specify the range of hue
	saturation	*<n1>, <n2>*	specify the saturation value of pixels
options:	**all/red, green, blue**		if full colour lut, specify the colour channel to be modified

Use **lset** to set the contents of the current display output look-up table.

Examples

```
lset 2 brightness 0,1
```

This command sets look-up table 2 to a linear grey-scale ramp, ranging from black to white.

```
lset brightness 1 saturation 1 hue 0,360
```

This command sets the current (false-colour) look-up table to a spectrum of fully saturated colours from red (hue=0) through green (hue=120) and blue (hue=240) and back to red (hue=360).

```
lset brightness 1 saturation 1,0 hue 120
```

This command sets the current (false-colour) look-up table to shades of green, from fully saturated at the minimum display level, through pastel shades at mid-level, to zero saturation (white) at the maximum display level.

```
min=-1 max=1 create display size 256
lset brightness 1,0 range 0,0.1 scaled display
```

This sequence of commands sets the current look-up table to an inverted grey-scale ramp over the range 0 to 0.1 translated via the black and white levels in the display

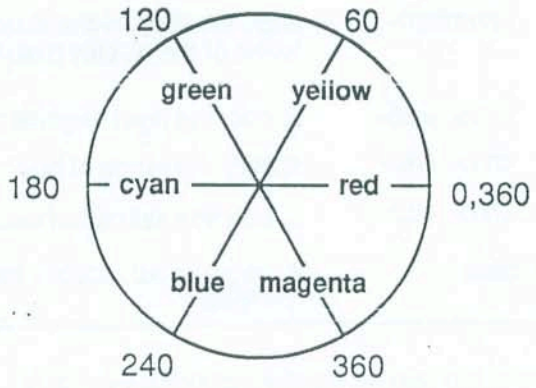
```
lset brightness -1,2 green red
```

This command sets the green and red components of the current (full-colour) look-up table to a high-contrast ramp. It leaves the blue component unchanged.

lset

Description

lset sets the contents of the current look-up table. If you are using a false-colour look-up table, you can use the keys **hue**, **brightness** and **saturation** to control the colour and pixel brightness. The **hue** values form a spectrum through the range 0 to 360, which is illustrated below.



Note that **hue** values are not restricted to the range 0 to 360. This allows you to specify colour scales that go through red, for example, 300 to 420 for magenta through red to yellow, or colour scales that cycle more than once through the colour spectrum, for example, 0 to 1000 goes over three times round the colour spectrum.

Use **lset...range** to specify the range of display levels that are to be set. The appearance of levels outside this range remains unchanged.

Notes

see also: **ladjust**, **lut**

Defaults and Ranges

keys/options	defaults	range
[number]	current look-up table number, held in the variable *clut*	valid look-up table number
range	total range of display levels supported in look-up table; often 0,255 or 0,127	range is hardware dependent
scaled	*none*	valid display picture
brightness	0,0	real number in range 0 to 1
hue	0,0	real number in range 0 to 360
saturation	0,0	real number in range 0 to 1

lut

keys:	**[number]**	*<number>*	look-up table number
	from/copy	*<number>*	copy new lut from specified picture or lut
	to	*<number>*	copy final lut to specified picture
	range	*<n1>, <n2>*	specify range of lut that is covered by ramp
	brightness	*<number>*	specify lut brightness
	contrast	*<number>*	specify lut contrast
	highlight	*<n1>, <n2>*	highlight a band of the lut
	scaled	*<number>*	display picture whose black/white scaling is assumed for ***range***, ***highlight*** values are supplied and returned
options:	**create**		create a new look-up table
	input		read lut from display hardware (at start of Semper session)
	delete		delete look-up table
	monochrome/false/colour		create monochrome, false or full-colour lut
	reset		reset lut (as if just ***created***)
	invert		invert lut (complementing or negating)
	zero		zero (clear) lut
	all/red,green,blue		modify specified lut colour(s) only
	keys		adjust lut interactively using keyboard
	enquire		return lut size and maximum output value

Use **lut** to generate, modify and store a look-up table (*lut*). Semper keeps a look-up table for use in your framestore hardware (if you have hardware luts). A look-up table controls how stored pixel values are presented on the display monitor, in terms of intensity or colour.

Examples

```
lut 5 create false
```

This command creates a look-up table number 5 as a false-colour lut.

```
lut 2 create range 50,200
```

This command creates a monochrome lut 2, black up to entry 50 and white above 200.

lut

```
lut zero red
```

This command zeros the red component of the current (false or full colour) lut.

```
lut from 51 keys
```

This command loads the current lut from a Semper *Lut* picture 51, and then allows interactive adjustment of the lut using the keyboard (type **?** for help). (Note that you can also use the Semper command **ladjust** for interactive lut adjustment using a mouse or cursor keys).

```
display 23; lut create false highlight 2.3,2.5 scaled display
```

This command displays picture 23, highlighting all values in the range 2.3–2.5 using white, with the range specified relative to the black/white intensity range of the current display picture.

Description

By default, **lut** works on the currently active look-up table. Use the command **type clut** to see the current lut number.

You can use the **lut** command in the following ways:

- to *specify* which look-up table to use
- to *modify* an existing look-up table
- to *store* a look-up table
- to delete a look-up table
- to find out about the hardware look-up table characteristics

To *specify* a look-up table, use the following keys and options:

lut n	operates on the existing look-up table *n*
lut n create	creates a new look-up table, according to the option **monochrome**, **false** or **full**
lut n copy c	copies existing lut *c* into lut *n*
lut n from p	copies from lut picture *p* into lut *n*
lut n input	reads a lut from the display hardware

To *modify* a look-up table, use the following keys and options:

lut...reset	resets a lut to the standard ramp
lut...invert	inverts the lut by reversing the minimum and maximum lut values
lut...zero	zeros the lut, making the picture invisible
lut...highlight h1, h2	forces lut entries in the range *h1–h2* to maximum brightness and reduces the rest of the table by a factor of 0.75 to highlight the band *h1–h2*

lut

lut...keys enters interactive mode in which you can alter lut range, brightness,contrast or highlight range using the keyboard (see *Further Information* for details). See also **ladjust**.

To *store* a look-up table, use the following keys and options:

lut stores the lut as software lut *n* and loads it into the display hardware if relevant

lut.. to p also stores a copy as a *Lut* picture p

To *delete* a look-up table, use the following option:

lut...delete deletes the look-up table

To *find out* about the size of look-up tables and the possible range of lut values, use the following:

lut...enquire sets the variables *lsize* and *lmax*

Setting look-up table parameters

By default, look-up tables are generated as simple ramps covering the input and output range appropriate to your display hardware. You can change a look-up table when you use **create** or **reset**, or when you are in the interactive **keys** mode, using the keys **range**, **brightness** and **contrast**.

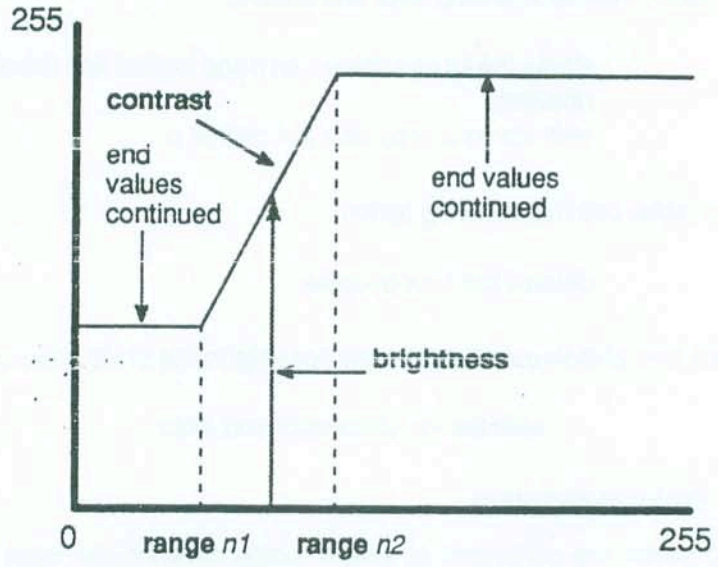
range specifies the minimum and maximum input values (horizontal range) over which the ramp extends, within hardware limitations. The end values of the ramp are repeated outwards as necessary. You can increase (but not reduce) the display contrast using **range**.

brightness specifies the output value (height) at the centre of the ramp. You can increase or reduce the display brightness using this key. Specify **brightness** as a real number in the range 0 to 1 (the default is 0.5).

contrast specifies the slope of the ramp, as a fraction of the maximum slope possible for the range and brightness values, that is, in the range –1 to 1.

lut

For example, the graph given below illustrates a hardware look-up table with 0-255 entries. **range** is set to 64, 140, **brightness** to 0.65 and **contrast** to 0.8.



Note that you can also specify **range** in terms of the pixel range scaled to the black and white levels of the display picture, so that the range is independent of the hardware's range of lut values. (Type **examine full** to see the original pixel range). Use the **scaled** key to do this, specifying the number of the display from which to take the scaling, for example:

```
min=0 max=1; lorentzian to display:4 size 200
lut range .2,.8 scaled display
```

False-colour look-up tables

False colour look-up tables are generated with an intensity and hue determined by what would be the output intensity for a monochrome lut. The hue varies smoothly from black through blue, red, magenta (red + blue), grey, green, cyan (green + yellow), and yellow (red + green) to white. You can stretch false colours scales using **range** in the same way as you can stretch a monochrome scale. However **brightness** and **contrast** do not affect them.

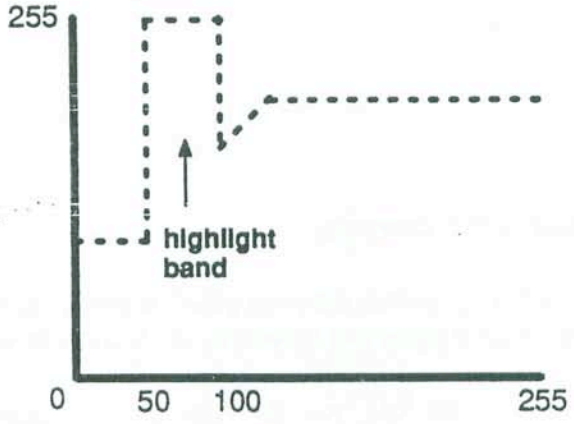
Full colour look-up tables

Full colour look-up tables are generated by treating each component as an independent monochrome lut. You can stretch or compress full colour scales using **range**, **brightness** and **contrast** in the same way as monochrome scales. By default, these change all three components identically, altering intensity and/or saturation levels rather than hue, but you can alter any combination of colour channels by quoting **red**, **green** and/or **blue**.

lut

Highlighting

You can also ask for a particular band of lut entries to be highlighted, that is, presented at maximum brightness or in a different colour – for example to allow you quickly to explore the effect of different threshold levels applied to a picture. Use the **highlight** key to specify the required band, for example, **highlight 50,100**. Note that lut values outside the band are reduced by a factor of 0.75 to ensure that highlighting stands out. For example, the diagram below shows the effect of the command **highlight 50, 100** on the previous look-up table graph:



Further Information

The interactive keystroke mode (**lut keys**) allows you access to most lut features instantly, though at present for monochrome luts only. (Note that the command **ladjust** allows you to change monochrome, false-colour and full colour lut parameters interactively using a mouse or cursor keys). Press **?** for a list of recognised keys and a summary of their functions, or *<return>* to leave the keystroke mode.

Broadly, you select a parameter to modify using the following keys:

b (brightness)
c (contrast)
r (range)
h (highlight)

Then you move the parameter up or down using the keys **>** and **<** (or other locally assigned keys, such as left and right arrows). The keys:

f (fine)
c (coarse)

allow you to control the step size. (Note that the **h** keys toggles, that is you can use it repeatedly to

lut

turn highlighting on and off alternately). For the range and highlight parameters, which involve two limits, you also select how you want to move them, using the following keys:

m (mean – both together)
s (separation)
u (upper only)
l (lower only)

For false and full colour luts, you can select one channel or all for modification using the following keys:

1 (red)
2 (green)
3 (blue)
a (all)

Note that you cannot select *pairs* of channels.

You can zero a channel in two ways: permanently using the key **z**(zero), or temporarily using the key **o** (the letter, not the digit), which toggles and allows you to clear and restore the lut alternately.

If you make mistakes and/or get lost, you can use the key **d**(discard) to discard any keystroke-controlled changes and recover the lut as it was originally generated.

The final brightness, contrast, range and highlight parameters are returned as Semper variables *b, c, r, r2* and *h, h2* for use in subsequent commands. If you are operating in the **scaled** mode, the values are inverse-scaled appropriately.

Notes

variables set: *lsize* (following **lut enquire**, lut size, that is, number of entries)
lmax (following **lut enquire**, maximum output value for lut)
b,c (following **lut keys**, final brightness, contrast parameters)
r, r2 (following **lut keys**, final range limits)
h, h2 (following **lut keys** with highlighting, final highlight limits)

see also: **ladjust**, **lset**

lut

Defaults and Ranges

keys/options	defaults	range
[number]	current look-up table number, held in the variable *clut*	valid look-up table number
from/copy	*none*	valid picture/lut number
to	*none*	valid picture number
range	full input range, usually 0–255	within hardware constraints
brightness	0.5	real number in the range 0 to 1
contrast	1	real number in the range –1 to 1
highlight	*none*	integer within range of lut entries
scaled	*none*	valid display picture number
monochrome/false /colour	monochrome	
all/red,green,blue	all three colour channels	

lvariance

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	over	*<number>*	width of block/ strip evaluated around each pixel

Use **lvariance** to calculate the local variance (mean square deviation) over a square block of neighbouring pixels around each source pixel. In effect, **lvariance** tells you the brightness variation around each pixel.

Examples

```
lvariance 1; display
```

This command presents a bright pixels wherever the source varies strongly within a 5 square neighbourhood – this operation is quite a good smooth edge detector.

Description

Note that with **lvariance**, execution time is more or less independent of the block size, so you can use very large block sizes if necessary. You use the **over** key for the size of block over which the local average is to be taken (default 5).

Edge pixels of the source, where the block averaged overflows the source, are processed as if the boundary values are repeated indefinitely outwards. If you specify an even value for the **over** key, the replaced source pixel is rounded to the bottom right from the block centre.

Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**lmean, lsd**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
over	width/strip 5	positive integer

macro

keys:	[ ]	*<number>*	store the rest of the input line as a numbered macro

The **macro** command causes Semper to store the rest of the input line as a macro. Macros are sequences of commands, command fragments or other text. To execute a macro, type @ followed by an identifying number.

Examples

```
macro 20; ask 'Picture: ' n; display n; ps full ln; display
@20
```

This sequence of commands defines and then executes macro 20.

```
macro 91; 'Subregion from 25 micron scan of stained specimen'
title 53 @91; title 57 @91
```

This sequence of commands defines macro 91 as containing text that is subsequently used for picture titles.

Description

Macros that are identified by number are user-defined macros, created using the **macro** command. Type **examine macros** to list all the macros on your current device and **list** to list the text of a specified macro. You can alter the text of a numbered macro using the **edit** command.

The following restrictions apply to the use of numbered macros within a program. (These restrictions do not apply to macros used directly in terminal input):

- macros that are executed within programs should not contain labels or **for** loops
- do not use the **macro** command within a program to (re-)define a macro

Macros that are identified by name, like *@xy* and *@region*, are built into the Semper command interpreter. Type **show macros** for a list of these standard macros.

Notes

see also: **edit, examine, list, show**

magnify

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	times	*<number>*	magnification factor
	size	*<x>, <y>*	dimensions of subregion to be magnified
	position	*<x>, <y>*	position/offset of subregion
	mark	*<number>*	mark source subregion
		<yes or no>	
options:	**repeating**		repeat pixels instead of interpolating
	left/right, bottom/top		magnify subregion at specified position

Use **magnify** to enlarge a picture or subregion by an integer factor. You can either interpolate or repeat source pixels. Repeating pixels emphasizes the effect of pixellation of an image.

Examples

```
magnify 4:23
```

This command replaces picture 4:23 with a double size version of itself.

```
xwires region; magnify @region times 5 to display
```

This command displays a region, marked with the cursor, magnified by 5.

```
magnify size 50 times 6 repeating
```

This command displays the central 50 square of the current picture, repeating each source pixel in a 6 by 6 block

Description

Use the **times** key to specify a magnification factor for an image. Use the standard subregion keys and options, **size**, **position**, **left/right**, **top/bottom** to specify a subregion for enlargement and the **mark** key to mark the subregion on the display. (For detail of the subregion keys and options and the **mark** key, refer to *Appendix C, Semper Keys and Options).* **magnify** interpolates pixels unless you specify the **repeating** option.

magnify

Note that the interpolation of pixel values can only be performed *between* source pixels. This means that the size of the output picture is:

$m*(n-1)+1$

where n is the source dimension and m is the magnification factor, rather than interpolation using $m*n$ as you might have expected.

The output origin is set so that it corresponds exactly to the source origin (unless the region that you magnify does not include the origin).

Notes

display marking: region, if subregion
multi-layer pictures: layers processed independently
forms used internally: fp, complex

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	source picture	valid picture number
times	times 2	positive integer
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0,0	within bounds of the picture (integers)
mark	mark off	see *Appendix C*

map

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	with	*<number>*	picture containing map function or histogram
	range	*<n1>, <n2>*	range of source picture spanned by map function
options:	**gaussian**		map source so that output is a histogram with a gaussian profile, if ***with*** specifies a histogram

Use **map** for arbitrary pixel value mapping, linear or non-linear, and for histogram equalisation or shape forcing.

Examples

```
map 1 to 2 with 3 range 1.2,3.5
```

This command replaces the pixel values for picture 1 with the values stored in the 1-D picture 3, over the range 1.2 to 3.5, and stores the mapped result as picture 2.

```
histogram 50 to 3; map 50 with 3
```

This command equalises the histogram of picture 50.

```
histogram 50 to 3; map 50 to display with 3 gaussian
```

This command displays picture 50 stretched so as to show a histogram with a gaussian profile.

Description

map works in the following two modes:

- arbitrary pixel value mapping
- histogram equalisation

Use the **with** key to specify a picture containing the mapping function or histogram.

Usually, you map using a tabulated mapping function that you provide directly as a 1-D picture. **map** treats the map pixels as defining replacement pixels over the source range *min* to *max*, using bilinear interpolation where necessary to extend these to form a continuous function. You can specify a range other than *min*, *max* using the **range** key.

Alternatively, you can map using a *Histogram* picture (produced from the picture you wish to map). In this case, **map** generates the actual mapping function itself in such a way as to equalise the

map

output picture histogram. This is a standard method of contrast-stretching in pictures with poor initial contrast. Since equalisation in fact often overdoes it, you may like to use the additional **gaussian** option, which produces an output histogram which is gaussian in shape, falling by a factor *exp(–2)* at both ends.

Note that finite numbers of distinct pixel values and/or histogram channels usually mean that the final histogram has the target shape only when averaged over a few neighbouring channels. The visual impact of this imperfection is however negligible.

Note that you can use the **xwires...graph** command to draw a map directly with the display cursor or use a display lut recovered using **lut**. *Byte* source pictures are mapped efficiently using a look-up-table approach.

Notes

multi-layer pictures: all layers processed
forms used internally: integer (for byte data), fp, complex

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	source picture	valid picture number
with	picture 999	valid picture number
range	values held in *min, max*	positive real numbers

mark

keys:	**[ ]**	*<number>*	mark specified picture/partition/frame
	text	*'<text>'*	mark specified text at ***position***
	size	*<x>, <y>*	dimensions of subregion to be marked
	to	*<x>, <y>*	mark line/arrow from ***position*** to given end point
	radius	*<number>*	mark circle/arc with specified radius and centre at ***position***
	with	*<number>*	mark positions/curve in specified *Plist* picture
	position	*<x>, <y>*	mark position in style determined by ***mkmode***, ***mksize*** unless ***text***, ***size***, ***to***, ***radius*** or ***with*** are set
	angle	*<x>, <y>*	mark arc with specified start and finish angles
	sampling	*<number>*	sampling factor for marked subregion
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**picture/partition/frame**		mark picture/partition/frame
	border		mark picture/partition/frame border
	line		mark a line from ***position*** to ***to***
	arrow		mark an arrow rather than a line
	circle		mark a circle with centre at ***position***, radius ***radius***
	arc		mark an arc over angular range ***angle***, in an anti-clockwise direction
	region		mark a subregion defined by subregion keys
	left/right, bottom/top		align specified subregion or text with picture/partition/frame border
	uv		mark subregion with sampling lattice defined by the base vectors *u,u2* and *v,v2*
	list/curve, open/closed		verify type of *Plist*, if ***with*** key used
	lj/rj		if ***text***, mark the text left-justified or right-justified at the specified position
	bj/tj		if ***text***, mark the text bottom-justified or top-justified at the specified position
	below/above		if ***text***, mark text centrally at top/bottom of picture
	outside/inside		if ***text*** and ***below/above***, mark text outside or inside picture border
	view		switch view to make the display region visible

mark

Use **mark** to mark positions, lines, arcs, etc. outline regions or write text in a display overlay.

Examples

```
mark @xy
```

This command marks a small cross on picture *display* at position *x,y*.

```
mark display:2 size 300,200 top left
```

This command outlines the indicated region of picture *display*:2.

```
mark partition display:2 border view
```

This command outlines the partition *display*:2, and shows it on the monitor.

```
mark radius 60 position 10,10 angle 0,pi/2
```

This command marks a quadrant of a circle upwards from 70,10 curving leftwards to 10,70.

```
mark text 'area b' bottom left
```

This command marks the text at the bottom left.

Description

With no options set, **mark** simply marks the origin of the current display picture. The type and size of mark is determined by the general keys **mkmode** and **mksize**. Use the general option **view** to ensure that what you mark is visible. For information on the general keys and options, refer to *Appendix C: Semper Keys and Options*.

By default, **mark** marks *display*, in picture coordinate mode. You can also use it in **partition** or **frame** mode. For example, **mark partition dis:7**. You specify positions using the standard subregion keys **position**, **left**, **right**, **top** and **bottom** etc.

Complex display pictures have their real and imaginary parts displayed side by side, and **mark** will mark both parts in the same way. You can restrict the marking to one part only using the general options **re** or **im**.

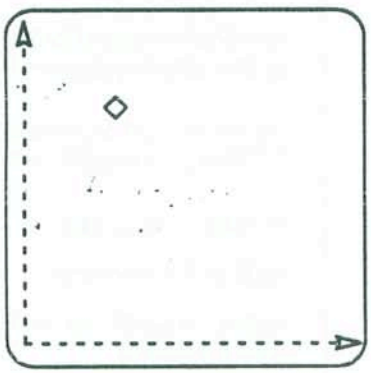
When annotating pictures or partitions, the annotation is output to all of the frames allocated to the partition. This means that annotation is correctly displayed on full colour framestores provided that the partitions have three successive frames allocated to them.

mark

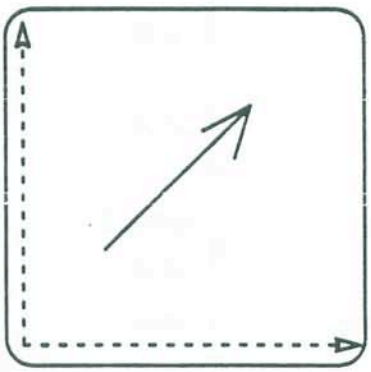
Note that the keys **line**, **circle**, **region** and **arc** are only included to make the command readable and are ignored by Semper. For example, the following commands perform identical functions:

```
mark arc angle 0, pi/4
mark angle 0, pi/4
```

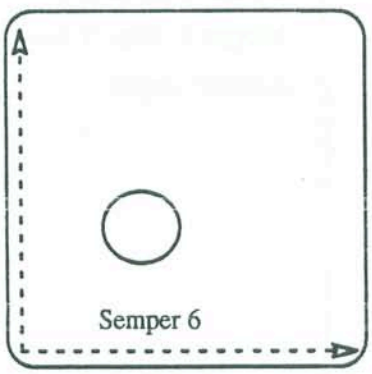
The following diagrams illustrate the effect of the **mark** command.



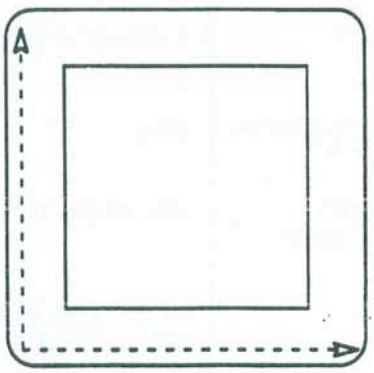
mark position 40,100 mkmode 4 mksize 2



mark position 40,40 to 100,100 arrow



mark position 50,50 radius 25 circle
mark text 'Semper 6' position 30,10



mark size 100,100 region

mark

Defaults and Ranges

keys/options	defaults	range
[ ]	*display* if picture or partition; current frame if frame, held in variable *cframe*	valid picture/partition/frame number
text	*none*	valid text string
size	*none*	less than or equal to the size of the picture (integers)
to	*none*	within bounds of picture/ partition/frame (integers)
radius	radius 2	positive real number
with	*none*	valid picture number
position	*none*	within bounds of picture (real numbers)
angle	*none*	real number in range 0 to 2π
sampling	factor of 1	positive real number
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
picture/partition/ frame	picture	
list/curve, open/closed	actual type of *Plist*	
lj/rj	left-justified, if ***left*** right-justified, if ***right***	
tj/bj	top-justified, if ***top*** bottom-justified, if ***bottom***	
outside/inside	text outside border	
view	view unchanged	

mask

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	radius	*<number>*	radius of circular mask
	position	*<x>, <y>*	centre position of circular mask
	width	*<number>*	width parameter for 'soft edge' of circular mask
	with	*<number>*	*Plist* picture containing mask boundary curve
	value	*<n1>, <n2>*	reset pixels to specified value
	mark	*<number>*	mark circle on display, or polygon if ***with***
		<yes or no>	
options:	**outside/inside**		reset pixels outside or inside the mask

Use **mask** to reset to a constant all pixels inside or outside a given subregion, for example, to eliminate unwanted features in an image, or to perform low or high pass filtering of a Fourier transform.

Examples

```
mask
```

This command resets all points more than two thirds of the way from the origin to the nearest picture edge, to the mean picture value at that radius.

```
xwires circle; mask inside @circle value 0
```

This command resets to zero all points inside a circle indicated using the cursor (**xwires** command).

```
mask outside radius 100 position 20,30 width 10
```

This command resets points more than 100 pixels from the point (20,30), fading smoothly over 10 or so pixels, to give a 'soft-edge' to the mask.

```
xwires curve to 51; mask with 51
```

This resets all points outside an arbitrary region indicated using the cursor.

mask

Description

mask operates in two modes, circular (the default) and polygonal (**mask..with** a *Plist*). It can reset pixels **outside** (the default) or **inside** the region in question. The default reset value is the source picture mean taken over the mask boundary, unless you force a particular value using the **value** key (which accepts a complex value for *Complex* pictures). For the circular mode only, you can include a 'soft edge' in the mask, with a gaussian profile extending outwards (or inwards if **inside**) from **radius**. Specifically, a pixel distance *r* from the centre of the circle, with initial value *p* is reset to *w.p + (1–w).***value**, where:

$$w = e^{-\left(\frac{(r-\mathbf{radius})^2}{\mathbf{width}^2}\right)}$$

Display marking includes a second circle 2***width** from the first when **width** is quoted.

If you indicate a display with the **mark** key, **mask** marks the circle on the display or marks a polygon if you are using the **with** key. For further detail of display marking, refer to *Appendix C, Semper Keys and Options.*

Notes

display marking:	mask boundary
multi-layer pictures:	layers processed independently
forms used internally:	integer, fp, complex

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	source picture	valid picture number
radius	two thirds of the minimum source direction	positive real number
position	centre of source picture	within bounds of picture (real numbers)
width	width 0	positive real numbers
with	circular mask	valid picture number
value	source picture average around mask boundary	real numbers
mark	mark off	see *Appendix C*
outside/inside	outside mask	

median

keys:	[from]	<number>	source picture
	[to]	<number>	output picture

The **median** command *smooths* binary pictures, removing isolated point and line objects and similar point and line holes within objects, at the same time.

Examples

```
median display
```

This command smooths objects in the current display picture.

```
median 50 to 51
```

This command smooths picture 50 and places the output in picture 51.

Description

The **median** command is part of the *morphology* group of Semper commands that are used to manipulate and measure 2-D shapes in binary pictures. The morphology commands include **analyse**, **erode**, **dilate** and **median**. (Note that you can make a binary picture using the **calculate** command, for example, **calculate :51>20** or **calculate :51>thr**).

Use the **median** command to remove noise from a picture by smoothing the picture. It resets each pixel to the median value of its 3x3 neighbourhood, retaining the original values when the 8 neighbours are evenly divided.

Notes

multi-layer pictures:	layers processed independently
forms used internally:	integer
see also:	**analyse**, **erode**, **dilate**,

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

menu

keys:	**id**	*<number>*	define which menu to use
	in	*<number>*	place menu in specified panel
	name	*'<text>'*	specify a name for the menu
	begins	*'<text>'*	specify the '*interaction begins*' action for a menu
	ends	*'<text>'*	specify the '*interaction ends*' action for a menu
	position	*<x>, <y>*	position a menu on its panel at the given coordinates
	size	*<x>, <y>*	define a maximum *x* and *y* size for a menu
	background	*<number>*	background colour of menu
	foreground	*<number>*	foreground colour of menu
options:	**create**		create a menu
	destroy		destroy a menu
	activate		activate a menu
	deactivate		deactivate a menu
	fixed/popup/pulldown		create a fixed/pop-up/pulldown menu
	choice/toggle		the menu is of style choice/toggle

The **menu** command controls the creation and operation of the Semper 6 *Plus* menu element. For a description of Semper 6 *Plus* elements, refer to the following manual:

Semper 6 Plus User Interface Guide

Examples

```
menu create name 'Display' fixed
```

This command creates a new fixed menu called *Display*.

```
menu activate id m35
```

This command displays the menu whose identifier is stored in the variable *m35*.

```
mouse query;menu id mnu position uix, uiy activate
```

This command finds the current position of the cursor, moves the menu so that it is positioned at the

menu

cursor position and displays the menu. The menu *mnu* must be a pop-up menu to be re-positioned in this way.

```
menu id m deactivate;menu id m name 'Next' activate
```

This command hides the menu , whose identifier is *m*, changes its name to *Next* and re-displays it. Note that this command only operates on pulldown and pop-up menus as fixed menus cannot be hidden and redisplayed.

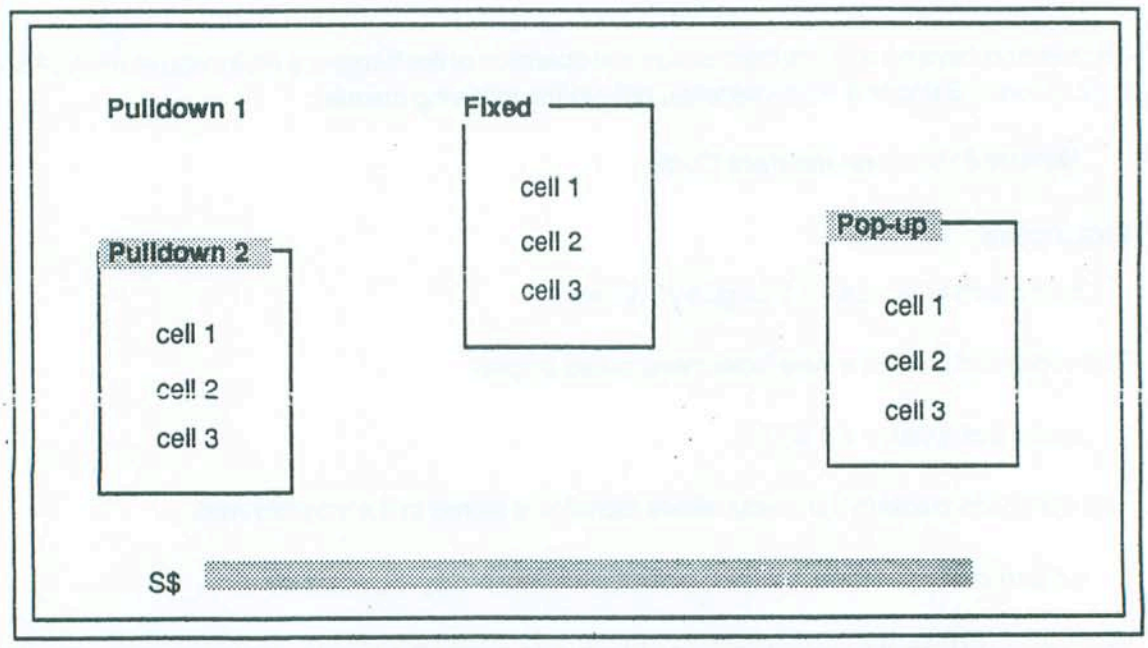
Description

A Semper 6 *Plus* menu can be created using the **create** option and removed using **destroy**. Use the **id** key to define a menu to act upon. Note that when you use the **destroy** option the specified menu and all the cells on that menu are destroyed. You do not have to destroy the cells on a menu individually.

There are three types of menu:

- **fixed**
- **popup**
- **pulldown**

The default menu type is **pulldown**. The diagram below illustrates the three types of Semper 6 *Plus* menu on a panel.



In the diagram above, **Pulldown 1** shows a deactivated pulldown menu and **Pulldown 2** illustrates an activated pulldown menu. A pop-up menu does not appear on the panel until it is activated.

menu

A fixed menu always appears on the panel. Note that pop-up menus belong to a panel like other elements, but may appear outside it. Both pop-up and pulldown menus take up a whole panel internally (and may be considered as special types of panel). This point is worth remembering if the system reports that all panels are in use when defining a user interface. (Use the **uif status** command to verify the state of the interface system).

There are two styles of menu:

- **choice**
- **toggle**

The default menu style is **choice**. On **choice** style menus the cells cycle through their highlight states as you move the mouse over them. On **toggle** type menus the cells do not cycle through their states until you select the cell by clicking on it.

Menus can be activated (selected) and deactivated. When pop-up and pulldown menus are activated they appear on the panel. To activate a menu use the **menu...activate** option. You can also activate a pulldown menu by clicking on its name (created using the **name** key).

To deactivate a menu, use the **menu...deactivate** command. To deactivate a pulldown or pop-up menu using the mouse either specify the **cell...drop** command (see **cell**), or program the deactivation directly. For example:

```
menu create pulldown
mid = eno
cell create add mid text 'function 1' drop row 1 column 1
c11 = eno
cell create add mid text 'function 2' drop row 2 column 1
c12 = eno
```

This program creates a menu with two elements on it. When either of the two cells *c11* or *c12* are selected, the menu is automatically deactivated.

A menu can go through the following transitional states:

- *interaction begins*
- *interaction ends*

You define the action taken for each of these states using the **begins** and **ends** keys. These keys allow you to specify as text the commands, keys and options to be executed by Semper when interaction with a menu begins or ends.

menu

The following table details how to begin and end interaction with a menu.

type	interaction begins	interaction ends
fixed	mouse enters menu	mouse leaves menu
pop-up	menu activated	menu deactivated
pulldown	menu activated	menu deactivated

You can define the size and position of a menu on the screen using the keys **size** and **position**. When you define the size of a menu, remember to include one character wide border to accommodate the box that is drawn around the menu. Using **position** on a pop-up menu when it is deactivated moves it to the required position before it appears. For example:

```
mouse query; menu id mid position mix,miy activate
```

causes the menu with id *mid* to appear at the current mouse position. This is the sort of action that can be assigned to a mouse button to cause pop-up menus to appear when a button is pressed. A similar action could be defined on a button to cause the menu to disappear. For example:

```
mouse query; menu deactivate
```

This command causes the menu under the mouse to be deactivated when the mouse button is pressed.

Use the keys **foreground** and **background** to change the foreground and background colours of a menu.

Notes

variables used: *eno (menu number,* if **id** not specified*)*
*pno (*panel number, if **in** not specified)
variables set: *eno (*set by the **create** option to the menu identifier number)
see also: **cell, mouse, ulf**

menu

Defaults and Ranges

keys/options	defaults	range
id	current menu, held in variable *eno*	valid menu number (positive integer)
in	current panel, held in variable *pno*	valid panel number (positive integer)
name	*none*	text string: length is machine dependent
begins	*none*	text string: length is machine dependent
ends	*none*	text string: length is machine dependent
position	position 0,0	within bounds of the panel (integer)
size	Semper 6 *Plus* determines appropriate size of menu	within bounds of the panel (integer)
background	background colour of panel	range is machine dependent (integer)
foreground	foreground colour of panel	range is machine dependent (integer)
fixed/popup/ pulldown	pulldown menu	
choice/toggle	choice style	

mindistance

keys:	**[from]**	*<number>*	source picture (1 or more layers)
	[to]	*<number>*	output picture (single layer)
	with	*<number>*	picture containing training statistics
	stride	*<number>*	sampling interval across and down the picture
	position	*<x>, <y>*	position, offset of subregion
	size	*<x>, <y>*	dimensions of subregion
	threshold	*<n1>...<n9>*	the threshold percentage to apply to each class
options:	**left/right, top/bottom**		position of subregion

The **mindistance** command classifies a picture using the *minimum distance to means* method. Use the **mindistance** command in *Remote Sensing* applications.

Examples

```
mindistance 1 2 with 3
```

This command classifies picture 1 using the training data contained in picture 3 and sends the output to picture 2.

```
mindistance 1 2 with 3 threshold 1, 2, 1
```

This command performs the same actions as in the above example and also applies thresholds of 1, 2 and 1 standard deviations to each of the classes. (Each threshold is the same for each layer of a class).

Description

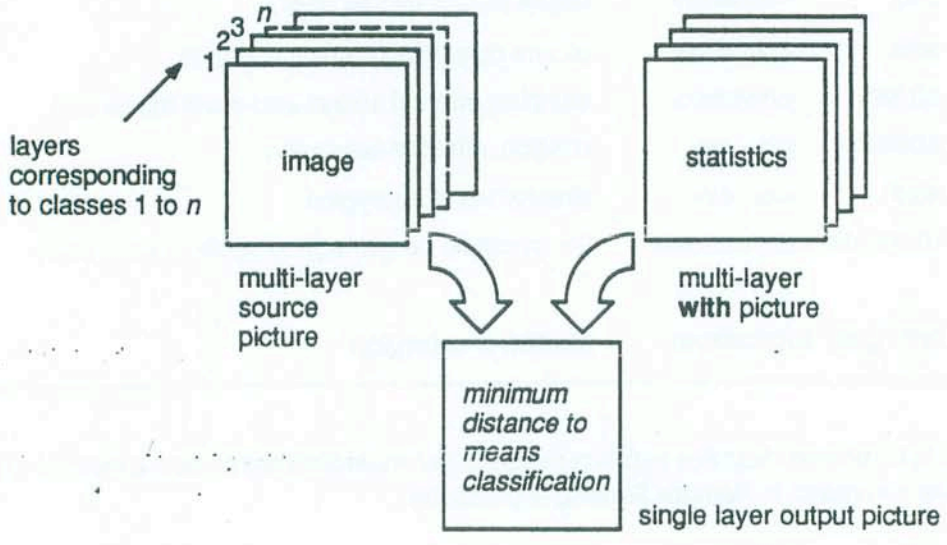
The **mindistance** command performs a *minimum distance to means* classification on the source picture using the training statistics given in the **with** picture. (The **with** picture is created using the **learn** command). The number of classes that the **mindistance** command can process depends upon the size of a Semper row buffer and on the number of layers in a source picture. Classes run from zero, which means unclassified, to *n*, where *n* is the number of layers in the source picture.

By default, no threshold values are applied and all pixels are classified. To apply thresholding, specify value(s) for the **threshold** key to denote the number of standard deviations from the means of each class. As with the **likelihood** command, if you specify one threshold it is applied to all classes.

You can use the standard subregion keys and options, **size**, **position**, **left/right** etc., to specify a subregion. See *Appendix C, Semper Keys and Options* for further detail.

mindistance

The following diagram illustrates the output of the **mindistance** command.



Use the **stride** key to perform a fast classification of a picture, without giving great detail. Specify the sampling interval across and down a picture with the **stride** key. The default interval is 1 so that the whole picture is sampled.

Notes

see also: **learn, likelihood**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none*	valid picture number
stride	stride 1	positive integer
position	position 0,0	within bounds of picture (integers)
size	whole picture	less than or equal to the size of the picture (integers)
threshold	all pixels are classified	integer in range 1 to 100

monitor

keys:	**channel** *<number>*	enable specified monitor channel (disable if negative number)
options:	**show**	print out monitor status
	on	turn monitoring on
	off	turn monitoring off
	all	enable all monitor channels
	none	disable all monitor channels
	rowio	enable monitoring of row input/output operations
	opens	enable monitoring of picture openings
	range	enable monitoring of calls to module RANGE
	processing	enable monitor output from new processing modules

Use the **monitor** command in installations where Semper has monitor support. Using this command, you can switch on and off the monitoring of system operation. Monitoring can be useful for debugging programs.

Note that the **monitor** facility is intended primarily for *Synoptics'* internal use. It is provided to customers only on an *as is* basis, in case it is found helpful, without any kind of warranty whatsoever.

Examples

```
monitor show
```

This command prints the current monitor status.

```
monitor rowio on
```

This command enables monitoring of picture row input/output operations and turns monitoring on.

```
monitor none range
monitor on; ..... ; monitor off
```

This command disables all monitoring except for **range** calls for a given series of commands.

```
monitor off
```

This command turns off all monitoring, but does not change the status of individual channels.

monitor

Description

Semper makes available the following named monitor channels:

No	Name	Description
1	*processing*	available for user-written extension routines
2	*rowio*	verifies calls to SEMROW (row input/output)
3	*opens*	verifies calls to SEMOPN/DEL (opens/deletes pictures)
4	*range*	verifies calls to RANGE, determines and records picture ranges

The options **on** and **off** switch all monitoring on and off. Printout usually goes to the terminal (to the monitor output stream in fact – see the **echo** command). All monitoring is turned off when Semper starts up.

Use the options **processing**, **rowio** etc. to enable or disable the monitoring of specific sets of information, for example, **monitor opens norange**. The options **all** and **none** turn all information on and off together. Use the **show** option to obtain a list of monitoring activities and monitor channels.

Additional monitor 'channels', numbered from 5 to 16, can be used by system or user routines. These are turned on or off using the **channel** key, for example, **monitor channel 5** to enable channel 5, or **monitor –8** to disable channel 8. Currently the following channels are in use:

5 verifies calls to routine DISC
6 verifies calls to routine TAPE
7 verifies calls to routine MCDC61

Note that you can add the following test to a Fortran routine that you are writing, to verify whether monitor output is to be generated:

```
IF (MONIT) THEN
    IF (SEMMON (TEXT, NAME, ICHAN) ) RETURN
ENDIF
```

The call to SEMMON passes the monitor message as a character string in TEXT, the name of the subroutine in NAME and the monitor channel number in ICHAN. SEMMON will output the monitor text if the specified monitor channel is enabled.

Defaults and Ranges

keys/options	defaults	range
channel	*none*	integer in range 1 to 16

motif

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	size	<x>, <y>	dimensions of output
	with	*<number>*	*Plist* picture containing list of positions to be averaged
	number	*<n1>, <n2>*	first and last position to be averaged
	mark	*<number>*	mark centre positions or, if ***regions***, mark region borders
		<yes *or* no>	
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**even/odd**		average even or odd numbered positions only
	regions		if ***mark***, mark regions rather than centre positions
	verify		print information on console about averaging process

Use **motif** to recover a clear *motif* from a picture containing many noisy instances of the motif. **motif** works by averaging regions centred at positions you have previously defined in a *Plist* picture.

Examples

```
motif to 10 size 60,40 with 3
```

This command produces a new picture 10 by averaging regions of the current picture around the positions listed in *Plist* 3. The dimensions of picture 10 are 60 points by 40.

```
motif with 3 numbers 1,80
```

This command averages only the first 80 listed positions.

```
xcf 1 with 2 to 3; survey full; peaks threshold 3*sd
motif 2 to 3 even; motif 2 to 4 odd
```

This command locates positions in picture 2 that match positions in a *reference* picture 1, (given suitable image sizes and noise levels). It produces two independent averages over the positions in pictures 3 and 4.

Description

motif averages source picture regions at each listed position and combines them in an average. Any regions that exceed the source picture bounds are omitted.

motif

Use the **with** key to specify the *Plist* containing the listed positions and use the **size** key to specify different dimensions for the output. The positions are rounded to integral values if necessary, and the number of averaged positions are stored in the variable *n*. Note that producing two independent averages using the **odd** and **even** options, as in the last command example, may help you distinguish signal from residual noise in the result.

If the positions are listed in a useful order (for example, the command **peaks** normally lists peak positions in height order), you may find it useful to restrict the average to the 'best' positions only, as in the second example.

If you set **mark**, the centres of the averaged regions are marked according to the general keys **mkmode** and **mksize** (see *Appendix C: Semper Keys and Options* for details). The **region** option outlines the boundary of a region instead of marking the region centre.

The **verify** option produces information about the averaging process on the console.

Notes

display marking:	averaged positions or borders of regions
multi-layer pictures:	faulted
forms used internally:	fp
variables set:	*n*, number of averaged regions

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
size	32 points square	positive integers
with	picture 999	valid picture number
number	all positions	within range of listed positions
mark	mark off	see *Appendix C*
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
even/odd	all positions	
verify	verification off	

mouse

keys:	**left**	*'<text>'*	define the actions to be taken for the left mouse button
	centre	*'<text>'*	define the actions to be taken for the centre button
	right	*'<text>'*	define the actions to be taken for the right button
	id	*<number>*	position the mouse on the specified object
	position	*<x>, <y>*	position the mouse pointer at a specified location relative to the top left of the display
options:	**query**		displays the current mouse position

The **mouse** command defines the actions and/or positions of a pointing device (mouse, graphics tablet, trackerball etc.) The word *mouse* is used for convenience. For a description of how you can use the **mouse** command together with Semper 6 *Plus* elements, refer to the manuals:

User Interface Guide
Tutor User Guide

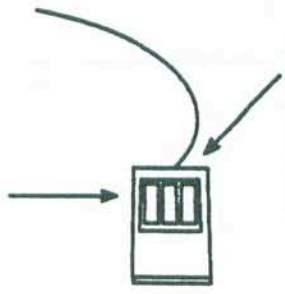
Examples

```
mouse right 'mouse query; type uix, uiy'
```

This command causes Semper 6 *Plus* to print the current position of the mouse each time you click the right hand mouse button.

Description

The **mouse** command assumes that a mouse has a maximum of three buttons, referred to by the keys **left**, **centre** and **right**. If you have a single button mouse, use the **left** key. If you have a two button mouse, use the **left** and **right** keys. The keys **left**, **centre** and **right** allow you to define the action to be taken when you press the appropriate button. You can specify any valid Semper command or sequence of commands as an action to be taken. To restore the default action, redefine the action by specifying the key and a null string (''). The following diagram shows the *default actions* for the mouse buttons when pressed:



Left: selects an element on the host display or framestore. If there are elements on both the displays at the selected position, the element on the *host screen* is selected.

Centre/Right: selects an element on the host display or framestore. If there are elements on both the displays at the selected position, the element on the *framestore* is selected.

mouse

Use the **position** key to move the mouse pointer to the specified position on the screen. Use the **id** key to position the cursor on an object (a panel, cell etc.). Note that when you specify the **id** key the cursor is positioned according to the justification that you used when creating the object (see the **justification** command).

Use the **query** option to display the current cursor position. This *x,y* position is held in the variables *uix* and *uiy*. The **query** option also sets the variables *eno* (current element number) and *pno* (current panel number) to the element and panel numbers at the cursor position.

If you use the command **device active** to limit interactions to a single display, then both the left and right buttons only select elements on the specified display. Use **device active 0** to reset the default settings.

Notes

variables set: *eno* (current element number set by the **query** option and **position** key)
uix, uiy (cursor x, y positions set by the **query** option)
pno (current panel number set by the **query** option and **position** key)

see also: **device**, **justification**

Defaults and Ranges

keys/options	defaults	range
left	select element on host display or framestore	valid Semper command
centre	select elements on host display or framestore	valid Semper command
right	select elements on host display or framestore	valid Semper command
id	*none*	positive integer
position	position 0,0	within bounds of the display (integers)

negate

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
options:	**preset**		rescale using existing values of *min, max*

Use **negate** to reverse the contrast of a picture, interchanging positive and negative forms. (If you just want to view a picture with reversed contrast, without changing its pixels, try the commands **display negated** or **lut invert** instead).

Examples

```
negate display to 20
```

This command places a reversed contrast version of **display** in picture 20.

```
negate 10
```

This command reverses the contrast of picture 10, retaining the same range.

```
negate 10 preset
```

This command rescales picture 10 so as to interchange the current values of *min,max*.

Description

negate reverses the contrast of a picture without altering the data range, that is, the minimum and maximum values present in the picture. It replaces source pixels *p* by an output value *p1*, where:

$$p1 = max + min - p$$

negate truncates byte values outside the range 0–255 to the nearer limit (like the command **scale**).

Use the **preset** option to rescale the picture using the existing values held in the variables *min* and *max*, rather than the actual minimum and maximum values found in the picture by **negate**.

negate

Notes

multi-layer pictures:	fully supported
forms used internally:	integer (for byte pictures), fp, complex
variables used:	*min, max* (if **preset**, values to be interchanged by scaling)
variables set:	*min, max* (unless **preset**, minimum and maximum pixel values)
see also:	**scale**, **display**, **lut**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

next

<variable name>	resume execution at the start of the next cycle in the named **for** loop

The **next** command causes Semper to resume execution at the start of the next cycle of the named **for** loop.

Examples

```
for n=1, 5; survey n; if sd<15 next n
type 'Standard deviation for picture ',n,' = ',sd; display n; loop
```

This sequence of commands reports and displays pictures with a standard deviation of 15 or greater.

Description

If you do not specify a loop variable name, the innermost active loop is assumed by Semper. For example:

```
for n=1,10; for m=n, 10; for x=-5,5; .. ; loop; next; loop; loop
```

in the above sequence the **next** command jumps to the next cycle of the *m* loop.

Notes

see also: **break**, **for**, **loop**

noise

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	width	*<number>*	width of noise distribution (standard deviation if gaussian or negative exponential noise, maximum if uniform)
	dose	*<number>*	number of quanta per pixel in poisson-noise simulation
options:	**uniform/exponential**		specify uniform or negative exponential noise distribution
	preset		if ***dose***, rescale using existing values of *mean*, *me2* instead of finding picture mean

noise adds pseudo-random noise to a picture, to simulate the noise that might be observed experimentally. Gaussian, poisson, uniform and negative exponential noise distributors are provided.

Examples

```
noise 1 to 3 width .5
```

This command adds gaussian distributed noise, with a standard deviation of 0.5, to picture 1, storing the result as picture 3.

```
noise 51 dose 8
```

This command replaces picture 51 by a poisson noise ('shot' noise) limited version, with an average of 8 quanta per pixel.

Description

By default, **noise** adds gaussian distributed noise with a standard deviation width of 0.1. You can also apply the following forms of noise distribution:

- poisson
- uniform
- negative exponential

Use the **dose** key to specify a poisson noise-limited version of the source picture. Semper rescales the output to have the same mean as the source (and therefore a similar range) . Use the **uniform** option to add uniformly distributed noise in the range 0 to **width**. If you use the **exponential** option it adds negative exponentially distributed noise with standard deviation **width**. This form of noise is typical of diffraction patterns, that is, power spectra.

noise

If you have a complex source picture, the real and imaginary parts are processed independently (with the same value of **width**). If you are adding poisson noise, the source picture must be non-negative as **noise...dose** faults any negative pixel values in the source picture. You can speed the process by using the **preset** option, if the mean of the picture is already recorded in the variables *mean* and *me2*. **preset** uses these mean values instead of scanning the picture directly.

The random number generator 'seed' used by **noise** is held in the Semper variable *rnm*, and you can initiate reproducible 'random' sequences of noise by setting this yourself.

Notes

multi-layer pictures:	all layers processed
forms used internally:	fp, complex
variables used:	*rnm* (random number generator 'seed') in the range 0 to 1
	mean, me2 (if **preset**, mean of picture)
variables set:	*mean, me2* (if not **preset**)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
width	0.1	real number in the range 0 to 1
dose	*gaussian* noise added	positive integer
uniform/ exponential	*gaussian* noise added	

ocf

keys:	**[from]**	*<number>*	first source picture
	with	*<number>*	second source picture
	to	*<number>*	output picture
	rings	*<number>*	number of ring sections that are compared
	radius	*<n1>, <n2>*	minimum, maximum radius of compared ring sections
	mark	*<number>*	mark rings on display
		<yes or no>	
options:	**full**		correlate full circular rings
	verify		prints information about the correlation process

Use **ocf** to determine the rotation that is necessary to bring two similar but misorientated pictures into register, or to detect rotational periodicity in a picture.

Examples

```
ocf 6 with 10 full
```

This command sets *theta* to the angle in radians by which picture 10 must be rotated clockwise for it to match picture 6.

```
ps 1 3; ps 2 4; ocf 3 with 4 verify
```

This command determines the misorientation of pictures 1 and 2, even if they are displaced laterally. It sends information about the process to the console.

```
ocf rings 3 radius 40,60 to 91
```

This command stores in picture 91 the auto-correlation function of the current picture with respect to orientation, calculated using samples taken on 3 rings with radii 40, 50 and 60.

Description

ocf rotates the two source pictures (**from** and **with**) about their origins, tabulating the cross-correlation coefficients between sets of samples. These sample sets are taken on 5 rings with radii stretching from one quarter to three quarters of the distance to the nearest picture edge. **ocf** returns the angle (in radians) giving the highest correlation level in the variable *theta*. If you specify an output picture as in the last command example, the correlation function is stored as a 1-D *Correlation* picture.

ocf

To register pictures that differ in position as well as orientation, you can eliminate the unknown displacements by considering power spectra (as in the second example) or auto-correlation functions, which are position independent. They are also centro-symmetric (implying an orientational ambiguity of 180 degrees). By default, **ocf** uses semi-circular rather than full circular ring samples . For registering other kinds of data (for example, pairs of images with an identifiable common point) use the **full** option.

A picture with fine detail has a spectrum that extends further out than one with largely coarse detail, and an auto-correlation function that shows the opposite behaviour. You select one or the other according to your application, and can also control the radii at which samples are taken using the **radius** key. If your pictures are noisy, you may need to increase the number of rings that **ocf** compares using the **rings** key.

If you specify a display with the **mark** key, **ocf** marks the circular ring samples on the display. For details of the **mark** key, refer to *Appendix C, Semper Keys and Options.*

Use the **verify** option to send information about the correlation process to the console.

Notes

display marking:	sample rings used
multi-layer pictures:	faulted
forms used internally:	fp, complex
variables set:	*theta* (angle, in radians, by which picture must be rotated clockwise to match the source picture)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
with	*none*	valid picture number
to	*none*	valid picture number
rings	5 ring sections	positive integer
radius	one and three quarters of the distance to the nearest picture edge	real number
mark	mark off	see *Appendix C*
full	semi-circular rings	
verify	verification off	

order

keys:	[]	*<n1>, <n2>...<n9>*	device numbers in new search order
options:	**verify**		verify the search order at the console

Use **order** to change the order in which Semper searches devices.

Examples

```
order
```

This command verifies the default device search order at the console.

```
order 2
```

This command moves device 2 to the front of the search order list, leaving the others unaffected. It verifies the new search order at the console.

```
order 2, 5, 3 noverify
```

This command moves devices 2, 5 and 3 (in that order) to the front of the list, leaving any others unaffected. If all devices are listed, this is equivalent to specifying the final search order. The new order is not printed at the console, because of the use of the **noverify** option.

Description

Semper maintains a default search order amongst the various assigned program libraries, at any given time. Use the **order** command without arguments to list the current search order. Libraries are placed at the front of this list when assigned, otherwise the search order is only altered by explicitly specifying an order using the **order** command.

Defaults and Ranges

keys/options	defaults	range
[]	current search order	integer in range 1 to system limits (type **show system**)
verify	verification on	

origin

keys:	**[ ]**	*<number>*	source picture
	position	*<x>, <y>, <z>*	position of new origin relative to current origin or to the position indicated by subregion options
options:	**left/right, bottom/top, near/far**		move origin to specified edge
	reset		move origin to default position

Use **origin** to move the coordinate origin of a picture.

Examples

```
origin top left
```

This command moves the origin of the current picture to the top left.

```
display 51; xwires; origin 51 @xy
```

This command moves an origin for picture 51 to a position specified using the cursor.

```
origin 4:23 reset
```

This command resets the origin of picture 4023 to its default position. The default origin of a picture depends on the *class* of a picture.

Description

Use the standard 3-D subregion keys and options to specify the origin position, assuming a **size** of 1. Refer to *Appendix C: Semper Keys and Options* for further detail.

The default position of the origin is at the centre of a picture, for most classes of picture. For pictures of class *Histogram*, *Plist* and *Lut* the default position is at the top left of the first (back) layer. Note that halfplane Fourier class pictures have their origin at centre-left, so you should use the command **origin left** to restore the default position for the origin, rather than **origin reset**, which would be suitable for fullplane Fourier pictures. For further detail of picture coordinates and picture origins refer to the *System Users' Guide* in the *Semper 6 Guide*.

The command **examine full** lists the **row, column** and **layer** position of the coordinate origin of a picture.

origin

Notes

see also: **examine full**

Defaults and Ranges

keys/options	defaults	range
[ ]	current picture, held in the variable *select*	valid picture number
position	position 0, 0, 0	within bounds of picture (integers)

output

This syntax is specific to...
IBM PC XT or AT compatible

keys:	**[from]**	*<number>*	source picture
	name	*'<text>'*	output file name
options:	**cut/dump/paint/tiff/raw**		output format (the default is to write a Semper 6 data file
	cut		write a Media Cybernetics CUT file
	dump		write a Micro Semper 1 dump file
	paint		write a greyscale Paintbrush PCX file
	tiff		write a greyscale TIFF file
	raw		write a raw binary data file
	new		replace the output file if it already exists
	unlabelled		if default output format, do not include the picture label information
	swap		if **raw**, write data in *Motorala* instead of *Intel* byte ordering
	byte/integer/fp/complex		if **raw**, output data format
	byte		write image data as bytes
	integer		write image data as 16-bit integers
	fp		write image data as floating-point values
	complex		write image data as complex values

Use **output** to output pictures in a fast binary form or for transfer to MicroSemper 1 and Figment 1 systems. You can also write images in Media Cybernetics CUT file format, in greyscale Paintbrush PCX format and in greyscale TIFF format.

Examples

```
output 23 name 'binary' unlabelled
```

This command writes picture 23 to the file *binary.pic* excluding the picture label.

```
output 17 dump name 'tosem1'
```

This command writes picture 17 to the MicroSemper 1 file *tosem1.dum*

output

```
output 11 new cut name 'export'
```

This command writes picture 11 to CUT format file *export.cut*, and the picture label to the file *export.lab*, overwriting any existing fiel of the same name.

```
output 12 raw name 'rawdata'
```

This command writes picture 12 to the file *rawdata.bin* as a byte stream.

```
output 2:4 paint name 'cells'
```

This command writes picture 4 on device 2 as a greyscale Paintbrush PCX file named *cells.pcx.*

```
output 42 tiff name 'image'
```

This command writes picture 42 as a greyscale TIFF file named *image.tif.*

Description

Use the **output** command to write a Semper 6 data file, unless you specify the option **cut, dump, paint, tiff** or **raw**. You specify the output file name with the **name** key. Note that **output** will only overwrite an existing file of the same name if you specify the **new** option.

The default file format contains exactly the same information as files handled by the **read** and **write** commands, but in raw binary form. If you specify the **unlabelled** option, the picture label information is not included in the output file.

If you specify the **raw** option, Semper produces a raw binary file. A raw data file is a byte stream starting with the top left pixel of layer 1, outputting along each row. Pixels are written to a raw data file using the form of the output picture, unless you specify one of the data form options **byte, integer, fp** or **complex.** Note that for compatability with other Semper systems, **integer** data is written as 16 bits (**not** 32 bits). This may cause data truncation if data is outside of the range –32767 to + 32767.

By default, the bytes are ordered in the Semper 6 data file so that the least significant byte appears at the first (lowest) address (*Intel* format). You can also write raw data files using *Motorola* byte ordering, if you specify the **swap** option in conjunction with **raw**.

If you specify the **cut** option, Semper produces a Media Cybernetics CUT file and a Semper LAB file. You can suppress the output of the LAB file by specifying the **unlabelled** option.

If you specify the **dump** option, a MicroSemper 1 dump file is produced.

If you specify the **paint** option, a greyscale Paintbrush PCX file is produced.

output

If you specify the **tiff** option, a greyscale TIFF (Tagged Image File Format) file is written, with a depth of 8 bits per pixel, black-to-white photometric interpretation and using no compression. The data is written contiguously so that it can be read by simple TIFF readers.

The following default file extensions are provided:

Semper 6 data file	*.pic*
raw binary file	*.bin*
MicroSemper 1 dump file	*.dum*
Media Cybernetics CUT file	*.cut*
Semper LAB file	*.lab*
Paintbrush PCX file	*.pcx*
TIFF file	*.tif*

You can recreate pictures that are produced by **output**, using the Semper command **input**.

Notes

see also: **input, read, write**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts for file name if interactive	valid file name
cut/dump/paint/ tiff/raw	write a Semper 6 data file	
new	if the output file already exists, output an error message	
swap	if **raw**, write data assuming Intel byte ordering	
byte/integer/fp/ complex	if **raw**, write data using the form of the output picture	

output

This syntax is specific to...
Sprynt systems and workstations running Unix

keys:	**[from]**	*<number>*	source picture
	name	*'<text>'*	output file name
options:	**raw/raster/tiff**		output format (the default is to write a Semper 6 data file)
	raw		write a raw binary data file
	raster		write a Sun Raster file
	tiff		write a greyscale TIFF file
	new		overwrite the output file if it already exists
	unlabelled		omit the picture label information from a Semper 6 data file
	swap		if **raw**, write data in *Motorola* instead of *Intel* byte ordering
	byte/integer/fp/complex		if **raw**, output data format
	byte		write image data as bytes
	integer		write image data as 16-bit integers
	fp		write image data as floating-point values
	complex		write image data as complex values

Use **output** to output pictures in one of a number of fast binary forms.

Examples

```
output 23 name 'binary' unlabelled
```

This command writes picture 23 to the file *binary.pic* excluding the picture label.

```
output 12 raw name 'rawdata'
```

This command writes picture 12 to the file *rawdata.bin* as a byte stream.

```
output 14 raw name 'local.dmp' swap
```

This command writes picture 14 to the file *local.dmp* using *Motorola* byte ordering.

output

```
output 303 raster name 'screen'
```

This command writes picture 303 as a Sun Raster file named *screen.rff.*

```
output 42 tiff name 'image'
```

This command writes picture 42 as a greyscale TIFF file named *image.tif.*

Description

Use the **output** command to write a Semper 6 data file, unless you specify the **raw** or **raster** options. You specify the output file name with the **name** key. Note that **output** will only overwrite an existing file of the same name if you specify the **new** option.

The default file format contains exactly the same information as files handled by the **read** and **write** commands, but in raw binary form. If you specify the **unlabelled** option, the picture label information is not included in the output file.

If you specify the **raw** option, Semper produces a raw binary file. A raw data file is a byte stream starting with the top left pixel of layer 1, outputting along each row. Pixels are written to a raw data file using the form of the output picture, unless you specify one of the data form options **byte**, **integer**, **fp** or **complex.** Note that for compatability with other Semper systems, **integer** data is written as 16 bits (**not** 32 bits). This may cause data truncation if data is outside of the range –32767 to + 32767.

By default, the bytes are ordered in the Semper 6 data file so that the least significant byte appears at the first (lowest) address (*Intel* format). You can also write raw data files using *Motorola* byte ordering, if you specify the **swap** option in conjunction with **raw**.

If you specify the **raster** option, Semper produces a Sun Raster file. Currently these are always written with a depth of 8 bits per pixel, using standard packing (type 1 – RT_STANDARD) and with no colormap information (RMT_NONE).

If you specify the **tiff** option, a greyscale TIFF (Tagged Image File Format) file is written, with a depth of 8 bits per pixel, black-to-white photometric interpretation and using no compression. The data is written contiguously so that it can be read by simple TIFF readers.

The default extension for Semper 6 files is *.pic*, for raw binary files it is *.bin*, for Sun Raster files it is *.rff*, and for TIFF files it is *.tif.*

Files written using the **output** command on a PC can be read into a UNIX workstation, linked over a *PC-NFS* type network. You can recreate pictures that are produced by **output**, using the Semper command **input.**

output

Defaults and Ranges

Notes

see also: **Input, read, write**

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts for file name if interactive	valid file name
raw/raster/tiff	write a Semper 6 data file	
new	if the output file already exists, output an error message	
swap	if **raw**, write data assuming Intel byte ordering	
byte/integer/fp/ complex	if **raw**, write data using the form of the output picture	

overlay

This syntax is specific to...
workstations running X Windows/DECwindows

keys:	**rgb**	*<n1>, <n2>, <n3>*	specify colour values in percentage terms
options:	**black/red/green/blue/cyan/ magenta/yellow/white**		specify colour for overlay plane
	on/off		specify overlay visible/invisible

The **overlay** command allows you to specify the colour displayed where the overlay is present.

Examples

```
overlay red
```

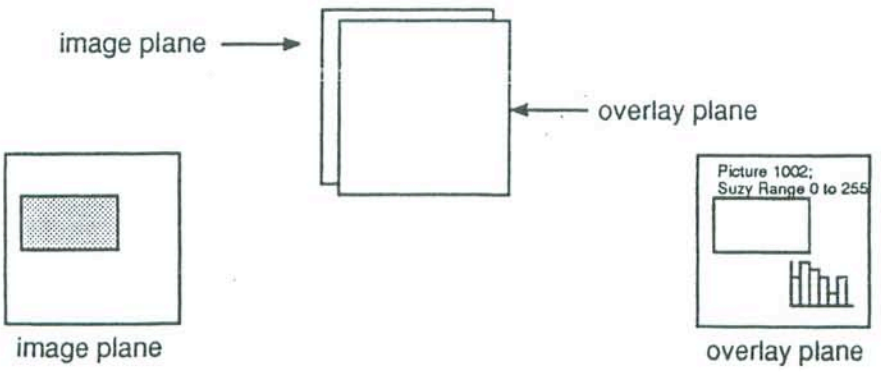
This command makes the overlay appear red (with maximum intensity).

```
overlay rgb 80,20,10
```

This command shows the overlay in a colour generated from 80% of maximum red, 20% of maximum green and 10% of maximum blue intensity.

Description

The display server must be a 6-plane (64 colour) or 8-plane (256 colour) workstation. Note that several of the possible colour values are reserved for the window manager desk top but that Semper uses the majority of the remaining colours. In 6-bit visual systems, two colours are reserved for the menu window and 40 for the image display. In 8-bit visual systems, eight colours are reserved for the menu window and 232 for the image display. Both systems also reserve one colour for the overlay. The overlay colour is used for all annotation and graphical output, for example, titles, borders, histograms, ymodulus plots, as illustrated below.



overlay

You can specify a colour for an overlay plane in two ways:

- use the **rgb** key to specify a colour composed of red, green and blue components. For example, the command **rgb 10, 60, 35** defines a colour composed of 10% of maximum red, 60% of maximum green and 30% of maximum blue intensity.
- use the options **black/red/green/blue/cyan/magenta/yellow/white** to define a specific colour.

You can also specify whether the overlay is visible or invisible (irrespective of the overlay colour) using the options **on** and **off**. This is achieved by rewriting the entire display image, and so may take a noticable amount of time on slower workstations.

Note that as Semper maintains a separate copy of the image contents, the overlay does not over-write the image and so the display can be used as an image source.

Notes

see also: **x11**

Defaults and Ranges

keys/options	defaults	range
rgb	*none*	real number expressing a percentage

overlay

This syntax is specific to...
Silicon Graphics workstations and workstations running X

keys:	**[number]**	*<number>*	overlay number
	rgb	*<n1>, <n2>, <n3>*	red, green and blue percentages of overlay colour
	hsv	*<n1>, <n2>, <n3>*	hue (0 to 360), saturation (percentage) and brightness (percentage) defining overlay colour
options:	**black/red/green/blue/cyan/ magenta/yellow/white**		overlay colour
	on/off		make the overlay visible/transparent
	graphics/rubberband/cursor		make the overlay the current graphics, rubberband or cursor overlay
	erase		clear the overlay
	show		list the current overlay settings

You use the **overlay** command to control the colour and visibility of the display window's eight overlays. Display annotation or graphics is directed to the current graphics overlay, rubberband lines are displayed in the rubberband overlay and the cursor is displayed in the cursor overlay. Overlays can be turned on or off without affecting the data stored in the overlay bitplanes. Where overlay information is overlapped, the highest numbered overlay is displayed.

Examples

```
overlay show
```

This command lists the current overlay settings.

```
overlay cyan
```

This command sets the colour of overlay 1 to cyan.

```
overlay 3 hsv 60,50,100 cursor
```

This command sets the colour of overlay 3 to a desaturated yellow and makes this the cursor overlay.

```
overlay 2 off
```

This command turns off overlay 2.

overlay

```
overlay 5 erase
```

This command clears overlay 5.

Description

The display window has eight overlay bit-planes numbered from 1 to 8. You use the **number** key to specify the overlay number. Note that the **number** key always defaults to 1.

The **show** option causes all the current overlay settings to be listed on the console.

When the display window is created, overlays 1, 7 and 8 are selected as the graphics, rubberband and cursor overlays, all overlays are turned on and cleared and the colours for overlays 1 to 8 are respectively set to white, red, green, blue, cyan, magenta, yellow and red.

The **overlay** command only changes the overlay settings specified in the command. The remaining overlay settings are left unchanged.

The overlay colour can be specified in one of three ways. If you need to specify the colour exactly, you can specify the RGB components of the colour as percentages with the **rgb** key, or the hue, saturation and brightness of the colour with the **hsv** key. The saturation and brightness values must also be specified as percentages. If the colour you want is one of the eight primary or secondary colours, you can use the corresponding colour option as a convenient alternative. You may not specify more than one of the colour options **black**, **white**, **red**, **green**, **blue**, **cyan**, **magenta** and **yellow** and the **rgb** and **hsv** keys in the same command.

An overlay can be turned on or off (made visible or invisible/transparent) by specifying the option **on** or **off**.

You may use one of the options **graphics**, **rubberband** or **cursor** to make the specified overlay the current graphics, rubberband or cursor overlay. You are not allowed to specify more than one function for a given overlay. For example, the following would not be allowed:

```
overlay 2 graphics; overlay 2 rubberband
```

All display annotation is directed into the current graphics overlay. Note that the **erase overlay** command erases only the current graphics overlay and leaves the other seven overlays unchanged. You can use the **erase** option with the **overlay** command to clear a specified overlay. To clear all the overlays, you should enter the following string of commands:

```
for i=1,8; overlay n erase; loop n
```

The cursor and any rubberband lines or boxes (see the commands **xwires**, **sketch**, **drag** and **pdraw**) are displayed by overwriting and erasing the data in the corresponding overlays. You are recommended to retain the highest numbered overlay for displaying the cursor so that it is not obscured by any of the other overlays. Beware also of turning off the cursor or rubberband overlays.

overlay

Notes

see also: **assign display, drag, erase, mark, sketch, xwires**

Defaults and Ranges

keys/options	defaults	range
[number]	1	integer in range 1 to 8
rgb	*none*	real numbers expressing a percentage
hsv	*none*	hue – real number in range 0 to 360 saturation – real number expressing a percentage brightness – real number expressing a percentage

overlay

This command is specific to...
Silicon Graphics IRIS 4D workstation
Sun 3 workstation
DEC VAXstation II GPX
PC + framestore/greystore

This syntax is specific to...
PC + Data Translation DT2861 framestore
PC + Matrox PIP512/PIP1024 framestore
Silicon Graphics IRIS 4D workstation
DEC VAXstation II GPX

keys:	**rgb**	*<n1>, <n2>, <n3>*	specify colour values in percentage terms
options:	**black/red/green/blue/cyan/ magenta/yellow/white**		specify colour for overlay plane

This syntax is specific to...
PC + Imaging Technology PCVISIONplus framestore

keys:	**rgb**	*<n1>, <n2>, <n3>*	specify colour values in percentage terms
options:	**black/red/green/blue/cyan/ magenta/yellow/white**		specify colour for overlay plane
	on/off		make the overlay plane visible/transparent

This syntax is specific to...
PC + MRC500 framestore
PC + Synoptics Synergy framestore
PC + Sun 3 workstation

keys:	**rgb**	*<n1>, <n2>, <n3>*	specify colour values in percentage terms
options:	**black/red/green/blue/cyan/ magenta/yellow/white**		specify colour for overlay plane
	cursor/text		specify the cursor or text overlay plane

overlay

This syntax is specific to...
PC + Synoptics Synapse framestore
PC + Metrabyte Corporation MV1 framestore

keys:	**rgb**	*<n1>, <n2>, <n3>*	specify colour values in percentage terms
options:	**black/red/green/blue/cyan/ magenta/yellow/white**		specify colour for overlay plane
	on/off		make the overlay plane visible/transparent
	cursor/text		specify the cursor or text overlay plane

This syntax is specific to...
PC + Quantimet 520 greystore

options:	**on/off**	make the overlay plane visible/transparent
	cursor/text	specify the cursor or text overlay plane

Use **overlay** to specify the colour displayed for each overlay plane, as some installations have both text and cursor overlays. If neither plane is visible, the image shows through. (Note that if you have a *Quantimet 520 Image Analysis System* the **overlay** command does not affect the display of colours. It can be used, however, to make overlay planes visible or transparent).

Examples

Silicon Graphics IRIS 4D workstation, Sun 3 workstation, DEC VAXstation II GPX, PC + framestore (excluding Quantimet 520)

```
overlay red
```

This command makes both overlays appear red (with maximum intensity).

```
overlay rgb 80,20,10
```

This command gives both overlays a colour generated from 80% of maximum red, 20%of maximum green and 10% of maximum blue intensity.

overlay

PC + Imaging Technology PCVISIONplus framestore only

```
overlay off
```

This command makes the overlay transparent.

PC + MRC500 and Synergy framestores/ Sun 3 workstation only

```
overlay cursor green
```

This command makes the cursor overlay appear green but does not affect the text overlay.

PC + Synoptics Synapse and Metrabyte Corporation MV1 framestores only

```
overlay cursor on
```

This command makes the cursor overlay visible.

PC + Quantimet 520 greystore

```
overlay off
```

This command makes both overlays (cursor and text) transparent.

```
overlay cursor on
```

This command makes the cursor overlay visible.

overlay

Description

Note that the following description is relevant for all installations except *Quantimet 520 Image Analysis Systems* (see the separate description given overleaf).

You can specify a colour for an overlay plane in two ways:

- use the **rgb** key to specify a colour composed of red, green and blue components. For example, the command **rgb 10, 60, 35** defines a colour composed of 10% of maximum red, 60% of maximum green and 30% of maximum blue intensity.
- use the options **black/red/green/blue/cyan/magenta/white** to define a specific colour.

The following installations have a single overlay plane, using one bit of annotation per image pixel:

- PC + *Data Translation* DT2861 framestore
- PC + *Imaging Technology* PCVISIONplus framestore
- PC + *Matrox* PIP512/PIP1024 framestore
- *Silicon Graphics* IRIS 4D workstation
- *DEC* VAXstationII GPX display + UIS window manager

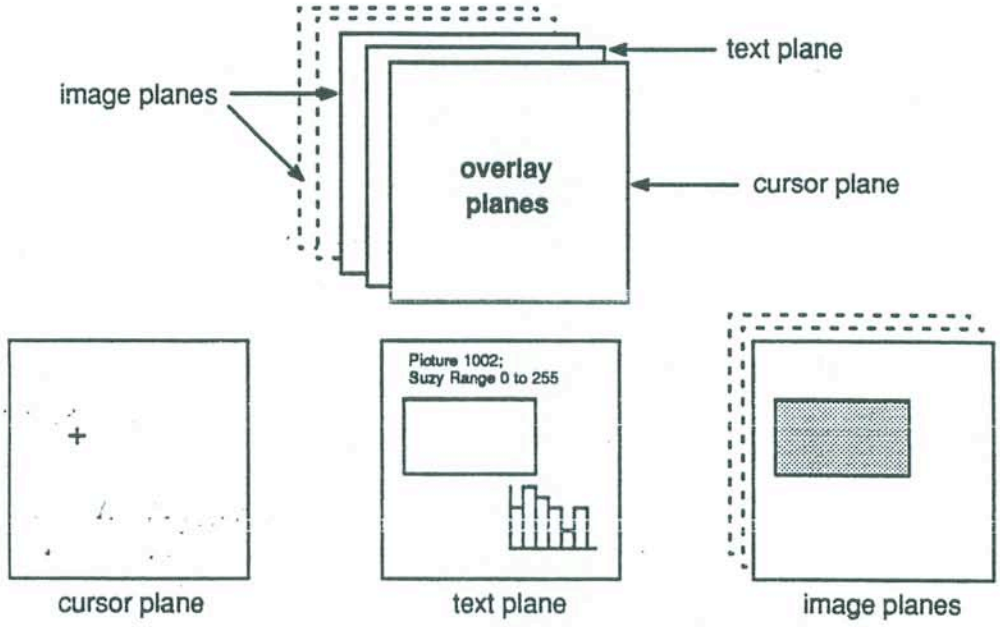
You can specify the colour for the overlay plane using the colour keys and options given above. Note that if you have a *Imaging Technology* PCVISIONplus framestore you can also use the options **on/off**. The option **off** makes the overlay plane transparent (as opposed to erasing it) and the option **on** makes the overlay plane visible again.

The installations given below have two overlay planes, using two bits of annotation per image pixel:

- MRC500 framestore
- *Synoptics* Synapse framestore
- *Synoptics* Synergy framestore
- *Metrabyte Corporation* MV1 framestore
- *Sun3* workstation + SUNVIEW window manager

Semper uses one of these overlay planes to hold the **xwires** cursor as it is being moved about. It uses the text overlay to hold all other graphical information such as text, picture borders, **histogram** and **ymod** plots. Use the options **cursor/text** to specify an overlay plane. If you have a *Synoptics* Synapse framestore or a *Metrabyte Corporation* MV1 framestore you can also make an overlay transparent using the **off** option and make it visible using the **on** option. The diagram overleaf illustrates the different overlay planes.

overlay



PC + Quantimet 520 greystore only

The Quantimet 520 greystore has two overlay planes, providing two bits of annotation per image pixel. Semper uses one of these, the **cursor** overlay, to hold the **xwires** cursor as it is moved about the screen. It uses the other plane, the **text** overlay, to hold all other graphical information such as text, picture borders, **histogram** and **ymod** plots.

The option **off** makes the overlay(s) transparent (as opposed to erasing them). The option **on** makes the overlay(s) visible again. Use the **text** and **cursor** options to specify which overlay plane is to be affected. If you do not specify a plane, Semper assumes both.

Defaults and Ranges

keys/options	defaults	range
rgb	*none*	real number expressing a percentage
cursor/text	Semper assumes both planes	
on/off	*none*	

ovread

keys:	[]	*<number>*	read back the overlay of the specified display picture, partition or frame
	[to]	*<number>*	output picture
	size	*<x>, <y>*	dimensions of a subregion
	position	*<x>, <y>*	position/offset of subregion
options:	**picture/partition/frame**		read back the display picture, partition or frame
	left/right, top/bottom		subregion positions
	re/im		read back the real or imaginary part of a complex display picture

Use the **ovread** command to read back the overlay plane of a display picture, partition or frame into a Semper picture in *Image* form.

Examples

```
ovread  display to 2
```

This command reads the contents of the overlay of the current display picture into picture 2.

```
ovread frame 1 size 200 top left to 3:1
```

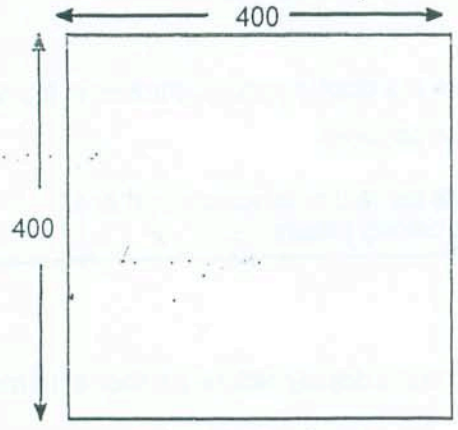
This command reads the contents of a 200 square region at the top left of display frame 1 into picture 1 on device 3.

Description

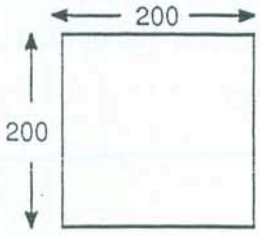
You can use the **ovread** command to read annotation back from the overlay. It creates a binary *Byte* form picture. The size of the picture corresponds to the dimensions of the source region in display pixels. The **ovread** command ignores under-sampling when reading back from the display, that is, a 400 square picture displayed with under-sampling of 2 occupies a region 200 square region on the display and **ovread** will create a 200 square picture when reading back from the display picture.

ovread

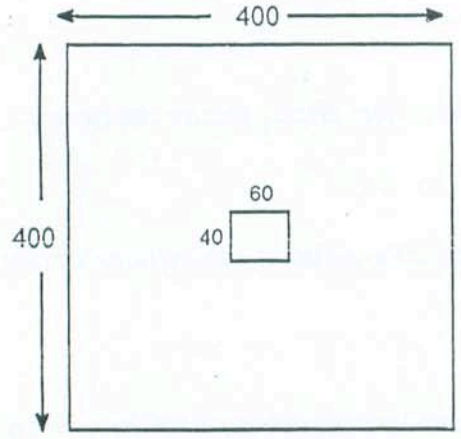
You can specify a 2-D subregion of a picture, partition or frame using the standard subregion keys and options **size, position, left/right** and **top/bottom**. Note that in the case of an under-sampled picture, the **size** key defines an area in terms of the original picture coordinates. For example, in the case of a picture undersampled by 2, specifying a subregion size of 60 by 40 gives an output picture of the dimensions 30 by 20. The effect of this is illustrated below:



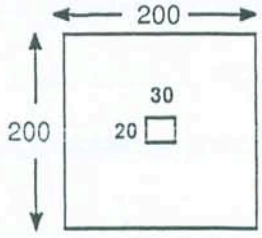
original disk picture



undersampled display picture



original disk picture



subregion size on display picture showing size of output picture

For complex display pictures, where the real and imaginary parts of the image are displayed side by side, **ovread** would normally read back the real part. If, however, you specify the option **im, ovread** will read back the imaginary part.

ovread

Notes

see also: **ovwrite**
form used internally: integer

Defaults and Ranges

keys/options	defaults	range
[ ]	if ***picture*** or ***partition***, current display picture held in the variable *display*; if ***frame***, current display frame held in the variable *cframe*	valid picture, partition or frame number
[to]	current picture held in the variable *select*	valid picture number
size	whole picture, partition or frame	less than or equal to the size of the picture, partition or frame (integers)
position	position 0,0	within bounds of picture, partition or frame (integers)
picture/partition/ frame	display picture	
re/im	read back real part of complex display picture	

ovwrite

keys:	[ ]	*<number>*	write to the specified picture, partition or frame
	[with]	*<number>*	source picture
	threshold	*<number>*	threshold value
	layer	*<number>*	layer of source picture to write
options:	**picture/partition/frame**		write to the display picture, partition or frame
	negated		invert the thresholding condition
	clip		clip data at border limits
	re/im		write to the real or imaginary part of a complex display picture
	view		switch the view to make the display region visible

Use the **ovwrite** command to write the data from a source picture into the overlay plane of the specified display picture, partition or frame.

Examples

```
ovwrite display with 1
```

This command overlays the image in picture 1 onto the current display picture, setting overlay pixels where the source pixels are greater than or equal to the mid-range value of the source picture.

```
ovwrite  frame 1 with 2 threshold 20
```

This command overlays the image in picture 2 onto the central region of display frame 1, using a threshold value of 20.

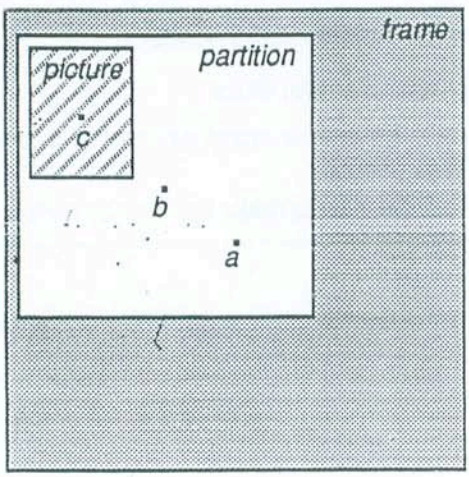
```
ovwrite display:2 negated clip
```

This command overlays the image in the current picture onto display picture 2, setting overlay pixels where the source pixels are below the mid-range value and clipping the output at the limits of the display picture.

ovwrite

Description

The **ovwrite** command writes the data from a source picture into the overlay plane of the specified display picture, partition or frame. It can be used to annotate pictures on the display by displaying an a binary image on top of a grey-scale image. The binary image is aligned so that its origin coincides with the graphics origin on the display. The three types of coordinate graphics origin are illustrated below.



a = frame coordinate origin
b = partition coordinate origin
c = picture coordinate origin

As the overlay plane is only 1 bit deep, the source image is converted into binary form by thresholding. By default, the threshold value is the mid range of the grey levels in the source picture. You can override this default using the **threshold** key. Overlay pixels are set to 1 where source pixels are greater than or equal to the threshold value and set to 0 elsewhere. If you specify the **negated** option the thresholding condition is reversed (overlay pixels are set where source pixels are less than the threshold value).

The **ovwrite** command will always clip the image at the graphics clipping limits (frame limits for a frame, and partition limits for a partition or picture). You can specify the **clip** option to blank out part of the display which lies outside the picture, partition or frame border.

Use the **layer** key to select a single layer if the source is a multi-layer picture. By default, **ovwrite** writes the first layer of the picture to the overlay.

For complex display pictures, where the real and imaginary parts of the image are displayed side by side, **ovwrite** would normally write to both parts. If, however, you specify the option **re** or **im**, only the real or imaginary part will be written to.

Notes

see also: **ovread**
form used internally: fp

ovwrite

Defaults and Ranges

keys/options	defaults	range
[ ]	if ***picture*** or ***partition***, current display picture held in the variable *display*; if ***frame***, current display frame held in the variable *cframe*	valid picture, partition or frame number
[with]	current picture held in the variable *select*	valid picture number
threshold	middle of source picture range	real number
layer	layer 1	valid layer number
picture/partition/ frame	display picture	
re/im	write to both parts of complex display picture	

pack

<variable name(s)>	type packed integer value for the specified variable name(s)

Use **pack** to type the packed integer value for a specified variable name.

Examples

```
pack pos, po2
```

This command types the packed integer values for the variables *pos* and *po2* (26219 and 26232).

Notes

see also: **unpack**

page

keys:	**width**	*<number>*	set terminal width
	length	*<number>*	set terminal page length
	aspect	*<number>*	set character aspect ratio
options:	**wrap**		wrap terminal output
	prompt		enable page prompt
	quit		enable the quit option in the page prompt
	enquire		set the variables *pwidth* and *plength* to the current page size

Use the **page** command to format output to the terminal.

Examples

```
page width 72 length 20 aspect 2.5
```

This command specifies a default terminal window size and a character aspect ratio.

```
page wrap
```

This command causes terminal output to wrap-around.

```
page noprompt
```

This command suppresses the page prompt.

```
page noquit
```

This command disables the quit option in the page prompt.

```
page enquire; type pwidth, plength
```

This sequence of commands sets the variables *pwidth* and *plength* to the current page size and prints out their values.

page

Description

Use the **width** and **length** keys to specify default dimensions for the terminal if the current defaults are inappropriate. If you specify dimensions that exceed the current size of the terminal window, the values are truncated. The **aspect** key specifies the aspect ratio for characters displayed on the terminal.

To find out the current page size, use the **enquire** option. This option sets the variables *pwidth* and *plength* to the current page dimensions.

By default, each line of text that is sent to the terminal is truncated to the current terminal width unless you specify the **wrap** option, in which case the line wraps-around onto subsequent lines.

A page prompt appears at the end of each page of information that is displayed on the terminal. This gives you the option of displaying the information one line or page at a time or of exiting from the current series of commands. Use the **noprompt** option to suppress this page prompt. You can also remove the quit option from the page prompt by specifying the **noquit** option. When the page prompt next appears on the terminal, the quit option is omitted.

Notes

variables set: *pwidth* (current page width, if ***enquire***)
plength (current page length, if ***enquire***)

Defaults and Ranges

keys/options	defaults	range
width	79	range is machine dependent (positive integer)
length	24	range is machine dependent (positive integer)
aspect	2	range is machine dependent (real number)
wrap	nowrap (lines are truncated)	
prompt	prompt on	
quit	quit enabled	

panel

keys:	**id**	*<number>*	define which panel to use
	name	*'<text>'*	specify a name for the panel
	on	*<number>*	specify a device number for the panel
	position	*<x>, <y>*	position a panel at the given coordinates
	size	*<x>, <y>*	define a minimum *x* and *y* size for a panel
	background	*<number>*	background colour of panel
	foreground	*<number>*	foreground colour of panel
options:	**create**		create a panel
	destroy		destroy a panel
	auto		the panel size is defined automatically
	hide		make a transient panel invisible
	mandatory		makes a panel mandatory, that is, Semper only allows interaction with elements on the specified panel when it is showing
	show		make a transient panel visible
	transient		specify that a panel is transient, not fixed

The **panel** command controls the creation and operation of a Semper 6 *Plus* panel object. For a description of Semper 6 *Plus* objects and elements, refer to the manuals:

User Interface Guide
Tutor User Guide

contained in the *Semper 6 Guide.*

Examples

```
panel create
```

This command creates a new, permanently visible panel on the default device.

```
panel create transient on fdi size 40, 10
```

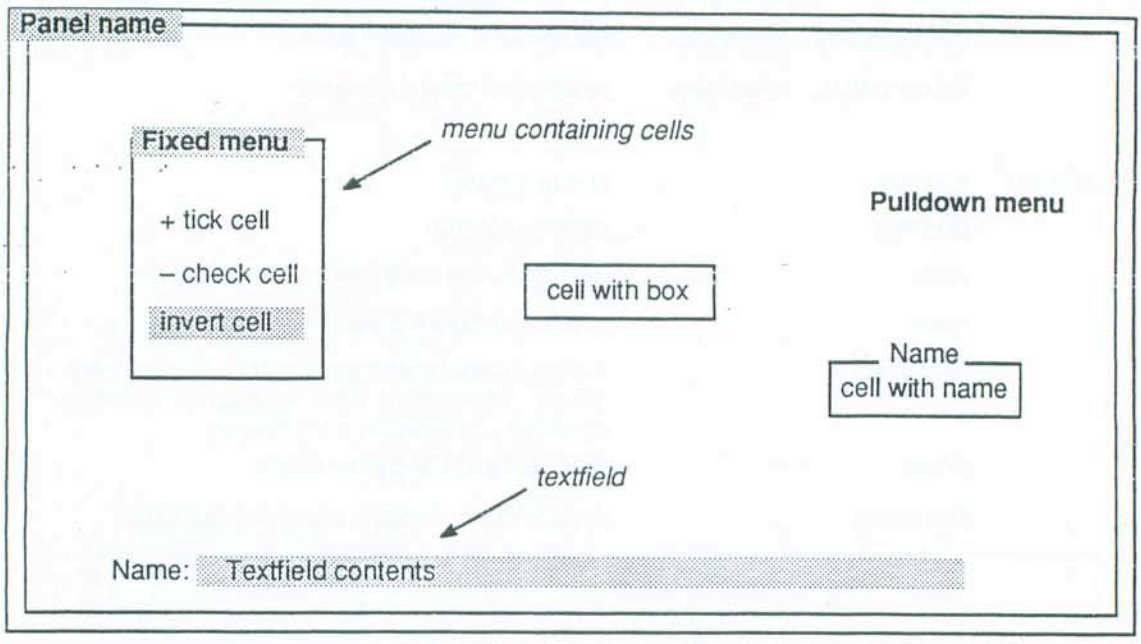
This command creates a new transient panel on the device *fdi* with the dimensions 40 by 10.

panel

Description

A Semper 6 *Plus* panel can be created using the **create** option and removed using **destroy**. Use the **id** key to define a panel to act upon. Note that when you use the **destroy** option the specified panel and all elements on that panel are destroyed.

The diagram below illustrates a Semper 6 *Plus* panel containing various objects and elements:



There are two types of panel:

- transient
- fixed

The default panel type is *fixed*. Fixed panels remain visible after they have been shown for the first time until you destroy them or leave Semper 6 *Plus*. You can create a *transient* panel using the **transient** option. A transient panel can be hidden using the **hide** option and redisplayed using the **show** option. You can move a transient panel using the **position** key.

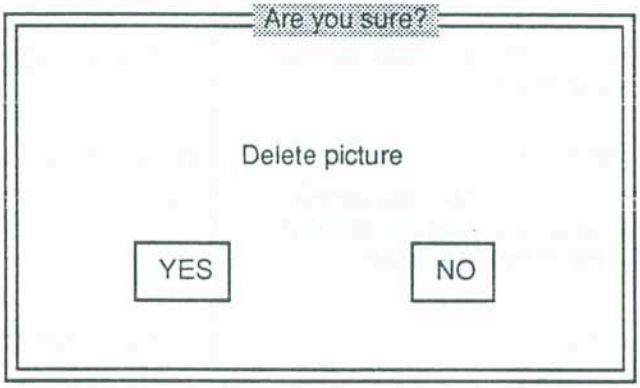
The **auto** option builds a panel so that it is just big enough to contain the elements placed upon it. The automatic sizing only takes place before a fixed panel is shown on the screen, or while a transient panel is hidden. Any elements on an **auto** panel retain their position relative to each other, but when the size of the panel is calculated, they are moved so that the topmost and leftmost element is at the top and left of the panel.

You can also use the **size** option to define your own size for a panel. Note that the size of the panel is assumed to include a one character wide border to allow for drawing a box around the panel. If you use the **auto** option, then the size given is treated as a minimum size for the panel.

panel

Use the **mandatory** option to include panels that gather information without which processing cannot continue, for example, a panel that asks for confirmation before beginning to delete data. Note that if a mandatory panel is transient and hidden, then interactions are allowed with any panel until the mandatory panel is made visible. If a mandatory panel is showing, then interactions are only allowed with this panel, regardless of the setting of **device active**.

Care should be taken to avoid displaying other panels when a **mandatory** panel is on the screen as this causes an error. The diagram below shows an example of a mandatory panel.



By default, the foreground colour for newly created panels is 1(usually white) and the background colour is 0 (usually black). You can override the default colours using the **foreground** and **background** keys. The **foreground** colour is used for lines and text. It is possible to change the colour of transient panels after they have been created but they must be hidden and redisplayed for the colour change to be visible.

Notes

variables set: *pno (*set by **panel create** to the newly created panel identifier and by **panel show** to the identifier of the displayed panel).

see also: **cell**, **menu**, **textfield**

panel

Defaults and Ranges

keys/options	defaults	range
id	current panel, held in variable *pno*	valid panel number (positive integer)
name	*none*	length is machine dependent (text string)
on	current device, held in the variable *cdi*	device numbers 1, 2
position	position 0,0	range is machine dependent (integer)
size	Semper 6 *Plus* determines appropriate size of panel if ***auto***, otherwise *none*	range is machine dependent (integer)
background	colour 0	range is machine dependent (integer)
foreground	colour 1	range is machine dependent (integer)

partition

keys:	**[number]**	*<number>*	partition number
	size	*<x>, <y>*	partition dimensions
	position	*<x>, <y>*	position/offset of partition
	frame	*<n1>, <n2>*	first, last frame number for partition
	lut	*<number>*	default viewing look-up table number for partition
options:	**left/right, bottom/top**		specify position of partition
	delete		delete partition
	enquire		report partition size and position by setting variables

Use **partition** to define or redefine the position and dimensions of a display partition and to establish a default viewing look-up table for a partition.

Examples

```
partition dis:3 size 256 top left
```

This command defines a partition *dis:3* as the top left 256 square of frame 1.

```
partition dis:11 frame 2 lut 3
```

This command defines partition *dis:11* as covering the whole of frame 2, and uses lut 3 as the default viewing look-up table.

```
partition dis:4 frame 1,3
```

This command defines the partition *dis:4* to take up the whole of the frames 1, 2 and 3 (suitable for the display of full-colour pictures).

```
library partitions
```

This command runs a program that defines partitions, dividing a frame into several rows and columns. The program prompts at the terminal for frame and partition details.

partition

Description

A partition defines a subregion of the display. Use the standard subregion keys, **size**, **position**, **left/right, top/bottom** to specify the dimensions of a partition in frame coordinates. Use the **frame** key to specify a range of frames for a partition. You display pictures by directing them to a partition. Partitions provide an easy way for you to display and store images and annotation within a framestore.

You can specify a default look-up table (monochrome, false colour, full colour) for a partition using the **lut** key. You can override the default look-up table for a partition by specifying a different number using the **view** command, for example, **view lut 2** or you can assign a different default, for example, **lut=2**.

If you redefine a partition any associated display picture is deleted, although not actually erased. Use the **create display** command to access a picture deleted in this way. You can overlap partitions and redefine a partition as often as you want, for example, you can define a new partition enclosing two distinct displays and treat them as one.

Use the **enquire** option to display the partition size, centre position and frames. This option causes the variables *psize*, *pposition* and *pframe* to be set (see *Notes* below).

Use the **delete** option to completely delete a partition, for example, **partition 3 delete**. Use the **erase...partition** command to erase a partition from the display.

Notes

variables set: *psize, ps2* (partition size)
pposition, pp2 (partition centre position)
pframe, pf2 (first and last frame of partition)

see also: **create display**, **erase**, **view**

Defaults and Ranges

keys/options	defaults	range
[number]	current display picture number, held in *display*	valid partition number
size	whole frame	within bounds of frame (integers)
position	position 0, 0	within bounds of frame (integers)
frame	1, *frame*	valid frame number
lut	lut 1	valid lut number

paste

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	position	*<x>, <y>, <z>*	position/offset of subregion for output picture
	layer	*<n1>, <n2>*	paste source as specified layer(s)
	mark	*<number>*	mark target region on display
		<yes> or *<no>*	
options:	**left/right, near/far**	**top/bottom,**	specify position of subregion

Use **paste** to paste one picture into another. It performs the opposite operation to **cut.**

Examples

```
paste 1 to 2 top left
```

This command inserts picture 1 at the top left of picture 2.

```
xwires; paste 50 to 51 @xy
```

This command inserts picture 50 into picture 51 centred at the position specified by means of the cursor.

```
paste 90 layer 2
```

This command inserts picture 90 into layer 2 of the current picture.

Description

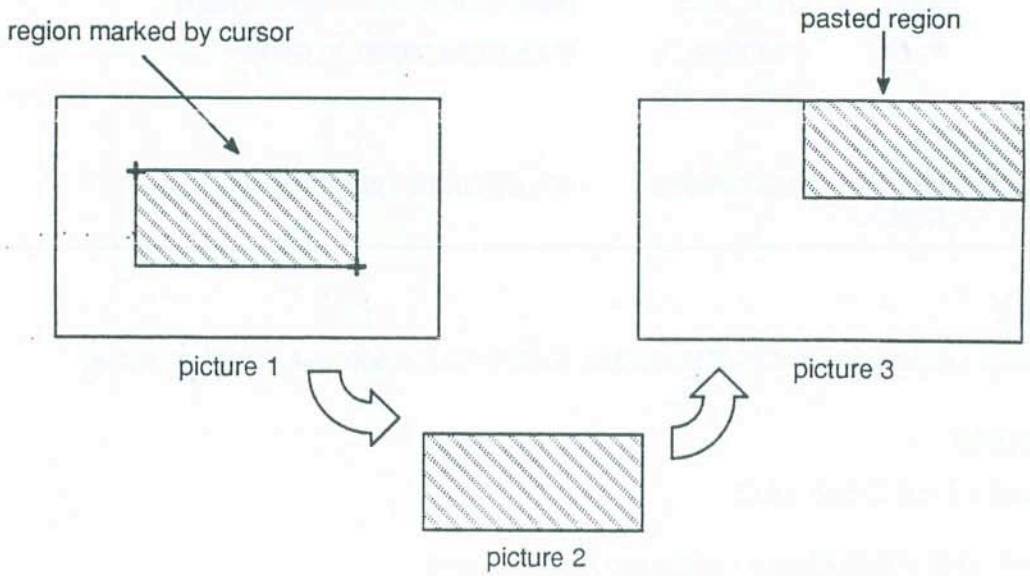
Use the standard subregion keys and options, **layer**, **left/right...**, to define the position at which the picture is pasted. See *Appendix C: Semper Keys and Options* for a full description of these keys. If you use the **position** key, the pasted picture is centred at the specified location. Any parts of the source picture that overflow the output picture or region are ignored. **paste** is often used in conjunction with **cut**.

If you specify a display using the **mark** key, for example, **mark display:2**, **paste** marks the target region. Refer to *Appendix C, Semper Keys and Options* for details of the **mark** key.

paste

The following diagram illustrates the **paste** command. It shows the effect of the sequence of commands:

```
xwires region; cut 1 to 2 @region; paste 2 to 3 top right
```



Notes

display marking:	target subregion
multi-layer pictures:	fully supported
forms used internally:	integer, fp, complex
see also:	**cut**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current picture, held in the variable *select*	valid picture number
position	position 0, 0, 0	within bounds of picture (integers)
layer	all unless variable *po3* is set	1 to number of layers (integer)
mark	mark off	see *Appendix C*

pcalculate

keys:	**[id]**	*<number>*	calculate parameters for specified particle
	[image]	*<number>*	original image containing particle
	[segment]	*<number>*	source segmented picture
	[plist]	*<number>*	source particle parameter list
options:	**verify**		send results to console

pcalculate operates on the particle parameter list (*ppl*) and the segmented picture produced by the **analyse** command. **pcalculate** calculates additional parameters for a specified particle, using the actual pixel values rather than a binary mask. Refer to *Appendix D, Particle Parameters* for a list of the parameters recorded by **analyse**.

Examples

```
pcalculate id 26
```

This command measures further parameters for the particle with identifier 26 and types the parameter values at the console.

```
pcalculate image 4 id 26 noverify
```

This command performs the same actions as the above example, but obtains pixel values from picture 4 instead. The **noverify** option suppresses the display of results at the console.

Description

pcalculate calculates additional parameters for a particle and returns their values in the following variables:

- *i1, i2* minimum and maximum intensity
- *si, mi* total and mean intensity
- *sdi* standard deviation for intensity
- *xcm, ycm* *x* and *y* coordinates of centre of mass
- *mm1, mm2* minimum and maximum principal second moments of mass
- *phi* orientation of long axis of mass

Use the keys **image**, **segment** and **plist** to specify a different original picture, segmented picture or *ppl* from the current defaults. By default, **pcalculate** sends the additional parameter values to the console. Use the **noverify** option to turn off this verification.

pcalculate

Notes

variables set:	*i1, i2*	intensity – *min, max*)
	si	(total intensity)
	mi	(mean intensity)
	sdi	(standard deviation for intensity)
	xcm, ycm	(centre of mass)
	mm1, mm2	(principal second moments of mass – *min, max*)
	phi	(orientation of long axis of mass)
see also:	**analyse**	

Defaults and Ranges

keys/options	defaults	range
[id]	current particle identifier, held in the variable *pid*	valid particle number
[image]	current image number, held in the variable *pimage*	valid picture number
[segment]	current segmented picture number held in the variable *psegment*	valid picture number
[plist]	*ppl* held in the variable *pplist*	valid picture number
verify	verification on	

Superseded by PICTURE (q.v.)

pcb

keys:	[from]	<number>	source picture

pcb produces *picture control block* information for a picture.

Examples

```
pcb 50; for x=x1,x2; p x,0=0; loop
```

This command sets the central row of picture 50 to zero.

```
pcb; unless class=3 | class=4 ..
```

This command tests whether the current picture is a class *Fourier* or *Spectrum* picture.

```
pcb; survey full; type mean*ncols*nrows
```

This command types the sum of all the pixels in a 2-D picture.

Description

pcb returns information about a picture. It sets the variables *x1*, *x2*, *y1*, y2 and *z1*, *z2* to the ranges of the x, *y* and *z* coordinates, *ncols*, *nrows* and *nlays* to the dimensions of the picture, *class* to the class number, *form* to the form number and *wpf*, write-protection flag to *yes* or *no*. You may like to use these variables in short command sequences as shown in the second command example. For details of picture class and form numbers refer to *Appendix A: Picture Types*.

Notes

variables set:

x1, *x2*, *y1*, *y2*, *z1*, *z2* (*x*, *y*, *z* coordinate range)
ncols, *nrows*, *nlays* (number of columns, rows and layers)
class (picture class number)
form (data form number)
wpf (write-protection flag: set to *yes* if protected)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number

pct

keys:	**[from]**	*<number>*	source picture (must be multi-layer)
	[to]	*<number>*	output picture containing the transformed image
	covariance	*<number>*	picture containing the covariance matrix
	eigen	*<number>*	picture used to store the eigenvalues and eigenvectors
	position	*<x>, <y>*	position, offset of subregion
	size	*<x>, <y>*	dimensions of subregion
options:	**hotelling**		perform a hotelling transform
	inverse		perform an inverse transform
	estimate		estimate the probable range of the output picture in order to rescale the output
	stretch		rescale the output
	left/right, top/bottom		subregion positions

The **pct** command performs a *Principal Component* transform or *Hotelling* transform. Use the **pct** command in *Remote Sensing* applications.

Examples

```
pct 1 2 covariance 3
```

This command performs a transform of picture 1 to picture 2, using the covariance details stored in picture 3.

```
pct 1 2 covariance 3 eigen 4
```

This command performs the same action as the above command and also stores the eigenmatrix in picture 4.

```
pct 2 5 covariance 3 inverse
```

This command performs an inverse transform from picture 2 to picture 5, using the covariance matrix stored in picture 3.

pct

Description

The **pct** command performs a *Principal Component* transformation (or *Hotelling* transform) used in *Remote Sensing* applications.

pct calculates the *forwards* **hotelling** transform as follows:

$$[o] = [g] * ([i] - [mo])$$

and its **inverse** is:

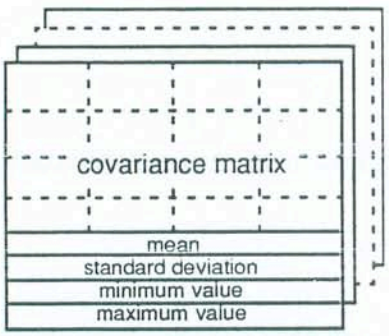
$$[o] = [g]^{-1} * [i] + mo$$

and the default non-mean adjusted transform is:

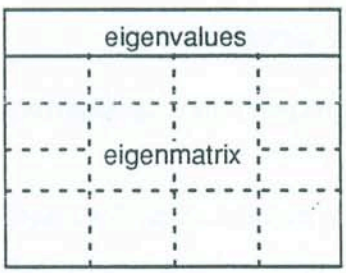
$$[o] = [g]\,[i]$$

Where [*i*] is the input picture, [*g*] is the eigenmatrix, [*o*] is the output picture and [*mo*] is the mean vector of the original picture. By default **pct** perform a *forwards* transform without adjusting the mean value of the picture layers.

The diagram below illustrates the layout of a **covariance** and an **eigenmatrix** picture.



covariance picture



eigenmatrix picture

Use the **estimate** option to estimate, from layer *minima* and *maxima*, the probable range of the output picture in order to rescale the output. If you do not use **estimate**, the actual layer ranges are determined and used. Estimated ranges are likely to overestimate the output picture range. Note that you can only rescale pictures that have a *Byte* form (see *Appendix A: Picture Types*).

pct

Use the **stretch** option to rescale output, from a picture in *Byte* form, so that the output covers the range 0 to 255. If you do not specify **stretch** the output picture is shifted so that all elements are positive, but not necessarily in the range 0 to 255. You can only use the **stretch** options on *forwards* transforms.

You must transform at least two layers of the picture. The maximum number of layers is dependent upon the number of columns in the input picture and the length of the Semper row buffers. Type **show system** to see your installation limits. Note that **pct** does not accept *Complex* pictures.

Notes

restrictions: *Complex* pictures are faulted
see also: **covariance, show system**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
covariance	*none*	valid picture number
eigen	*none*	valid picture number
position	position 0,0	within bounds of picture (integers)
size	whole picture	within bounds of picture (integers)
hotelling	perform transform without adjusting the mean values of the picture layers	
estimate	actual minimum and maximum layer ranges are used	

pcurve

keys:	**[with]**	*<number>*	*Plist* containing closed curve definition
	image	*<number>*	use pixels contained in specified picture to measure the additional intensity weighted parameters
options:	**verify**		send results to console

Use **pcurve** to make measurements of a particle outlined by an arbitrary curve you supply in the form of a (closed curve) *Plist* picture. The measurements are equivalent to those made by **analyse** and **pcalculate**. Refer to *Appendix D, Particle Parameters* for a description of the parameters calculated by **analyse**.

Examples

```
xwires closed curve; pcurve
```

This command measures the equivalent set of parameters usually recorded by **analyse** for the region enclosed by the closed curve input with the cursor. It stores the parameter values in variables and displays these values on the console.

```
pcurve image display
```

This command performs the same actions as in the above example and includes the 10 additional parameters usually calculated by **pcalculate**, taking pixel values from the display.

```
pcurve 51
```

This command measures the parameters for the curve in *Plist* 51.

Description

pcurve calculates most parameters exactly rather than on the basis of the pixels included within the curve, with the exception of the centre of area, second moments of area and the direction of the lowest second moment of area. An additional variable *np*, number of pixels, returns the area calculated on this basis as well. The variables *pid, pa, h, bg* and *ec* (which the commands **pid** and **pset** would set) are not meaningful for a closed curve taken in isolation, and are not set by **pcurve.**

Specifying a source picture with the **image** key causes **pcurve** to calculate 10 additional intensity-weighted parameters, in the same way as **pcalculate** and their values are returned in the variables *i1, 12, si, mi, sdi, xcm, ycm, mm1, mm2* and *phi.*

By default, **pcurve** sends the parameter values to the console; use the **noverify** option to turn off this verification.

pcurve

Notes

variables set:

np	(number of pixels enclosed by curve)
xr, yr	(reference point)
x1, x2	(*x* limits – minimum and maximum)
y1, y2	(*y* limits – minimum and maximum)
hf, vf	(feret diameters – horizontal, vertical)
af, bf	(feret diameters – 45 degrees, 135 degrees)
hp, vp	(projection – horizontal, vertical)
p	(perimeter)
a	(area)
xc, yc	(centre of area – *x*, *y* coordinates)
m1, m2	(principal second moments of area – *min*, *max*)
theta	(orientation of long axis of area)
c	(circularity)
i1, i2	(intensity – *min,max*)
si	(total intensity)
mi	(mean intensity)
sdi	(standard deviation for intensity)
xcm, ycm	(centre of mass – *x*, *y* coordinates)
mm1, mm2	(principal second moments of mass – *min*, *max*)
phi	(orientation of long axis of mass)

see also: **analyse, pcalculate**

Defaults and Ranges

keys/options	defaults	range
[with]	*Plist* 999	valid picture number
Image	additional parameters are not measured	valid picture number
verify	verification on	

pdraw

keys:	**[picture]**	*<number>*	draw on specified display picture
	[image]	*<number>*	modify pixels in specified image
	value	*<number>*	reset pixels to specified value
	position	*<x>, <y>*	initial cursor position
options:	**verify**		verify cursor input on display

Use **pdraw** to make minor adjustments to pictures, drawing a sequence of lines to separate or join particles when thresholding does not give the desired result.

Examples

```
pdraw image 4
```

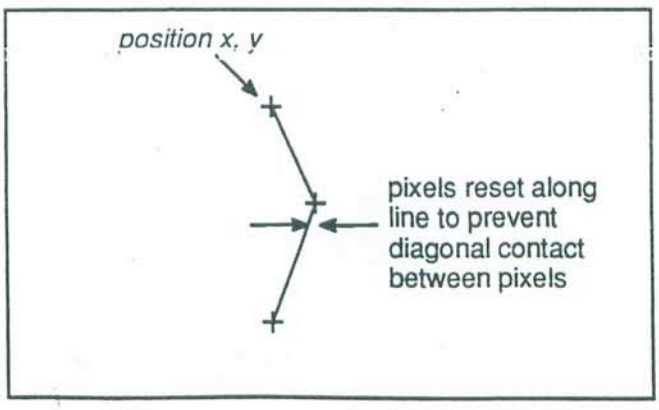
This command lets you draw a sequence of lines on the current display, along which the pixels in picture 4 are set to zero, to separate particles that were originally touching.

```
pdraw 11 picture dis:3 value max
```

This command lets you mark a curve on *display:3* along which picture 11 is set to *max*, so as to join the particles that were originally separated.

Description

By default, **pdraw** modifies the image specified by the **Image** key. Use two or more points to specify a sequence of lines and end the sequence by repeating the end point . The band of pixels that are reset in the picture is wide enough to prevent diagonal contact between points on the two sides of the curve. The diagram below illustrates the **pdraw** command.



pdraw

Defaults and Ranges

keys/options	defaults	range
[picture]	current display held in the variable *display*	valid picture number
[image]	current picture, held in the variable *pimage*	valid picture number
value	value 0	real number
position	position 0,0	within bounds of picture (integers)
verify	verification on	

peaks

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	threshold	*<number>*	minimum height of peaks
	sradius	*<number>*	radius from each peak within which lower peaks are discarded
	radius	*<number>*	radius around each peak over which a local centre of mass is calculated to replace the raw peak position
	mark	*<number>*	mark peak positions on the display
		<yes> *or* <no>	
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**negative**		include negative pixels in centre of mass search
	squared		find centre of mass of squared pixels
	iterated		repeat the centre of mass of squared pixels
	verify		list information about peak positions at the console

Use **peaks** to locate local peaks in a picture and make a list of them, sorted in order of descending size.

Examples

```
max=1; library molecule
display times 4; peaks threshold .5 mark display
```

This sequence of commands displays a small test picture, marks all the peaks in the picture, and stores their positions in *Plist* 999.

```
peaks threshold 3*sd radius 8 verify
```

This command locates and lists peaks higher than the standard deviation multiplied by three and calculates the centres of mass over regions 8 points in radius.

```
ps 1; survey; peaks threshold max/5 sradius 6
```

This command locates all the peak positions in a power spectrum, suppressing any peaks within 6 pixels of a higher peak.

peaks

Description

peaks searches from the top left of a picture for points higher than the threshold (default is zero) at which the value is also greater than the eight neighbouring values (or 2 neighbouring values in 1-D pictures). The peaks are stored by default, in *Plist* 999. You can specify a threshold value for the search using the **threshold** key.

Use the **mark** key to specify a display to mark and the keys **mkmode** and **mksize** to determine the style and size of marking. If the **mark** key is set, all the peak positions found are marked on the display. Refer to *Appendix C, Semper Keys and Options* for details of the keys **mark**, **mkmode** and **mksize**.

If your picture is noisy, you can use the keys **radius** and **sradius**:

- If you set **sradius** (*suppress radius*), **peaks** suppresses the lower peaks within **sradius** of any existing peak.

- If you set **radius**, raw peak positions are refined by a local centre of mass determination over a region with the specified radius, as if you had used the command **find cm**. You can use the **squared**, **negative** and **iterated** options relevant to the centre of mass mode.

The final list is sorted into descending order by peak height (or mass, if centre of mass calculation is included). The sorted positions, heights or masses are output as a *Plist* picture. The total number of peaks is also stored in the variable *n*. Semper lists information about peak positions at the console if you specify the **verify** option.

Notes

display marking:	positions found
multi-layer pictures:	faulted
forms used internally:	fp
variables set:	*n* (total number of peaks)
see also:	**find**

peaks

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	*Plist* 999	valid picture number
threshold	height 0	positive real number
sradius	*none*	real number
radius	radius 0	real number
mark	mark off	see *Appendix C*
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
negative	negative pixels included	
verify	verification off	

pedit

keys:	[plist]	*<number>*	source particle parameter list
	[to]	*<number>*	output particle parameter list
	segment	*<number>*	segmented picture (source and output the same)
		<n1>, <n2>	source and output segmented picture
	if	*<expression>*	logical expression specifying which particles to include
	unless	*<expression>*	logical expression specifying which particles to exclude

pedit operates on the particle parameter list (*ppl*) produced by the **analyse** command. **pedit** allows you to revise a *ppl* and/or a segmented picture to retain only the particles that satisfy your specified conditions. Refer to *Appendix D, Particle Parameters* for a list of the parameters recorded by **analyse** and the names that you should use to refer to them in **if/unless** expressions.

Examples

```
analyse...; pedit if hferet/vferet<5
```

This command revises the current *ppl*, only retaining particles with a horizontal feret diameter less than the vertical feret diameter times 5, that is, only long, thin particles that are vertically orientated.

```
pedit 4 5 unless circularity>.7
```

This command copies the *ppl* in picture 4 to 5, removing particles of a near-circular shape.

```
pedit 50 to 51 segment 4,5 if xcen>0
```

This command revises both the *ppl* 50 and the segmented picture 4, retaining only particles in the right-hand side of the analysed field. The final *ppl* is stored as picture 51, and the new segmented picture as 5.

Description

analyse records 25 particles for each particle and stores these details in a particle parameter list (*ppl*). **pedit** allows you to revise the *ppl* by specifying the type of particles to be retained using the keys **if/unless**. Use a valid Semper expression with these keys, as defined in *Appendix B, Semper Expression Syntax.*

To revise a *ppl*, specify an output *ppl* number using the key **to**. To revise a segmented picture use the **segment** key followed by the numbers of the source and output segmented pictures. If you omit the output picture number, it defaults to the source picture number.

pedit

Notes

see also: **analyse**

Defaults and Ranges

keys/options	defaults	range
[plist]	*ppl* held in the variable *pplist*	valid picture number
[to]	*none*	valid picture number
segment	no updating of segmented picture	valid picture number
if	true	valid Semper expression
unless	false	valid Semper expression

pextract

keys:	[id]	<*number*>	extract particle with specified identifier (source background, if zero)
	[segment]	<*number*>	segmented picture produced by ***analyse***
	[plist]	<*number*>	source particle parameter list containing results from ***analyse***
	[to]	<*number*>	output picture
	value	<*number*>	background pixel value
	image	<*number*>	fill particle outline with intensity values taken from the specified image

Use the **pextract** command to isolate a particle of interest. **pextract** operates on the particle parameter list (*ppl*) and the segmented picture produced by the **analyse** command. **pextract** extracts a single particle and places it in its own picture, in either grey level or binary form.

Examples

```
pextract id 5 to 12
```

This command extracts a particle with the identifier 5 and produces the binary image of the particle in a minimal size picture 12.

```
pextract image 2 to 3 value -1
```

This command places in picture 3 a region of picture 2 just large enough to contain the current particle (identifier stored in the variable *pid*). It preserves the original intensity values within the particle boundary and sets the intensity of other pixels to –1.

Description

By default, **pextract** produces a binary picture (containing the values 0 and 1) which is just large enough to contain the specified particle, with a one pixel border on all sides. Pixels that belong to the particle are given the value 1 and background pixels are set to 0. To preserve the original pixel values within the particle boundary, specify a source picture using the **image** key.

If you give a zero value to the **id** key, **pextract** extracts the picture background. By default, the output is in the same form as the source *Image* or in Byte form if one is not specified.

Notes

forms used internally: fp
see also: **analyse**

pextract

Defaults and Ranges

keys/options	defaults	range
[id]	current particle identifier, held in the variable *pid*	positive integer or zero
[segment]	current segmented picture, held in the variable *psegment*	valid picture number
[plist]	*ppl* held in the variable *pplist*	valid picture number
[to]	picture number 997	valid picture number
value	value 0	real number
image	binary image generated	valid picture number

pferet

keys:	**[id]**	*<number>*	calculate feret diameter for specified particle
	[segment]	*<number>*	source segmented picture
	[plist]	*<number>*	source particle parameter list containing results from ***analyse***
	angle	*<n1>...<n9>*	calculate feret diameters at specified angle(s), in radians anti-clockwise from the positive *x* axis
options:	**verify**		send results to console

pferet operates on the particle parameter list (*ppl*) and the segmented picture produced by the **analyse** command. **pferet** calculates up to nine distinct feret diameters for a single particle and returns the values in the variables *f*, *f2*, *f3...f9*.

Examples

```
pferet angle pi/8
```

This command sets the variable *f* to the feret diameter of the current particle (identifier *pid*) at an angle $\pi/8$ anti-clockwise from the positive *x* axis and verifies the result at the console.

```
pferet id 4 angle 0, .3, .6, .9, 1.2
```

This command sets the variables *f*, *f2*, *f3*, *f4*, *f5* to the feret diameters of the particle with identifier 4 in the specified directions and verifies the result at the console.

Description

pferet allows you to obtain feret diameters at angles other than the four angles that are measured by the **analyse** command. Note that the **hferet** and **vferet** parameters that are measured by **analyse** correspond to the angles of 0 and $\pi/2$ respectively and the **aferet** and **bferet** parameters correspond to the angles of $\pi/4$ and $3\pi/4$.

By default the values that **pferet** finds are sent to the console. To suppress this information, use the **noverify** option.

Notes

variables set:	*f*, *f2*, *f3*,*f4*...*f9* (calculated feret diameters)
see also:	**analyse**

pferet

Defaults and Ranges

keys/options	defaults	range
[Id]	current particle identifier, held in the variable *pid*	positive integer
[segment]	picture number held in the variable *psegment*	valid picture number
[plist]	*ppl* held in the variable *pplist*	valid picture number
angle	*none*	real number in range 0 to 2π
verify	verification on	

pfilter

keys:	**[from]**	*<number>*	source picture containing list of positions (*Plist*)
	[to]	*<number>*	output picture (*Plist*)
	tolerance	*<number>*	specify the tolerance for approximating the curve

Use the **pfilter** command to edit the list of positions produced by the **sketch** command. It allows you to reduce the number of points in a curve without distorting the curve beyond a specified tolerance.

Examples

```
pfilter
```

This command filters the curve defined by the current picture using the default tolerance of 1 and replaces the current picture with the result.

```
pfilter 1 to 2 tolerance 3.5
```

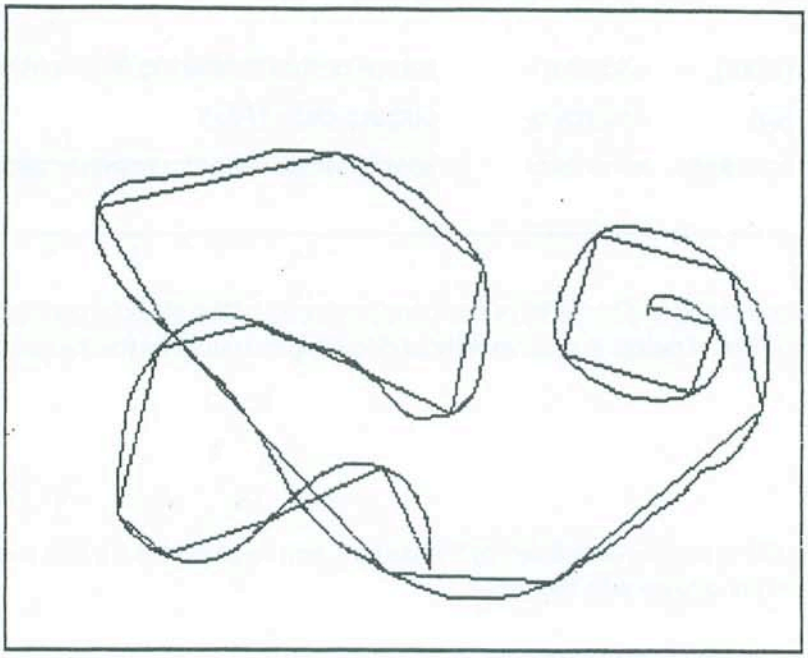
This command generates a more coarse approximation of the curve described in picture 1 and stores the result in picture 2.

Description

The **pfilter** command provides an easy-to-use curve filtering facility. Use this command to reduce the data produced by the **sketch** command after free-hand sketching with the mouse. The filtering process is controlled by the **tolerance** key which specifies the maximum departure of the approximation to the curve from the original curve.

The **pfilter** command moves along the curve until the line between the current point to the start point departs from the curve by more than the specified tolerance. The point before the current point is retained and all previous points except the start point are discarded. The process is then repeated starting from the last point to be retained. This is illustrated in the diagram given overleaf. A **tolerance** of eight pixels is specified to be the maximum departure from the original curve and the filtered version.

pfilter



The **pfilter** command requires a *Plist* (position list) source picture of the type produced by **sketch**, with at least two columns and layers and only one row. The data from all source layers are transferred to the output picture.

This command returns the length of the edited curve in the variable *p*, and the area for a closed curve in the variable *a*.

Notes

see also: **sketch**
variables set: *p* (length of curve)
a (area enclosed by a closed curve)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid *Plist* picture number
[to]	source picture	valid picture number
tolerance	1.0	real number greater than or equal to zero

phistogram

keys:	[plist]	*<number>*	source particle parameter list
	[to]	*<number>*	output histogram
	if	*<expression>*	logical expression specifying which particles to include
	unless	*<expression>*	logical expression specifying which particles to exclude
	channels	*<number>*	number of histogram channels
	height	*<number>*	histogram height in framestore pixels
	times	*<number>*	display magnification factor
	aspect	*<number>*	if **type/log**, text aspect ratio
	width	*<number>*	if **type/log**, number of characters per line
options:	xref, yref, id, parents, holes, background, contact, xmin, xmax, ymin, ymax, hferet, vferet, aferet, bferet, hproj, vproj, perimeter, area, xcen, ycen, mmin, mmax, angle, circularity		produce histogram for specified particle parameter
	preset		set histogram limits from *min*, *max*
	repeating		repeat histogram counts when magnifying, instead of interpolating
	letter		mark top of display partition with picture number and title
	border		mark picture border
	type/log		output histogram in character form to console or log output stream

phistogram operates on the particle parameter list (*ppl*) produced by the **analyse** command. **phistogram** displays a histogram of the parameters measured by **analyse.** Refer to *Appendix D, Particle Parameters* for a list of the recorded parameters and the names that you should use to refer to them in **if/unless** expressions.

Examples

```
analyse...; phistogram circularity
```

This command displays a histogram of the circularity value of all particles.

phistogram

```
phistogram area to 51 if perimeter > 200
```

This command creates a *Histogram* picture 51 containing a histogram of the areas of particles with perimeters greater than 200.

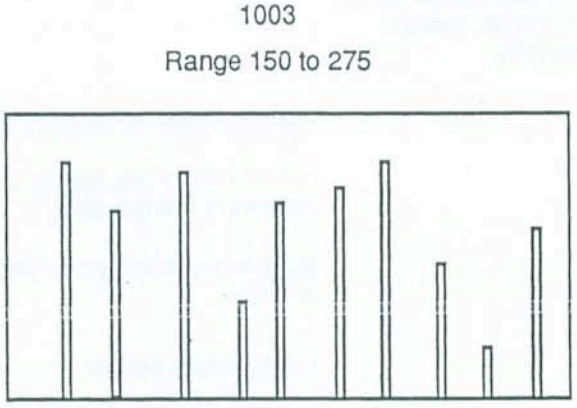
```
phistogram 46 area to dis:3
```

This command draws a histogram in *dis:3* showing the areas of all particles in *ppl* 46.

Description

analyse records 25 particles for each particle and stores these details in a particle parameter list (*ppl*). **phistogram** allows you to display a histogram of a named parameter, for example, **xmin, xmax**. You can use the keys **If/unless** to select only particles that meet particular conditions. The diagram below illustrates a histogram produced by the following sequence of commands:

```
analyse 1 to 2 ge h le h2 area 10; phist 2 to 3 area; display 3
```



By default, **phistogram** displays a histogram in graphical form on the display. You can also store a histogram in a class *Histogram* picture using the **to** key. Use the keys **channel, height** and **times** to specify the number of histogram channels and the height and magnification of a graphical display.

Use the **type/log** option to output the histogram in character form to the console or to the log output stream. Use the **aspect** and **width** keys to specify the aspect ratio and number of characters per line for character form displays.

You can turn off the default lettering and border marking using the options **noletter** and **noborder**.

Notes

see also: **analyse**

phistogram

Defaults and Ranges

keys/options	defaults	range
[plist]	*ppl* held in the variable *pplist*	valid picture number
[to]	current display picture, held in the variable *display*, histogram shown in graphical form	valid picture number
if	true	valid Semper expression
unless	false	valid Semper expression
channels	*max,min* if in range 20,256; otherwise 256	positive integer
height	lesser of half histogram width and half partition height	positive integer
times	1	positive integer
aspect	default given by the **page** command	positive real number
width	default given by the **page** command	positive integer
xref,yref,id,parents, holes,bacground, contact,xmin,xmax, ymin,ymax,hferet, uferet,uferet,aferet, bferet,hproj,uproj, perimeter,area,xcen, ycen,mmin,mmax, angle,cirularity	*none*	
preset	lowest and highest selected parameter values	
repeating	interpolate between histogram counts when magnifying	
border	bordering on	
letter	lettering on	
type/log	histogram shown in graphical form on display	

photo

This command is specific to...
PC + selected versions of Synoptics Synergy framestore

keys:	**partition**	*<number>*	specify partition to output
	frames	*<number>*	specify number of frames to output
	ignore	*<n1>, <n2>*	ignore pixels until specified clock time and line
	yreplicate	*<number>*	specify *y*-axis magnification

In Synergy systems that are enhanced by the addition of the *Photo Output* facility, the **photo** command allows you to output the contents of the display (or a display partition) to a recording *c.r.t.*, such as those frequently found in scanning electron microscopes. If required, you can replicate each line of the display several times at the output.

Examples

```
photo
```

This command outputs partition 1 (which is usually the whole frame) to the photo output unit, in synchronism with the slow-scan clocks normally used for input.

```
photo partition 2 frames 10 ignore 100,200
```

This command outputs partition 2, ten times, on the photo output port. The first pixel will appear 100 clock times into the 200th line of the scan.

```
photo yreplicate 3.5
```

This command outputs the first line of the display three times, the second line four times, etc. to give an overall *y*-axis magnification of 3.5.

Description

You can interrupt the output process at any time by pressing the keys *<control-c>* or *<control-break>*.

photo

Defaults and Ranges

keys/options	defaults	range
partition	partition 1	valid partition number
frames	frame 1	positive integers
ignore	0, 0	positive integers
yreplicate	1	positive real number

phr

keys:	[from]	<number>	source picture
	[to]	<number>	output picture

Use **phr** to *phase-randomise* a *Fourier* transform, that is, to assign it random phases while preserving its modulus. You can use this type of filter to test for significant local ordering in disordered pictures with strongly 'coloured' spectra.

Examples

```
phr 1 to 2
```

This command randomises the phase of picture 1 and stores the result as picture 2.

Description

You must specify a *Fourier* source picture with **phr**. If the origin is at the left of the picture, **phr** assumes a half-plane transform with conjugate symmetry and preserves the symmetry.

You use the *phase randomisation* test in the following way. Take an original picture showing signs of local order (for example, small patches of fringes throughout the image) and compare it visually with a phase-randomised version that you produce using the command sequence **fourier; phr; image.** The two images will invariably differ in specific detail, but if they have the same general texture, you can deduce that the structures apparent in the original picture only reflect *white noise*, suitably filtered to impose a particular power spectrum. This is because the transform of white noise has random phases.

For **phr** to preserve conjugate symmetry, ensure that the column length of the picture is not greater than the maximum row length supported by your installation. Type **show system** to see your installation limits.

The random number generator 'seed' is held in Semper variable *rnm*, and you can initiate reproducible 'random' sequences by setting this yourself.

Notes

multi-layer pictures:	faulted
forms used internally:	complex
variables set and used:	*rnm* (random number generator seed – in the range 0 to 1)
see also:	**fourier, image, show system**

phr

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

picture

keys:	**[from]** *<number>*	source picture
options:	**size**	return picture dimensions
	limits	return X, Y and Z picture coordinate limits
	class	return picture class
	form	return data form
	wp	return write-protect flag
	date	return creation date
	time	return creation time
	range	return intensity range
	sum	return intensity sum
	statistics	return intensity mean and standard deviation
	bw	return black and white levels (display picture only)
	sampling	return sampling interval (display picture only)
	type	return position list type (position list picture only)
	all	return all available information

You use the **picture** command to obtain information about a given picture. The information is returned in Semper variables. You select the required parameters by specifying the corresponding option. This command supersedes the **pcb** command.

Examples

```
picture size
```

This command returns the dimensions of the current picture in the variables *nx*, *ny* and *nz*.

```
picture 2:5 all
```

This command returns all available information for picture 5 on device 2.

```
threshold gt 100; picture sum
```

This example returns the number of pixels in the current picture whose intensity is greater than 100 in the variable *s*.

picture

Description

The **picture** command sets only the variables associated with the data you request. If the data is not defined (for example, the position list type for a picture which is not a position list), the corresponding variables are returned unset. Use the **set** function to check for an unset variable.

With the options **sum** and **statistics**, a separate sum and mean is returned for the real and imaginary parts of a complex picture. The standard deviation for a complex picture is measured in terms of the modulus of the data with respect to the complex mean.

For a non-complex picture, zero values are returned for the imaginary sum and mean (variables *s2* and *m2*.

The names of variables have been chosen so that they do not clash with key or option names for any of the existing commands. By using the individual options, you can control directly which variables are set . You should use this command in preference to the **pcb** command which sets all of its variables in one go (including the *form* variable which can interfere with the operation of other commands).

If you specify the option **all**, all available information about the source picture is returned.

Notes

variables set:	*nx,ny,nz*	picture dimensions
	x1,x2	X coordinate limits (x1 < = x2)
	y1,y2	Y coordinate limits (y1 < = y2)
	z1,z2	Z coordinate limits (z1 < = z2)
	c	picture class
	f	data form
	w	write-protect flag (0 = off, 1 < = on)
	yd,md,dd	creation date – year,month,day
	ht,mt,st	creation time – hours,minutes,seconds
	i1,i2	intensity range
	s	intensity sum (real part for compex picture)
	s2	intensity sum (imaginary part for complex picture)
	m	intensity mean (real part for complex picture)
	m2	intensity mean (imaginary part for complex picture)
	sd	intensity standard deviation
	db,dw	display intensity scaling – black,white levels
	ds	display sampling interval
	pt	position list type

picture

Defaults and ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number

pid

keys:	**[segment]**	*<number>*	segmented picture produced by ***analyse***
	position	*<x>, <y>*	identify particle at specified position
options:	**verify**		verify identifier at console

pid operates on the segmented picture produced by the **analyse** command. It identifies the particle at a specified position and types the identifier number on the console. **pid** stores the identifier in the variable *pid*.

Examples

```
analyse...segment 51; xwires; pid @xy; ptype all if id=pid
```

This command sets the variable *pid* to the identifier of the particle you select using the cursor. **ptype** then prints all the parameters of the particle.

```
pid segment 23 position x,y
```

This command types the identifier of the particle at position *x, y* in the segmented picture 23.

Description

If the position that you specify does not contain a particle, the variable *pid* is set to zero and the message *'Background hit'* is shown at the console. You can suppress any console messages using the **noverify** option.

Notes

variables set: *pid* (particle identifier)
see also: **analyse**, **ptype**

Defaults and Ranges

keys/options	defaults	range
[segment]	current segmented picture held in the variable *psegment*	valid picture number
position	position 0,0	within bounds of picture (integers)
verify	verification on	

pixel, p

<pixel coordinates> = <pixel values>	***pixel*** uses a special syntax. Items before the equals sign are taken to be the coordinates of the pixel to change. Items afterwards are taken as value pairs to insert.

Use the **pixel** or **p** command to change the value of any pixel in the currently selected picture. Pixels are *picture elements*, that is, the brightness sample values making up a picture.

Examples

```
pixel x, y, z=value, ivalue
```

This command sets the pixel with coordinates *x, y, z* in the current picture to the specified value pair.

```
create 1 size 256, 1; origin left; for x 0,255; pixel x=x; loop
```

This sequence of commands creates a linear ramp in picture 1.

```
xwires; type pixel(x,y)
```

This command types the value of the pixel pointed to by the cursor.

Notes

forms used internally: complex

pmark

keys:	**[picture]**	*<number>*	display picture to be marked
	[plist]	*<number>*	source particle parameter list
	if	*<expression>*	logical expression specifying which particles to include
	unless	*<expression>*	logical expression specifying which particles to exclude
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**xref,yref,id,parents,holes, background,contact,xmin, xmax,ymin,ymax,hferet, vferet,aferet,bferet,hproj, vproj,perimeter,area,xcen, ycen,mmin,mmax,angle, circularity**		mark with the specified particle parameter value
	cm		place annotation at particle centres of area

pmark operates on the particle parameter list (*ppl*) produced by the **analyse** command. **analyse** records 25 particles for each particle and stores these details in a *ppl*. **pmark** allows you to mark on a display the position of a set of particles or the values of one of their parameters. Refer to *Appendix D, Particle Parameters* for a list of the parameters recorded by **analyse** and the names that you should use to refer to them in **if/unless** expressions.

Examples

```
analyse...; pmark area
```

This command marks, as text, the area of all particles on picture *display*.

```
pmark if perimeter > 200
```

This command marks the position of all particles with a perimeter greater than 200.

```
pmark picture dis:3 hferet cm
```

This command marks in *dis:3* the horizontal feret diameters of all particles, at the centre of area rather than at the reference point.

pmark

Description

pmark marks the positions of the particles recorded by **analyse** in the style determined by **the** general keys **mkmode** and **mksize** (see *Appendix C: Semper Keys and Options* for details). **pmark** also allows you to mark the value of a particle parameter by specifying parameter names, for example, **xcen**, **ycen**. You can use the keys **If/unless** to select only particles that meet a specified condition.

To mark a display other than the current display, use the **picture** key. Use the **cm** option to mark a particle at its centre of area instead of at its reference point.

Notes

see also: **analyse**

Defaults and Ranges

keys/options	defaults	range
[picture]	mark current display, held in the variable *display*	valid display picture number
[plist]	*ppl* held in the variable *pplist*	valid picture number
If	true	valid Semper expression
unless	false	valid Semper expression
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
xref,yref,ld,parents, holes,background, contact,xmin,xmax, ymin,ymax,hferet, vferet,aferet,bferet, hproj,vproj,perimeter, area,xcen,ycen,mmin, mmax,angle,circularity	*none*	
cm	place annotation at particle reference points	

pointer

keys:	**gearing**	*<n1>, <n2>*	specify the ratio of mouse to display pixels (how quickly the cursor moves compared to mouse movements)
	sensitivity	*<n1>, <n2>*	specify the sensitivity ratio of the mouse

Use the **pointer** command to set the condition of the pointing device (usually a mouse). You can specify the ratio of pointer steps to display pixels (the gearing) and the size of movements below which pointer movement is ignored (sensitivity) in the *x* and *y* directions.

Examples

```
pointer gearing 4 sensitivity 2
```

This command selects a 4:1 gearing ratio and ignores moves of less than two mouse pixels.

```
pointer gearing 1,100 sensitivity 1,10
```

This command selects a 1:1 ratio with maximum sensitivity in the *x* (horizontal) direction, but uses a 100:1 ratio and low sensitivity in *y*. This is useful if you must have a horizontal line in **xwires**.

Description

Use the **gearing** key to specify how quickly the cursor moves in comparison with the movements of the pointer (high gearing allows precise positioning). Use **sensitivity** to threshold the pointer movements. Any movement that is less than the sensitivity is ignored, so a high value for sensitivity causes less jitter, but makes the positioning less precise.

Note that if you are using a *Sun3* workstation or a *Silicon Graphics IRIS 4D* workstation the gearing and sensitivity settings are ignored if the workstation's mouse is being used as the pointing device by Semper.

Defaults and Ranges

keys/options	defaults	range
gearing	4, ***gearing***	positive integers
sensitivity	2, ***sensitivity***	positive integers

postscript

keys:	[ ]	*<number>*	create a *PostScript* file for the specified source picture, or display frame/partition/picture if ***frame/partition/picture*** is set
	name	*'<text>'*	output filename
	copies	*<number>*	number of copies to print
	times	*<number>*	output scaling factor
	size	*<x>, <y>*	dimensions of subregion
	position	*<x>, <y>*	position/offset of subregion
	layer	*<n1>, <n2>*	range of source picture layers
	text	*'<text>'*	caption string
options:	**encapsulated**		generate *Encapsulated PostScript* file
	frame/partition/picture		output display frame/partition/picture
	preset		scale black-white levels according to existing values of *min*, *max*
	black/white		output display annotation as black/white
	portrait/landscape		output orientation
	left/right/top/bottom		subregion positions
	re/im		output real/imaginary part of a complex display picture
	border		print border around image
	origin		print tick marks aligned with source picture origin
	above/below		print caption string above/below image
	header		print header information
	new		replace an existing output file
	old		re-use an existing output file, starting from the beginning; you can use this option for write-only output devices, for example, printers

Use **postscript** to write out Semper images or screen dumps in a form that can be sent to a *PostScript* printer.

postscript

Examples

```
postscript 5 name 'image' times 2.3 landscape
```

This command creates a *PostScript* file called *image.ps*, containing picture 5 magnified by a factor of 2.3 and in landscape orientation. This file can be sent to a *PostScript* printer.

```
postscript frame name 'screen' encapsulated
```

This command creates the encapsulated *PostScript* file *screen.eps*, containing the image displayed in the current frame, together with display annotation. This file can be included in another *PostScript* document.

```
postscript 2:33 name 'fig10.6' text 'Fig 10.6 Original Sample'
```

This command creates a *PostScript* file containing picture 2:33 and its caption.

Description

The **postscript** command allows you to write images in *PostScript* format. An image can be either all or part of a Semper source picture (stored on disc) or can be a display frame, partition or picture (shown on screen). To specify a display frame, partition or picture use the options **frame/partition/picture**.

Specify a name for the output *PostScript* file using the **name** key. You can choose one of the two following forms of output:

- *PostScript*
- *Encapsulated PostScript*

Both of the above forms follow the *PostScript* Document Structuring Conventions defined by *Adobe Systems Incorporated*.

By default, **postscript** creates a complete *PostScript* program that prints the source image together with annotation. The image is scaled so that each pixel is 1 unit square, or the value you specify using the **times** key. A unit is 1 printer's point (1/72 inch). If you do not specify an extension for the name of the *PostScript* file, it defaults to *.ps*.

Use the **encapsulated** option if you wish to include a Semper image in another *PostScript* document, for example, a document used by a desktop publishing system. The **encapsulated** option creates a *PostScript* program that prints the image onto the current page with the bottom left-hand corner of the image at the origin and with the image scaled so that each pixel is one unit square. The default extension for an *Encapsulated Postscript* file is *.eps*.

postscript

You can use the standard subregion keys, **size**, **position** etc to define a subregion of an image to be output. Refer to *Appendix C, Semper Keys and Options* for further detail about the standard subregion keys. Use the **layer** key to specify a range of layers for the source picture. However, if you specify **encapsulated**, only the first specified layer is output. If you specify the **frame** option, only the specified frame is output, or the start frame if you specify **partition** or **picture.**

By default, **postscript** prints a Semper picture so that the lowest pixel intensity appears black and the highest pixel intensity appears white. You can alter these black and white levels using the **preset** option, which takes the black and white intensity levels from the variables *min* and *max*. The options **black/white** determine whether any display annotation is output as black or white.

Note that the following keys and options can only be used with the default *PostScript* output and cannot be specified with the **encapsulated** option:

- **copies** (number of copies of the document to be printed)
- **times** (scale of output)
- **text** (adds a caption to the printed output)
- **portrait/landscape** (defines the orientation of the output)
- **border** (draws a border around the image)
- **origin** (marks the position of the origin on the printed output)
- **above/below** (positions a caption above/below the image)
- **header** (prints information about the source picture)

Also note the following restrictions on the use of **postscript** keys and options:

- the **black/white** option can only be used if you specify the **frame/partition/picture** option
- the **re/im** option can only be used if you specify the **picture** option
- the **preset** option and **layer** key can only be used with a source picture (they are ignored if you specify the **frame/partition/picture** option)

postscript

Defaults and Ranges

keys/options	defaults	range
[ ]	if source picture, current picture, held in the variable *select* if ***frame***, current frame held in the variable *cframe* if ***partition/picture*** number held in the variable *display*	valid picture/partition/frame number
name	*none*	valid filename
copies	1	positive integer
times	1	positive real number
size	entire picture/frame/partition	less than or equal to the size of picture/frame/partition (integers)
position	position 0,0	within bounds of picture/frame/ partition (integers)
layer	all layers (ignored if ***frame /partition/picture***)	integers in range1 to number of layers
text	*none*	length is machine dependent (text string)
black/white	white	
portrait/landscape	portrait	
border	border is printed	
above/below	below	
header	header is printed	

print

keys:	**[from]**	*<number>*	picture to be printed
	size	*<x>, <y>*	size of subregion to be printed
	position	*<x>, <y>, <z>*	position/offset of subregion
	layer	*<number>*	subregion layer
options:	**left/right, bottom/top, near/far**		subregion positions
	mp		print complex values as modulus and phase form rather than as real and imaginary parts

Use **print** to print small blocks of pixels to the console. (Use the **echo** command to direct output to the console or to the log output stream).

Examples

```
xwires; print @xy size 5
```

This command prints a 5 by 5 blocks of values around a position marked using the cursor.

```
print top left mp layer 4
```

This command prints a block of values (for a *Complex* picture) from the top left of layer 4, in modulus and phase form.

```
print size 7,3
```

This command prints a central block of values, 7 points by 3, to the console.

Description

Use the standard 2-D subregion keys and options, **size**, **position** etc. to specify the block of pixels to be printed. (Refer to *Appendix C: Semper Keys and Options* for further detail). You can also use the **layer** key to select one layer from a multi-layer picture.

The block size defaults to the largest odd number that fits the terminal width (as determined by the **page** command), which is typically nine. The *x* and *y* coordinates are written at the side and across the top of the printed block. Imaginary parts, where present, are printed separately, following the real parts. Use the **mp** option to print complex values in modulus and phase form rather than in real and imaginary form.

print

Notes

multi-layer pictures: one layer, selected by **layer**
forms used internally: integer, fp, complex
see also: **echo**, **page**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
size	typically 9, *size*; the first value depends on the terminal width	less than or equal to the size of the picture (integers)
position	position 0, 0 ,0	within bounds of picture (integers)
layer	layer that includes the origin	integer in range 1 to number of layers

project

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	angle	*<number>*	projection direction, in radians anti-clockwise from the positive *x* axis
	mark	*<number>*	mark projection line and direction on the display
		<yes> *or* <no>	
options:	**horizontally/vertically**		project horizontally/vertically
	average		average the intensity projection over the area

Use **project** to project or average a picture, in any direction, to form a 1-D picture. For example, you can use **project** to obtain a relatively noise-free image of linear structures (edges or interfaces). You can also use it, in conjunction with **backproject**, to reconstruct a pictures from its projections.

Examples

```
project 1 to 2 vertically
```

This command sums the columns of picture 1 to form a 1-D picture 2.

```
xwires line; project display to 10 angle theta average
```

This command averages the display picture along the direction specified by the cursor.

```
project 50 to 51 horizontally
```

This command projects horizontally, that is along the rows of picture 50 to form picture 51.

Description

Semper automatically defines the width of the output to just accommodate all projected source pixels, and the origin is placed where the source origin projects. The positive *x* direction in the projection lies to your right as you look in the projection direction. If you use the **mark** key to specify a display, **project** marks the projection line and direction on the display. See *Appendix C, Semper Keys and Options* for further detail of the **mark** key.

The **horizontal** and **vertical** options perform efficient row and column sums/averages, equivalent to **angle 0** and **angle pi/2.** The command **project .. angle ..** allows you to indicate an arbitrary projection direction instead, in radians, anti-clockwise from the positive *x* axis. Each source pixel is

project

shared between two projection columns in proportion to the distance from the two column centres – fluctuations in the number of points that happen to project into each column can otherwise lead to uneven projections even from smooth source data. Note that there are certain directions that will still give uneven results (when the projection shape is equal to the ratio of small integer values – 1:1 or 45° being the worst case).

The **average** option causes the projection sums to be divided by the area of the source picture that contributes to each sum.

Beware of the possible clash of the **vertical** option with the option **verify**.

Notes

display marking:	projection line and direction
1-D pictures:	faulted
multi-layer pictures:	faulted
forms used internally:	complex
see also:	**backproject**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
angle	horizontal projection	real number in range 0 to 2π
mark	mark off	see *Appendix C*

ps

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
options:	**ln**		produce (natural) logarithm of transform intensity

Use **ps** to calculate the power spectrum (Fourier transform intensity) of a picture. This is a widely used tool for detecting and measuring periodic features, and for assessing resolution or image formation conditions.

Examples

```
ps 1 4
```

This command obtains the spectrum of picture 1 as picture 4.

```
ps ln; survey full; min=mean; display preset
```

This command replaces the current picture with the (natural) log of its spectrum, to accommodate the wide dynamic range of the spectrum, and displays this, suppressing the very small values.

Description

The **ps** command performs a Fourier transform of a picture and then takes its squared modulus (intensity). If you supply the picture already transformed, as a *Fourier* picture, **ps** simply takes the squared modulus.

ps produces a *Spectrum* class picture in floating-point form. **ps** follows the same pattern as **fourier** with respect to real/complex data. If the source data is real (real *Image* or auto-*Correlation*, or half-plane *Fourier*), only the right half-plane is output. If the data is complex (*Complex Image* or auto-*Correlation*, or full-plane *Fourier*), the full-plane is output. Use the commands **fullplane** or **halfplane** to convert these further if required.

Notes

restrictions:	image sizes must be powers of two unsuitable for direct output to display
multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**fourier, fullplane, halfplane**

ps

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

pset

keys:	[plist]	*<number>*	source particle parameter list
	sort	*<parameter name>*	name of parameter by which particles are sorted
	if	*<expression>*	logical expression specifying which particles to include
	unless	*<expression>*	logical expression specifying which particles to exclude
	index	*<number>*	ranking of the selected particle
options:	**xref, yref,id,parents,holes, background,contact,xmin, xmax,ymin,ymax,hferet, vferet,aferet,bferet,hpfroj, vproj,perimeter,area,xcen, ycen,mmin,mmax,angle, circularity**		return the values of the specified parameters using the corresponding variables *xr, yr, pid, pa, h, bg, ec, x1, x2, y1, y2, hf, vf, af, bf, hp, vp, p, a, xc, yc, m1, m2, theta, c*
	count		return the number of selected particles in the variable *n*
	all		return the values of all the parameters for one particle
	ascending/descending		if ***sort***, sorton specified parameter in ascending or descending order

pset operates on the particle parameter list (*ppl*) produced by the **analyse** command. **pset** allows you to set variables to the values of the various parameters recorded in a *ppl*. Refer to *Appendix D, Particle Parameters* for a list of the parameters recorded by **analyse** and the names that should be used to refer to them in **if/unless** expressions and with the **sort** key.

Examples

```
analyse...; pset all index 3
```

This command returns values for all 25 parameters recorded for the third particle found by **analyse**. The values are returned in the variables *xr, yr...c* etc.

```
pset perimeter if id=4
```

This command returns in the variable *p* the perimeter of the particle with an identifier of 4.

pset

```
pset count if area>100
for index = 1,n
   pset xcen ycen if area>100 sort area descending
   ...
loop
```

This example returns in the variable *n* the number of particles with area greater than 100 and then retrieves as *xc,yc* the centre position of each of the particles in turn, starting with the particles with the greatest area.

Description

analyse records 25 parameters for each particle and stores these details in a particle parameter list (*ppl*). **pset** returns the values of these 25 parameters in corresponding variables. Use the **all** option to set all 25 variables. By default, **pset** returns values for the first particle encountered. If you need values for a different particle set the **Index** key to the required ranking of particles (1,2,3...).

Use the keys **If/unless** to set criteria for selecting/rejecting particles. The **sort** key allows you to select a particle parameter on which to sort the particle list. The parameter names you can specify are **xref, yref, id , parent, holes, background, contact, xmin, xmax, ymin, ymax, hferet, vferet, aferet, bferet, hproj, vproj, perimeter, area, xcen, ycen, mmin, mmax, angle** and **circularity.** You can also use these parameter names as options to recover specific parameter values for a given particle.

You can use the **count** option to return the number of selected particles in the variable *n*. Note that if you initially type the command **pset count** you can then write a loop recovering parameters for all listed particles in turn, as in the last command example.

Notes

variables set:	*n*	(number of selected particles)
	xr, yr	(reference point)
	pid	(identifier)
	pa	(parent id)
	h	(number of holes)
	bg	(background flag)
	ec	(edge contact flag)
	x1, x2	(*x* limits – minimum and maximum)
	y1, y2	(*y* limits – minimum and maximum)
	hf, vf	(feret diameters – horizontal, vertical)
	af, bf	(feret diameters – 45 degrees, 135 degrees)
	hp, vp	(projection – horizontal, vertical)
	p	(perimeter)
	a	(area)
	xc, yc	(centre of area – *x*, *y* coordinates)
	m1, m2	(principal second moments of area – *min*, *max*)
	theta	(orientation of long axis of area)
	c	(circularity)

pset

see also: **analyse**

Defaults and Ranges

keys/options	defaults	range
[plist]	*ppl* held in the variable *pplist*	valid picture number
sort	particles ordered as in *ppl*	valid parameter name (see Appendix D)
if	true	valid Semper expression
unless	false	valid Semper expression
index	1	integer in range 1 to number of selected particles
xref,yref,id, parent,holes, background,contact, xmin,xmax,ymin, ymax,hferet,vferet, aferet,bferet,hproj, vproj, perimeter,area, xcen,ycen,mmin, mmax,angle,circularity	*none*	
ascending/ descending	*ascending*	

pshow

keys:	[plist]	*<number>*	source particle parameter list
	sort	*<parameter name>*	name of parameter by which particles are sorted
	If	*<expression>*	logical expression specifying which particles to include
	unless	*<expression>*	logical expression specifying which particles to exclude
	saturation	*<number>*	colour saturation of highlighted particles
	hue	*<number>* *<n1>,<n2>*	colour hue of highlighted particles hue for first and last highlighted particle
options:	ascending/descending		if ***sort***, sort on specified parameter in ascending or descending order

pshow operates on the particle parameter list (*ppl*) produced by the **analyse** command and an unscaled display of the corresponding segmented picture. **analyse** records 25 particles for each particle and stores these details in a *ppl*. **pshow** allows you to alter the display look-up tables to highlight any particles that meet the conditions that you specify. Refer to *Appendix D, Particle Parameters* for a list of the parameters recorded by **analyse** and the names that should be used to refer to them in **If/unless** expressions and the **sort** key.

Examples

```
analyse...segment...
display psegment noscale
pshow if area>100
```

This sequence of commands analyses a field and displays the segmented picture, in an installation with a display device 8 bits deep. **pshow** shows the the larger particles as white, and the smaller as grey, against a black background.

```
pshow if id=25
```

This command highlights the particle with an identifier of 25.

```
pshow if hferet<50 hue 0,240 saturation 0.8 sort area
```

This command highlights the selected particles with slightly de-saturated colours, spanning a range of hues from red to blue with increasing area.

pshow

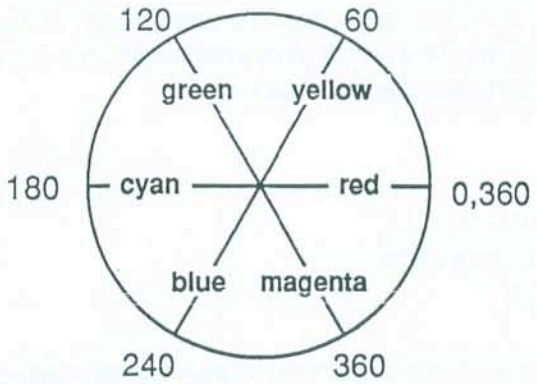
Description

pshow relies on false colour look-up table (*lut*) manipulation and can only be used when the number of particles to be highlighted is no larger than the number of *lut* entries, that is, the number of distinct grey levels that each display frame can store. Type **show luts** to list the currently defined look-up tables.

To use **pshow** you need to display the segmented picture produced by **analyse** as an unscaled picture. The best way to do this is to use the following command:

```
display...noscale
```

pshow works by using a look-table that shows the particles you select as white (in default) and all other particles as grey, against a black background. If you specify a non-zero value for the **saturation** key the selected particles are displayed in colour at the specified colour saturation level, using the hue given by the **hue** key. If you specify a pair of values for **hue**, **pshow** uses the specified range of colours to highlight the selected particles. The hue values form a spectrum through the range 0 to 360 degrees, which is illustrated below.



Use the keys **if/unless** to set criteria for selecting/rejecting particles. The **sort** key allows you to select a particle parameter on which to sort the particle list. The parameter names you can specify are **xref, yref, id, parent, holes, background, contact, xmin, xmax, ymin, ymax, hferet,vferet, aferet, bferet, hproj, vproj, perimeter, area, xcen, ycen, mmin, mmax, angle** and **circularity**.

Notes

see also: **analyse, show luts, display**

pshow

Defaults and Ranges

keys/options	defaults	range
[plist]	*ppl* held in the variable *pplist*	valid picture number
sort	particles ordered as in *ppl*	valid parameter name (see Appendix D)
if	true	valid Semper expression
unless	false	valid Semper expression
saturation	saturation 0 (no colour)	real number in range 0 to 1
hue	red	real number in range 0 to 360
ascending/ descending	ascending	

ptype

keys:	**[plist]**	*<number>*	source particle parameter list
	sort	*<parameter name>*	name of parameter by which particles are sorted
	If	*<expression>*	logical expression specifying which particles to include
	unless	*<expression>*	logical expression specifying which particles to exclude
	Index	*<number>*	if **all**, ranking of the selected particle
options:	**xref, yref,id,parents,holes, background,contact,xmin, xmax,ymin,ymax,hferet, vferet,aferet,bferet,hproj, vpoj,perimeter,area,xcen, ycen,mmin,mmax,angle, circularity**		list the values of the specified parameters (up to a maximum of 5 parameters)
	all		print all parameters for one particle
	ascending/descending		if ***sort***, sort on specified parameter in ascending or descending order

ptype operates on the particle parameter list (*ppl*) produced by the **analyse** command. **ptype** lists the values of one or more of the measured parameters for any particle or set of particles. Refer to *Appendix D, Particle Parameters* for a list of the parameters recorded by **analyse** and the names that should be used to refer to them in **If/unless** expressions.

Examples

```
analyse...; ptype area perimeter
```

This example lists the area and perimeter for all particles in the current picture.

```
ptype id angle unless circularity>.5 sort mmax
```

This command types the identifier and 'long' direction of any particles with a circularity of less than 0.5, in sort order of the 'long' axis length.

```
ptype 23 all if id=4
```

This command types all parameters for the particle with an identifier of 4.

Description

analyse records 25 particles for each particle and stores these details in a particle parameter list (*ppl*). **ptype** types a list of the specified parameters for a particle or number of particles recorded by the **analyse** command. You can ask for up to 5 of the 25 parameters at one time. Alternatively, use the **all** option to see a list of all parameters for a single particle.

ptype

Use the keys **if/unless** to set criteria for selecting/rejecting particles. The **sort** key allows you to select a particle parameter on which to sort the particle list. The parameter names you can specifying are **xref, yref, id, parents, holes, background, contact, xmin, xmax, ymin, ymax, hferet,vferet, aferet, bferet, hproj, vproj, perimeter, area, xcen, ycen, mmin, mmax, angle** and **circularity.** You use these same parameter names as options to specify which parameter values you wish to be typed.

If you are using the **all** option, you can specify an **index** key to specify the ranking (1, 2, 3...) of the particle that you want within the list, dictated by the keys **if, unless** and **sort.** Note that if you initially type the command **pset count** you can then write a loop reporting parameters for all listed particles in turn, for example:

```
pset count if area>99; for index=1,n; ptype all if area>99; loop
```

Notes

see also: **analyse, pset**

Defaults and Ranges

keys/options	defaults	range
[plist]	*ppl* held in the variable *pplist*	valid picture number
sort	particles ordered as in *ppl*	valid parameter name (see Appendix D)
if	true	valid Semper expression
unless	false	valid Semper expression
index	1	integer in range 1 to number of selected particles
ascending/ descending	ascending	

qget

This command is specific to...
PC + Quantimet 520 greystore

keys:	**[to]**	*<number>*	output Semper picture
	name	*'<text>'*	source image file (Quantimet 520 format)

Use **qget** to read a Quantimet 520 format image file into a Semper picture.

Examples

```
qget name '520PIC' to 2
```

This command reads the Quantimet 520 format disc file called *520PIC.IMG* into Semper picture 2.

```
qget 2 name '520PIC.IMG'
```

This command is identical in effect to the example given above.

Description

If you do not specify a filename using the **name** key, Semper prompts for a name. Note that the file extension defaults to *.IMG* but you can specify other extensions explicitly.

qget searches for files in the current directory and then throughout the DOS PATH. You can avoid a time-consuming path scan by specifying a full path name with the filename.

Notes

see also: **qput**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
name	*none;* if you do not specify a name, Semper prompts for a filename	valid filename

qput

This command is specific to...
PC + Quantimet 520 greystore

keys:	**[from]**	*<number>*	source Semper picture
	name	*'<text>'*	output image file (Quantimet 520 format)
options:	**new**		over-write existing Quantimet 520 disc file

Use **qput** to write a Semper picture to a disc file in Quantimet 520 format.

Examples

```
qput name '520PIC' from 3
```

This command writes Semper picture 3 out to a Quantimet 520 format disc file called *520PIC.IMG.*

```
qput 3 name '520PIC.IMG'
```

This command is identical in effect to the command example given above.

```
qput name '520PIC' from 3 new
```

This command writes Semper picture 3 out to a Quantimet 520 format disc file called *520PIC.IMG.* The **new** option allows an existing file with the name *520PIC.IMG* to be overwritten.

Description

If you do not specify a filename using the **name** key, Semper prompts for a name. Note that the file extension defaults to *.IMG* but you can specify other extensions explicitly. The **new** option allows you to overwrite an existing file of the same name.

Complex, floating point and integer Semper pictures are first scaled to have the range 0 to 255 before being converted to Quantimet 520 format. Byte form pictures are not scaled. Note that a Quantimet image that is derived from a complex form Semper picture may not have data over the entire range of 0 to 255. Although Semper scales the data so that values that correspond to the maximum real or imaginary value are mapped to 255 and values corresponding to the minimum real or imaginary values are mapped to 0, only the *real* part is contained in the Quantimet 520 format image.

Notes

see also: **qget**

qput

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none;* if you do not specify a name, Semper prompts for a filename	valid filename

qshade

This command is specific to...
PC + Quantimet 520 greystore

options:	**on/off**	turn the shading corrector on/off
	set	set the shading corrector to the current image (also turns off the shading corrector)

Use **qshade** to control the Quantimet 520 shading corrector. You use the shading corrector to reduce the effect of background image distortions.

Examples

```
qshade set
```

This command sets the shading corrector store to the current image. (It then turns the shading corrector off).

```
qshade on
```

This command activates the shading corrector.

Description

Use the **on** option to turn the shading corrector on. Use the **off** option to turn the corrector off. The **set** option sets the shading corrector to the current image. It is this image that is used for shading correction when you specify **qshade on** for another image. Note that the **set** option also turns the shading corrector off.

ramps

keys:	[]	*<number>*	picture/partition/frame to be filled by ramp
	size	*<x>, <y>*	dimensions of subregion to be filled
	position	*<x>, <y>*	position/offset of subregion
	times	*<number>*	number of ramps to display
options:	**picture/partition/frame**		fill picture/partition/frame with a ramp
	left/right, top/bottom		subregion position
	full		produce 'full' ramp, with unit intensity increments
	view		switch view to make the display region visible

Use **ramps** to generate full-range grey level ramps on the display, for test purposes or to help you to adjust the monitor contrast and brightness controls.

Examples

```
ramps frame 1 view
```

This command fills the entire frame with four ramps, and presents it on the monitor.

```
ramps partition dis:5 size 256,20 top
```

This command writes ramps of the indicated size at the top of the partition.

```
xwires dis:3 region; ramps @region
```

This command writes ramps to the region of picture *dis:3* marked by the cursor.

Description

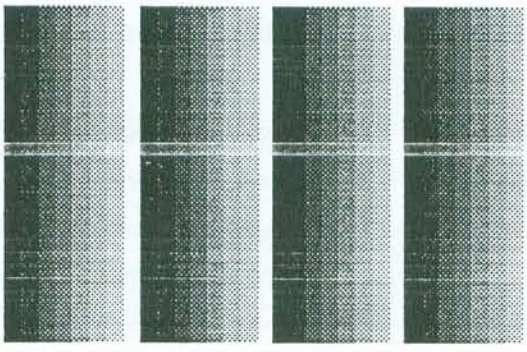
By default, **ramps** works on *display*, in picture coordinate mode. You can also use **ramps** to display a ramp in a partition or frame, for example, **ramps partition dis:7**. You can specify a subregion using the standard subregion keys: **size**, **position**, **left**, **right**, **top** and **bottom**. Refer to *Appendix C, Semper Keys and Options* for detail of the subregion keys.

By default, the **ramps** command generates four grey-scale ramps, rising from left to right, across the region width. You can vary the number of ramps using the **times** key, and if you use the **full** option the ramps are generated with unit intensity increments. Use the general **view** option if you want to ensure that the ramps are visible.

Note that the values of *min* and *max* do not affect **ramps**. The ramps are not recorded as a display picture; if you want to store them, use the **create display** command after the **ramps** command.

ramps

The diagram below illustrates the default grey level ramps, created using the **ramps** command.



Notes

see also: **create display**

Defaults and Ranges

keys/options	defaults	range
[ ]	*display* if picture or partition, current frame if frame, held in the variable *cframe*	valid picture/partition/frame number
size	entire picture/partition/frame	less than or equal to the size of the picture/partition/frame
position	position 0, 0	within bounds of picture/ partition/frame
times	4	positive integer
picture/partition/ frame	picture	
view	view disabled	

rank

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	over	*<number>*	pixel neighbourhood size (3 or 5)
	position	*<number>*	ranking position required as output
options:	**erode/dilate**		apply erosion or dilation filter

Use **rank** for local median, minimum or maximum filtering, and multi-level erosion and dilation. This command ranks the pixels in a small neighbourhood around a given pixel in order of increasing brightness, and replaces the pixel by the 1st, 2nd.. middle.. or last ranked pixel, depending on the options selected.

Examples

```
rank display
```

This command applies a 3x3 median filter to the display picture, removing isolated spurious values without softening edges.

```
for n=1,3; rank erode over 5; loop
```

This command erodes all bright regions of the current picture by about 6 pixels.

```
rank position 9
```

This command replaces each pixel by the maximum (brightest) within its 3x3 neighourhood.

Description

rank sorts the pixels within a 3x3 pixel neighbourhood (or 5x5 if you specify 5 for the **over** key) into ascending numerical order, and replaces the source pixel by the value found to lie at a particular place in the ranking.

By default, the output is the middle ranked, or median, value. This is a widely used means of removing isolated spurious values, that is, removing one kind of noise, without degrading the sharpness of lines and edges in the picture in the way that linear filters such as **lmean** or **flr** do.

Selecting a low-ranked value causes dark regions of a picture to expand, while conversely a high-ranked value causes bright regions to expand. Therefore you can achieve a multi-grey level form of erosion and dilation. The **position** key gives you complete control of the output ranking,

rank

while options **erode** and **dilate** select positions just short of the extremes (2 and 8 for 3x3 kernels, 4 and 21 for 5x5 kernels) as this makes the result less noise sensitive and achieves better rounding of corner features. (For 1–D pictures, **erode** and **dilate** simply select the extreme positions).

The algorithm used by **rank** requires data in the range 0–255. You can apply it to other data reasonably easily by **display**ing the data, and then recreating the display with *min*=0 and *max*=255 (see **create**). If you want your data treated as binary, use **calculate** to threshold it suitably, or output to the display with *min*=0 and *max*=1. **rank** runs slightly faster with such data.

Notes

multi-layer pictures:	faulted
forms use internally:	integer
see also:	**calculate, flr, lmean**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	source picture	valid picture number
over	3 neighbouring pixels	3 or 5
position	middle ranked for median mode, low value for erosion and high for dilation; but exact value depends on **over** and whether source is 1-D or 2-D	positive integer
erode/dilate	median filtering	

rcf

keys:	**[from]**	*<number>*	first source picture
	[to]	*<number>*	output picture
	with	*<number>*	second source picture
	over	*<number>*	width in pixels of bands of Fourier components compared at each spatial frequency
	radius	*<number>*	radius of real-space mask applied to compared images; affects significance threshold only
options:	**phaseresidual**		produce phase-residual instead of radial correlation function

rcf performs a *radial correlation function* on an image. You use **rcf** to assess the useful resolution of an image of which you have two copies with independent noise, by tabulating how well their Fourier components agree at different spatial frequencies. You can choose either cross-correlation coefficients or phase-residual values.

Examples

```
rcf 1 with 2 to 3; display times 4
```

This command displays the *radial correlation function* between transforms 1 and 2, with a significance threshold curve superposed.

```
rcf 1 with 2 phaseresidual over 3
```

This command puts in picture 999 the *phase residual function* between transforms 1 and 2, calculated over bands 3 pixels wide.

```
for n=1,2; mask n radius 45 width 5; fourier; loop
rcf 1 with 2
```

This sequence of commands generates the *rcf* for two *Images*, including the smooth-edged masks necessary to prevent spurious high frequency mismatches.

Description

You must specify two source pictures with the **rcf** command, using the keys **from** and **with**. These source pictures must be half-plane Fourier transforms of square pictures.

rcf

rcf produces 1-D output, with zero spatial frequency at the left (the origin) and maximum spatial frequency (0.5/pixel) at the right. You can apply two types of function using the **rcf** command:

- radial correlation function
- phase-residual function

The *radial correlation* values, generated by default, are standard cross-correlation coefficients between corresponding (complex) Fourier components, normalised to lie in the range –1 to 1.

The *phase-residual* values are the *rms* phase difference (in radians) between corresponding Fourier components, with the differences weighted in proportion to the sum of the two relevant moduli. Use the **phaseresidual** option to generate phase-residual values.

By default, the Fourier components of each picture are grouped into bands that are one pixel wide. Use the **over** key to specify a greater width, as in the second command example, where wider overlapping bands are used, which effectively smooth the output.

You can use a smooth-edged mask, as in the last command example, to make both images fade out smoothly towards the picture edges beyond the features of interest before using **rcf**. This prevents opposite-edge mismatches or sharp features at mask edges from generating artificially raised high-frequency agreement.

The output of the **rcf** command has two layers. The second layer contains a *limiting significance threshold* curve to assist assessment of the first layer. (Use the command **display layer 1** to see the *rcf* only, without the significance threshold superposed). For the cross-correlation mode, the threshold is $2/\sqrt{n}$ where n is the number of independent Fourier components in the relevant band. Where correlation levels are low, this is twice the expected standard deviation of the correlation values. For the *phase-residual* mode, it is simply a constant value of $\pi/4$ – a less optimistic threshold corresponding to a correlation level of 0.71.

If the original images are severely masked off, the number of independent Fourier components is reduced, and **rcf** adjusts the significance threshold curve appropriately, if you tell it the mask **radius** (for example, **rcf 1 with 2 radius 45** in the last command example).

Notes

multi-layer pictures:	faulted
forms use internally:	complex
see also:	**fourier, mask**

rcf

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in variable *select*	valid picture number
[to]	picture 999	valid picture number
with	*none*	valid picture number
over	width 1	positive integer
radius	infinite	positive integer
phaseresidual	produce *radial correlation function*	

read

keys:	[to]	*<number>*	picture to be read
	name	*'<text>'*	file containing picture data
options:	unformatted		use Fortran's unformatted reading mode
	again		if **name** is omitted and was used in a command to open a file for reading, open the same file again

Use **read** to read pictures from files that you created using a non-Semper program, or to recover pictures from files that you created using the **write** command.

Examples

```
read 20 name 'sec34'
```

This command reads picture 20 from the file sec34.dat.

```
read unformatted name 'binary.pic'
```

This command prompts at the terminal for the file name, and reads to the current picture, using the *unformatted* option. If you do not specify an extension for the filename the default extension.*unf* is assumed.

Description

You can use the **read** command to transfer images between Semper systems on different host computers. In this case the images must be written to file using the **write** command. You must specify the **unformatted** option when using the **read** command, if the same option was used with the **write** command.

If you do not specify an extension for the filename, a default extension of *.dat* for formatted files and *.unf* for unformatted files is assumed.

The **again** option allows you to re-open a file without having to specify the file name again with the **name** key. If the **name** key was not used before to open a file for reading, the **read** command will prompt for the file name.

You can also read in images from non-Semper systems, provided that the files are suitably formatted. A precise description of the possible file formats can be found in the document:

Semper 6 REA/WRITE file format

read

which can be obtained from Synoptics.

Note that to recover pictures that were output with the **save** command simply use the **assign** and **copy** commands.

Notes

multi-layer pictures:	fully supported
forms use internally:	integer, fp, complex
see also:	**assign, copy, write**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename
unformatted	formatted	

read

keys:	[to]	*<number>*	picture to be read
	name	*'<text>'*	file containing picture data
options:	**unformatted**		use Fortran's unformatted reading mode

Use **read** to read pictures from files that you created using a non-Semper program, or to recover pictures from files that you created using the **write** command.

Examples

```
read 20 name 'sec34'
```

This command reads picture 20 from the file *sec34.dat.*

```
read unformatted name 'binary.pic'
```

This command prompts at the terminal for the file name, and reads to the current picture, using the *unformatted* option. If you dp not specify an extension for a filename the default extension *.dat* is assumed.

Description

You can use the **read** command to transfer images between Semper systems on different host computers. In this case the images must be written to file using the **write** command. You must specify the **unformatted** option when using the **read** command, if the same option was used with the **write** command.

You can also read in images from non-Semper systems, provided that the files are suitably formatted. A precise description of the possible file formats can be found in the document:

Semper 6 READ/WRITE file format

which can be obtained from *Synoptics.*

Note that to recover pictures that were **saved**, simply use the **assign** and **copy** commands.

Notes

multi-layer pictures:	fully supported
forms use internally:	integer, fp, complex
see also:	**assign, copy, write**

read

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename
unformatted	formatted	

reclass

keys:	[]	*<number>*	picture to be reclassed
options:	**image/fourier/spectrum /correlation/undefined/ walsh/histogram/plist/lut**		new picture class
	list/curve		type of position list, if ***plist***
	open/closed		type of curve, if ***plist curve***
	as		ignored

Use **reclass** to override Semper's default classification of pictures, to make Semper treat a picture as if it contained data of a different or special kind.

Examples

```
reclass 53 plist
```

This command causes Semper to treat picture 53 as a *Plist* picture (this is sensible only if you have set its contents appropriately).

```
ps 1; reclass image; fourier; weight with..; image
```

This command reclasses a *Spectrum* picture so that it can be Fourier-filtered.

Description

reclass only alters pictures labels; it does not effect the data in the pictures or any other characteristic, such as the origin. You can specify any of the class names, however you cannot reclass *Macro* pictures, or change other classes to *Macro*. See *Appendix A: Picture Types* for detail of picture classes.

If you are reclassing a picture as a *Plist* picture, you can use the additional options **list**, **curve**, **open** and **closed** to force the type as follows:

`reclass ... plist` or `... plist list`	list
`reclass ... plist curve` or `... plist open curve`	open curve
`reclass ... plist closed curve`	closed curve

Note that the **as** option is provided to make the command line more readable, for example, the following commands perform identical functions:

```
reclass 51 as lut
reclass 51 lut
```

reclass

Defaults and Ranges

keys/options	defaults	range
[ ]	current picture, held in the variable *select*	valid picture number
image/fourier/ spectrum...	*Image*	
list/curve	list	
open/closed	open	

reinitialise

keys:	**device**	*<number>*	number of device to be reinitialised
	slots	*<number>*	number of directory slots to be reserved
options:	**verify**		verify process at the console

Use **reinitialise** to wipe a picture disc clean and also to alter the number of slots in its directory.

Examples

```
reinitialise slots 100
```

This command reinitialises the current device *cd* with a 100 slot directory and verifies the process at the console.

```
reinitialise device 5 noverify
```

This command reinitialises device 5 with the largest possible directory. Information about the process is not displayed at the console, because of the **noverify** option.

Description

reinitialise asks for confirmation at the terminal before reinitialising a picture disc and cannot be used on a write-protected disc (see the **wp** command).

You cannot reinitialise disc devices that are not picture discs. Program libraries must be deleted and re-created if necessary (but see **compress** before proceeding). Help libraries are managed by an independent help library utility.

By default, information about the reinitialising process is sent to the console. To suppress this information, use the **noverify** option.

Notes

see also: **compress**

reinitialise

Defaults and Ranges

keys/options	defaults	range
device	current device, held in variable *cd*	valid device number
slots	maximum useful or possible if picture device; 32 if program library	positive integer
verify	verification on	

rename

keys:	**program**	*'<text>'*	old name for program
	as	*'<text>'*	new name for program
options:	**verify**		verify process at the console

Use **rename** to change the name of a program in a program library, without altering or copying the program text.

Examples

```
rename program 'combine' as 'old$combine'
```

This command renames program *combine* to *old$combine*. The command process is verified at the console.

```
rename program 'combine' noverify
```

This command prompts at the terminal for a new name for program *combine*. Information about the command process is not sent to the terminal because of the **noverify** option.

Description

If more than one copy of a program exists in your libraries, **rename** acts on the first one found in the current search order.

As Semper only considers the first three letters of a command name, **rename** cannot be distinguished from **renumber** (which renumbers pictures) except by the keys following the **ren...** command. Therefore you need to supply at least one of the keys **program** or **as** (Semper prompts at the terminal for an omitted key).

By default, Semper sends information about the command process to the console. Use the **noverify** option to suppress this information.

Defaults and Ranges

keys/options	defaults	range
program	*none*; prompts if interactive	valid program name
as	*none*; prompts if interactive	valid program name
verify	verification on	

renumber

keys:	**[from]**	*<number>*	old number for picture
	[to]	*<number>*	new number for picture
options:	**verify**		verify the command process at the terminal

Use **renumber** to change the number of a picture, without altering or copying its contents.

Examples

```
renumber 1 to 10
```

This command changes the number of picture 1 to 10 and verifies the change at the console.

```
renumber to 772 noverify
```

This command changes the number of the current picture to 772. Semper does not display information about the command process at the console, because of the **noverify** option.

Description

Note that you can only change picture numbers within a device and that the specified new number must not be the same as an existing picture number.

Defaults and Ranges

keys/options	defaults	range
[from]	current picture number, held in the variable *select*	valid picture number
[to]	old picture number	valid picture number
verify	verification on	

report

options:	**error/trap**	select the last error message/error trap for output
	exit	force exit from command sequence by setting error 10

Use **report** to report the last error message or to type the message for the last error that was trapped.

Examples

```
report error
```

This command types the last error message.

```
report trap exit
```

This command types the error message for the last trap and forces an exit from a command sequence by setting error 10.

Description

Although you can use **report** to repeat an error message that has disappeared from the screen, **report** is mainly used to generate an error message after a trap has been taken. For example:

```
trap = -1 examine picno          ;! trap any error
n = rc                           ;! save error code
if n = 30 jump nopicture         ;! test for no picture
if n = 34 jump nodisc            ;! test for no device
report trap exit                 ;! other error - report
```

By default, **report** prints the last error message, or the last trap message if an error has not been reported yet. Note that **report** only gives the final line of any error report sequence – any context or prior explanatory messages are not preserved.

return

return takes no arguments

Use **return** to return execution from a program or *run* file. (The **end** command at the end of a run file is treated as a **return**. Use **end** to mark the end of a program or run file).

Examples

The following program fragment illustrates the structure of a typical program:

```
seriesrange()
Local s,s2,m,m2
   if set(s) & set(s2) jump scan
   ask 'First, last picture in series ' s,s2
   unless set(s) & set(s2) return
   scan: survey s notype; m=min,max; for n=s+1,s2; survey n notype
         m=min(m,min),max(m2,max); loop
   min=m max=m2; type 'Series range ',min.max
   end
```

This example program determines the minimum and maximum of a series of pictures and also illustrates the use of local variables and terminal dialogue.

rewind

keys:	device	<*number*>	rewind specified tape device

rewind initiates the rewinding of a tape device, and can be used to avoid later delays in some installations.

Examples

```
rewind device 4
```

This command rewinds device 4.

Description

You do not need to use the **rewind** command, as Semper rewinds or backspaces tapes whenever it is necessary. However, if your installation returns control to the terminal as soon as a rewind operation has begun, you may be able to save time by anticipating a rewind that will be needed later.

Note that **rewind** will not cause loss of data or position.

Defaults and Ranges

keys/options	defaults	range
device	current device, held in the variable *cd*	valid device number

rf

keys:	[from]	<number>	source picture
	[to]	<number>	output picture
	a	<number>	filter width parameter

Use **rf** to apply two point recursive filters to pictures. The effect is smoothing or sharpening as if by convolution with an infinitely wide point response.

Examples

```
rf 1 to 2
```

This command mildly smooths picture 1 to picture 2.

```
rf to display a 0.9
```

This command displays a strongly smoothed version of the current picture.

```
rf a -0.3
```

This command replaces the current picture by a (mildly) sharpened version.

Description

Use the **a** key to specify a filter width in the range –1 to 1. A positive value for the filter smooths the picture (retains the low frequencies only), and a negative value sharpens it (retains the high frequencies). The filter action becomes progressively stronger as it approaches 1 or –1.

rf is based on a simple 1-D filter recursive filter. An output pixel *f'(i)* is calculated as a weighted average of the original value *f(i)* and the last calculated filtered value *f'(i–1)*:

$$f'(i) = a.f'(i-1) + (1-a).f(i)$$

This has a one-sided point response (negative exponential in shape). **rf** applies it in each direction in turn along rows and columns (along the row only if 1-D) so as to achieve a symmetric point response.

The 1-D spatial frequency response is:

$$\frac{(1+a^2-2a)}{(1+a^2-2a.\cos(\frac{2\pi.i.k}{n}))}$$

for a transform pixel *k* points from the origin and where *n* is the picture dimension. This is unity (1) for

rf

zero frequency, and falls or rises to $\left(\frac{1-a}{1+a}\right)^2$ at the maximum frequency. For positive values of the weighting factor **a**, the point response has the approximate form:

$$e^{-\frac{|x|}{w}} \text{ with } w = \frac{1}{|(\ln(a)|} \text{ pixels, which has a full width at half height of about } \frac{1.4}{|\ln(a)|}$$

(The 2-D responses are simply the product of the 1-D responses in each direction).

Notes

multi-layer pictures: all layers processed
forms used internally: fp

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
a	filter width .3	real number in range –1 to 1

rgb

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	lut	*<number>*	if not **monochrome** and output is not to display, output picture for look-up table data
	levels	*<n1,<n2>,<n3>*	if ***false***, number of levels for red, green and blue data
options:	**monochrome/false/ sampled/averaged**		specify type of conversion from full colour to monochrome or false colour picture

The **rgb** command converts a full colour image into one of a number of forms suitable for displaying the image on a monochrome or false colour framestore. Each output format makes a different compromise in trying to preserve as much as possible of the spatial and colour information in the original image.

Examples

```
rgb 1 2; lut create monochrome; display 2
```

This command converts the full colour byte image in picture 1 into a monochrome picture 2 and then displays the result.

```
rgb 3:1 false
```

This command sends a false colour conversion of the source image directly to the display and modifies the current look-up table in order to view the result.

```
rgb 3:1 false levels 4,5,3
```

This command performs the same function as the above example, but explicitly assigns the number of levels for the red, green and blue data.

```
rgb 3:1 sampled to 3:2 lut 3:3
```

This command outputs a sampled or dithered representation of the source image to picture 3:2 and outputs the corresponding display look-up table to picture 3:3.

rgb

Description

The **rgb** command allows you to generate a representation of a full colour image that can be viewed on monochrome or false-colour displays in the following ways:

- **monochrome**
- **false**
- **sampled**
- averaged

The default option is **monochrome**.

Note that the source picture to the **rgb** command must be in byte form and have precisely three layers.

The conversions for the **monochrome** and **false** options combine the red, green and blue information for each pixel to produce a single output value that will fit within the range of output values for the display. Both of these conversions preserve full spatial resolution at the expense of the combined colour and intensity resolution. Note that you can use the commands **show system** or **lut enquire** to find out how many output levels your display will support.

For the **monochrome** option, the result for each pixel is a weighted sum of the red, green and blue intensities:

*output value = 0.299 * red + 0.587 * green + 0.114 * blue*

This retains the luminance (perceived intensity) and discards the colour information.

For the option **false**, the red, green and blue intensities are converted to the corresponding level within the number of intensity levels specified for red, green and blue by means of the **levels** keys (represented by *nr*, *ng* and *nb* below). The three discrete colour intensities are then combined into a single output value:

*r = round((nr – 1) * red / 255)*
*g = round((ng – 1) * green / 255)*
*b = round((nb – 1) * blue / 255)*

*output value = (ng * nb * r) + (nb * g) + b*

This approach sacrifices intensity resolution to introduce some colour information.

rgb

If you omit the **levels** key when you specify the **false** option, Semper provides a default that is suitable for your particular display. For an 8-bit display, the default is 8,8,4 and for a 7-bit display the default is 4,8,4. For any other displays, the levels are chosen to give as near as possible a 2:2:1 ratio between the number of red, green and blue intensity levels. When using the **levels** key, you must specify at least 2 levels for each colour and the product of the three numbers must not exceed the maximum number of output levels for the display.

The conversions made for the **sampled** and **averaged** options sacrifice some spatial resolution to give full resolution for colour. Each output pixel displays a red or a green or a blue intensity that is either taken directly from the corresponding source pixel (**sampled** option) or is obtained by taking the average of three adjacent pixels in the source image (**averaged** option). The results are approximately the same for both options except that the **averaged** option will avoid any of the aliasing effects that the **sampled** option may introduce when the source image is sampled. Although much better colour resolution is obtained by this approach, the brightness of the displayed image will be reduced to about a third (since only one of the three colour phosphors is excited at each output pixel on the display).

When output from the **rgb** command is displayed, you have to load a display look-up table that corresponds to the output representation used. For the **monochrome** option, a grey-scale ramp is required (use the command **lut create monochrome** or **lut reset** to generate one). For the other conversions, the **rgb** command generates the appropriate look-up table. If the image is output to the display, the current look-up table is modified. Otherwise, the look-up table date is output to the lut class picture specified by the **lut** key.

Notes

see also: **lut**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number (the picture must be 3 layer and in byte form)
[to]	current display picture, held in the variable *display*	valid picture number
lut	999	valid picture number
levels	if 8-bit framestore 8,4,4 if 7-bit framestore 4,8,4 otherwise dependent on number of framestore levels	positive integers (at least two values are required)
monochrome/false/ sampled/averaged	monochrome	

rhistogram

```
rhistogram 1 type band 4,7 type width 40 aspect 2
```

This command forms a 2-D histogram of picture 1 and displays the output on the terminal. The output pictures is 40 characters wide and is displayed with an aspect ratio of 2:1.

Description

A 2-D histogram (or feature space plot) is formed from the two specified layers of the source picture. Use the **band** key to specify the two layers. You can specify the number of histogram channels using the **channels** key. The number of histogram channels are limited by the size of a Semper row buffer (proportional to the squre root). Note that the number of channels determines the size of the output picture. If you do not specify a number, **rhistogram** defaults to the maximum possible number of channesl. By default the **rhistogram** command displays the output histogram as a 2D image.

If you use the **preset** option, you need to specify the values using the variables *min, mi2* and *max, ma2* to delimit the ranges over which the command operates. If you do not specify **preset** then the minimum and maximum variables are set to the values found during the histogram creation. If you specify **type**, the command displays a character based histogram on the terminal, rather than displaying a histogram in graphical form on the display.

Notes

variables used:	*min, mi2, max, ma2* (if **preset**, pixel range shown by channels)
variables set:	*min, mi2, max, ma2* (unless **preset**, pixel range channels)

rhistogram

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current display picture, held in the variable *display*, histogram shown in graphical form	valid picture number
channels	maximum possible value	possible integer
band	*none;* band must be specified	possible integer
times	1	positive integer
aspect	default given by the **page** command	positive integer
width	default given by the **page** command	positive real number
position	0, 0	within the bounds of the picture (integers)
size	whole picture	less than or equal to the size of the picture (integers)
repeating	interpolate between histogram counts when magnifying	
letter	lettering on	
border	bordering on	
type/log	histogram shown in graphical form on display	
verify	verification off	

rotate

keys:	[from]	<number>	source picture
	[to]	<number>	output picture
	angle	<number>	angle by which picture is rotated clockwise, in radians
	value	<number>	constant value used for filling regions that overflow rotated source

Use **rotate** to rotate pictures by 'large' angles (that is angles between 45 and 135 degrees). Note that **rotate** is much quicker than **extract** in this instance.

Examples

```
rotate display angle pi/2
```

This command rotates the display 90 degrees clockwise.

```
rotate 50 to 51 angle rad(50) value 57
```

This command rotates picture 50 to 51 by 50 degrees anti-clockwise, padding regions that overflow the source picture with the value 57.

Description

The picture to be rotated must be square, and its size must be one of the factorisable sizes listed in response to the command **show sizes**. You can of course **cut** a picture, if necessary, to a suitable larger size, and then **rotate** and **cut** again afterwards. The rotation is about the centre of the picture, not its origin, but the recorded origin is updated appropriately (rounded to the nearest pixel).

The algorithm used involves a relatively fast multi-radix transposition step, and the execution time is independent of the involved angle. The field of view is reduced slightly – only by 2% or so for angles in the range 60 to 120 degrees, but by progressively larger amounts for angles outside this range. For this reason, **rotate** faults angles within 30 degrees of 0 or 180 degrees.

Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
restrictions:	picture must be square and a factorisable size

rotate

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
angle	*none*	real number in range 30 to 150 and 210 to 330
value	0	positive integer

rs232

Due to unresolved problems associated with multiplexed interrupts, access to the COM2 port is not supported. Any attempt to use this port will be faulted. You should connect your mouse to the COM2 port (most mouse drivers will support this) and collect data on the COM1 port.

This command is specific to ...
IBM PC XT or AT compatible

keys:	**port**	*<number>*	comms port to use, 1=COM1, 2=COM2
	band	*<number>*	baud rate
	stop	*<number>*	number of stop bits
	bits	*<number>*	number of data bits
options:	**on/off**		switch comms port on or off
	clear		clear any error conditions
	empty		empty (flush) data buffers
	parity/noparity		use or ignore parity bits
	even/odd/none		select even or odd or no parity
	stick		select forced parity
	status		return port status in Semper variables
	verify		if **status**, type port status information on the console

The **rs232** command allows you to control and access the standard communications ports, COM1 and COM2 directly. Data can be sent and received asynchronously, allowing you to control external instrumentation in parallel with other operations within Semper.

Examples

```
rs232 on port 1 baud 9600 stop 1 bits 8 noparity
```

This command sets up COM1 at 9600 baud, 8 data bits, 1 stop bit and no parity.

```
rs232 status port 1
```

This command returns the status of COM1 in Semper variables.

```
rs232 clear
```

This command clears any outstanding errors for the currently selected port.

rs232

Description

The command, together with **rsget** and **rsput**, gives you extensive control of the standard RS232 communications ports, COM1 and COM2. On startup, Semper reads the current configuration of the ports, and will reset them on exit. Data transfer takes place asynchronously under interrupt contol, thus ensuring that incoming data is not missed and allowing other processing while data is being transmitted. Both ports can be controlled independently and at the same time. Semper will not allow you to use a port which apears to be in use by another program (for example a mouse driver), however, you should still ensure that no resident programs attempt periodic acces to the port.

Semper maintains a concept of the currently selected port. You specify which one with the **port** key. Before using the port to send or receive data you must enable it with the **on** option. You can select the baud rate, data bits, stop bits and parity at any time the port is off, or when the port is being switched on. You turn the port off with the **off** key.

The baud rate is set with the **baud** key. Select a value that matches your external equipment, but be aware that rates above 9600 may not work due to hardware limitations. The number of data bits is set with the **bits** key and can be 5,6,7 or 8. The number of stop bits is set with the **stop** key and can be 1, 1.5 (with 5 data bits) or 2.

The **parity** option is used to introduce parity settings. These can be even or odd (**even** or **odd** options), forced (**stick** option) or none (option **none** or option **noparity**).

The **clear** option is used to clear any outstanding communcations errors.

The **empty** option is used to flush the data buffers. This is useful when synchronsing read operations where unsolicitated data may have been sent previously.

The option **status** causes the current port status to be returned in Semper variables *po, pb, ps, pc, pr, pt, p$s* and *p$e*. The option **verify** can be used with the **status** option to verify the current settings on the terminal.

rs232

Notes

variables set:

p$o	on/off state, 0=off, 1=on
p$b	baud rate
p$s	number of stop bits
p$c	number of data bits
p$r	number of characters in the receive buffer
p$t	number of characters in the transmit buffer
p$p	parity settings (bit 0 is least-signficant bit) bit 0=odd/even parity, 0=even, 1=odd bit 1=forced parity, 0=off, 1=on bit 2=parity checking, 0=off, 1=on
p$e	error status (0=no error) bit 0=data overun bit 1=parity error bit 2=framing error bit 3=line break

see also: **rsget, rsput**

Defaults and Ranges

keys/options	defaults	range
port	*none*	1 or 2
baud	*none*	baud rate supported by your external equipment
stop	*none*	1, 1.5 (if 5 data bits) or 2
bits	*none*	5, 6, 7 or 8

rsget

This command is specific to ...
IBM PC XT or AT comaptible

keys:	**name**	'<text>'	name of file into which received data is written
	timeout	*<number>*	timeout in seconds
	until	*<number>*	integer ASCII code for character to terminate read operation
options:	**binary**		read characters without processing record terminators
	number/key		read a decimal number or a single character
	keep		if **number/key**, retain the terminating character in the receive buffer
	old/new		re-use existing file or create new output file

The **rsget** command allows you to receive data from a communication port established with the **rs232** command. Data can be read directly into a file, or as individual decimal numbers or characters.

Examples

```
rsget number
```

This command reads a decimal number into the variable *nv* and stores the terminating character code in variable *nk*.

```
rsget name 'results.log' until 4 timeout 30
```

This command reads data into the file *results.log* until a byte containing the value 4 (Control-D) is read or no data has been read for 30 seconds. The terminating character code (or –1 on timeout) is returned in the variable *nk*.

Description

Together with the **rs232** and **rsput** commands, this command gives you extensive control of the standard RS232 communications ports, COM1: and COM2:. Data transfer takes place asynchronously under interrupt control, thus ensuring that incoming data is not missed. Both ports can be controlled independently and at the same time.

Semper maintains a concept of the currently selected port, and this port is used by **rsget** to read data. You can switch ports using the **rs232** command.

rsget

All the modes of operation of the **rsget** command are affected by the **timeout** key, which specifies an interval (in seconds) after which the transfer is timed out. A time out will cause the variable *nk* to be set to –1. If a timeout value is not specified it defaults to 300 (5 minutes).

The option **key** specifies that a single ASCII character code is to be read into the variable *nk*. The option **keep** specifies that the character code should also be left in the receive buffer to be read again. The option **number** is used to specify that a decimal number is to be read into *nv*, the terminating (non-numeric) character is stored in *nk*, and can be reread if the option **keep** is given. A number is defined as a sequence of ASCII characters in the range 0 to 9 with an optional decimal point.

The **name** key is used to specify the name of a file into which lines of text are transferred from the receive buffer. Records are delimited by carriage return or line feed or carriage return followed by line feed. Records written to the file will be separated by carriage return and line feed (the format for MS-DOS text files). If the option **binary** is given, the data is transferred into the file without any processing of record terminators. The options **old** or **new** allow you to overwrite an existing file (otherwise an error message is output).

The port is read until a timeout occurs or until the character specified by the key **until** is encountered. *nk* is set to the terminating character code or –1 if a timeout occurs.

You can flush the receive buffer up to a specified character by using **rsget** with the key **until.**

Notes

variables set:	*nk*	character code for terminating character (-1 on timeout)
	nv	if **number**, decimal number read
see also:		**rs232, rsput**

Defaults and Ranges

keys/options	defaults	range
name	*none*; prompts for file name if interactive	valid file name
timeout	300 seconds	positive real number
until	4 (Control-D)	integer in the range 0 to 255
binary	write data to file as separate records	
number/key	read until terminating character is reached or until a timeout	
old/new	fault existing file, otheriwse, create new output file	

rsput

This command is specific to ...
IBM PC XT or AT comptible

keys:	**name**	'<text>'	name of file which contains data to be sent
	text	*'<text>'*	text string to send
	char	*<number>*	integer ASCII code of character to send
options:	**again**		if **name** is omitted and was used in a command to open a file for reading, open the same file again
	binary		if **name**, send the contents of the file without processing record terminators
	cr		terminate each record/item with carriage return
	lf		terminate each record/item with line feed
	wait		wait for transmission to complete

The **rsput** command allows you to transmit data through a communication port established with the **rs232** command. Data can be sent as single characters, as a text string or as data from a file.

Examples

```
rsput text 'go' cr
```

This command sends the characters 'g', 'o' and carriage return to the current port.

```
rsput name 'setup.dat' cr lf
```

This command sends the contents of the file *setup.data* to the current port with carriage return and line feed between records.

```
rsput name 'new.obj' binary
```

This command sends the file *new.obj* as a binary byte stream. The number of bytes sent is exactly the number in the file.

Description

Together with the **rs232** and **rsget** commands, the **rsput** command gives you extensive control of the RS232 communciations ports, COM1: and COM2:. The transfer of data takes place asynchronously under interrupt control, freeing the Semper session to perform other operations while the data is transmitted. Both ports can be controlled independently and at the same time.

rsput

Semper maintains a concept of the currently selected port, and this port is used by **rsput** to send data. You can switch ports using the **rs232** command.

The key **char** specifies a single ASCII character to send. Any character can be sent (providing that the port has a suitable number of data bits selected).

The text key specifies a text string to send. Note that this can be any kind of Semper textstring, such as

```
rsput text 'found item number ',n,' at position ',x,',',y
```

The **name** key specifies the name of a file containing data to be sent. If the **binary** option is given, the contents of the file are sent unmodified, otherwise, the data is sent as a series of records.

The options **cr** and **lf** are used to send carriage return and linefeed characters after any text or character. They are also used to specify how records are terminated when sending a non-binary file. Note that the default for file transmission is **cr** and **nolf**.

The **again** option allows you to re-open a file without having to specify the file name again with the **name** key. If the **name** key was not used before to open a file for reading, the **rsput** command will prompt for the file name.

The option **wait** is used to make Semper wait until all of the data has been transmitted before continuing (the default is to put the data in the transmit buffer for asynchronous transmission). This option is useful if you need to synchronise with some external events, for example, lighting changes or microscope stage movements.

Notes

see also: **rs232, rsget**

rsput

Defaults and Ranges

keys/options	defaults	range
name	*none*; prompts for file name if interactive	valid file name
text	*none*	valid Semper textstring
char	*none*	integer in the range 0 to 255
binary	if **name**, read data from file as separate records	
cr/lf	if **name**, record terminator is carriage return, otherwise, *none*	
wait	put data in transmit buffer for asynchronous transmission	

run

keys:	**name**	*'<text>'*	name of file containing commands to be run
options:	**again**		if **name** is omitted and was used in a command to open a file for reading, open the same file again

Use **run** to read and execute commands from the beginning of a *run file*, untile a **return** or **end** command, or end-of-file condition is reached.

Examples

```
run name 'spectra'
```

This command runs the file *spectra.run*

Description

No special format is required for a *run file*. The commands are executed exactly as if they were typed at the keyboard, except that **jump** commands can be used to refer to labels anywhere in the file before or after a **jump** command. You can also call programs from a *run file*. For further information on programs and run files refer to *Chapter 6, Indirect Interpretation* of the following manual:

Advanced Users's Guide

contained in the Semper 6 Guide.

If you do not specify an extension for the filename, the default extension *.run* is assumed.

A **run** command should be the last command on a line, as text after a **run** command is lost in the process of switching command input to the run file.

The **again** option allows you to re-open a file without having to specify the file name again with the **name** key. If the **name** key was not used before to open a file for reading, the **run** command will prompt for the file name.

Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**end, jump, return**

run

Defaults and Ranges

keys/options	defaults	range
name	*none*; prompts if interactive	valid filename

save

keys:	**[from]**	*<number>*	source picture
		<n1>,<n2>	range of source pictures
	name	'<text>'	name of file to be created
	device	*<number>*	new device number
	size	*<number>*	size of disc file in kilobytes
	slots	*<number>*	number of directory slots
options:	**deassign**		deassign file on exit
	renumber		renumber output pictures from 1 upwards
	verify		verify assignment, picture copy and deassignment at the console

Use **save** to dump copies of pictures to newly created disc files for backup or achive purposes. You can recover the pictures using the commands **assign, examine** and **copy.**

Examples

```
save 20 name 'paper.figs'
```

This command saves a copy of picture 20 in a file called *paper.figs.*

```
save 1,999 name 'diffgrams' nodeassign
examine device n
deaassign device n
```

The example copies all existing pictures in the current device *cd* into the file *diffgrams.dsk*, listing details on the console before deassigning.

Description

save combines the facilities of the commands **assign new, copy** and **deassign** providing a convenient way to save or collate pictures into picutre discs. In its simplest form, the **save** command carries out the following sequence of operations:

It searches for any pictures that exist in the range of picture numbers specified with the **from** key. Note that the range must lie entirely within a single picture device. By default, the current picture is saved.

save

It creates a new picture disc file with overall file size and directory size just large enough to accommodate the number of pictures to be saved. The file name is specified with the **name** key and the default extension for the file name is '.dsk'.

All the specified pictures are copied to the new picture disc with the same picture number and exactly the same information (except that all write-protect flags are turned off).

The new picture disc is deassigned.

The **renumber** and **nodeassign** options and the **device, size** and **slots** keys may be used to alter the **save** command's default behaviour.

If the **renumber** option is set, the pictures are numbered from 1 upwards instead of retaining their original picture numbers.

If the **nodeassign** option is given, the new picture disc is left assigned for further use during the current session and the device number is returned in the Semper variable *n*. You can specify a device number for the picture disc with the **device** key, otherwise, the **save** command assigns the lowest free device number.

With the **size** and **slots** keys you can cause the **save** command to assign the new picture disc with a larger file size and directory size. A size which is smaller than the minimum required to save the specified pictures will be faulted (the error message will state what is the minimum requirement). If the **size** key is set, the **save** command performs exactly the same function as the **assign new** command. In this case, the default for the **slots** key is changed to be the maximum number of directory slots for a picture disc.

If the **verify** option is set, the **save** command verifies its operation on the console (device assign/deassign and source and destination picture numbers).

Notes

variables set: *n* (if **nodeassign**, device number assigned)
see also: **assign, copy, examine**

save

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture numbers refering to the same device
name	none; prompts if interactive	valid filename
device	lowest free device number	integer in range 2 to system limits (type **show system**)
size	minimum size required to store output pictures	positive integer
slots	if **size,** 2002, otherwise, number of output pictures + 2	positive integer
deassign	deassign file	
renumber	pictures output with same number	
verify	verification off,	

scale

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	range/msd	*<n1>, <n2>*	output range or required mean/standard deviation
options:	**preset**		assume source picture range given by existing values of *min*, *max*

Use **scale** to re-scale a picture linearly so that it contains a different pixel range. For example, use **scale** to scale a picture into the range 0–255 before conversion to *Byte* form, or to standardise grey scales between several different pictures which you want to combine into a montage.

Examples

```
scale 1 byte
```

This command scales picture 1 into the range 0–255 and converts it to *Byte* form. Refer to *Appendix C, Semper Keys and Options* for detail of the general key **byte**.

```
scale 50 51 msd 1, 0.2
```

This command scales picture 50 to 51, with a final mean of 1 and a standard deviation of 0.2.

```
scale range 1e3,1.3e3
```

This command scales the current picture into the range 1000–1300.

```
min=0 max=2; scale preset range 20,10
```

This command scales the current picture so that the range 0–2 becomes 20–10 (which means a contrast reversal).

Description

If necessary, **scale** scans the source picture to determine the initial range or mean/standard deviation. Note that an *abandon* request at this stage simply causes it to continue with estimated parameters.

scale records the final range in the output picture label, and returns it in the variables *min*, *max* whenever it is known reliably (that is, unless you specify **msd** or **preset** or make an *abandon* request during an initial range scan). Otherwise, the variables *min* and *max* (and in the **msd** mode *mean,me2,sd* also) are reset to the source picture values.

scale

If the output is in *Byte* form, **scale** truncates pixels outside the range 0–255 at the nearer limit.

For more general rescaling operations, including histogram equalisation, use the **map** command.

Notes

multi-layer pictures:	fully supported
forms used internally:	integer (for *Byte* data), fp, complex
variables set:	*min*, *max* (unless **preset** or **msd**, final pixel range) *mean*, *me2*, *sd* (if **msd**, source picture mean and standard deviation)
variables used:	*min*, *max* (if **preset**, source picture pixel range)
see also:	**map**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
range/msd	range 0–255	real numbers
preset	actual minimum and maximum pixel range	

scib

This syntax is specific to Sprynt systems, using the SCIB as the image grabber

keys:	mode	*'<text>'*	name of SCIB mode to load in
	rdr,gdr,bdr		specify the red, green or blue channel A/D convertor upper digitised voltages
	rcg,gcg,bcg		select red, green or blue channel gains
	roff,goff,boff		set red, green or blue channel offset voltages
	vi		select video input for green channnel
	sync		select sync source
	hue, sat		set Spectra hue or saturation
	cvi		select Spectra composite video input
	repeat, exposure		set MIFB/EMIFB exposure repeat rate count or period count
options:	enquire		set variable **acamera**(2) and **abppixel** to the dimensions of and the number of bytes per pixel of current SCIB mode. When the number of bytes per pixel is 2 set variable **ad16as8src** to the value of d16as8src for the current SCIB mode

The **scib** command allows you to control the SCIB board interactively.

Examples

```
scib mode 'mode monoccir'
```

This command loads mode 'mode monoccir' as the current SCIB mode.

```
scib gdr 128
```

This command selects upper digitised voltage to128 for the green channel. This setting is appropriate for monochrome cameras with 0.7 V.

```
scib cvi 0
```

If the Spectra Colour Decoder is configured for remote control then Spectra video input 1 is selected.

scib

Signal descriptions:

Analog composite	0.3V pk–pk composite with video (active low)
High level analog	2–4V pk–pk active low
TTL	0–5V logic signal

Connectors:

SK1 is the Video Connector (9–way D–type socket on the end plate)
SK2 is the Facilities connector (25–way d_type socket on the end plate)
PL4 is the Auxiliary Video Timing Connector (20 way ribbon cable plug on the PCB)

TTL sync option

TTLCHSYNC\ is an internal signal whose source and polarity is selected by the parameter **vtg_cfg** in the scibmode file. The function of this parameter is described in the summary of the C library function scib_cfg_vtg(). For more detail consult the the 'Sprynt Colour Input Board Users Guide'.

The key **vi** is used to select which is used as the green (monochrome) input.

When a Spectra Colour Decoder , which has two composite video inputs and an S–Video input, is used and is under remote control, the input board can provide power for it and remotely control which video input is selected using key **cvi.** Keys **hue** and **sat** can be used for the hue (NTSC only) and saturation adjustment. These keys can also be used to control other external equipment if Spectra is not being used or is not under remote control.

The (Extended) Megaplus Interface Board can be used to interface between the Videk Megaplus camera and the Sprynt input board. Keys **repeat** and **exposure** can then be used to set repeat rate and exposure period counts for integrating or sequence capture.

At the start of each semper session, a default scib mode is read from the configuration file 'sprynt.cfg' and loaded accordingly. During a semper session, different scib modes can be load by **scib** command with the **mode** key. A mode loaded must have its description in the SCIB mode file scibmode. Failure to load a scib mode will prevent all further access to SCIB until a mode is successfully loaded.

Notes

Whenever a scib mode is loaded all the parameters(gains and offsets, etc.) are set to the default values specified in the scibmode file. A **scib** command with other keys over these settings temporarily. It is recommended that you put a suitable mode set for each camera type you use frequently in the scibmode file. Please read Appendix A of 'Sprynt Colour Input Board User's Guide' to get a general idea of how to generate modes for analogue and digital cameras.

scib

defaults and Ranges

keys/options	defaults	range
rdr, gdr, bdr	*none*	integer in range 0 to 255
rcg, gcg, bcg	*none*	integer in range 0 to 7
roff, goff, boff	*none*	integer in range 0 to 255
vi	*none*	integer in range 0 to 1
sync	*none*	integer in range 0 to 7
hue, sat	*none*	integer in range 0 to 255
cvi	*none*	integer in range 0 to 3
repeat	*none*	integer in range 2 to 32767
exposure	*none*	integer in range 1 to 32767

section

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	position	*<x>, <y>*	coordinates of centre of averaged circle or sector
	angle	*<number>*	central direction of averaged sector, in radians anti-clockwise from the positive *x* axis
	width	*<number>*	angular width of averaged sector, in radians
	mark	*<number>* *<yes> or <no>*	mark the section centre line and angular limits on the display

Use **section** to extract radial sections from pictures, optionally averaged over a specified sector or a whole picture, in the form of a 1-D picture.

Examples

```
ps 50; section to 51; display
```

This command displays the rotationally averaged power spectrum of picture 50.

```
section 21 position 10,20 angle pi/2 width 0.4
```

This command replaces picture 21 with an averaged radial section over a vertical sector of angular width 0.4 radians, radiating from the point 10,20.

```
section angle theta width 0
```

This command produces an interpolated linear section radiating from the origin in the indicated direction.

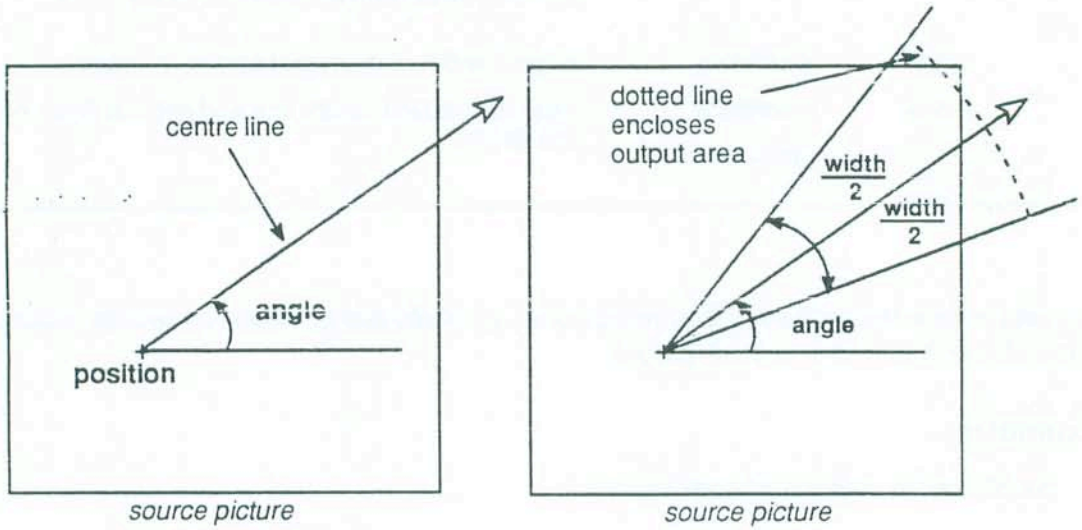
Description

By default, **section** averages the picture over the right-hand half of the picture. You can restrict the average to a sector only by quoting a sector **width** (in radians) and central **angle** (the anti-clockwise direction from the *positive* x axis, in radians). If you specify a display using the **mark** key, **section** marks the section centre line and angular limits on the display. Refer to *Appendix C, Semper Keys and Options* for details of the **mark** key.

The averaged sections radiate from the picture origin, or from any other **position** within the picture that you specify. The output picture is 1-D, with the origin at the left. Semper determines the size of the output picture so that it spans all source pixels that fall within the specified sector.

section

section works by extracting an initial section by bilinear interpolation along the centre line. It then averages any pixels falling within the specified sector with this section, the contribution of each being shared between two section pixels appropriately. If you specify **width 0, section** outputs only those values obtained by interpolating along the centre line (linear section). The diagram below illustrates the **section** command:



Notes

display marking:	averaged sector
multi-layer pictures:	faulted
forms used internally:	complex

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
position	position 0,0	within bounds of picture (integers)
angle	angle 0	real number in range 0 to 2π
width	π	real number
mark	mark off	see *Appendix C*

select

keys:	[]	<number>	picture number

Use **select** to set the protected variable *select* to a specified picture number, making that picture *current*. The default picture number for many of the processing commands is taken from *select*.

Examples

```
select 15; fourier
```

This command makes picture 15 the current picture and then performs a *Fourier* transform on the current picture.

```
select 4:53
```

This command makes picture 4053 the default source for the next command.

Defaults and Ranges

keys/options	defaults	range
[]	current picture, held in the variable *select*	valid picture number

separate

keys:	**[from]**	*<number>*	source picture (multi-layer)
	[to]	*<number>*	output picture, or first of output series
	layer	*<number>*	single layer to be separated
		<n1>, <n2>	first and last separated layers

Use **separate** to split up part or all of a multi-layer picture as separate 2-D pictures.

Examples

```
separate 50 to 51
```

This command separates all the layers of picture 50 as pictures 51, 52..

```
separate layers 3,5 to 101
```

This command separates layers 3 to 5 of the current picture as pictures 101,102 and 103.

```
separate layer 9
```

This command replaces the current picture by layer 9 of the picture only (like **cut**).

Description

The **separate** command produces several output pictures, one for each layer, beginning with the output picture number that you specify. Take care not to over-write existing pictures when using this command.

Notes

forms used internally: all
see also: **cut**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
layer	all layers	integers in range 1 to number of layers

sget

This syntax is specific to...
Silicon Graphics workstations

keys:	[to]	*<number>*	output picture
	name	*'<text>'*	source file name
options:	**again**		if **name** is omitted and was used in a command to open a file for reading, open the same file again

You use **sget** to read images from a *Stanford* format image file.

Examples

```
sget 20 name 'stanford.pic'
```

This command reads data from the file *stanford.pic* into picture 20 on the current device.

Description

Files are searched for in the current directory and then throughout the search path. You can specify a full path name to avoid the path scan.

Notes

see also: **sput**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

sharpen

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	over	*<number>*	width of averaged block/strip around each pixel during high frequency extraction
options:	**horizontally/vertically**		use horizontal/vertical strip averages around each pixel

sharpen applies a simple picture sharpening filter, effectively doubling the high spatial frequencies (fine detail) by adding the difference between the original picture and a locally averaged version.

Examples

```
sharpen display
```

This command sharpens the picture display.

```
sharpen 50 51 over 3
```

This command doubles frequencies that have a period no greater than 3 pixels, from picture 50 to picture 51.

Description

Use the **over** key to specify the size of block over which the local average is to be taken (the default is 5). You can specify 1-D **horizontal** and **vertical** forms as well as square forms. It is not useful to assign a very large value to **over**, as this does little more than double all pixels.

Note the possible clash of the **vertical** option with the general option **verify**, as Semper only takes into account the first three characters of a command, key or option.

The general form of the filter kernel is illustrated by its value in the case of **over 3**:

$$\frac{1}{9}\begin{bmatrix}(-1 & -1 & -1)\\(-1 & 17 & -1)\\(-1 & -1 & -1)\end{bmatrix}$$

sharpen processes edge pixels of the source, where the averaged block overflows the source, as if the boundary values are repeated indefinitely outwards. If you specify an even value for **over**, the replaced source pixel is rounded to the bottom right from the block centre.

sharpen

Note that **sharpen** is closely related to the **lmean** command, in fact, it performs the following action:

original picture + (original picture – **lmean***)*

Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	lmean

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
over	5	positive integer

sheet

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	range	*<n1>, <n2>*	range of *z* values to be represented by actual source pixel range
	border	*<number>*	*z* value of constant border to be added around the edge
	zorigin	*<number>*	*z* value for view origin
	theta	*<number>*	first rotation applied to surface before viewing, anti-clockwise about the positive *z* axis, in radians
	psi	*<number>*	second rotation applied to surface before viewing, anti-clockwise about the new positive *x* axis, in radians
	size	*<x>, <y>*	output picture dimensions
	times	*<number>*	magnification factor for output
	value	*<number>*	background value for use outside view of surface
	ltheta	*<number>*	angle between main light source direction and viewing direction, in radians
	lphi	*<number>*	azimuth of main light source direction, anti-clockwise from positive *x* axis, in radians
	main	*<number>*	brightness of main light source
	forward	*<number>*	brightness of subsidiary (forward) light source
	ambient	*<number>*	brightness of non-directional ambient lighting
	dcontrast	*<number>*	depth contrast, brightness difference between front and rear of surface
	sdr	*<number>*	ratio of specular to diffuse reflection, describing surface polish

Use the **sheet** command to generate a shaded image of a 3-D surface (*sheet*), the height of which is tabulated in a 2-D picture. **sheet** allows you to vary the viewing orientation, output image size etc.

Examples

```
lorentzian 1 size 150; min=0 max=255; sheet range 10,100 border 0
```

This command generates a shaded image on the display of a 2-D Lorentzian 'spike'. The image is a 100 pixels high with a border that is 10 pixels high.

sheet

```
sheet 1 to 5 theta pi/3 psi pi/6
```

This command generates in picture 5 a differently oriented image of the surface (rotated 60 degrees clockwise in-plane, then tilted 30 degrees, with the top away from you).

```
sheet size 300,200 times 3
```

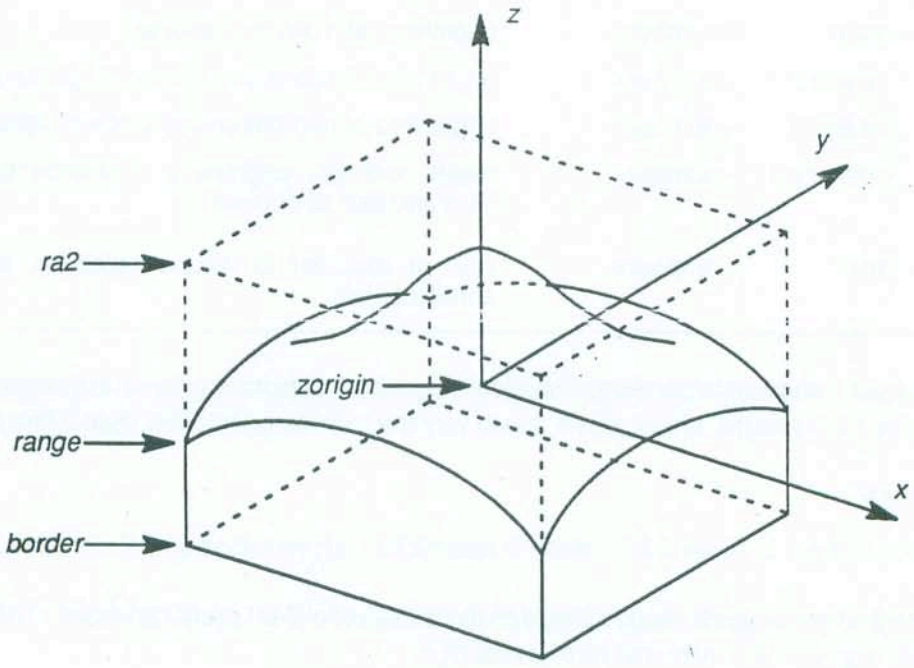
This command generates an image magnified by three, in an output picture sized 300 by 200.

Description

The **sheet** commands allows you to specify keys that affect the following:

- 3-D object (**range, border, zorigin**)
- orientation of the surface (**theta, psi**)
- 2-D output picture (**times, size, value**)
- illumination (**itheta, iphi, main, forward, ambient, dcontrast, sdr**)

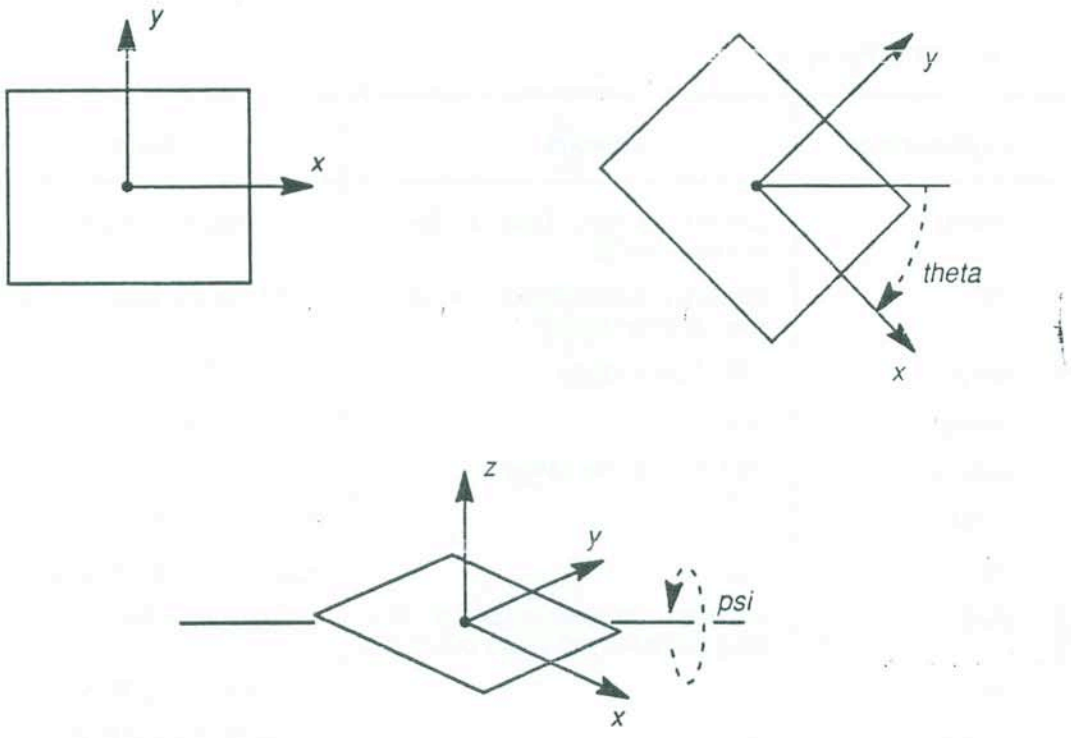
sheet generates a shaded image of the surface defined by interpreting the source picture pixels as height (*z*) values above the *x-y* plane. You can scale the source picture to represent a different range of heights using the **range** key. If you specify a **border** value, as in the first command example, **sheet** adds a border that extends to the specified height around the sheet so that vertical walls link the edges of the displayed surface to the specified horizontal level. The **zorigin** key allows you to position the view centre in the *z* direction, if the default of the mid-point of the height range is not appropriate. The diagram below illustrates these concepts.



sheet

When a 2-D image is displayed, the viewing direction is along the negative *z* axis, with the *x* axis to the right and the *y* axis pointing upwards. With the keys **theta** and **psi** you can alter the orientation of the shaded image away from the standard 2-D orientation. The surface is rotated first by angle **theta** clockwise about the *z* axis and then by angle **psi** anti-clockwise about the new positive *x* axis (with the top away from you). The default value for both **theta** and **psi** is $\pi/4$. You should not specify the magnitude of **psi** to exceed $\pi/2$ (that is, attempt to look at the underside of the sheet) as **sheet** does not produce correct results in this instance.

The diagram below illustrates the effects of **theta** and **psi**.



Use the **times** key to increase or decrease the size of the output picture. The **times** key takes fractional as well as well as integral values, for example **sheet..times .75** or **sheet..times 1.5**. By default, the output picture is created large enough to accommodate the 3-D box enclosing the surface, in the orientation you select.

If you want to force the size of the output picture, for example, to stop the size changing with the orientation, use the **size** key to specify the dimensions. If necessary, **sheet** truncates the shaded

sheet

image to fit the specified **size**. Note that the default values for the illumination keys ensure that the output pixel values lie in the range 0–255 allowed for *Byte* pictures, and Byte is accordingly the default output form. The surface is normally presented on a background of zero (dark) pixels, but you can change the background value using the **value** key.

A full description of the lighting parameters **ltheta**, **lphi**, **ambient**, **dcontrast**, **forward**, **main** and **sdr** is to be found in *Appendix H, Illumination*. With these you can determine the following aspects of the appearance of a shaded surface:

- ambient lighting (**ambient**)
- depth contrast (**dcontrast**)
- light source intensity/direction (**forward**, **main**, **ltheta**, **lphi**)
- diffuse/specular reflection (**sdr**)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current picture display, held in the variable *display*	valid picture display number
range	actual pixel range	real numbers
border	*none*	real number
zorigin	mid-point of the ***range***	real number
theta	$\pi/4$	real number in range 0 to 2π
psi	$\pi/4$	real number in range 0 to 2π
size	depends on source picture size, height range, and viewing direction	positive integers
times	1	positive real number
value	0	positive real number
ltheta	$\pi/4$	real number in range 0 to 2π
lphi	0	real number in range 0 to 2π
main	154	positive real number
forward	42	positive real number
ambient	40	positive real number
dcontrast	30	positive real number
sdr	0.4	positive real number

show

keys:	[ ]	*<number>*	device/partition/lut/error number
		<n1>, <n2>	range of devices/partitions/luts/error numbers
	usage	*<text>*	name of item to locate within current session
options:	**variables**		show all variables that are set and their values
	devices		show all assigned devices (discs, tapes, display libraries and log files)
	partitions		show defined display partitions
	luts		show defined look-up-tables
	time		show date and time of day
	programs		show names of all programs in assigned libraries
	path		show the search path used to locate files
	commands		show all available Semper commands
	macros		show all available named macros, with replacement texts
	system		show installation details (size limits, i/o units etc.)
	errors		show standard error messages, with numerical error codes
	sizes		show all *factorisable* sizes
	echo		show current echo settings for the terminal and for any text files
	page		show current terminal output settings
	full		with **programs** option, shows the program index usage

show prints information about your session on the console.

Examples

```
show devices      (or show device 3, or show devices 3,5)
```

This command reports the currently assigned devices and provides detailed information about them.

show

```
show partitions     (or show partition fs:3 etc. )
```

This command reports the currently defined display partitions, with details.

```
show path
```

This command reports the file search path.

```
show usage time
```

This command reports the usage of the **time** item within the current session.

```
show luts          (or show lut 2 etc.)
```

This command reports the currently defined look-up-tables.

```
show time
```

This command reports the date and time.

```
show variables
```

This command reports the value of all the variables (alphabetically sorted) that are currently set.

```
show commands
```

This command lists the names of all the available commands in alphabetical order.

```
show programs
```

This command reports the names of all programs in the assigned program libraries.

```
show programs device n
```

This command reports the names of any programs in the library on device *n*.

```
shows programs full
```

This command the names of all programs and also the program index usage.

```
show
```

This command lists the **show** options (**macros**, **commands**, **system** etc).

show

Description

Use the **show** command to list information about your Semper environment. For example, the **show system** command reports information that is specific to your installation. To see a list of the available options, simply type **show**.

Note that the **show usage** command is intended for users who write their own Semper programs. It provides a list of the instances in which a Semper command, key, option or variable is used. It allows you to ensure that a name in your program does not conflict with an existing name used by Semper. For example, the command **show usage pi** gives the following information:

```
'tim' is used in this session:
 - as a command
 - as an option to:
   sho
 - as a key to:
   bas con dil ero his mag
   nul phi pos ram rhi she sol
   str vie ymo
```

Defaults and Ranges

keys/options	defaults	range
[]	*none*	integers
usage	*none*	valid Semper command, key, option or variable name

sketch

keys:	**[ ]**	*<number>*	draw curve on the specified picture/partition/frame
	[to]	*<number>*	output picture
	tolerance	*<number>*	specify the tolerance for approximating the curve
	position	*<x>,<y>*	specify a start position for the cursor
options:	**picture/partition/frame**		sketch a curve on the picture, partition or frame
	closed		generate a closed curve
	re/im		operate with respect to the real or imaginary part of a complex display picture
	view		switch view to make the display region visible

The **sketch** command allows you to draw curves on the display using the mouse or pointing device.

Examples

```
sketch frame 1 to 2:3
```

This command draws a curve on display frame 1 and stores the result in picture 3 on device 2.

```
sketch to 1 tolerance 1.5
```

This command draws a curve on the current display picture and stores the filtered version as disk picture 1.

Description

With the **sketch** command you can draw curves free-hand on the display. The curve can be drawn in a display frame or partition, or it may be drawn on top of a display picture using the underlying image to guide the sketching process. This allows you to digitise features in an image with considerable ease. You start the sketching process by moving the mouse to the start of the curve and then draw the curve by pressing one of the mouse buttons and moving the mouse. Data for the curve is recorded until you release the mouse button. Alternatively, you may press any key on the keyboard to start and end the sketching process.

The data generated by the **sketch** command is stored in a Semper picture, specified by the **to** key. This picture is created as a *Plist* picture, which is a list of positions recording the points of the curve. You can edit this picture using the **pfilter** command, to reduce the number of data points.

Note that some curves may be very detailed and **sketch** may generate more data than can be stored in a picture. The **sketch** command incorporates a point filtering facility, with the **tolerance**

sketch

key, which allows longer or more detailed curves to be digitised. The resulting curve departs from the original curve by a distance that does not exceed the tolerance value that you specify.

By default **tolerance** is set to zero, which filters out redundant points that lie in straight lines. The filtering process is described in more detail in the entry for the **pfilter** command.

If you specify the **closed** option,the curve is closed at the end by a straight line running from the end point to the start point of the curve. You can specify the start position for the cursor using the **position** key, otherwise the cursor will appear at the default position which is the centre of the picture, partition or frame. Also, if you set the **view** option the view is switched so that the area of interest is made visible.

For complex display pictures, where the real and imaginary parts of the image are displayed side by side, **sketch** would normally display the cursor on the real part and draw the curve on both parts. If, however, you specify the option **re** or **im**, sketching will be carried out in just the real or imaginary part.

The **sketch** command returns the length of the curve in the variable *p*, and the area for a closed curve in the variable *a*.

Notes

see also:	**pfilter**
variables set:	*p* (length of curve)
	a (area enclosed by a closed curve)

sketch

Defaults and Ranges

keys/options	defaults	range
[ ]	if picture or partition, current display picture, held in the variable *display*; if frame, current display frame held in the variable *cframe*	valid picture/partition/frame number
[to]	picture 999	valid picture number
tolerance	0.0	positive real number
position	position 0,0	within bounds of the current picture/partition/frame (integers)
picture/partition/ frame	picture	
closed	open curve	
re/im	display cursor in real part, draw curve in both parts of complex display picture	
view	view disabled	

skiz

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
options:	**binary/label**		treat source picture as binary, labelled image
	fg		label foreground regions if binary source image
	bg		label background regions if binary source image
	circle/diamond/square/octagon		use Euclidean, 4-connected, 8-connected or octagonal metric for measuring distances

You use the **skiz** command to generate the *skeleton by zones of influence* or skiz associated with the source image. The skiz is the locus of points which are equidistant from the labelled regions defined by the source image and is the complement to the zones of influence associated with the same source image (see the **zone** command). A binary image is output with 1's for pixels that lie on the skiz and 0's elsewhere. By default, the source picture is treated as a binary image and a temporary, labelled image is obtained from it (see the **label** command for details about the labelling process). If the **label** option is set, the source image is treated as being already labelled. Distances between pixels are measured, by default, using the true Euclidean metric. Faster and possibly acceptable results can be obtained with simpler metrics by specifying one of the options **diamond, square** or **octagon** (see the **dt** command to find out more about distance transforms and metrics).

Examples

```
skiz
```

This command replaces the binary image in the current picture with the skiz associated with the connected foreground regions of the binary image.

```
skiz 1 2 label diamond
```

This command outputs to picture 2 the skiz associated with the labelled regions in picture 1 using the 4-connected, *city block* metric.

Description

If the source picture is treated as a binary image (the default), a temporary, labelled image is obtained from it in exactly the same way as the **label** ccommand. By default, the foreground regions are labelled. You can use the options **fg** and **bg** to control whether foreground and/or background regions are labelled. Note that foreground regions are treated as being 8-connected, while background regions are 4-connected.

skiz

By specifying the **label** option, you can suppress the labelling pass. This allows you to supply source images which have already been labelled, including images generated by any of the classification commands (**box, mindistance** and **likelihood**). Note that differently labelled regions are allowed to touch each other.

The skiz is the locus of points which are equidistant from the labelled regions associated with the source image. Because pixels occupy discrete positions in the image plane, there will only be a few pixels which intersect the skiz exactly. In order to produce a connected representation of the skiz, all pairs of 4-connected pixels which lie on either side of the skiz are also included in the output image. This produces a result which is equally distributed about the true skiz. However, the result will be one to two pixels thick and its level of connectivity will vary.

An entirely 4-connected or 8-connected result can be obtained by using the commands **bmlut** and **bmmap**, but the end result may be biased with respect to the true skiz by up to one pixel.

To obtain an 8-connected skiz, use the following commands:

```
bmlut to <map> if p4 unless  c8=1&n8~=1&(~p1&p7),+
                             c8=1&n8~=1&(~p7&p1),+
                             c8=1&n8~=1&(~p3&p5),+
                             c8=1&n8~=1&(~p5&p3)

bmmap <skiz> with <map>
```

where <skiz> contains the results from the skiz command and <map> contains the sequence of look-up tables defining the necessary sequence of 3 by 3 neighbourhood transformations.

To obtain a 4-connected skiz, use the following commands:

```
bmlut to <map> if p4|(~p0&p1&p3)|(p2&p1&p5)|(~p6&p3&p7)|(~p8&p5&p0),+
                  p4&~(c4=1&n4~=1&(~p1&p7)),+
                  p4&~(c4=1&n4~=1&(~p7&p1)),+
                  p4&~(c4=1&n4~=1&(~p3&p5)),+
                  p4&~(c4=1&n4~=1&(~p5&p3))

bmmap <skiz> with <map>
```

The **bmlut** command for the 4-connected case sets up a sequence of 5 neighbourhood transformations. The first transformation is a diagonal fill which makes any 8-connected components of the skiz 4-connected. The remaining transformations specify a 4-connected thinning transformation.

skiz

Distances between pixels are calculated using one of several possible metrics. By default, the exact Euclidean metric is used. A simpler metric requiring less computation, can be selected with one of the options **diamond, square** or **octagon.** For more details about the different metrics and distance transforms, consult the documentation for the **dt** command.

Notes

see also: **zone, dt, label, box, mindistance, likelihood, bmult, bmmap**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
binary/label	treat source picture as a binary image	
fg/bg	label foreground regions	
circle/diamond square/octagon	use the Euclidean metric	

slowscan

This command is specific to...
PC + Data Translation DT2861 framestore

keys:	**wait**	*<number>*	capture frames from video input for specified number of seconds
	frame	*<number>*	frames are grabbed into specified framestore frame
options:	**snap**		grab the next frame from the video input into the framestore

Use **slowscan** to grab frames continuously from a slowscan source for a fixed or variable period.

Examples

```
slowscan
```

This command captures frames from the video input into the currently viewed frame, until you press a key or the mouse button. At this point, it freezes the result in the framestore.

```
slowscan wait 5
```

This command captures the frames from the video input for five seconds and then freezes the result in the framestore.

```
slowscan snap
```

This command grabs the next frame from the video input into the framestore.

```
slowscan frame 3
```

This command grabs the frames into framestore frame 3, irrespective of which frame you are currently viewing.

Description

When you use the **slowscan** command, you destroy the previous contents of the image plane of the framestore frame that receives the video input. If you want to use the grabbed image you must create a display partition to use. You can then store the image on disc, for example:

```
partition frame 1;...
slowscan wait 1.5 frame 1; min=0 max=255
create display:1 size 512 byte
copy display:1 to :400
```

slowscan

This sequence of commands transfers a grabbed frame in frame 1 to disc picture :400. The image is scaled so that black becomes value 0 and white becomes value 255.

Note that you can use the **frame** key so that any frames are grabbed into a specific frame, instead of the frame that you are currently viewing.

Notes

see also: live, video

Defaults and Ranges

keys/options	defaults	range
wait	–1 (wait until a key or the mouse button is pressed)	real number
frame	frame that is currently viewed	valid frame number

snap

This command is specific to...
PC + Imaging Technology PCVISIONplus framestore
PC + Synoptics Synapse framestore
PC + Quantimet 520 greystore
PC + Metrabyte Corporation MV1 framestore

This syntax is specific to...
PC + Imaging Technology PCVISIONplus framestore

keys:	**partition**	*<number>*	capture the image into the specified display partition
	ilut	*<number>*	specify an input lookup-table
	channel	*<number>*	specify video input channel (1 or 2)
	mask	*<number>*	set the video input write-protect mask
	izoom	*<number>*	specify the sub-sampling factor
options:	**preset**		use the existing values of *min*, *max* for black and white scaling
	pll/crystal		the framestore uses its phase locked loop with an external sync source, or its own internal crystal, as the sync source

This syntax is specific to...
PC + Synoptics Synapse framestore

keys:	**partition**	*<number>*	capture the image into the specified display partition
options:	**preset**		use the existing values of *min* and *max* for black and white scaling

This syntax is specific to...
PC + Quantimet 520 greystore

keys:	**partition**	*<number>*	capture the image into the specified display partition
	gain	*<number>*	specify the gain applied to the video input signal
	offset	*<number>*	specify the offset applied to the video input signal
options:	**preset**		use the existing values of *min* and *max* for black and white scaling

snap

This syntax is specific to...
PC + Metrabyte Corporation MV1 framestore

keys:	**partition**	*<number>*	capture the image in the specified display partition
	ilut	*<number>*	specify an input look-up table
	channel	*<number>*	specify the video input channel (1 or 2)
	gain	*<number>*	specify the gain setting to be applied to the video input signal
	offset	*<number>*	specify the offset to be applied to the video input signal
	roffset	*<number>*	specify the offset to be applied to the video input of the red framestore (full colour systems only)
	goffset	*<number>*	specify the offset to be applied to the video input of the green framestore (full colour systems only)
	boffset	*<number>*	specify the offset to be applied to the video input of the blue framestore (full colour systems only)
options:	**preset**		use the existing values of *min* and *max* for black and white scaling
	pll/crystal		the framestore uses its phase locked loop with an external sync source, or its own internal crystal, as the sync source

Use the **snap** command to take a snapshot of a single frame from a video-rate source.

Examples

```
snap partition 1
```

This command grabs the next frame from the video input and places it into partition 1 of the framestore.

```
min=0 max=200; snap partition 2 preset
```

This command grabs the next frame into partition 2. The black and white scaling values are recorded as having the values of *min* and *max* respectively.

snap

PC + Imaging Technology PCVISIONplus framestore only

```
snap channel 2 partition 1 izoom 2
```

This command grabs the next frame from video channel 2 into partition 1. **Izoom** causes the live image to be sub-sampled to half its original size.

```
snap pll
```

This command grabs the next frame into the current display partition. The framestore uses its *phase locked loop* with an external sync source as the sync source.

PC + Quantimet 520 greystore only

```
snap gain 7 offset 5 partition 1
```

This command grabs the next frame from the video input into partition 1 applying a gain of 7 and an offset of 5.

PC + Metrabyte Corporation MV1 framestore only

```
snap channel 4 gain 1.5 offset 20 partition 1
```

This command grabs the next frame from video channel 4 into partition 1. It applies a gain factor of 1.5 and an offset of 20 to the video input.

Description

Use the **partition** key to specify the display partition into which the snapshot image is captured. In the case of the PCVISIONplus framestore, Semper attempts to centre the grabbed image on the specified partition. Semper displays an error message if the captured image overlaps the frame boundary.

snap

The **preset** option allows you to define the black and white scaling levels of a display picture. When Semper captures a live image the black and white levels default to the maximum values that can be stored in the image plane, as detailed below:

- *Synoptics* Synapse and *Metrabyte Corporation* MV1 framestore: 0 to 255
- *Imaging Technology* PCVISIONplus framestore: 0 to 127
- *Quantimet* 520 greystore: 0 to 63

Use the **preset** option to override these defaults. In this case the black level is set to the current value of the variable *min* and the white level to *max*.

PC + Imaging Technology PCVISIONplus framestore only

Use the **crystal** and **pll** options to determine the sync source for your live input. Use the **ilut** key to specify an input look-up table and the **channel** key to specify the video input channel.

The **mask** key allows you to set up the video input write-protect mask. A value of 0 allows you to write to all bit planes, a value of 127 allows you to write to the most significant bit plane etc. Use the **izoom** key, to sub-sample the video input.

PC + Quantimet 520 greystore only

Use the **gain** and **offset** keys to control the gain and offset of the Quantimet scanner. Use these keys to compensate for variations in the lighting conditions and camera sensitivity, so as to make the best use of the available greyscale.

PC + Metrabyte Corporation MV1 framestore only

A number of additional keys (**ilut**, **channel**, **gain**, **offset**, **roffset**, **goffset**, **boffset**) and options (**pll/crystal**) are available with this framestore, to determine the form of live input.

The **ilut** key allows you to specify the input look-up table which is used to map the video input. The **channel** key allows you to specify one of two input channels. Use the **gain** and **offset** keys to control the gain factor and offset applied to the video input on the framestore. The keys **roffset**, **goffset** and **boffset** allow you to select the offset applied to the video input on the red, green and blue framestores respectively (full colour systems only).

snapshot

This syntax is specific to Sprynt systems, using SCIB as the image grabber

keys:	[to]	*<number>*	display picture to copy captured images to
	tod	*<number>*	picture on a memory device or picture disc device
	sigma	*<number>*	number of images to integrate over
	skip	*<number>*	number of images to skip between those integrated or in a sequence
	sequence	*<number>*	number of images to capture
	gap	*<number>*	number of images to ignore between those integrated in a sequence
	timeout	*<number>*	approx maximum time in seconds to wait for an image to be captured before erroring
	size	*<n1>, <n2>*	if not capturing to the display then specifies the size of disc picture
	window	*<n1>, <n2>*	specifies the size of subregion of the SCIB scan to be captured
options:	exttrig		only to capture when external trigger received
	preset		record display picture black and white levels as current values of **min** and **max.**
	initialise		if it is first invokation of snapshot or live command in current Semper session then a couple of dummy captures will be done before proper captures

This syntax is specific to Sprynt systems, using SYNAPSE as the image grabber

keys:	[to]	*<number>*	display picture to copy captured image to
	subsample	*<number>*	subsample factor to use in copy
	window	*<n1>, <n2>*	Synpapse acquisition scan position with which to align top left of copy region when copying
	size	*<n1>, <n2>*	specifies the size of disc picture
options:	synchronous		if acquisition turned on then temporarily halt acquisition while doing the copy
	preset		record picture black and white levels as current values of **min** and **max**

snapshot

Use **snapshot** to grab a single or sequence of images from a video–rate source.

Examples

PC + SCIB framestore only

```
assign scratch size 2000; cd=n; cache dev n memory
snapshot tod cd:1 size 256,256
```

This command captures the centre 256,256 region of SCIB to disc picture CD:1.

```
snapshot to dis:1 tod cd:1 sequence 10 skip 4
```

This command captures a sequence of 10 images. Every 4th image from the camera is captured. Each captured image is simultaneously captured to DIS:1 and CD:1 . CD:1 will be a 10 layer picture. DIS:1 will be a 1 layer picture.

```
snapshot tod cd:3 sigma 4 skip 1 sequence 3 gap 2
```

This command captures a sequence of 3 integrated images to disc. Each integrated image is the sum of 4 images. The first image in the sequence is the sum of images 0,2,4 and 6. The second is sum of 9,11,13 and 15 and the third is the sum of images 18,20,22 and 24. CD:3 will be a 3 layer integer picture.

PC + Spryn using Synapse as the grabber only

```
snapshot to cd:3 size 256,256
```

This command captures the centre 256,256 region of Synapse to disc picture CD:3.

```
partition 1 size 128,128; snapshot to dis:1 subsample 4
```

This command captures and subsamples the 512 by 512 Synapse image to an 128 by 128 image in partition 1.

---

snapshot

Description

The **snapshot** command for SCIB is similar to the **live** command for SCIB. When Synapse is used as the grabber the **snapshot** additionally allows the transfer of images directly to disc. When SCIB is used you can capture colour images to the display and capture sequences of images, or perform image integration on monochrome images.The keys and options for SCIB and Synapse grabbers are not exactly the same.

The **to** key can used to specify both display and disc picture if Synapse is the grabber, but can only be used to specify display picture if SCIB is the grabber.

If you only want to capture a subregion of a camera scan use the **window** key to specifiy the position in the camera scan from which the top left pixel in the captured images is to come, where window 0,0 denotes the top left pixel in the camera scan.

If the **preset** option is set the black and white levels recorded with a Semper display picture are set to the current value of *min* and *max*. When it is not set and an image is captured into the display the black and white levels and variables *min* and *max* are set to 0.0 and 254.0 if the display is true colour, and 0.0 and 127.0 if the display is false colour if the SCIB is the grabber; the black and white levels and variables *min* and *max* are set to 0.0 and 255.0 if Synapse is used as the grabber.

PC + SCIB framestore only

When SCIB is used as the grabber, capturing to display can be done with both colour and monochrome images. When capturing monochrome images display partitions should only have one frame.

Capturing to disc can only be done on monochrome images when Semper is running in a false colour display mode. The key **tod** is used to specify the disc picture. The disc picture must either be on a memory device or on a picture disc device with its own memory buffer. When capturing to disc only, the key **size** can be used to specify the size of the disc picture.

Integrating and/or sequence capture requires capture to disc. Simultaneous capture to the display is optional. Which and how many images to capture are controlled by the keys **sigma, skip, sequence** and **gap**.

The **timeout** key is used to set the maximum time in seconds to wait for image to be captured before giving a time out error. If a time out error occurs it is likely to be due either to the camera not being connected/turned on or that the way the sync is connected to the input board is not as selected in the mode being used.

snapshot

Capture can be triggered by an external signal. If the option **exttrig** is specified, capture will only be done when an external trigger is received.

The SCIB board requires either 1or 2 captures to initialise its memory after power up. When the **initialise** option is set and it is the first invokation of the live or snapshot command in the current Semper session, a couple of dummy captures from SCIB will be done before the proper capture.

PC + Sprynt using Synapse as the grabber only

The key **subsample** specifies a subsampling factor to use in the copy. The value of subsample affects the dimensions of the area of the acquisition scan of that is samples. eg if the display partition or the disc picture is specified to have size128,128 and subsample has value 2 then a 256,256 area of the Synapse acquisition scan is subsampled.

When the option **synchronous** is set and acquisition is on then it is temporarily halted at the beginning of the next first field while the copying from Synapse to Sprynt takes place. This ensures that the image copied consists only of a single complete first field, second field pair.

---

snapshot

Defaults and Ranges

keys/options	defaults	range
[to]	*the current display picture*	valid picture number
window	*copy region positioned at centre of camera scan*	integer pair in range 0,0 to the camera scan size
tod	*none*	valid memory or disc picture number
size	*the size of camera scan*	integer pair in range 0,0 to the camera scan size
subsample	*1*	positive integer
sequence	*1*	positive integer
gap	*0*	positive integer
sigma	*1*	positive integer
skip	*0*	positive integer
timeout	*4*	–1 or positive integer
initialise	*on*	
preset	*no*	
synchronous	*yes*	
exttrig	*no*	

snap

The options **pll/crystal** allow you to specify an external or internal sync source. If you specify **pll**, the framestore uses its *phase locked loop* with an external sync source as the source. If you specify **crystal**, the framestore uses its own internal crystal as the sync source.

Notes

see also: live, video

Defaults and Ranges

keys/options	defaults	range
partition	current display partition, held in the variable *display*	valid partition number
ilut	*none*	valid input look-up table number
channel	*none*	channel 1 or 2, if *Metrabyte MV1*, channel 1, 2, 3 or 4
mask	*none*	integer in the range 0 to 255
izoom	1	1 or 2
gain	*none*	integer in the range 0 to 15 if *Metrabyte MV1*, gain factors of 0.5, 1.0, 1.5 and 2.0
offset	*none*	integer in the range 0 to 15 if *Metrabyte MV1*, integer in the range 0 to 255
roffset	*none*	integer in the range 0 to 255
goffset	*none*	integer in the range 0 to 255
boffset	*none*	integer in the range 0 to 255
preset	uses actual black/white pixel values for scaling	
pll/crystal	uses internal sync source (***crystal***)	

solid

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	threshold/with	*<number>*	single threshold level applicable to all layers, or 1-D picture containing independent thresholds for each layer in turn
	theta	*<number>*	first rotation applied to object before viewing, anti-clockwise about the positive *z* axis
	psi	*<number>*	second rotation applied to object before viewing, anti-clockwise about the new positive *x* axis
	size	*<x>, <y>*	output picture dimensions
	times	*<number>*	magnification factor for output
	value	*<number>*	background value for use outside view of object
	ltheta	*<number>*	angle between main light source direction and viewing direction, in radians
	lphi	*<number>*	azimuth of main light source direction, anti-clockwise from positive *x* axis, in radians
	main	*<number>*	brightness of main light source
	forward	*<number>*	brightness of subsidiary (forward) light source
	ambient	*<number>*	brightness of non-directional ambient lighting
	dcontrast	*<number>*	depth contrast, brightness difference between front and rear of object
	sdr	*<number>*	ratio of specular diffuse reflection, describing surface polish

Use **solid** to generate a shaded image of a 3-D solid object, defined by thresholding a density distribution tabulated in a multi-layer 3-D picture. **solid** allows you to vary the viewing orientation, output image size etc.

Examples

```
gaussian 1 size 40,40,40; min=0 max=255; solid threshold .02
```

This command generates a shaded image on the display of a sphere truncated at the edges of a cube.

solid

```
solid 1 to 5 theta pi/3 psi -pi/6 threshold .03
```

This command generates an image of the object differently oriented in picture 5. The image is rotated by 60 degrees clockwise in-plane, then tilted 30 degrees with the top towards you.

```
solid size 300,200 times 3
```

This command generates an image magnified by three, in an output picture of the specified size.

```
solid 1 with 37
```

This command generates an image of the object with independent thresholds for each layer, as given in the pixels of the 1-D picture 37.

Description

The **solid** commands allows you to specify keys that affect the following:

- the 3-D object (**threshold**, **with**)
- orientation of the object (**theta**, **psi**)
- 2-D output picture (**times**, **size**, **value**)
- illumination (**itheta**, **iphi**, **main**, **forward**, **ambient**, **dcontrast**, **sdr**)

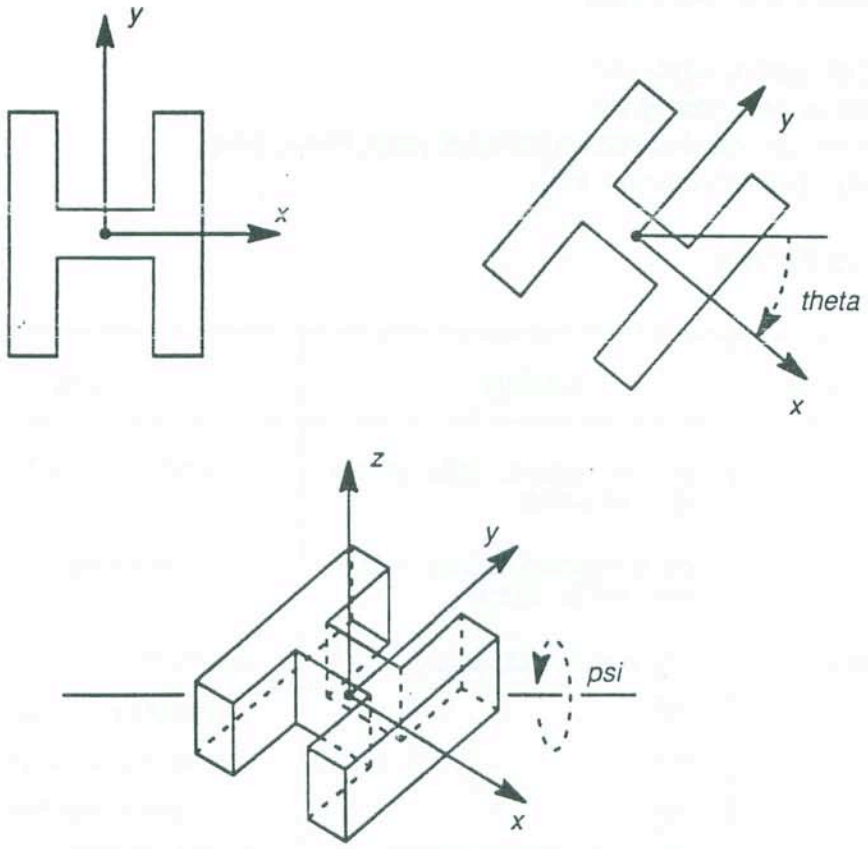
The object, for which solid will generate a shaded image, is defined by thresholding the pixels in the source picture. The source picture should be a multi-layer picture with its data describing a 3-D density distribution. All pixels with values greater than or equal to the threshold value (specified with the **threshold** key) are treated as part of a solid object and all the remaining pixels are treated as being transparent.

You can also give different thresholds for each layer of the 3-D picture using the **with** key. The **with** key specifies a 1-D picture whose pixels (from left to right) are the threshold levels to be applied to the various layers (from back to front). This is useful, for example, in viewing the 3-D reconstruction of a 2-D crystal.

When a 2-D image is displayed the viewing direction is along the negative *z* axis, with the *x* axis to the right and the *y* axis pointing upwards. With the keys **theta** and **psi** you can alter the orientation of the object to be viewed, away from the standard 2-D orientation. The object is rotated first by angle **theta** clockwise about the *z* axis and then by angle **psi** anti-clockwise about the new positive *x* axis (with the top away from you). The default value for both **theta** and **psi** is $\pi/4$.

solid

The diagram below illustrates the effect of **theta** and **psi**.



Use the **times** key to increase or decrease the size of the output picture. The **times** key takes fractional as well as well as integral values, for example **sheet..times .75** or **sheet..times 1.5**. By default, the output picture is created large enough to accommodate the 3-D box enclosing the object, in the orientation you select.

If you want to force the size of the output picture, for example, to stop the size changing with the orientation, use the **size** key to specify the dimensions. If necessary, **solid** truncates the shaded image to fit the specified **size**. Note that the default values for the illumination keys ensure that the output pixel values lie in the range 0–255 allowed for *Byte* pictures, and Byte is accordingly the default output form. The object is normally presented on a background of zero (dark) pixels, but you can change the background value using the **value** key.

solid

A full description of the lighting parameters **ltheta, lphi, ambient, dcontrast, forward, main** and **sdr** is to be found in *Appendix H, Illumination*. With these you can determine the following aspects of the appearance of a shaded surface:

- ambient lighting (**ambient**)
- depth contrast (**dcontrast**)
- light source intensity/direction (**forward, main, ltheta, lphi**)
- diffuse/specular reflection (**sdr**)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current picture display, held in the variable *display*	valid picture display number
threshold	single threshold value 0	real number
with	*none*	valid picture number
theta	$\pi/4$	real number in range 0 to 2π
psi	$\pi/4$	real number in range 0 to 2π
size	depends on source picture size, height range, and viewing direction	positive integers
times	1	positive real number
value	0	real number
ltheta	$\pi/4$	real number in range 0 to 2π
lphi	0	real number in range 0 to 2π
main	154	positive real number
forward	42	positive real number
ambient	40	positive real number
dcontrast	30	real number
sdr	0.4	positive real number

spawn

This command is specific to...
IBM PC XT or AT compatible

***spawn** uses a special syntax. If you specify an argument, the rest of the command is treated as a string to be used by the local operating system. If you do not specify an argument, **spawn** runs interactively.*

Use **spawn** to escape temporarily from Semper to the local operating system. This is useful, for example, for editing programs that you can only list in Semper or for returning to the operating system without losing picture data from any scratch disks that are assigned.

You can also use **spawn** to run operating system commands, specified on the Semper command line.

Examples

```
spawn
```

This command creates an interactive DOS command interpreter.

```
spawn 'format a:'
```

This command runs the DOS utility to format drive A.

```
n=27
spawn 'cd \results\run',n
```

This sequence of commands changes the current directory to \RESULTS\RUN27.

Description

To return to Semper after an interactive spawn use the DOS command **exit**, as shown in the following sequence of commands:

```
spawn
dir a:
type results.log
exit
```

Do not run any programs that remain resident whilst spawned from Semper, as this will fragment the DOS memory until you next reboot your PC.

If there is not enough memory for the spawning process, the system may halt. To verify that there is sufficient memory, run the DOS utility CHKDSK before running Semper. If there is less than 540,000 bytes available you may not be able to spawn.

spawn

This command is specific to...
DEC VAX

***spawn** uses a special syntax. If you specify an argument, the rest of the command is treated as a string to be used by the local operating system. If you do not specify an argument, **spawn** runs interactively.*

Use **spawn** to escape temporarily from Semper to the local operating system. This is useful, for example, for editing programs that you can only list in Semper or for returning to the operating system without losing picture data from any scratch disks that are assigned.

You can also use **spawn** to run operating system commands, specified on the Semper command line.

Examples

```
spawn
```

This command creates an interactive DCL command interpreter.

```
spawn 'show system'
```

This command runs the DCL utility to show processes on the local system.

```
n=27
spawn 'rename output.dat run',n'.results'
```

This sequence of commands renames the file *output.dat* to *run27.results*.

Description

To return to Semper after an interactive spawn use the DCL commands **eoj**, **logout** or **logoff**, as shown in the following example:

```
spawn
dir
ed results.log
eoj
```

spc

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	with	*<number>*	intensity contours from specified picture are marked on source
	levels	*<number>*	number of contours to be marked
	range	*<n1>, <n2>*	draw contours between specified minimum and maximum values
	value	*<number>*	value used for marking contours
options:	**ln/list**		mark contours with equal ratio between heights, or heights given by the values of the variables *l, l2...l9*
	verify		verify marked contour heights at the console

Use **spc** to mark intensity contours in a picture, or to superimpose the contours of one picture onto another, with some flexibility as to the contour heights that are marked and the intensity of the contour lines. **spc** marks contours by resetting the actual pixels; use the **contour** command if you wish to draw contours on a display overlay instead.

Examples

```
display; spc display levels 9
```

This command displays the current picture and superimposes 9 contours on it. The contours are bright where the picture is dark and vice versa.

```
spc to display value min
```

This command displays the current picture, with 5 dark contours, covering the range held in the variables *min,max*.

```
spc ln range 1,64
```

This command marks 7 contours in the current picture, covering the range **ln(1)** to **ln(64)** (that is, contours at values 2,4,8,16 and 32).

```
l=32.4,38.7,41.3,44.6,59.3,61; spc 50 levels 6 list to dis with 51
```

This command displays picture 50, with contours of picture 51 superimposed at the six listed heights.

spc

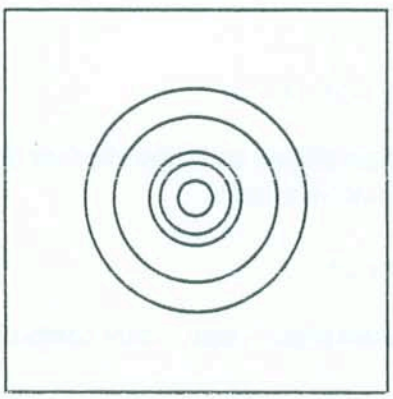
Description

By default, **spc** marks contours by setting pixels to whichever range limit is further (so that contours are bright where the picture is dark and vice versa). You can force any particular value for marking contours using the **value** key. You can superimpose the contours of one picture on another by setting **with** to the number of the picture you want to contour, as in the last command example.

You can specify the contour heights in several ways. Usually, the number of contours specified by **levels** is drawn (5 in default), covering the **range** indicated (*min, max* in default). The contours are evenly spaced. This does not include the extreme values, for example, **levels 3 range 1, 5** generates contours at heights 2, 3 and 4. If you specify **ln**, the contour heights are spaced with a constant ratio, effectively contouring the picture logarithm instead. This requires an entirely positive range: If you specify **list**, Semper takes an arbitrary set of heights from the variables *l,l2,l3..l9* as in the last command example.

spc marks contours by resetting the outermost pixels horizontally and vertically of any region as high as, or higher than, each contour height. Note that you can list the contour heights at the console using the **verify** option.

The diagram below shows contours created by **spc** on a lorentzian profile peak.



Notes

multi-layer pictures:	faulted
forms used internally:	fp, complex
variables used:	*l, l2, l3...l9* (if **list**, contour heights used by **spc**)
see also:	**contour**

spc

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	source picture	valid picture number
levels	5	positive integer
range	values held in *min*, *max*	integer
value	further of *range*, *ra2* from each individual contour height	real number
verify	verification off	

sput

This syntax is specific to...
Silicon Graphics workstations

keys:	[from]	*<number>*	source picture
	name	*'<text>'*	output file name
options:	**new**		overwrite the output file if it already exists

You use **sput** to write a picture to a *Stanford* format image file.

Examples

```
sput 20 name 'stanford.pic'
```

This command writes data to the file *stanford.pic* from picture 20 on the current device.

```
sput 20 name 'stanford.pic' new
```

As for the above example but also allows an existing file named *stanford.pic* to be overwritten.

Description

Files are created in the current directory, unless you specify a pathname.

Notes

see also: **sget**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

sscan

This command is specific to...
PC + MRC500 framestore
PC + Synoptics Synergy framestore

keys:	**channel**	*<number>*	slow-scan video channel, 1 or 2
	partition	*<number>*	input to specified display partition
	frames	*<number>*	number of frame scans to be captured
	time	*<number>*	time constant, if using recursive filter
	ignore	*<x>, <y>*	specifies the top left pixel of the window to digitize; the first *y* lines of the scan are ignored and the first *x* pixels of each scan
	dwell	*<number>*	if ***movie***, specify the number of scans for which the input stays on a frame
	a	*<number>*	if ***coefficient***, specifies the first coefficient for the recursive filter
	b	*<number>*	if ***coeffiecient***, specifies the second coefficient of the recursive filter
options:	**movie**		record a sequence of frames
	coefficient		specify coefficients for the recursive filter
	sum		average the input signal over ***frames*** frame-times
	kalman		make the output displayed after each frame-time the average of all frames since the start
	saturate		if the result of an input filter calculation overflows, write into the framestore as white (saturate value)
	stop		terminate input at the end of the current scan if an input filter calculation overflows
	immediate		terminate input immediately, rather than at the end of the current scan, if requested or when an input filter calculation overflows
	preset		use existing values of *min*, *max* for black and white scaling
	dual		display channel 1and 2 on the screen simultaneously
	ratio		multiply channel 1 input with channel 2 input and use a 16 bit value filter on the product

sscan

Use **sscan** to grab frames, or part of a frame, continuously from a slowscan source for a fixed or variable period. You can grab a frame into the whole screen or a partition of the screen. You can also apply noise filtering (recursive, averaging or *Kalman*) to the input.

Examples

```
sscan
```

This command captures frames directly into the whole screen area until you press any key or the mouse button.

```
sscan channel 2 partition 2 frames 25 time 3
```

This command captures frames from slowscan input channel 2 over the area of partition 2 for 25 frame-times, using a recursive filter with a time constant of 3 frame-times.

```
sscan movie partition 3 dwell 3 kalman frames 15 ignore 100,200
```

This command captures frames in **movie** mode. In this mode, Semper ignores the position of the specified partition but uses its size to define the size of each movie frame. The input stays on each movie frame for 3 scans and ends after a total of 15 scans. **sscan** ignores the first 100 pixels of each line and the first 200 lines.

Description

Use the **channel** key to specify the slow-scan video input channel (1 or 2). Use the **partition** key to define the display partition that is to receive input. **sscan** creates a display of the same size in the the specified partition. By default this is recorded as representing a *Byte* picture with black and white recorded as representing picture values 0 and 255 respectively.

Use the **frames** key to specify the total number of frames to capture. If you do not specify a value for **frames** you are asked to terminate the input. Press any key or the mouse button to interrupt the input. Input usually stops at the end of the current scan, unless you specify the **immediate** option.

You can apply three types of filter with **sscan**:

- recursive-type noise reduction filter
- averaging filter
- *Kalman* filter

If you are using the first type of filter, use the **time** key to specify the time constant. Specify the time in scan frame-times. For example, if you specify a time constant of 1, pixels will reach approximately 65% of their final value after one frame scan and approximately 90% after two.

sscan

Alternatively, use the **coefficient** option together with the keys **a** and **b**. This places a new pixel value into framestore memory, using the equation:

*new pixel value = (**a** coefficient * old value) + (**b** coefficient * input value)*

If you are applying an averaging filter, specify the **sum** option. The input signal is averaged over **frames** frame-times, except in the **movie** mode, in which case it is averaged over **dwell** frame-times.

If you are applying a **kalman** filter, Semper updates the **a** and **b** coefficients each frame time, according to the expressions:

$b = 1/n$, $a = (n-1)/n$

where *n* is the frame number.

This means that the output that is displayed after each frame-time is the average of all frames since the start.

The **movie** option allows you to record a region of a sequence of frames. In this mode the framestore packs as many sub-frames as possible , having the same size as the specified **partition**, into the entire display area, starting at the top left. The input dwells on each sub-frame for **dwell** frame-times and stops after a total of **frames** frame-times.

The **ignore** key allows you to specify the position of the top left pixel of the window of the slow scan which the framestore is to digitize. For example, if display partition 2 is 256 by 256 and its top left pixel is at the position 100,200, the command:

```
sscan partition 2 ignore 50,300...
```

extracts a 256 by 256 window of the slow scan input, starting at the 51st pixel along the 301st line, and writes it into partition 2 (starting at 100,200). By default, the first two pixels of each line, and the first two lines of each frame are ignored.

Use the **saturate** option to specify that, if the result of any input filter calculation overflows (that is, becomes 'whiter than white'), the value saturates and is written into the framestore as white.

The **stop** option causes input to end if any input filter calculation overflows.

You can override the default scaling values using the variables *min* and *max* and the **preset** option. For example:

```
min=-100 max=100; sscan partition 1 preset times 3 frames 10 fp
```

sscan

Notes

see also: **live**
variables used: *min*, *max* (if **preset**)
variables set: *min*, *max* (if not **preset**)

Defaults and Ranges

keys/options	defaults	range
channel	channel 1	channel 1 or 2
partition	partition 1	valid partition number
frames	0	positive integer
time	0	positive integer
ignore	2,2 (first two pixels of each line and first two lines of each frame are ignored)	positive integers
dwell	1	positive integers
a	*none*	real number
b	*none*	real number

ssgen

This command is specific to...
PC + selected versions of Synoptics Synergy framestore

keys:	**clock**	*<number>*	specify the clock speed
	xactive	*<number>*	number of active pixels in each horizontal line
	xretrace	*<number>*	retrace period in clock times
	xpresync	*<number>*	specify pre-sync period in clock times
	xsync	*<number>*	specify sync period in clock times
	xpostsync	*<number>*	specify post-sync period in clock times
	yactive	*<number>*	number of active lines in a frame
	yretrace	*<number>*	number of retraced lines in a frame
	ypresync	*<number>*	number of lines for the pre-sync period
	ysync	*<number>*	number of lines for the sync period
	ypostsync	*<number>*	number of lines for post-sync period
	xleft	*<number>*	xscan output value at the left hand side of the scan
	xright	*<number>*	xscan output value at the right hand side of the scan
	ytop	*<number>*	yscan output value at the top of the scan
	ybottom	*<number>*	yscan output value at the bottom of the scan

If your Synergy system has a *Scan Generator* option, you can control the scan waveforms and pixel clocks using the **ssgen** command.

Examples

```
ssgen
```

This command causes the scan generator to start generating a scan of 520 lines of 780 pixels, with a pixel clock frequency of approximately 680 KHz. (This format is suitable for capturing full-screen images using the **sscan** command).

```
ssgen xactive 100 yactive 200
```

This command causes the scan generator to start generating a scan of 100 pixels by 200 lines.

```
ssgen xleft 1000 xright 3000 ytop 4095 ybottom 0
```

This command approximately halves the *x* scan waveform in amplitude and reverses the *y* scan in polarity.

ssgen

Description

The Synergy *Scan Generator* option provides a digital scan generation facility for Synergy which can be used in conjunction with a scanning instrument to both scan a sample and capture the picture waveform. Use the **ssgen** command to control this facility.

Once you start the scan generator using the **ssgen** command, it continues to generate the waveforms and synchronisation signals until you type another **ssgen** command.

The **ssgen** command gives control over the speed, format, and voltage levels of the scan, using the following parameters:

- **clock** — to set the basic clock duration of the scan generator
- **yactive, yretrace, ypresync, ysync, ypostsync** — to compose the vertical format of each frame
- **xactive, xretrace xpresync, xsync, xpostsync** — to compose the horizontal lines
- **xleft, xright, ytop, ybottom** — to control the range and position of the output waveform

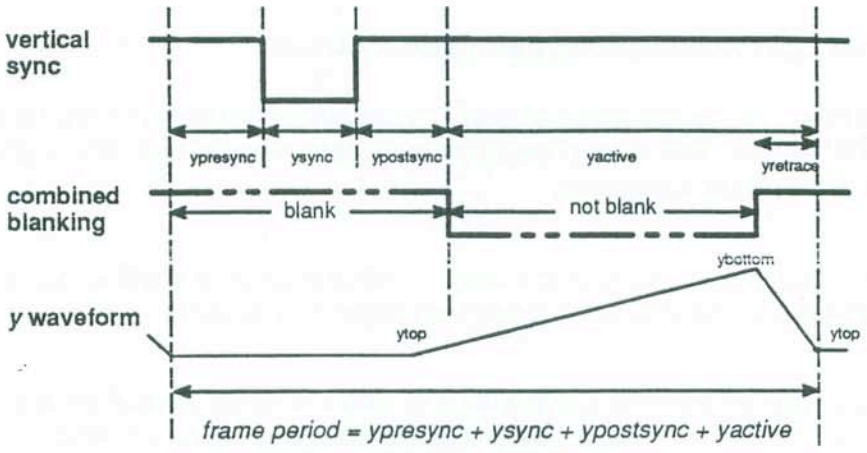
Use **clock** to set the basic clock duration of the scan generator. All pixel times and horizontal signal durations are expressed in terms of this interval. Vertical signal durations are expressed in terms of the line period, which is also derived from this clock. The clock period is:

*(clock+1)/(7.5*10^6) seconds*

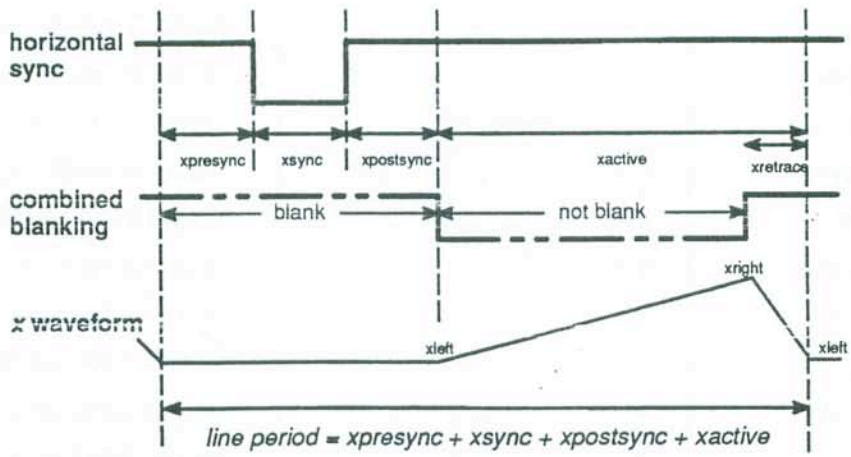
Take care when choosing a value for **clock**, as it is possible to set the clock speed so that Synergy's slow-scan input cannot cope with the data rate. The maximum clock clock speed is 1MegaHertz; the minimum clock period is 1µs (1 microsecond).

Each frame consists of **yactive** active lines (during which pixel clocks and the *y* scan waveform are generated), followed by a retrace period of **yretrace** line times (during which the *y* scan waveform retraces its steps). This is followed by a pre-sync period of **ypresync** line times, followed by a sync period of **ysync** line times, followed by a post-sync period of **ypostsync** line times. The diagram given opposite illustrates these concepts:

ssgen



Each horizontal line consists of an active period of **xactive** pixels (during which, if within the active *y* period, pixel clocks and the *x* scan waveform are generated), followed by a retrace period of **xretrace** clock times (during which the scanning waveform retraces its steps). This is followed by a pre-sync period of **xpresync** clocks, followed by a sync period of **xsync** clocks, followed by a post-sync period of **xpostsync** clocks. Note that all horizontal timings are in **clock** units. The diagram given below illustrates these concepts.



The scan waveforms that are generated in the *x* and *y* directions are linear ramps of duration **xactive** and **yactive** respectively. The waveforms are generated by 12-bit digital-to-analogue converters with selectable output voltage ranges.

ssgen

Refer to the manual:

Synergy Image Processing Subsystem Hardware Manual

for details of how to change the output ranges. You can control the range and position of the output waveform within the 0 to 4095 range of the output converters using the keys **xleft**, **xright**, **ytop**, and **ybottom** (see the previous diagrams):

- The *x* scan waveform begins at a value of **xleft** and ramps to **xright** at the end of the line. During the *x* retrace period, it ramps from **xright** back to **xleft**.
- The *y* scan waveform begins at a value of **ytop** and ramps to **ybottom** at the end of the frame. During the *y* retrace period, it ramps from **ybottom** back to **ytop**.

Defaults and Ranges

keys/options	defaults	range
clock	10	integer in the range 1 to 32767
xactive	780	integer in range 1 to 800
xretrace	10	integer in range 1 to 50
xpresync	5	integer in range 1 to 10
xsync	225/(1+clock)	integer in range 1 to 100
xpostsync	5	integer in range 1 to 10
yactive	520	integer in the range 1 to 900
yretrace	10	integer in the range 1 to 50
ypresync	1	integer in the range 1 to 10
ysync	5	integer in the range 1 to 10
ypostsync	1	integer in the range 1 to 10
xleft	0	integer in range 0 to 4095
xright	4095	integer in range 0 to 4095
ytop	0	integer in range 0 to 4095
ybottom	4095	integer in range 0 to 4095

ssheet

This syntax is specific to...
Silicon Graphics workstations

keys:	[from]	<*number*>	source picture
	to	<*number*>	start picture number for saving displayed images
	with	<*number*>	picture containing an image specifying the surface colour values for the colour map rendering mode
	emission	<*number*>	display look-up table number defining the colours to be used in the emission contour mode
	colour	<*number*>	display look-up table number defining the colours to be used in the colour contour mode
	map	<*number*>	display look-up table number for mapping the colours to be used in the colour map mode
	vfrom	<*number*>	picture containing initial view parameter settings
	vto	<*number*>	picture for saving viewing parameters

Use the **ssheet** command to display 3D surfaces using the Silicon Graphics workstation's 3D graphics capability. The source picture intensity values are interpreted as Z or height information and the resulting surface is rendered into a separate display window. You control the viewing, lighting and clipping parameters as well as the way in which the surface is rendered by manipulating sliders and menus with the workstation's mouse.

Examples

```
ssheet 2:1
```

This command displays the surface described by the intensity values in picture 2:1.

```
ssheet from 4 to 5
```

This command displays picture 4 as a 3D surface and allows you to save displayed images in pictures 5, 6, 7, etc.

```
ssheet 9 colour 2 emission 5
```

This command displays picture 9, using display look-up table 2 for the colour contoured rendering mode and look-up table 5 for the emission contour mode.

```
ssheet 34 vfrom 999 vto 998
```

This command displays picture 34 according to the viewing parameters saved in picture 999. The modified viewing parameters on exit from **ssheet** are saved in picture 998.

ssheet

Description

The command **ssheet** interprets the source picture intensity values as Z or height information and displays the surface in one of five ways:

- wireframe
- smooth shaded surface
- emission contoured surface
- colour contoured surface
- colour mapped surface

A control panel is displayed in a second window, allowing you to modify the viewing parameters, data sampling interval and rendering mode, and to save/restore all these settings. You may also save the displayed view as a Semper picture. Separate control panels can be selected to control the lighting conditions and set the clipping limits. All of these controls are driven by the workstation's mouse.

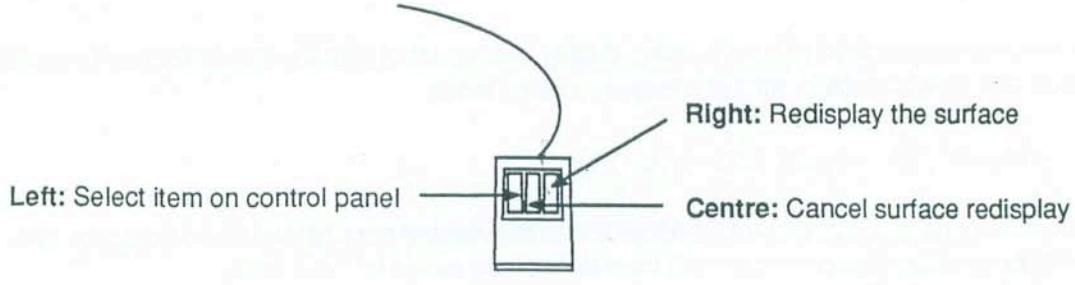
When the command starts up, the source picture data is copied into an array in virtual memory (4 bytes per pixel). For very large images (1K by 1K or larger), this will take some time and there may also be a time penalty when rendering the image, if the available amount of physical memory is such that much of the image data is paged from disc.

How to use the mouse

All of the normal window controls may be accessed with the mouse. You may reposition any of the control panels but not resize them. The surface display window may be repositioned and also resized at any time. Selecting the **Close** or **Quit** option in a window's pop-up menu will cause the **ssheet** command to be abandoned. The normal way to exit the **ssheet** command is to select the **QUIT** option on the main control panel.

When the cursor lies within the display window, or any of the control panels, you can press the right mouse button to cause the surface to be redisplayed. Pressing the middle mouse button will cancel this operation.

When the cursor is positioned inside any of the control panels, you can adjust sliders or select menu options by pointing to the relevant object with the mouse and the pressing the left mouse button.



ssheet

To move a slider, you must must place the cursor on or near the slider's pointer before pressing the the left mouse button. As long as you hold down the mouse button, the pointer will track any movement of the mouse. There are three types of sliders:

- circular sliders, for setting angular values
- linear sliders
- logarithmic sliders

Only the field-of-view and vertical scaling parameters are controled by logarithmic sliders.

When you select a menu option, the option is highlighted in red. Only when you release the left mouse button does the menu option take effect. This means that you can slide the cursor up and down the menu, over different options, until you are satisfied with the selection. If you move the cursor right out of the menu box, the menu is restored to its original state, allowing you to cancel the selection of any menu option.

When the image is being redisplayed and the single redraw mode is selected, the appearance of the cursor changes to an hourglass. This does not happen when the continuous redraw mode is selected because it would cause the cursor to switch rapidly between the hourglass cursor and the default arrow cursor.

The display window

The display window is the second window to appear on the screen after the **ssheet** command starts up. You will have to position and size the window so that it does not obscure the main control panel and any other windows you may also wish to see. The display window may be repositioned and resized at any time. The title of the source picture will appear in the window's title bar.

The 3D surface is rendered in the display window so that, with the default field-of-view of 45 degrees, the image occupies the middle third of the window and should not exceed the limits of the window even with extreme settings for the vertical scaling and view direction. Decreasing the field-of-view zooms the view in towards the surface. Any part of the surface extending beyond the limits of the window will be clipped.

The main control panel

The main control panel appears in the first window to appear on the screen after the **ssheet** command starts up. The window occupies most of the width of the screen so you should position it either at the top or the bottom of the screen. All of the normal viewing controls are to be found on this control panel. Further controls can be displayed up by selecting the **Lighting** and **Clipping** menu options.

The vertical scaling factor for the surface can be varied on a logarithmic scale from 0.01 to 10.0. With unit vertical scaling, the surface Z values are scaled so that the Z range is a quarter of the X or Y dimension, whichever is larger.

ssheet

The view is positioned with respect to the surface by means of a 3D cursor. This is red cross-hair which indicates the view centre position. Where the cursor is obscured by the surface, it appears dotted. When the surface is redisplayed, the view centre position will appear at the centre of the display window. The cursor position is controlled by the **Cursor X**, **Cursor Y** and **Cursor Z** sliders. The X and Y slider values are displayed in terms of the source picture coordinates and the Z slider value in terms of the source picture data range. The 3D cursor may be turned on or off.

Circular sliders allow you to set the azimuth and elevation angles for the viewing direction. The surface is displayed so that, as far as possible, the Z axis is projected projected vertically upwards on the screen. The azimuth is the anti-clockwise angle between the positive X axis and the projection of the viewing direction on the X-Y plane. The elevation is the angle between the viewing direction and the X-Y plane.

The surface can be rendered in one of five ways which you select by means of the **Rendering mode** menu:

- wireframe — a grid is drawn joining all adjacent vertices in the X and Y directions
- smooth shaded surface — a smoothly shaded grey surface is rendered, subject to the current lighting conditions
- emission contoured surface — the surface is rendered so that it appears to emit light whose colour varies with the height of the surface
- colour contoured surface — the surface is rendered so that the surface colour varies with the height of the surface, subject to the current lighting conditions
- colour mapped surface — the surface is rendered with the surface colour defined by a separate source image, subject to the current lighting conditions

Note that the **Colour map** option will only appear in the **Rendering mode** menu if the **with** key is specified with the **ssheet** command. The picture specified with this key should be either a single layer picture defining a false colour image or a three layer picture defining a full colour image. The X and Y dimensions of this picture must be the same as the source picture dimensions.

The **emission**, **colour** and **map** keys allow you to specify display look-up tables to define or map the colours used in each of the last three rendering modes. The default is to use the current look-up table (given by Semper variable **clut**) for the colour contour mode and to use linear grey-scale ramps for the emission contour and colour map modes. Each look-up table entry defines the colour for a single contour band for the emission and colour contour modes. The total assemblage of bands spans the entire Z range of the surface. When several adjacent look-up table entries are the same, they will be merged into a single colour band. The time taken to render a contoured surface varies in direct proportion to the number of bands. The **Band coarseness** slider allows you to set an interval for sampling the look-up table data so that the number of bands is reduced. The default band coarseness results in a maximum of 16 contour bands.

ssheet

You can also control the source data sampling interval with the **Model coarseness** slider. The coarser the interval, the faster the surface is redisplayed. The wireframe mode is the fastest rendering mode, followed by smooth shading and then the contoured surface or colour map rendering modes, depending on the band coarseness. The default model coarseness setting results in a maximum of 32 sampling points in the X direction. With this sampling interval the wireframe rendering mode is fast enough to allow continuous adjustment of the viewing parameters to be displayed in real time. Note that when the **Continuous** redraw mode is selected, the display is double buffered: the display window switches instantaneously to the new image. In **Single** redraw mode, the old image is erased first and you can then watch the new image being rendered in the display window.

The surface is rendered as a series of square facets with a source pixel at each corner. You can redisplay part of the surface by swithcing on the **Clipping** mode. Any facets which lie entirely outside the clipping limits are skipped over as well as any facets which fall outside the limits of the display window. The Clipping limits themselves are set by means of a separate control panel which you select with the **Clipping** menu option. Reducing the clipping limits reduces the amount of time to redisplay the surface.

The surface is rendered with all hidden surfaces removed by making use of the workstation's Z-buffer facility. The 3D nature of the image can be further enhanced by switching on the **Border** mode. This adds a border surface or "plinth" under the surface which extends down as far as the minimum Z-extent of the surface. The effect is as if the surface is "sculpted" out of a solid block of material. When the **Emission contour** or **Colour contour** rendering mode is selected, the contours extend around the border surface.

The normal way to cause the surface to be redisplayed after making changes to the viewing parameters is to press the right mouse button. However, if you select the **Continuous** redrawing mode, the surface will be redisplayed each time a viewing parameter is changed. The following events also cause the surface to be redisplayed:

- changing **Rendering mode**
- changing **Redraw mode**
- switching **Clipping** on or off
- switching **Border** on or off
- selecting **Centre view** on the clipping control panel
- repositioning or resizing the display window
- exposing any part of the display window

If the **to** key is specified when starting up the **ssheet** command, the **Save Image** menu option will appear on the main control panel. This allows you to to save the current image into the specified Semper picture. The picture will contain three layers of byte data representing the red, green and blue intensities of the surface image. You can save the current image as often as you like. Each time the output picture number is incremented, up to the maximum of 999. In this way you can save a whole sequence of views.

ssheet

You can also save all the current viewing parameters into a buffer and then restore them later on. If you use the key **vfrom** when starting up the **ssheet** command, the buffer is initialised to the values contained in the specified Semper picture, otherwise, the following default values are used:

- Azimuth 315 degrees
- Elevation 45 degrees
- Field of view 45 degrees
- Cursor position centre of X, Y and Z range
- Vertical scale 1.0
- Model coarseness (source picture row length) / 32
- Contour band coarseness (display look-up table length) / 16
- Rendering mode wireframe
- Redraw mode single
- Cursor on/off on
- Border on/off off
- Clipping on/off on
- Autocentring on/off off
- Clipping limits limit of X and Y range
- Front section plane 100 %
- Back section plane –100 %
- Ambient lighting 10 %
- Light source azimuth 0 degrees
- Light source elevation 90 degrees
- Light source colour white (on), black (off), black (off)

The contents of the buffer are used to set the viewing parameters when you start up **ssheet** and whenever you select the **Restore View** menu option. If you specify the **vto** key, the contents of the buffer are copied into the specified Semper picture when you quit from **ssheet**.

The lighting control canel

All the controls to set the lighting parameters are contained in a separate control panel which will appear if you select the **Lighting** menu option on the main control panel. You may position this control panel anywhere you like on the screen. When you have finished adjusting the lighting parameters, you can clear away the window by selecting the **HIDE WINDOW** menu option.

The lighting parameters affect the appearance of the surface (border surface as well as the top surface) when rendered in the smooth shaded, colour contour or colour map mode. There is ambient lighting and three separate point light sources. You can control the orientation, colour and brightness of each light source. If you set the red, green and blue intensity of a light source to zero, the light source is turned off. This bypasses the small time overhead associated with each light source when rendering the surface. The default lighting consists of 10% red, green and blue ambient light and one white light source positioned vertically above the surface. The orientation of the light sources is specified in the same way as the viewing direction, that is, with an azimuth and elevation defined with respect to the source picture axes.

ssheet

The clipping control panel

Controls to set the clipping limits and the front and back sectioning planes are located in a separate control panel which will appear if you select the **Clipping** menu option on the main control panel. You may position this control panel anywhere you like on the screen. When you have finished adjusting the clipping parameters, you can clear away the window by selecting the **HIDE WINDOW** menu option.

The clipping limits determine which surface facets are rendered. Any facet which falls completely outside the clipping limits is not rendered, together with any facets falling outside the limits of the display window. You can use the clipping facility to expose parts of a surface which would not otherwise be visible. You can also speed up the display of small sections of the surface, so that rendering modes other than the wireframe mode can also be used when the **Continuous** redraw mode is selected.

The **Centre view** menu option causes the view centre position to be set mid-way between the X and Y clipping limits. This is useful when the clipping limits get too far away from the view centre position so that the part of the surface that is not clipped disappears off the edge of the display window. Turning on the **Autocentring** mode causes the view centre position to be centred on the clipping limits whenever these are adjusted.

The front and back sectioning planes allow you to set clipping planes perpendicular to the view direction. The slider value for each plane is arranged so that zero intersects the view centre position and 100% is guaranteed to lie beyond the limits of the surface.

The clipping parameters are not allowed to cross over: the slider controls are interlocked so that, for example, if the minimum X clipping limit is moved to the right of the maximum X limit, the pointer for maximum X slider will also move to maintain the correct relationship with the minimum X slider. The sliders for the Y clipping limits and the sliders for the front and back sectioning planes are similarly interlocked.

ssheet

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
to	*none*	valid picture number
with	*none*	valid picture number
emission	linear grey-scale look-up table	valid look-up table number
colour	current look-up table, held in the variable *clut*	valid look-up table number
map	linear grey-scale look-up table	valid look-up table number
vfrom	*none*	valid picture number
vto	*none*	valid picture number

stack

keys:	[from]	<n1>, <n2>	first, last picture in range to be stacked
	[to]	<number>	multi-layer output picture
	layer	<number>	single layer taken from each source picture
		<n1>, <n2>	first, last layer taken from each source picture

You can combine layers from several 2-D or multi-layer pictures as a single multi-layer picture using **stack**.

Examples

```
stack 1,9 to 11
```

This command stacks all the layers of pictures 1 to 9 as picture 11.

```
stack 1,9 to 11 layers 2,3
```

This command stacks layers 2 to 3 of pictures 1 to 9 as picture 11.

Description

You create a new multi-layer output picture by each use of **stack**. If you need to replace several layers in an existing multi-layer picture, either use **paste** repeatedly as necessary, or **stack** to an intermediate picture in the first instance and **paste** this picture into the final output.

All the source pictures must have layers of the same size.

Notes

forms used internally: all
see also: **paste**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select, from*	valid picture numbers
[to]	source picture	valid picture number
layer	all layers	integers in range 1 to number of layers

stop (quit, exit)

***stop** takes no arguments*

To finish a Semper session, type **stop** (or **quit**, or **exit**).

Description

Type **stop**, **quit** or **exit** to leave a Semper session. You leave *run* and *library* programs with a **return** or **end** command. **stop** (or **quit**, or **exit**) in a program ends the session altogether. When a Semper session stops, all assigned devices are automatically deassigned (or deleted in the case of workspace devices and scratch picture discs/program libraries).

Note that you usually place the word **end** at the end of library and run files to mark their last line. However, if you use **end** interactively it has the same effect as **stop** and ends the Semper session.

strain

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	tolerance	*<number>*	fractional deviation from initial (estimated) lattice sites, beyond which sites are excluded from fit
	times	*<number>*	if ***mark***, magnification of display marking
	mark	*<number>*	mark deviations from lattice on display
		<yes> or <no>	
options:	**verify**		list details of fitting process at console

Use **strain** to deduce local strain levels for distorted crystals from a list of unit cell positions, and re-sort the list into ascending strain order.

Examples

```
u=10.0 v=0,10 w=0,0; strain 50 to 91
```

This command refines the lattice parameters *u,v,w* so as to fit the positions that are listed in picture 50. It outputs a revised list of positions, sorted in order of increasing strain level, to 91.

```
strain 999 tolerance 0.2 verify
```

This command re-sorts picture 999 by local strain level, excluding any positions with a fractional deviation from the nearest lattice site that exceeds 0.2. Information about the command process is sent to the console.

Description

Note that in the initial stages of processing (the lattice fitting) **strain** works in the same way as **base**, although it has fewer options.

strain also evaluates a local strain measure for each position having at least two neighbours (defined as other listed positions for which the lattice coordinates differ from those at the first by values within **tolerance** of unity). The local strain measure is the *r.m.s.* fractional displacement difference between the position itself and each of its neighbours. **strain** stores the strain values in the list instead of the original height or mass values, if these are present, and resorts the list, omitting positions for which the strain cannot be estimated. You can then use the **motif** command to make averages on the basis of selected strain levels only.

strain

If you specify a display using the **mark** key, **strain** marks all deviations from the lattice on the display. See *Appendix C, Semper Keys and Options* for details of the **mark** key. Use the **times** key to specify the magnification factor for the display marking.

Notes

variables used: *u, u2, v, v2* (initial estimates for lattice base vectors)
w, w2 (initial estimate for lattice origin)

variables set: *u, u2, v, v2* (base vectors of fitted lattice)
w, w2 (origin of fitted lattice)
n (number of sites included in fit)
r (r.m.s. deviation in source pixels between included sites and final fitted lattice)

see also: **base**, **motif**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
tolerance	tolerance 0.3	real number
times	times 5	positive integer
mark	mark off	see *Appendix C*
verify	verification off	

survey

keys:	**[from]**	*<number>*	source picture to be surveyed
	size	*<x>, <y>, <z>*	dimensions of subregion to be surveyed
	position	*<x>, <y>, <z>*	position/offset of subregion
	layer	*<n1>, <n2>*	range of layers in subregion
options:	**full**		perform fuller form of survey, including mean, standard deviation (also median and mode if *Histogram* source)
	left/right, near/far	**top/bottom,**	subregion positions
	verify		list results found at the console

Use **survey** to scan a picture or subregion of a picture, to determine the data range and other statistics.

Examples

```
survey 19
```

This command types the size and data range of picture 19 on the terminal and sets the variables *min* and *max* accordingly.

```
survey layer 3 noverify
```

This command sets *min* and *max* to the data range within layer 3 of the current picture. The values are not listed at the console.

```
survey full size 100 bottom left
```

This command lists the range, mean and standard deviation of the bottom left 100 square of the current picture, and sets the variables *min*,*max*,*mean*,(*me2*) and *sd*.

Description

Use the standard subregion keys, **size**, **position** etc. to specify the region you want surveyed. See *Appendix C, Semper Keys and Options* for further details. Normally, **survey** only determines the range of pixels and where possible, recovers the range from the picture label.

survey full ignores the label (providing a recovery route in case of difficulty), and determines the mean (complex, if the picture is *Complex*) and standard deviation of a picture.

survey

If the source picture is a histogram, **survey** sets *min, max* to the range represented by the histogram channels, not the minimum and maximum channel counts. It also sets *mean, (me2), sd, median* and *mode* to values estimated from the histogram. Any subregion that you request is ignored.

Note that **survey** returns estimated values if you *abandon* while it is scanning data.

Use the **noverify** option to prevent **survey** listing the results on the console.

Notes

multi-layer pictures:	fully supported
forms used internally:	integer, fp, complex
variables set:	*min*, *max* (minimum and maximum values found)
	mean, *me2* (if **full**, mean and standard deviation found)
	median, *mode* (if **full** and *Histogram* source, median and mode found)

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0, 0 , 0	within bounds of the picture (integers)
layer	all layers, unless variables *si3*, *po3* are set	integers in range 1 to number of layers
verify	verification on	

syntax

keys:	*<command name>*	displays syntax of specified Semper command

syntax gives a short summary of the syntax of a Semper command. This is the precise command definition used by Semper, and is always available, even when no help libraries are assigned.

Examples

```
syntax base
```

This command details the syntax of the **base** command. Semper produces the following output:

```
Options:
   ver   uvo
Keys:
   tol=.3  num  nu2  rad  pos  po2  tim=5  $2=0  to=$2  $1=sel
   fro=$1
Picture opens:
   (lp1, old)=fro
```

The syntax of the **base** command is given below in full for comparison:

keys:	**[from]**	*<number>*	*Plist* picture
	[to]	*<number>*	output picture
	position	*<x>*, *<y>*	centre of region to which fit is restricted
	radius	*<number>*	radius of region to which fit is restricted
	numbers	*<n1>*, *<n2>*	serial number of positions to which fit is restricted
	tolerance	*<number>*	fractional deviation from initial (estimated) lattice sites beyond which sites are excluded from fit
	mark	*<number>* <yes> *or* <no>	mark the final deviations of all positions from the lattice on the display
	times	*<number>*	if ***mark***, magnification of display marking
options:	**uvonly**		fit lattice base vectors *u*, *u2* and *v*, *v2* only, preserving lattice origin *w*, *w2*
	verify		verify details of fitting process at the console

syntax

Description

syntax lists the keys, options and implicit *picture opens* of a Semper command. **syntax** lists only the first three characters of options and keys, as this is as much as Semper needs to know to decode the command line.

Keys

Keys are shown either by their name, or as *key=val*, where *val* is the default value that is substituted if you do not specify the key in the command line. The special keys **$1,$2,$12,$13** etc. are used to read from the command line values that are entered without any prefixed key name. This mechanism is used by many commands. For example, **display** includes the following in its key list:

```
$1=sel fro=$1 $2=dis to=$2
```

This means that Semper assigns the first value found on the line without a keyword to the key **$1**, and by default to the key **fro(m)**, and the second value found to the key **$2**, and by default to the key **to**. Note that the default values for these keys are *sel(ect)* for **$1** and *dis(play)* for **$2**. This means that the when you do not specify any keys, **display** displays the currently selected picture on the currently selected display.

In some cases **syntax** lists no default value, as Semper does not immediately provide a default for the key. This is because no useful default exists, or because the default cannot be expressed simply. In the latter case, the syntax definition in the help library will explain the default action taken by the command.

Notes

Picture opens refers to pictures that the command interpreter opens for the command before processing. The most common item listed by **syntax** is:

```
(lp1,old)=fro
```

which means that the picture number held in the key *fro(m)* is opened in *old* mode as logical picture number 1. *Old* mode means that the picture must already exist.

The other form given by **syntax** is:

```
(lp2,new,lp1)=to
```

which means that a new picture, number *to*, with the default characteristics of logical picture 1 is opened as logical picture 2.

textfield

keys:	**id**	*<number>*	define which textfield to use
	in	*<number>*	place textfield in specified panel
	name	*'<text>'*	specify a name for a textfield
	begins	*'<text>'*	specify the *interaction begins* action for a textfield
	changes	*'<text>'*	specify the *state change* action for a textfield
	ends	*'<text>'*	specify the *interaction ends* action for a textfield
	contents	*'<text>'*	replace the contents of a textfield with the text
	append	*'<text>'*	add the text to the existing contents of a textfield
	prior	*'<text>'*	insert the text in front of the existing contents of a textfield
	length	*<number>*	specify maximum number of characters in a textfield
	value	*<number>*	replace the contents of a textfield with the number
	assign	*'<text>'*	assign the contents of a textfield to the specified variable
	position	*<x>, <y>*	position a textfield on its panel
	size	*<x>, <y>*	define a minimum *x* and *y* size for a textfield
	tab	*<number>*	define the 'tab-target' of a textfield
	background	*<number>*	number of textfield background colour
	foreground	*<number>*	number of textfield foreground colour
options:	**create**		create a textfield
	destroy		destroy a textfield
	clear		clears the contents of the current textfield
	execute		executes the contents of the current textfield
	immediately		executes the contents of the textfield immediately when used with the ***execute*** option
	activate		activates a textfield, ready to accept input
	wp/wp off		write-protects a textfield/ turns off write-protection

The **textfield** command controls the creation and operation of the Semper 6 *Plus* textfield element. For a description of Semper 6 *Plus* elements, refer to the *User Interface Guide* and *Tutor User Guide* contained in the *Semper 6 Guide*.

textfield

Examples

```
textfield create name 'Commands' length 10
```

This command creates a new textfield with the name *Commands* on the current panel. The textfield has space for a maximum of 10 characters. The default values for positioning the textfield are used.

```
textfield assign 'con'
```

This command assigns the contents of the current textfield to the variable *con*. Note that you must specify a variable name in quotes, otherwise the variable is evaluated. The contents of the textfield must be a valid numeric expression.

Description

A textfield is placed on a panel and you can type text into it from the keyboard. The contents of a textfield can be executed as a Semper 6 *Plus* command. The following diagram illustrates a textfield.

You create a Semper 6 *Plus* textfield using the **create** option and remove a textfield using **destroy**. Use the **id** key to define a textfield to act upon. The *active textfield* accepts input that you type at the keyboard. A textfield is made *active* when:

- it is first created; the latest textfield to be created is activated by default
- it is activated explicitly using **activate**
- it is activated using the mouse

You can define the contents of a textfield using the keys **clear**, **contents**, **append**, **prior**. Use the **contents** key to replace the contents of the current textfield with a specified text string. Use the **append** key to add text to the end of the existing contents of a textfield and the **prior** key to insert text before the existing contents. Use the **clear** option to clear the contents of the textfield.

The **execute** option causes the contents of a textfield to be executed *as if they were entered at the keyboard*. It is important to remember this point if you are using a **textfield...execute** command within a run file or library program. For example, consider the following fragment of a library program:

```
textfield id tx contents 'display 1' execute
display 2
return
end
```

textfield

After executing the **textfield** command, Semper 6 *Plus* executes the **display 2** command and not the **display 1** command that is contained in the textfield. The **display 1** command is only executed at the end of the library program. In order to execute the command that is contained in a textfield immediately (as if the **textfield** command is a library command executing the textfield contents) you need to specify the **immediately** option. For example, if the previous command sequence is changed to the one given below, then picture 1 is displayed before picture 2:

```
textfield id tx contents 'display 1' execute immediately
display 2
return
end
```

The command **textfield...assign** evaluates the contents of a textfield as a numeric expression and stores the results in the named Semper variable. You must specify the name of the variable in quotes. The **value** key replaces the contents of the current textfield with the value of a specified numeric expression.

A textfield can go through the following transitional states:

- *interaction begins*
- *state changes*
- *interaction ends*

You define the action taken for each of these states using the **begins**, **changes** and **ends** keys. These keys allow you to specify as text the commands, keys and options to be executed by Semper when interaction with a textfield begins or ends, or a textfield changes state. The following table shows how to initiate a **begins**, **changes** or **ends** action for a textfield.

activate textfield	change contents of a textfield	enter text and select
begins	**changes**	**end**

Use the **length** key to define how many characters can be entered into a textfield. If you exceed this self-defined limit, Semper does not give an error but truncates the string to the specified number of characters. You can only alter the length of a textfield when the panel containing the textfield is not showing. You can name a textfield using the **name** key. This allows you to identify the purpose of a textfield on a panel.

Use the **tab** key to define another textfield as the 'tab target' of the current textfield. The 'tab target' is the textfield that is selected when the current textfield is active and you press the <TAB> key. This key allows you to define a form using textfields.

textfield

You can change the attributes of a textfield using the **size**, **foreground** and **background** keys. The **size** key determines the minimum size of a textfield. Use **foreground** and **background** to change the foreground and background colours of a textfield.

Notes

variables used: *eno (textfield number,* if **id** not specified*)*
*pno (*panel number, if **in** not specified)
variables set: *eno (*set by the **create** option to the textfield identifier number)
see also: **ceil**, **panel**

Defaults and Ranges

keys/options	defaults	range
id	current textfield, held in variable *eno*	valid textfield number
in	current panel, held in variable *pno*	valid panel number
name	*none*	text string
begins	*none*	text string containing valid Semper commands
changes	*none*	text string containing valid Semper commands
ends	*none*	text string containing valid Semper commands
contents	*none*	text string
append	*none*	text string
prior	*none*	text string
length	5	integer: range is machine dependent
value	*none*	valid numeric expression
assign	*none*	valid variable name in quotes
position	position 0,0	within bounds of panel (integers)
size	Semper determines textfield size	within bounds of panel (integers)
tab	*none*	valid textfield identifier
background	background colour of panel	integer: range is machine dependent
foreground	foreground colour of panel	integer: range is machine dependent

threshold

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
	ge	*<n1>...<n9>*	lower threshold(s) (including threshold value)
	gt	*<n1>...<n9>*	lower threshold(s) (excluding threshold value)
	le	*<n1>...<n9>*	upper threshold(s) (including threshold value)
	lt	*<n1>...<n9>*	upper threshold(s) (excluding threshold value)

You use **threshold** to produce a binary image according to one or more intensity thresholds. If no threshold values are specified, the mid-range intensity value for the source picture is used as a lower limit for thresholding.

Examples

```
threshold
```

This command converts the current picture into a binary image where non-zero pixels represent source pixels with value greater than or equal to the mid-range value for the source picture.

```
threshold 2:3 2:4 ge i1 le i2
```

This command thresholds picture 2:3 for intensities in the range i1 to i2.

```
threshold 99 ge 70,50,50,10,90 le 20,40,60,100,80
```

This command thresholds picture 99 for intensity bands 10 to 20, 30 to 40, 50 to 60, 70 to 80 and 90 to 100 (note how values have been sorted into the correct sequence).

```
threshold lt -35 gt 35
```

This command thresholds the current picture for all pixels with absolute value greater than 35.

Description

You use the multi-valued keys **ge** or **gt** to specify lower threshold limits and the **le** or **lt** key to specify upper limits. Up to 9 upper and 9 lower limits may be specified. The limits are sorted into adjacent pairs and a check is made to ensure that they are consistent with each other, e.g.

```
ge 10,20 lt 30
```

would be faulted because the lower limit of 10 does not pair up with an upper limit (or plus infinity).

threshold

On the other hand the limits

```
ge 20 lt 10 le 30
```

are valid, specifying intensity bands less than10 and 20 to 30.

Threshold values should be specified in adjacent pairs with the exception of a single, isolated upper or lower limit. The ordering of the **ge, gt,** and **le** and **lt** keys on the command line does not matter as they are sorted into numerical order before being checked for consistency.

Notes

see also: **calculate**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
ge	middle of source picture range	real number
gt	*none*	real number
le	*none*	real number
lt	*none*	real number

time

options:	**reset**	reset the elapsed time clock to zero
	verify	verify the time on the console

Use **time** to measure the elapsed time taken by different Semper operations, for example, the total duration of your session. (If you simply want to know the time of day, type **show time**).

Examples

```
time
```

This command reports the number of seconds that have elapsed since the start of your session, or since you last typed **time reset**.

```
time reset
```

This command resets the elapsed time clock to zero.

```
time noverify
```

This command sets the variable *t* to the number of seconds of elapsed time but does not confirm the time on the console.

Description

The precision of the result shown by **time** is, of course, installation dependent. The time is printed to the nearest centi-second.

Notes

variables set: *t* (number of seconds of elapsed time)
see also: **show**

Defaults and Ranges

keys/options	defaults	range
verify	verification on	

title

keys:	[]	*<number>*	change/set the title of the specified picture
	text/from	*'<text>'/<number>*	text string for title or picture number containing title to use
options:	**add**		append text to a title, rather than replacing a title

Use **title** to change the text, or add to the text, of a picture title.

Examples

```
title 51 text 'Difference between methods'

title 51
Text (as textstring) : 'Difference between methods'
```

These identical commands specify a new title for picture 51. In the second example, the terminal prompts for the title text, as none is specified.

```
title add text 'serial number ',ns
```

This command appends the specified text, including the value of variable *ns*, to the end of the title recorded for the current picture.

```
title text ''
```

This command deletes any recorded title for the current picture.

```
title 51 from 50
```

This command copies the title of picture 50 to that of picture 51.

Description

Semper prompts at the terminal for the title text if you omit the **text** key, as in the first example.

title

Defaults and Ranges

keys/options	defaults	range
[]	current picture, held in the variable *select*	valid picture number
text/from	prompts for text if interactive, otherwise faults command	if ***text***, text string; length is machine dependent, if ***from***, valid picture number

transpose

keys:	[from]	<number>	source picture
	[to]	<number>	output picture

Use **transpose** to transpose pictures, that is, to interchange the *x* and *y* axes so that they are reflected in the top left/bottom right diagonal. **transpose** will transpose an image much faster than **extract**.

Examples

```
transpose display:3
```

This command transposes display picture 3.

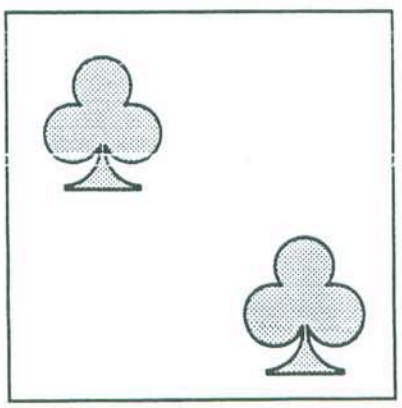
```
transpose 50 to 51
```

This command transposes picture 50 to 51.

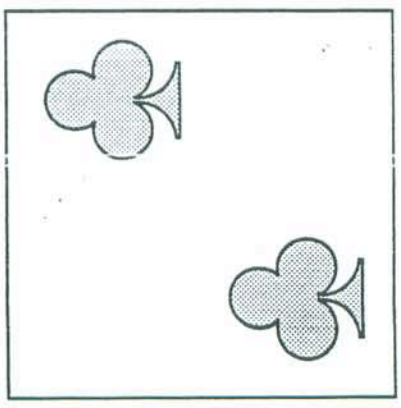
Description

transpose requires a square picture of a factorisable size. (Type **show sizes** for a list of suitable sizes). You can, of course, **cut** if necessary, to a suitable larger size, **transpose** and then **cut** again afterwards. The recorded picture origin is updated appropriately (rounded to the nearest pixel).

The diagram below shows the effect of the **transpose** command on an image.



source picture



output picture

transpose

Notes:

restrictions:	image size must be square and factorisable
multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**cut**, **extract**, **show**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

turn

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
options:	**anticlockwise**		turn 90 degrees anti-clockwise rather than clockwise
	upsidedown		turn 180 degrees
	over		reflect *x* axis (interchanging left and right sides)

Use **turn** to perform simple picture reflections and rotations. **turn** is much quicker at these tasks than **extract.**

Examples

```
turn 1 to 2   (or turn 1 to 2 anticlockwise)
```

This command rotates picture 1 by 90 degrees clockwise (or anti-clockwise) and places the output in picture 2.

```
turn 50 over
```

This command reflects the *x*–axis of picture 50 (like turning the page of a book).

```
turn display upsidedown
```

This command rotates the display picture by 180 degrees (turning it upsidedown).

```
turn over upsidedown
```

This command reflects the *y* axis (turns over top to bottom rather than side to side).

Description

To **turn** a picture by 90 degrees, the picture must be square, and its size must be a power of two. You can of course **cut** a picture, if necessary, to a suitable (larger) size, **turn** and then **cut** again afterwards.

Note that if you use the options **over** and **upsidedown** together, they reflect the *y* axis rather than the *x* axis.

Rotations are about the picture centre rather than at the origin, so the source picture origin is not always exactly preserved.

turn

The diagram below illustrates the effect of the **turn** command and the different combinations of options. Note that some combinations have the same effect on the image.

original image

turn (default)

turn anticlockwise

turn upsidedown

turn over

turn over upsidedown

turn over anticlockwise

turn anticlockwise upsidedown

turn over anticlockwise upsidedown

Notes:

restrictions:	image size must be square and a power of two
multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**cut**, **extract**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

type

<variable name(s)> *'<text string>' <variable name(s)>*	***type*** uses a special syntax. It types the value of the specified variable(s) and any given text string at the terminal

Use **type** to print text or numeric values on the console.

Examples

```
type min,max
```

This command types the value of the variables *min* and *max* on the console.

```
type '350 degrees is ', 350*pi/180, 'radians'
```

This command types the the specified text strings and the value of the numeric expression on the console.

Description

You can use **type** interactively or within a program to display the values of variables. **type** allows you to mix expression values and text. Specify text within single quotes and use a comma to separate values or expressions.

Notes

see also: **ask, log**

uif

keys:	**read**	*'<text>'*	read a user interface from a file
	save	*'<text>'*	save the current user interface to a file
options:	**enable**		initialises the user interface system
	exit		leaves the user interface system and returns to Semper
	go		starts up the user interface system
	status		displays the current status of the interface system
	stop		leaves the user interface system and returns to the operating system

The **uif** command controls the loading, saving and execution of Semper 6 *Plus* user interfaces. For detail of Semper 6 *Plus* interfaces, read the following manuals:

- *User Interface Guide*
- *Tutor User Guide*

contained in the *Semper 6 Guide*.

Examples

```
uif enable; uif read 'standard.uif'
```

This command reads a pre-defined user interface stored in the file *standard.uif*.

```
uif go
```

This command starts the currently defined user interface.

Description

You must type the command **uif enable** before typing any other user interface command. **uif enable** initialises the user interface system by setting up internal variables. You can then define your own user interface using the commands **panel**, **menu**, **cell** etc.

To start the user interface system type **uif go**. This command displays the defined user interface or standard interface if you have not defined your own interface.

uif

You can leave the user interface system by typing one of the following commands:

- **uif exit**
- **uif stop**

The command **uif exit** returns to Semper. To re-enter the user interface type **uif go**. This sequence of commands is useful when developing or debugging a user interface. The command **uif stop** returns to the operating system.

The **status** option details the number of different *objects* that are available or already used (objects include panels, menus, cells and textfields) and how much of the text space remains. You use the text space for storing names, actions and other strings.

Use the **read** and **save** keys to read and save a user interface. The **save** key saves the current interface together with all currently set Semper variables. The **read** key restores a previously defined user interface. This key also restores all variables that are not write-protected.

Notes

variables set: **read** sets all variables that were set when the specified user interface was saved

see also: **cell**, **device**, **execute**, **justification**, **menu**, **mouse**, **panel**, **textfield**

Defaults and Ranges

keys/options	defaults	range
read	*none*	valid filename
save	*none*	valid filename

unix

This syntax is specific to... workstations running Unix

keys:	[]	*'<text>'*	Unix command as textstring

Use the **unix** command to execute non-interactive Unix commands from within a Semper session.

Examples

```
unix 'ls .'
```

This command lists the contents of the current directory on the console.

```
unix 'rm junk.dat'
```

This command deletes the file *junk.dat*.

```
unix 'xclock &'
```

This command displays the X clock.

Description

The Unix command string is specified as a Semper textstring immediately following the command name. If it is omitted, Semper prompts for the command string.

The string is passed to a separate process running a C shell (csh) and is executed by the shell. Standard input for the shell is connected to */dev/null*. Any attempt to read standard input will generate an end-of-file condition (this means that you can't interact with the C shell – to do that you should open a new terminal window using the window manager menus or a Unix command like *xterm*). Standard output is copied to Semper's console output stream and standard error is connected to Semper's standard error (normally the terminal window from which Semper starts up).

The **unix** command waits until the commands passed to the shell are completed and return control to the shell. Some commands (like *xclock* in the last example above) do not return to the shell until they are explicitly terminated by other means, in which case, they should be executed as background processes. If the **unix** command is abandoned, all dependent processes will be terminated.

Defaults and Ranges

keys/options	defaults	range
[]	*none*; prompts for command string if interactive	valid Unix or C shell command string

unpack

<number(s)>	types the three character name for the specified packed integer value(s)

Use **unpack** to type a three-character name for a specified packed integer value. It performs the opposite operation to **pack**.

Examples

```
unpack 26219, 26232, 30786, 30792
```

This command unpacks the specified integer values. Semper produces the following output:

```
26219 unpacks to 'pos'
26232 unpacks to 'po2'
30786 unpacks to 'siz'
30792 unpacks to 'si2'
```

Notes

see also: **pack**

unset

<variable name(s)>	unset the specified variable(s)

Use **unset** to unset permanently a variable, option or key that was set previously using an assignment, for example, *name=value*.

Examples

```
unset type
```

This command unsets the variable called *type*.

```
unset mtpass, erase
```

This command unsets the variables called *mtpass* and *erase*.

Description

Semper maintains a list of the variables that are set at any given time. A variable that is not on this list has no value at all and is called *unset*. You can use the **unset** command to unset a variable but note that you cannot unset a *protected* or *fixed* variable. See *Appendix F, Protected and Fixed Variables* for further details.

user

*The **user** command has no keys or options*

The **user** command is provided to help you to incorporate your own Fortran modules into Semper as new commands. The command itself does nothing, but is included to show you where to place user-written routines within Semper and how to compile and link them into the system.

For a detailed description of how to write your own commands that can be added to Semper, consult the following manual:

Semper 6 Plus Fortran Programmers' Guide

The subroutine called by the **user** command is listed below:

```
C Semper 6 null processing module USER
C
      SUBROUTINE USER
C This is a minimal routine used as a place marker for
C including new user written routines. The user is free to
C modify this routine into something more useful!
C
      RETURN
C
C Copyright (C) 1987-1990: Synoptics Ltd, All Rights Reserved
C
       END
```

This is a silly change from V6.2, throwing away a useful guide to new programmers.

vfc

This syntax is specific to...
SPARCstations running X Windows/OpenWindows

keys:	**port**	*<number>*	specify RCA video port number to use
	to	*<number>*	specify picture number for storing image
	size	*<number>,<number>*	specify size of image subregion
	position	*<number>,<number>*	specify position of image subregion
options:	**on/off**		assign/deassign the VideoPix device
	mono/colour/automatic		specify monochrome/colour/automatic capture
	grab/live		specify single/continuous acquisition
	svideo		specify Super Video input port

The **vfc** command allows you to directly control a **Sun VideoPix** framegrabber and collect images from it.

Examples

```
vfc on port 1
```

This command allocates the VideoPix and switches input to RCA port 1

```
vfc grab to 3
```

This command grabs a full frame image from the VideoPix and copies it to picture 3.

```
vfc live
```

This command previews incoming video data on the Semper display window.

Description

The display server must be an 8-plane (256 colour) workstation. The VideoPix device must be assigned using the **on** option before any images can be acquired. When you have finished acquiring images you can free the device for other programs and users with the **off** option.

The input port can be one of the two RCA ports or the Super Video port. You can choose which using the **port** key and the **svideo** option.

vfc

The incoming video is sensed automatically to determine the TV standard (NTSC or PAL) and whether there is any colour information. You can override the colour sense information using the options **monochrome** and **colour**. The option **automatic** restores the default sense operation.

Images can be acquired as single frames with the option **grab**, or continuously with the option **live**. If the **live** option is used the Semper display is used to preview the images and hence must be assigned. Continuous acquisition is terminated by pressing any key or mouse button when the focus is in a Semper window, at this point the original contents of the display are restored.

The last image acquired can be stored into a Semper image specified by the key **to**. The size of this image defaults to full frame, but can be overidden by the **size** key. In addition the position of the acquired region relative to the top left of the frame can be specified with the **position** key. The size and position keys are both in units of pixels. So, for example, the command:

```
vfc to 1 size 256,256 position 100,200
```

will store a 256 pixel square region from the last acquired image located 100 pixels from the left edge of the image and 200 pixels from the top.

Notes

The TV standard is determined when you issue a *vfc on* command. If you have different standards on different ports you should deallocate the VideoPix between port changes. For example:

```
vfc on port 1 grab to 1; vfc off; vfc on port 2 grab to 2
```

Defaults and Ranges

keys/options	defaults	range
port	*none*	1 or 2
size	*full frame for TV standard*	positive integers
position	*0,0*	positive integers
to	*none*	any valid picture number

vfc

This syntax is specific to...
SPARCstations running X Windows with a Sun VideoPix frame grabber

keys:	**port**	*<number>*	RCA video port number to use
	to	*<number>*	picture number for storing image
	size	*<number>,<number>*	size of image subregion
	position	*<number>,<number>*	subregion offset from top left of frame
options:	**on/off**		assign/deassign the VideoPix device
	mono/colour/automatic		select monochrome/colour/automatic capture
	grab/live		select single/continuous acquisition
	svideo		select Super Video input port

The **vfc** command allows you to directly control a *Sun VideoPix* framegrabber and collect images from it.

Examples

```
vfc on port 1
```

This command allocates the VideoPix and switches input to RCA port 1

```
vfc grab to 3
```

This command grabs a full frame image from the VideoPix and copies it to picture 3.

```
vfc live
```

This command previews incoming video data on the Semper display window.

Description

The display server must be an 8-plane (256 colour) workstation. The VideoPix device must be assigned using the **on** option before any images can be acquired. When you have finished acquiring images you can free the device for other programs and users with the **off** option.

The input port can be one of the two RCA ports or the Super Video port. You can choose which using the **port** key and the **svideo** option.

vfc

The incoming video signal is sensed automatically to determine the TV standard (NTSC or PAL) and whether there is any colour information. You can override the colour sense information using the options **monochrome** and **colour**. The option **automatic** restores the default sense operation.

Images can be acquired as single frames with the option **grab**, or continuously with the option **live**. If the **live** option is used, the Semper display is used to preview the images and hence must be assigned. Continuous acquisition is terminated by pressing any key or mouse button when the focus is in a Semper window. At this point the original contents of the display are restored.

The last image acquired can be stored into a Semper image specified by the key **to**. The size of this image defaults to full frame, but can be overidden by the **size** key. In addition the position of the acquired region relative to the top left of the frame can be specified with the **position** key. The size and position keys are both in units of pixels. So, for example, the command:

```
vfc to 1 size 256,256 position 100,200
```

will store a 256 pixel square region from the last acquired image located 100 pixels from the left edge of the image and 200 pixels from the top.

The TV standard is determined when you issue the command **vfc on**. If you have different standards on different ports you should deallocate the VideoPix between port changes. For example:

```
vfc on port 1 grab to 1
vfc off
vfc on port 2 grab to 2
```

Defaults and Ranges

keys/options	defaults	range
port	*none*	1 or 2
size	full frame for TV standard	positive integers
position	0,0	positive integers
to	*none*	valid picture number

video

This syntax is specific to...
Silicon Graphics workstations with an Imaging Technology FG-100 frame grabber

keys:	**channel**	*<number>*	video input channel
	ilut	*<number>*	input look-up table
options:	**pll/crystal**		obtain sync from external signal using dual phase-locked loop or from the internal crystal oscillator
	show		list current video input settings

Use the **video** command to change or examine the current video input settings.

Examples

```
video show
```

This command lists the current video input settings.

```
video pll
```

This command configures the frame grabber to synchronise with the input signal.

```
video channel 1 ilut 2
```

This command selects video input channel 1 and input look-up table 2.

Description

The current video input settings consist of:

- the video input channel from which the frame grabber obtains its signal
- the input look-up table used to map the input data before storing it in the frame grabber's memory
- the source for synchronising the frame grabber and the input signal

video

At the start of a Semper session the video settings are as follows:

- video input channel 0
- input look-up table 1
- sync source internal – crystal oscillator

When you change any of these settings, using the **video** command's keys and options, they remain as specified until you explicitly change them again. Type the command **video show** to list the current video input settings.

There are four possible video input channels numbered from 0 to 3. Video input channels 0, 1 and 2 are always available for use. The last channel may be used for passing a sync signal, depending on the jumper settings on the frame grabber board, in which case it should not be selected for input. Consult the *FG-100 User's Manual* for details about the jumper settings. Video input channel 0 is the default at the start of a Semper session. Use the **channel** key to change the input channel.

The frame grabber stores sixteen input look-up tables (numbered from 1 to 16). You can use any *one of these* to transform the data as it is input into the frame grabber's memory. Specify the look-up table number using the **ilut** key. All the look-up tables are initialised to linear ramps that leave the data unmodified.

It is necessary to synchronise the frame grabber and the input source for images to be correctly digitised. The FG-100 board provides numerous options for this and you should establish which one you intend to use before installing the framestore. Specify the **crystal** option if the input source synchronises itself to the frame grabber's internally generated signal, and the **pll** option if the frame grabber needs to be synchronised with an external signal.

Notes

see also: **grab, ilut**

Defaults and Ranges

keys/options	defaults	range
channel	*none*	positive integer (0 to 3)
ilut	*none*	positive integer (1 to 16)
pll/crystal	*none*	

video

This command is specific to...
PC + Data Translation DT2861 framestore
PC + Imaging Technology PCVISIONplus framestore
PC + Matrox PIP512/1024 framestore
PC + Quantimet 520 framestore
PC + Metrabyte MV1 framestore

This syntax is specific to...
PC + Data Translation DT2861 framestore

keys:	**start**	*<n1>, <n2>*	specify start of image capture window limits
	end	*<n1>, <n2>*	specify end of image capture window limits
options:	**pll/crystal**		the framestore uses its phase locked loop with an external sync, or its own internal crystal, as the sync source
	reset		sets the sync source to internal (***crystal***) and the image capture window limits to be the entire 512 by 512 frame

This syntax is specific to...
PC + Imaging Technology PCVISIONplus framestore

keys:	**partition**	*<number>*	centre the live image on the specified display partition
	ilut	*<number>*	specify an input lookup-table
	channel	*<number>*	specify video input channel (1 or 2)
	mask	*<number>*	set the video input write-protect mask
	izoom	*<number>*	specify the sub-sampling to be applied to the video input
options:	**setup**		makes the framestore go *live* and allows you to alter the input gain and offset using the keyboard whilst viewing a live image
	reset		sets the sync source to internal (***crystal***) and the video input write-protect mask to 1
	pll/crystal		the framestore uses its phase locked loop with an external sync source, or its own internal crystal, as the sync source

video

This syntax is specific to...
PC + Matrox PIP512/1024 framestore

keys:	**camera**	*<number>*	specify framestore video input channel
	offset	*<number>*	specify the offset to be applied to the input signal
	gain	*<number>*	specify the gain to be applied to the input signal
options:	**setup**		makes the framestore go *live* and allows you to alter the input gain and offset interactively on a live image using the cursor keys
	pll/crystal		the framestore uses its phase locked loop with an external sync source, or its own internal crystal, as the sync source

This syntax is specific to...
PC + Metrabyte Corporation MV1 framestore

keys:	**lut**	*<number>*	specify an input look-up table
	channel	*<number>*	specify the video input channel (1 or 2)
	gain	*<number>*	specify the gain setting for the video input signal
	offset	*<number>*	specify the offset to be applied to the video input signal
	roffset	*<number>*	specify the offset to be applied to the video input of the red framestore (full colour systems only)
	goffset	*<number>*	specify the offset to be applied to the video input of the green framestore (full colour systems only)
	boffset	*<number>*	specify the offset to be applied to the video input of the blue framestore (full colour systems only)
options:	**pll/crystal**		the framestore uses its phase locked loop with an external sync source, or its own internal crystal, as the sync source
	reset		resets the sync source to internal (crystal)
	setup		causes the framestore to go live and allows you to alter the input offset using the cursor keys while viewing a live image

video

This syntax is specific to...
PC + Quantimet 520 greystore

keys:	**time**	*<number>*	select the time constant for the recursive filter
	partition	*<number>*	specify partition into which the image is to be grabbed
	gain	*<number>*	specify the gain to be applied to the video input signal
	offset	*<number>*	specify the offset to be applied to the video input signal
options:	**setup**		adjust the video gain and offset interactively
	start/stop		start/stop video acquisition
	variables		record the values of the video input gain and offset in the Semper variables *fsg* and *fso*

Use the **video** command to set up your framestore in preparation for grabbing frames from an image source.

Examples

PC + Data Translation DT2861 framestore only

```
video start 0,0 end 255,511
```

This command sets the left hand half of the 512 by 512 screen as the image capture limits.

```
video reset
```

This command is equivalent to **video crystal start 0,0 end 511, 511**. It sets the sync source to internal (crystal) and the image capture window limits to the entire 512 by 512 screen.

---

video

PC + Imaging Technology PCVISIONplus framestore only

```
video ilut 1 channel 1 pll
```

This command selects input look-up table 1, video input channel 1 and the framestore phase locked loop to lock onto the input signal sync as the sync source.

PC + Matrox PIP512/1024 framestore only

```
video camera 2 offset 5.8 gain 0.9 pll
```

This command selects framestore video input 2, applies an offset of 5.8 Volts and a gain of 90% of the maximum available gain to the input video signal. The framestore uses its phase locked loop to lock onto an external sync source.

PC + Quantimet 520 greystore only

```
video setup
```

This command allows you to set up the gain and offset using the cursor keys, whilst you are viewing a live image.

```
video offset 5 gain 7
```

This command applies an offset of 5 and a gain of 7 to the video input signal.

PC + Metrabyte Corporation MV1 framestore only

```
video ilut 1 channel 1 pll
```

This command selects input look-up table 1, video input channel 1 and the framestore phase locked loop to lock onto the input signal sync as the sync source.

---

video

Description

PC + Data Translation DT2861 only

The **video** command allows you to:

- specify the image capture window limits (using **start** and **end**)
- select the sync source, external (**pll**) or internal (**crystal**)
- reset to the default framestore state (**reset**)

The image capture window limits define the area of the frame that accepts live input. The area that you define applies to any frame used for live or slowscan input. To define the image capture window limits, specify **start** *xstart*, *ystart* and **end** *xend*, *yend* where *xstart*, *ystart*, *xend* and *yend* are coordinates relative to the top left of the frame:

- *xstart* can take the values 0, 16, 32, 48, 64 ..... 224, 240
- *ystart* can take the values 0, 1, 2, 3, 4 ..... 254, 255
- *xend* can take the values 15, 31, 47, 63, 79 ..... 239, 255
- *yend* can take the values 0, 1, 2, 3, 4 ..... 254, 255

PC + Imaging Technology PCVISIONplus framestore only

The **video** command is designed to allow you to set up the framestore for frame grabbing before using **live** or **snap**. A number of keys and options allow you to:

- specify the display partition on which the live image is to be centred (**partition**)
- specify the input look-up table to use for video input (**ilut**)
- specify video input channel 1 or 2 (**channel**)
- set input write-protect mask (**mask**).
- sub-sample the live image (**izoom**)
- alter the input gain and offset interactively (using **setup**)
- reset to framestore defaults (**reset**)
- select a sync source, external (**pll**) or internal (**crystal**)

Note that a zero value for **mask** allows you to write to all bit planes, a value of 127 allows you to write to the most significant bit plane etc.

video

PC + Matrox PIP512/PIP1024 framestore only

The **video** command allows you to:

- select the camera input channel (**camera**)
- set the gain and offset to be applied to the input signal (**gain**, **offset** and **setup**)
- select the sync source, external (**pll**) or internal (**crystal**)
- alter the gain and offset whilst viewing a live image, using the cursor keys (**setup**)

PC + Quantimet 520 greystore only

The **video** command allows you to:

- select the gain and offset to be applied to the input signal (**gain**, **offset** and **setup**)
- select a time constant for the recursive filter (**time**)
- start and stop video acquisition (**start** and **stop**)
- record the values of the video gain and offset in Semper variables *fsg* and *fso* (**variables**)

The **video** command is designed to allow you to set up the video input gain and offset, before using the commands **live** or **snap**. You can select the gain and offset using the **gain** and **offset** keys or by using the cursor keys whilst viewing a live image if you specify the **setup** option.

You can use the **time** key to select the time constant for the recursive filter. Values of 1, 2 and 3 correspond to time constants of 0.5, 0.125 and 1.0 seconds (these figures may vary for some greystore versions).

The **start** and **stop** options allow you to start and stop video acquisition as two separate commands.

The **variables** option records the current values of the video input gain and offset of the Quantimet 520 in the Semper variables *fsgain* and *fsoffset*. These variables are set just before the **video** command returns to Semper.

video

PC + Metrabyte Corporation MV1 framestore only

The **video** command is designed to allow you to set up the framestore for frame grabbing before using **live** or **snap**. A number of keys and options allow you to:

- specify the input look-up table to use for video input (**ilut**)
- specify a video input channel 1, 2, 3 or 4 (**channel**)
- set the gain and offset to be applied to the input signal (**gain**, **offset, roffset, goffset, boffset**)
- alter the input gain and offset interactively (using **setup**)
- reset to framestore defaults (**reset**)
- select a sync source, external (**pll**) or internal (**crystal**)

The **video** command is designed to allow you to set up the video input gain and offset, before using the commands **live** or **snap**. You can set up the gain and offset by specifying values with the **gain** and **offset** keys. The offset can also be set using the cursor keys whilst viewing a live image after specifying the **setup** option.

Notes

see also: **live, slowscan, sscan, snap**

variables set:
fsg (If *Quantimet 520* and **variables** option is set, recorded video input gain)
fso (if *Quantimet 520* and **variables** option is set, recorded video input offset

video

Defaults and Ranges

keys/options	defaults	range
start	*none*	see ***Description***
end	*none*	see ***Description***
time	time=0 (no filtering)	0, 1, 2 or 3
partition	current display partition, held in the variable *display*	valid partition number
ilut	*none*	valid input look-up table number
channel	*none*	channel 1 or 2, if *Metrabyte MV1*, channels 1, 2, 3 or 4
mask	*none*	integer in the range 0 to 225
izoom	*none*	1 or 2
camera	*none*	1, 2 or 3
gain	*none*	if *Quantimet*, an integer in the range 0 to 15, if *Matrox* a real number in the range 0 to 1 to express a percentage, if *Metrabyte MV1*, a gain factor of 0.5, 1.0, 1.5 and 2.0
offset	*none*	if *Quantimet*, an integer in the range 0 to 15, if *Matrox* a positive real number in the range 4.26 to 5.98 to express a voltage, if *Metrabyte MV1*, an integer in the range 0 to 255
roffset	*none*	integer in the range 0 to 255
goffset	*none*	integer in the range 0 to 255
boffset	*none*	integer in the range 0 to 255
pll/crystal	internal sync source (crystal)	

view

keys:	[ ]	*<number>*	picture, partition or frame to be viewed
		<n1>...<n9>	list of pictures/partitions/frames to be viewed in turn
	lut	*<number>*	look-up table to be used for viewing
	zoom	*<number>*	(integral) zoom factor for viewing
	times	*<number>*	number of viewing cycles in rotating view mode
	wait	*<number>*	number of seconds for which each view is presented in rotating viewing mode
	size	*<x>, <y>*	dimensions of subregion to be viewed
	position	*<x>, <y>*	position/offset of subregion
	pan	*<x>, <y>*	position for centre of monitor screen
options:	picture/partition/frame		view picture, partition or frame
	left/right, top/bottom		position of subregion
	clip		blank out view outside picture/partition/frame border
	enquire		return current view parameters in variables *f,z,x1,x2,y1,y2*
	re/im		centre view on real or imaginary part of complex display picture

view allows you to select the region of display memory presented on the viewing monitor screen, the zoom factor and the look-up-table that is used. **view** also allows you to rotate between two or more views in succession. For information on the general key called **view**, refer to *Appendix C, Semper Keys and Options*.

Examples

```
view zoom 8 pan x,y
```

This command views the current display picture at 8 times the normal magnification with position *x, y* centred on the monitor screen.

```
view partition display:5 lut 2
```

This command views partition *display:5* using look-up table number 2.

```
view display:3,display:4
```

This command views pictures *display:3* and *display:4* five times in alternation, switching between the two at one second intervals.

view

```
lut 3 create colour; view display:5 lut 3
```

This command views picture *display:5* which should consist of red, green and blue layers displayed in three successive frames in full colour mode. Use the command **lut 3 create false** for false colour viewing.

```
view frame 3 top right
```

This command views the top right of frame 3 (or the current frame, held in the variable *cframe*, if you omit the frame number) using the current look-up-table, held in the variable *clut*.

Description

view (and **lut**) do not alter display picture data as stored or as seen by Semper. They simply alter how they are presented on the screen.

Note that the full facilities provided by **view** may not be available on your installation and some details are determined by choices made at the time of installation. Ideally, the region that you request is presented centrally and the rest of the screen is blanked, but you may find that your installation views without blanking, 'wraps around' at frame boundaries, or does not always centre the view as you request. Type **help installation** for detail or just experiment with the **view** command on your installation.

The graphics mode that you select (**picture**, **partition** or **frame**) determines the blanking limits, and what is centred on the screen by default, that is, the appropriate coordinate origin. You can move the centre using the key **pan** (as in the second example), which *pans* the monitor to the specified position (in the appropriate coordinate system).

You can specify a viewing region using the standard 2-D subregion keys and options, for example, **size**, **position** etc. (For further detail, refer to *Appendix C, Semper Keys and Options*). The viewing region also depends on the appropriate coordinate system and the size of the monitor (subdivided if **zoom** is not 1). Therefore you can use the **position** key as a synonym for **pan** whenever the implied region is not truncated at the picture, partition or frame boundary.

For complex display pictures, where the real and imaginary parts of the image are displayed side by side, the view will normally be centred on the region which includes both parts. You can restrict the view to just the real or imaginary part by specifying the option **re** or **im.**

You can view displays in *rotation* by specifying up to nine picture, partition or frame numbers. You can alter the number of times that you view the set of displays using the **times** key and the number of seconds for which Semper waits on each display using the **wait** key, for example:

```
view display:2,display:3 times 3 wait 2
```

view

You can list up to nine pictures for successive viewing in this way.

If your display hardware allows, you can specify the **clip** option to blank out part of the display which lies outside the picture, partition or frame border.

If the **enquire** option is set, the current view parameters: the number of the frame currently in view, the zoom factor and the extent of the viewable area in the specified coordinate frame (**picture, partition** or **frame** option) are returned in the Semper variables *f,z,x1,x2,y1* and *y2.*

Notes:

variables set:	*f*	frame number of frame currently in view
	z	zoom factor
	x1,x2	X limits of view
	y1,y2	Y limits of view
see also:	**lut**	

Defaults and Ranges

keys/options	defaults	range
[ ]	display, if ***picture*** or ***partition***, current frame, if ***frame***, held in the variable *cframe*	valid picture/partition/frame number
lut	if ***partition*** or ***picture***, partition default, otherwise current look-up-table held in the variable *clut*	valid look-up-table number
zoom	1	positive integer
times	times 5	positive integer
wait	1 second	real number
size	smaller of picture/partition/frame size and monitor size	less than or equal to the size of the picture/partition/frame (integers)
position	position 0,0	within bounds of the picture /partition/frame (integers)
pan	position 0,0 or subregion centre	within bounds of the picture /partition/frame (integers)
picture/partition/ frame	picture	
re/im	centre view on both parts of complex display picture	

wait

keys:	[for]	<*number*>	specify number of seconds to wait

You can make Semper wait for a fixed time, or until you press <*return*>, using the **wait** command.

Examples

```
wait 5
```

This command causes Semper to wait 5 seconds.

```
wait
```

This command types a prompt message and waits until you press a key or mouse button.

Description

Specify the number of seconds delay using the **[for]** key. Note that some installations may not have this timed wait facility.

If you specify **wait 0**, it is not faulted and causes no delay. This allows you to enable or disable program pauses by assigning a value to a variable such as *delay* and using commands such as **wait delay** in your program.

Defaults and Ranges

keys/options	defaults	range
[for]	*none*; waits for the user to press a key before continuing	integer

walsh

keys:	[from]	<number>	source picture
	[to]	<number>	output picture

Use **walsh** to calculate *a Walsh/Hadamard* transform of a picture. This transform is a decomposition into square wave components rather than sinusoidal and are real (for real source pictures). You can use **walsh** for data compression and texture classification. To recover the original image, that is to perform an inverse transform, use the **image** command.

Examples

```
walsh 1 4
```

This command places the Walsh transform of picture 1 into picture 4.

Description

The transform is 'sequency-ordered' with the origins central in both planes irrespective of any different origins that are recorded. You should not send the output directly to the display (**walsh to display**) but must send it to a specified output picture, because of the incompatible (complex) ranges that are used during the intermediate stages of the transform.

Note that the output of **walsh** is complex if the source is complex, and floating-point otherwise.

Notes:

restrictions:	image sizes must be powers of two unsuitable for direct output to display
multi-layer pictures:	faulted
forms used internally:	fp, complex
see also:	**fourier, image**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number

warp

keys:	[from]	<number>	source picture
	[to]	<number>	output picture
	image	<number>	*Plist* containing image control points
	map	<number>	*Plist* containing map control points
	order	<number>	order of mapping polynomial
	layer	<number>	resample specified picture layer
	position	<x>,<y>	position in map image space of the output image
	size	<x>, <y>	size of output picture
	stride	<number>	scale factor in output image
	errmax	<number>	maximum error of control points
options:	bilinear/bicubic		resampling method
	verify		verify information about geometric correction at the console

The **warp** command corrects geometrical distortions in images by fitting a polynomial grid to a set of control points. This command can be used in *Remote Sensing* applications.

Examples

```
warp 2 3 map 4 image 5
```

This command resamples picture 2 to picture 3 using map control points in picture 4 and image control points in picture 5.

```
warp 2 3 map 4 image 5 bicubic order 2
```

This command performs the same functions as the above command but uses a second order polynomial and resamples using bicubic resampling.

```
warp 2 3 map 4 image 5 position 120,100 size 64 stride 10
```

This command resamples image 2 to image 3 using *nearest neighbour* interpolation. A first order polynomial is used with pictures 4 and 5 containing the control points. The area covered by the output image is:

(120.0, 100.0) .. (760.0, 740.0)

since (64 * 10.0 = 640.0)

warp

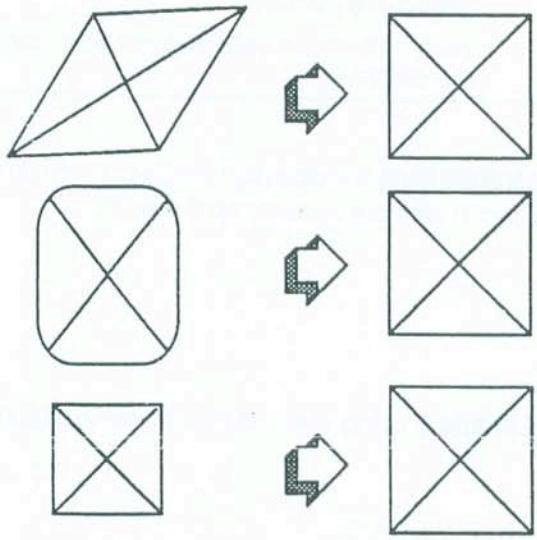
Description

The **warp** command warps the source picture using a polynomial, the coefficients of which are calculated from the control points using the least squares method. The maximum order of polynomial is at least 3 but may be more, depending on the length of a Semper row buffer. You supply the control points in a *Plist* picture, specified by the **map** or **image** keys. You can specify three methods of interpolation:

- *nearest neighbour* (the default)
- **bilinear** interpolation
- **bicubic** interpolation

Use the **layer** key to specify a layer of the picture to resample.

The diagram below illustrates some uses of the **warp** command:



By default, the output picture is the same size as the input image. To change the dimensions, use the **size** key. Use the **position** key to change the position of the output image in map (reference image) space. The point specified by **position** is the bottom left of the image. Use the **stride** key to define a scale factor for the image. Note that if **stride** is greater than 1 then for similarly scaled images, features will shrink.

Use the **verify** option to list the registration accuracy at the console. This consists of the control points and their calculated position as compared with their given position. The (standard) error in the *x* and *y* directions is also listed.

Use the **errmax** key to define the maximum displacement that is allowed between a control point and its calculated position. If **warp** finds a point that exceeds this value then the point is dropped and

warp

the coefficients are re-calculated. If not enough 'good' points remain the command gives an error message. This process is repeated until all points are within the required accuracy or not enough points remain. If you specify the **verify** option, the accuracy information is displayed for each cycle.

The first order polynomial is as follows:

$$u = a0 + a1.x + a2.y$$
$$v = b0 + b1.x + b2.y$$

The second order polynomial is given below:

$$u = a0 + a1.x + a2.y + a3.x^2 + a4.x.y + a5.y^2$$
$$v = b0 + b1.x + b2.y + b3.x^2 + b4.x.y + b5.y^2$$

where *u, v* refer to uncorrected image coordinates and *x, y* refer to map reference coordinates.

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
image	*none*	valid *Plist* picture number
map	*none*	valid *Plist* picture number
order	1	positive integer
layer	all layers	integer in range 1 to number of layers
position	position 0,0	positive integers
size	size of input image	positive integers
stride	1	positive integer
errmax	*none*	positive integer
bilinear/bicubic	*nearest neighbour*	
verify	verification off	

weight

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	with	*<number>*	1-D picture defining radial function
	position	*<x>,<y>*	position from which distance to pixels is measured
options:	**add**		add function of radius

Use **weight** to multiply a picture by an arbitrary function of distance from a given point – a *radial* function. You supply this function in a 1-D picture. You can also add the radial function, instead of multiplying. (If you want to multiply by a function with a simple mathematical form, use the **calculate** command instead).

Examples

```
weight 50 with 3 to 51
```

This command uses the 1-D picture 3 to define a function of radius by which picture 50 is multiplied to form picture 51.

```
weight 1 with 2 add
```

This command adds to picture 1 the radial function defined in 1-D picture 2.

```
fourier 1 2; ps to 3; section; display; xwires graph to 4
weight 2 with 4; image
```

This sequence of commands applies a hand-drawn Fourier transform filter to picture 1.

Description

weight multiplies source pixels that are distance *r* from the origin, or from another position you specify using the **position** key. The source pixels are multiplied (increased) by the value of the 1-D picture at position *r* (linearly interpolated if necessary). The 1-D picture does not need to have the same size as the source: the rightmost pixel is treated as repeated indefinitely if necessary, and pixels left of the origin are simply ignored.

Notes

multi-layer pictures:	faulted
forms used internally:	complex
see also:	**calculate**

weight

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	picture 999	valid picture number
position	position 0,0	within bounds of picture (integers)
add	multiply by radial function	

window

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	output picture
	position	*<x>,<y>*	lattice origin
	width	*<number>*	width in lattice coordinates, of regions retained around each lattice site
	radius	*<number>*	maximum radius of lattice sites that are retained
options:	**verify**		verify results on the console

Use **window** to perform a *window array* Fourier transform filter that retains only components close to a site of a predefined lattice. This command allows you to perform *lattice averaging* of a noisy image of a periodic structure, by filtering out Fourier components that do not have the right periodicity.

Examples

```
u=20,5 v=3,18; window display
```

This command masks the display picture retaining only pixels near sites of the lattice defined by *u, v*. The above command is not in itself useful, but will show you clearly what **window** does.

```
window 50 to 51 width .1 radius 120
```

This command masks picture 50 with windows whose sides are a fraction 0.1 of the corresponding lattice base vectors in length, also setting to zero any pixels that are more than 120 units from the origin.

```
fourier; min=0 max=1e8; ps to display; library lattice
window; image; display
```

This sequence of commands performs a complete filtration process on the current picture (though you may need to adjust the display grey level to suit your data). You indicate the lattice using the display cursor, when prompted by the **library** program.

Description

You can apply **window** to full-plane and half-plane Fourier pictures.

The transform of an image of a periodic structure is localised near an array of peaks arranged in a (*reciprocal*) lattice, while the image noise is spread throughout the transform. You can therefore improve the image substantially by setting to zero all pixels that are not near a lattice site – which can be seen as applying a mask consisting of an array of small windows centred on the lattice sites.

window

The lattice is defined by base vectors *u*, *u2* and *v*, *v2*. You can use the **position** key to move its origin. The generated windows are parallelograms with sides **width** (0.2 in default) multiplied by the corresponding base vector length. Note that smaller windows perform better filtering, but are more likely to cause artefacts if you do not know the base vectors accurately or if the image is imperfectly periodic. In any case, some signal distortion is likely if the windows are smaller than 3 pixels wide. Use the **verify** option to send details of the filtering operation to the console.

You can also see the effect of the filter as a local averaging of unit cells in real space. Transform windows of width *1/n* correspond roughly to averaging the image over regions *n* cells wide. For alternative ways of effecting the average, see the commands **lattice**, **motif**.

If you need to interchange real and reciprocal base vectors, use the program **library reciprocal**.

Notes

multi-layer pictures:	faulted
forms used internally:	complex
variables used:	*u, u2, v, v2* (lattice base vectors)
see also:	**lattice**, **motif**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current picture, held in the variable *from*	valid picture number
position	position 0,0	within bounds of picture (integers)
width	width 0.2	real number
radius	infinite	within bounds of picture (integer)
verify	verification off	

wp

keys:	[]	*<number>*	write-protect specified picture
		<n1>, <n2>	write-protect existing pictures in specified range of picture numbers
options:	**on/off**		turn write-protection on or off

You write-protect a picture with **wp**, and remove the protection using **wp off**.

Examples

```
wp 40
```

This command protects picture 40 against subsequent deletion or alteration.

```
wp 100, 200
```

This command protects all pictures in the range 100 to 200.

```
wp off 4:3, 4:5
```

This command disables write-protection from pictures 4003, 4004 and 4005.

Defaults and Ranges

keys/options	defaults	range
[]	current picture, held in the variable *select*	valid picture number
on/off	turn write-protection on	

write

keys:	**[from]**	*<number>*	picture to be written
	name	*'<text>'*	name of file to which picture is to be written
	format	*'<text>'*	Fortran format used for ***write*** statements which outputs each row in turn
options:	**new**		overwrite an existing output file
	unformatted		write using Fortran unformatted mode
	unlabelled		write without including picture label information

You use **write** to output pictures to files independent of Semper, for transfer to other programs or other machines, or possibly for archive purposes (although the command **save** is usually faster and more convenient for this purpose).

Examples

```
write 23 name 'sec.', n
```

This command writes picture 23 to a file named *sec.dat* followed by the value of the variable *n*.

```
write format '(10F8.2)' name 'scantransform'
```

This command writes the current picture in 10F8.2 format to the file *scantransform.dat*.

```
write unformatted
```

This command prompts for a filename on the console and writes the current picture into the Fortran unformatted mode.

```
write scr:12 unlabelled new name 'toxyz.dat'
```

This command writes picture 12 from device *scr* to a new file named *toxyz.dat* with no picture label in the file.

Description

Pictures produced by **write** can be recovered by **read**.

If you do not specify an extension for the filename, the default extension *.dat* is assumed. Use the **new** option to overwrite an existing file, if an output file of the specified name already exists. Semper displays an error message if you try to overwrite an existing file without this option.

In most cases, **write** produces a character file that you can easily inspect and subsequently edit. The **unformatted** option generates an unformatted Fortran file, which is faster and more compact, but unintelligible except to Fortran **read** statements.

write

The default formats for byte, integer, floating point and complex pictures are as follows:

- 1X,24I3 for byte pictures
- 1X, 12I6 for integer pictures
- 1X, 1P6E12.6 for complex and floating point pictures

Note that you must specify a complete Fortran format string, including opening and closing brackets, if you are using the **format** key in a **write** statement. For example:

```
write name 'newprog' format '(24I3)'
```

If you wish to omit the picture label information from the output file, use the **unlabelled** option.

For a precise description of the possible file formats, you should refer to the following document:

Semper 6 READ/WRITE file format

which is available from *Synoptics*.

Notes

multi-layer pictures:	fully supported
forms used internally:	integer, fp, complex
see also:	**read**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename
format	depends on picture form	text string containing valid Fortran format string

xcf

keys:	**[from]**	*<number>*	first picture to be cross-correlated
	[to]	*<number>*	output picture
	with	*<number>*	second picture to be cross-correlated
	via	*<number>*	intermediate picture, receiving Fourier transform of ***with*** picture
	radius	*<number>*	if ***search***, radius around peak over which a local centre-of-mass is calculated to replace the raw peak position
	mark	*<number>* *<yes> or <no>*	mark cross-correlation peaks on display or, if ***radius***, the circular region
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**search**		search *xcf* for peak, returning position as *x*, y
	lowest		if ***search***, locate lowest rather than highest point
	negative		if ***search*** and ***radius***, include negative pixels in centre-of-mass search
	squared		if ***search*** and ***radius***, find centre-of-mass of squared pixels
	iterated		if ***search*** and ***radius***, repeat the centre-of-mass location once, with a region shifted to be centred at the centre-of-mass found initially
	verify		verify cross-correlation process at the console

Use **xcf** to calculate cross-correlation functions between pairs of pictures. A cross-correlation function allows you to:

- find a lateral shift needed to align (register), the pictures
- locate motifs occurring at many places in a picture
- detect features buried in high noise levels
- measure signal-to-noise ratios

xcf creates an output picture of class *Correlation*, in floating-point form.

xcf

Examples

```
xcf 1 with 2 to 3; extract 2 to 3 @xy
```

This command calculates the *xcf* of pictures 1 and 2 in picture 3 (class *Correlation*), and produces a shifted version of picture 2 in register with picture 1.

```
xcf 1 with 2; type x,y,t,root(t/(1-t))
```

This command replaces picture 1 with the *xcf* between pictures 1 and 2, and reports the correlation peak position and height, and the signal-to-noise ratio, on the assumption that they are identical apart from noise.

```
fourier 1; xcf 2 with 1; cut 2 size 400 @xy
xcf 3 with 1; cut 3 size 400 @xy
xcf 4 with 1; cut 4 size 400 @xy
```

This sequence of commands produces subregions from pictures 2, 3 and 4 in turn, that are in register with picture 1.

```
create 1 size 512; paste 2; mask radius 35; xcf with 3 nosearch
```

This command produces a correlation map showing positions in a 512 square picture 3 that matches a radius 35 motif in picture 2.

Description

The cross-correlation (*xcf*) of picture *f* (**from**) with picture *w* (**with**) is a map of the **with** picture showing how well *f* matches *w* when its centre is shifted to the position being mapped. The output picture may indicate the following:

- if *f* and *w* are similar apart from a relative displacement, the output picture contains a single strong peak. The position of the peak identifies the displacement.
- if *w* contains many copies of a motif at the centre of *f*, the *xcf* has local peaks marking the position of each such copy.

xcf also searches for the highest correlation peak (unless you specify **nosearch**). You can also search for the lowest peak using the **lowest** option. If you specify a value for the **radius** key, **xcf** also finds a local centre-of-mass around the peak.

If you use the **mark** key to specify the display, **xcf** marks the cross-correlation peak on the display, in the style and size determined by **mkmode** and **mksize**. For details of the keys **mark**, **mkmode** and **mksize** refer to *Appendix C, Semper Keys and Options*.

xcf

In more detail, the form of the normalized *xcf* calculated by **xcf f with w** is:

$$xcf(x') = \frac{\langle f(x)w(x-x')\rangle - \langle f\rangle\langle w\rangle}{\sqrt{(\langle f^2\rangle - \langle f\rangle^2)\,(\langle w^2\rangle - \langle w\rangle^2)}}$$

where <..> denotes an average over the whole picture. The normalization makes the result independent of the grey-scaling of the two pictures (unchanged by constant addition, subtraction, multiplication or division). The results lie in the range –1 to 1:

- values near zero indicate there is little agreement between the pictures
- values near 1 indicate that the pictures agree closely
- values near –1 indicate that the pictures agree closely but with opposite contrast

In general, correlation functions are remarkably powerful at detecting a match in the presence of noise. (Correlating images *n* points square detects a common signal buried in noise with a level *root(n)/2* times that of the signal). Note that substantial differences in imaging conditions between *f* and *w* do not usually prevent **xcf** from finding a match.

Note that the options **negative**, **squared** and **iterated** are relevant only in the centre-of-mass mode, that is, when you set **radius**.

Further information

To calculate the cross average, Semper transforms both source pictures:

- **from** to the output picture **to**
- **with** to the intermediate picture **via**

Semper multiplies the conjugate of the first transform with the second transform before inverse transforming the product.

You can also supply pictures that are already transformed, that is *Fourier* pictures instead of *Images* and **xcf** picks up the calculation at the appropriate point. This allows you to include additional filters to manipulate *xcf* peak shapes and/or noise levels and is also useful if you are aligning several pictures in turn with a reference picture, since you need only transform the reference once.

If the images are not correctly registered, correlation levels are reduced in proportion to a fraction of the image area that is common to both images. If this matters, for example, because you are using the peak height to deduce the signal-to-noise ratio as in the second command example, you can correct the height before proceeding by a sequence of commands such as:

```
xcf..; pcb; t = t * ncols/(ncols-mod(x)) * nrows/(nrows-mod(y))
```

xcf

When you use discrete transform methods, as above, to calculate *xcfs*, the correlated *Images* and the generated *Correlation* are both treated as if repeated periodically in both directions. This means that features shifted off one edge of an *Image* reappear at the opposite edge, and may give rise to spurious correlation. Also, if the *Images* are mis-registered by more than half the field of view, the *Correlation* peak (which should be displaced off one edge of the picture) in fact reappears at the opposite edge in a spurious position. To prevent this, align a larger area in the first instance, more coarsely sampled if necessary.

Notes

restrictions:	image sizes must be powers of two unsuitable for direct output to display
display marking:	peak position, region searched for centre-of-mass
multi-layer pictures:	faulted
forms used internally:	fp (complex in transforms)
variables set:	*x, y* (position of xcf peak found) *t* (height of xcf peak found – or mass if **radius**)
see also:	**fourier**, **pcb**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
with	*none*	valid picture number
via	picture 999	valid picture number
radius	radius 0	positive real number
mark	mark off	see *Appendix C*
mkmode	1 (upright cross)	integer in the range 1 to 5
mksize	2	positive integer
search	search on	
negative	negative pixels included in search	
verify	verification off	

xwires

keys:	[ ]	*<number>*	picture/partition/frame number
	[to]	*<number>*	output picture for ***curve***, ***llst***, ***graph***
	size	*<x>, <y>*	dimensions of skewed subregion if ***region uv***, or number of sampling points to use along ***curve*** or ***graph***
	sampling	*<number>*	sampling interval for ***region***, ***curve*** or ***graph***
	position	*<x>, <y>*	position at which cursor first appears
	mkmode	*<number>*	mark mode
	mksize	*<number>*	mark size
options:	**picture/partition/frame**		use specified coordinate system
	line		set the variables *x, y, xn, yn, r, theta* from marked line
	circle		set the variables *x, y, r* from the marked circle
	arc		set *x, y, r, theta, th2* from marked circular arc
	region		set *x, y, r, r2* from marked subregion
	oned		if ***region***, accept angled 1-D subregion (sets *r2*=1) and set *theta*
	angled		if ***region***, accept rotated subregion and set *theta* also
	uv		if ***region***, accept skewed subregion and set *u, u2, v, v2* also
	section		same as ***region oned***
	llst/curve		generate *Plist* of specified type
	open/closed		if ***curve***, specify type of curve
	graph		generate 1-D picture from marked graph
	symmetric		if ***graph***, generate symmetric data from right hand side only
	antisymmetric		if ***graph***, generate anti-symmetric data from right hand side only
	verify		mark information about the process on the display
	re/im		restrict operation to real/imaginary part of complex display picture
			if **verify**, mark information or real/imaginary part only
	view		switch view to make the display region visible

xwires

xwires allows you to enter positions, directions, regions, curves etc. directly onto the display device, for use in subsequent commands.

Examples

```
xwires; print @xy
```

This command prints a block of pixel values around the point that you mark with the cursor.

```
xwires section; extract @section to 999; display
```

This command extracts a line scan between two points that you mark.

```
xwires display:4 region; cut @region
```

This command cuts out a rectangular region which you mark by a pair of opposite corners.

```
xwires curve closed to 51; type p,a; mask with 51
```

This command types the perimeter *(p)* and the area of a polygonal region *(a)* that you mark, and masks the current picture outside this region.

```
xwires line; type r
```

This command defines a line on the display from two marked points and types the length of the line.

```
xwires circle; mask @circle
```

This defines a circle passing through three marked points and masks the area outside the circle.

```
xwires region angled; extract @region angle theta
```

This command defines a rotated rectangular subregion from three points. The first two points define the opposite corners of the subregion and the third point defines the direction of the new *x* axis with respect to the first point. The region is then extracted from the picture.

```
xwires region uv size 50, 30; extract @region uv
```

This command defines a skewed (parallelogram) subregion. The first two marked points define the opposite corners of a rectangle and the third and fourth points define directions of the *u* and *v* axes with respect to the first point. When the region is extracted, these sides become the horizontal and vertical sides of the subregion.

xwires

```
xwires list to 51
```

This command defines a list of marked positions and stores them as an output *Plist* picture 51. (You end the list by repeating the last point).

Description

If you do not specify an option, **xwires** simply returns the coordinates of the point that you mark (as the coordinates *x,y*). By default, **xwires** works on *display*, in picture coordinate mode. **xwires** also works in **partition** or **frame** mode. For example, **xwires partition display:7**. If you specify the **view** option, **xwires** makes visible the part of the display that you would like to point at.

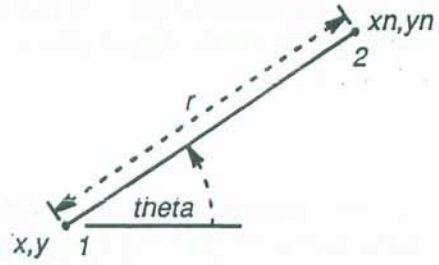
Use the keys **mkmode** and **mksize** to define the type of display marking used by **verify**. Refer to *Appendix C, Semper Keys and Options* for further detail. You can also prevent any verification at all by specifying **noverify**. In some installations, you can control where the display cursor first appears by setting its initial position using the **position** key, for example, **xwires position 10,20** or **xwires @xy**. If the specified position lies outside the frame limits, the cursor appears centrally instead.

On complex display pictures, where the real and imaginary parts of the image are displayed side by side, the cursor normally operates with respect to the real part of the image. If, however, you specify the option **im**, the cursor will operates with respect to the imaginary part of the image. By default, the input data is verified by marking it on both the real and imaginary parts. If you specify the option **re** or **im**, verification is restricted to just the real or imaginary part.

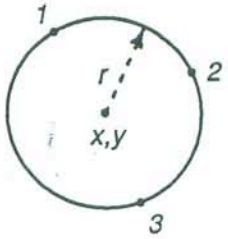
You can use **xwires** to define the following items on the display:

- line
- circle
- arc
- region

xwires line defines a line from the first marked point (*x, y*) to the second marked point (*xn, yn*). It returns the length of the line in the variable *r* and the line direction in *theta* (in radians anti-clockwise from the positive *x* axis).

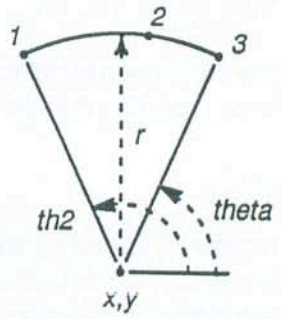


xwires circle defines a circle passing through three marked points. It returns the centre of the circle in the variables *x, y* and the radius in *r*.



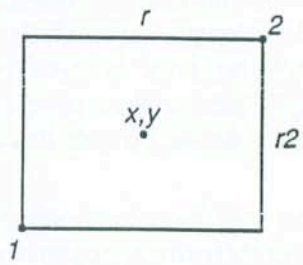
xwires

xwires arc defines a circular arc passing through three marked points, between the first and last points and through the second. It returns the centre of the arc in variables *x, y*, the radius in *r* and the angular range in *theta, th2* (in radians, anti-clockwise from the positive *x* axis). The angular range is set so that the arc is always traversed in an anti-clockwise direction.

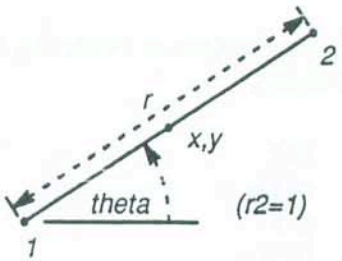


xwires region defines a rectangular region from two marked points which represent the opposite corners of the rectangle. It returns the centre of the rectangle in the variables *x, y* and the size in *r, r2*. You can use the following options with **xwires region**:

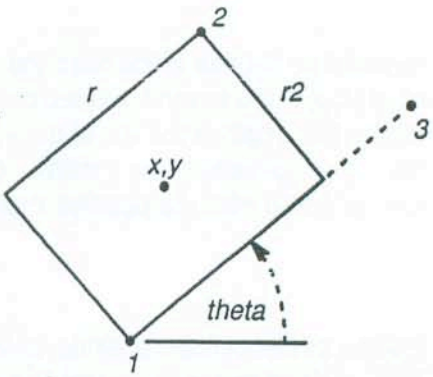
- oned
- angled
- uv
- section



Use the **oned** option to define a 1-D subregion, that is, a line, between two marked points. The command **xwires section** is treated in exactly the same way as **xwires region oned**.

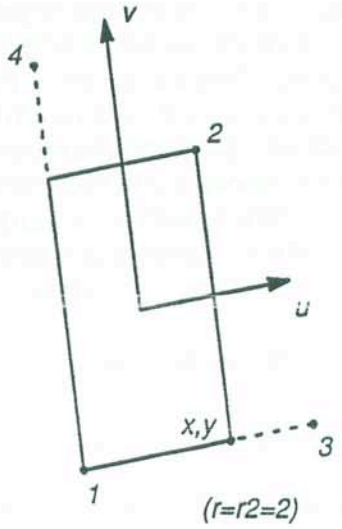


Use the **angled** option to define a rotated rectangular subregion. The first two points define the opposite corners of the rectangle and the third point defines the direction of the new *x* axis, in radians anti-clockwise from the positive *x* axis.

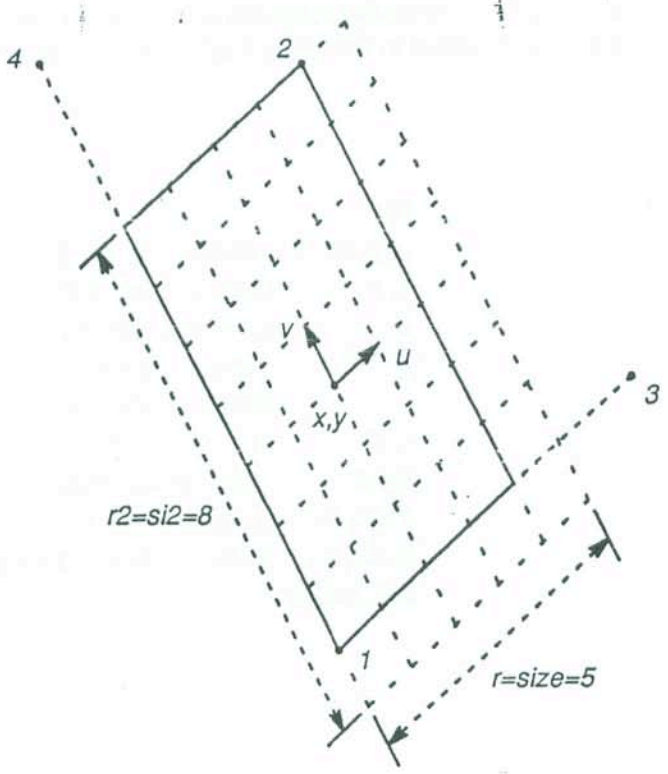


xwires

Use the **uv** option to define a skewed (parallelogram) subregion. The first two points define the opposite corners of the parallelogram and the third and fourth points define the directions of the *u* and *v* axes.



With any of the **xwires region** commands, you may use the **sampling** key to obtain a size for the region you have marked which depends on the sampling interval you have specified. You may also use the **size** key with the **xwires region uv** command, to obtain base vectors adjusted for the region size that you specify, rather than the default size of 2 by 2. The following diagram illustrates the results returned with the command **xwires region uv size 5, 8**.



xwires

Use the option **list** or **curve** to generate a *Plist* picture from positions that you mark with the cursor. **xwires list** defines a list of marked positions and stores them as an output *Plist* picture. You end the list by repeating the last point. You can also use **xwires curve** to define a curve that lists marked positions and stores them as an output *Plist* picture. You end the curve by repeating the last point (which creates an **open** curve) or the first point (which creates a **closed** curve). Note that if you draw a curve, **xwires** returns the perimeter or length in the variable *p*. If the curve is closed the area is also returned in the variable *a*. If you draw points in a clockwise direction, then *a* is set to minus the area. You can force **xwires** to generate a closed curve by using the **closed** option. In this case, repeating the last point will close the curve.

Use the **graph** option to create a 1-D picture from a marked graph. For example, the command:

```
xwires graph size 123
```

outputs a 1-D picture with 123 evenly spaced samples from the drawn function. You can specify the **symmetric** and **antisymmetric** options with the **graph** option. After the first point you mark, any points you mark to the left of the displayed graph (or its *x*-origin, if you are drawing a **symmetric** or **antisymmetric** graph) are ignored. Similarly any points that you mark to the left of a previously marked point are ignored. You terminate the graph either by repeating a point, or by marking a point beyond the right hand edge of the graph. If you do not draw a graph extending to both ends of the displayed graph, the end points of your graph are continued in both directions.

The size of the output picture is the same as that of the displayed graph, unless you specify the keys **size** or **sampling**. Its origin is recorded in the same position as that of the display (rounded if necessary).

Notes

forms used internally:	fp	
variables set:	*x, y, xn, yn*	(start/end points of line)
	x, y	(centre of circle/arc/region)
	a	(area enclosed by closed curve)
	p	(length of curve)
	r	(length of line, radius of circle/arc)
	r, r2	(size of region)
	theta	(angle of line/region)
	theta, th2	(angular range of arc)
	u, u2, v, v2	(base vectors for skewed region)
see also:	**cut, extract**	

xwires

Defaults and Ranges

keys/options	defaults	range
[ ]	if ***picture*** or ***partition***, current display held in the variable *display*, if ***frame*** current frame held in the variable *cframe*	valid picture/partition/frame number
[to]	picture 999	valid picture number
size	if ***region uv***, 2, *size*, if ***curve*** or ***graph***, actual length	less than or equal to the size of the picture/partition/frame (integers)
sampling	1 if ***region***, none if ***curve***	positive real number
position	position 0,0	within bounds of picture/partition /frame (integers)
mkmode	1 (upright cross)	integer in range 1 to 5
mksize	2	positive integer
picture/partition/ frame	picture	
open/closed	open	
verify	verification on	
re/im	operate with respect to real part of complex display picture, verify input data on both parts	
view	view disabled	

x11

This syntax is specific to...
workstations running X Windows

keys:	ilimit	*<number>*	image refresh threshold
	vlimit	*<number>*	vector/annotation refresh threshold

You use the **x11** command to control the rate at which the display is updated.

Examples

```
x11 ilimit 8
```

This command causes the display to be refreshed after every eight image writes (the default).

```
x11 vlimit 0
```

This command causes vectors and annotation to be refreshed only at the end of display operations or on *display flush* (the default).

```
x11 vlimit 1 ilimit 1
```

This command causes the display to be updated after every operation.

Description

In order to display overlay planes that do not over-write the image plane, Semper maintains separate copies of the image data and the overlay data and uses them to update the visible areas of the window.

To keep the display up-to-date after each operation involves an additional processing time overhead of about 80% (images) and up to 10000% (annotation). To reduce this overhead, Semper delays writing to the server until one of the following occurs:

- the number of operations exceeds the appropriate threshold
- Semper requires a *display flush* or a *display close*
- the server requests a refresh (for example, on exposing a window)
- the internal table of updates overflows

The **x11** command allows you to set the refresh thresholds or to turn off the threshold mechanism. This allows you to make a trade-off between interactive display rate and processor time. The default threshold values are usually acceptable for normal monitoring purposes.

x11

Use the **ilimit** key to specify the image refresh threshold. A positive value species a threshold. A zero or negative value turns off the threshold mechanism so that the image only appears when the current command is completed. The threshold value for image refresh is initially set to 8.

Use the **vlimit** key to specify the vector/annotation refresh threshold. A positive value specifies a threshold. A zero or negative value turns off the threshold mechanism. The annotation refresh threshold is turned off by default, so complex annotation appears to be painted from top to bottom. If you require the annotation to appear at a more steady rate use a threshold value of 8 (say) but be prepared to wait for the results to appear on the display.

Notes

see also: assign display, overlay

Defaults and Ranges

keys/options	defaults	range
ilimit	8	integer
vlimit	0	integer

ymod

keys:	**[from]**	*<number>*	display specified picture
	[to]	*<number>*	output display picture
	size	*<x>, <y>*	dimensions of subregion to be displayed
	position	*<x>, <y>*	position/offset of subregion
	layer	*<number>*	layer containing subregion
	times	*<number>*	integral magnification factor for display
	mark	*<number>*	mark subregion of source picture on the display
		<yes> or <no>	
options:	**left/right, top/bottom**		position of subregion
	preset		use current values of *min*, *max* to set height limits
	letter		letter top of partition with picture number

Use **ymod** to produce 'perspective' displays of pictures or picture subregions, as line drawings that plot intensity as surface height.

Examples

```
ymod 23 to display:2
```

This command displays picture 23 on the overlay of display picture *display:2*

```
ymod times 2
```

This command displays the current picture, interpolated by a factor of 2 so as to fill out a sparse basic structure.

```
xwires region; min=0 max=2; ymod preset @region layer 2
```

This command displays the specified subregion of layer 2, scaling 0 and 2 to the bottom and top of the surface.

Description

Note that you can display a (single layer) subregion using the standard subregion keys and options, for example, **size**, **position**, **layer** etc. For further detail of subregion keys and options, refer to *Appendix C, Semper Keys and Options.*

ymod acts in the same way as the **display** command, for display erasure and lettering. The real and imaginary parts of Complex pictures are drawn one beneath the other (real at the top). Note that **ymod** displays cannot be annotated or measured with a cursor.

ymod

If you specify a display using the **mark** key, **ymod** marks the source subregion on the display. Refer to *Appendix C, Semper Keys and Options* for details of the **mark** key.

Notes

restrictions:	not 1-D
display marking:	region, if subregion
multi-layer pictures:	single layer subregions only
forms used internally:	fp, complex
see also:	**display**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	current display, held in the variable *display*	valid display picture number
size	whole picture	less than or equal to the size of the picture (integers)
position	position 0,0	within bounds of the picture (integers)
layer	layer containing origin	integer in range 1 to number of layers
times	times 1	positive integer
mark	mark off	see *Appendix C*
preset	actual *min*, *max* pixels present	
letter	lettering on	

zget

This syntax is specific to... Silicon Graphics workstations

keys:	[to]	*<number>*	output picture
	name	*'<text>'*	source file name
options:	**again**		if **name** is omitted and was used in a command to open a file for reading, open the same file again

You use **zget** to read an image from a *Sarastro* binary image file.

Examples

```
zget 20 name 'zimage'
```

This command reads data from the file *zimage* into picture 20 on the current device.

Description

Files are searched for in the current directory and then throughout the search path. You can specify a full path name to avoid the path scan.

Notes

see also: **zput**

Defaults and Ranges

keys/options	defaults	range
[to]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

zput

This syntax is specific to... Silicon Graphics workstations

keys:	[from]	*<number>*	source picture
	name	*'<text>'*	output file name
options:	new		overwrite the output file if it already exists

You use **zput** to write a picture to a *Sarastro* binary image file.

Examples

```
zput 20 name 'zimage'
```

This command writes data to the file *zimage* from picture 20 on the current device.

```
zput 20 name 'zimage' new
```

As for the above example but also allows an existing file named *zimage* to be overwritten.

Description

Files are created in the current directory, unless you specify a pathname.

Notes

see also: **zget**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
name	*none*; prompts if interactive	valid filename

zone

keys:	**[from]**	*<number>*	source picture
	[to]	*<number>*	destination picture
options:	**binary/label**		treat source picture as binary or labelled image
	fg		label foreground regions of binary source image
	bg		label background regions of binary source image
	circle/diamond/square/octagon		use Euclidean, 4-connected, 8-connected or octagonal metric for measuring distances

You use the **zone** command to generate the zones of influence associated with the source image. Each pixel in the output image is assigned the label of the nearest labelled pixel in the source image. By default, the source picture is treated as a binary image, and a labelled image is obtained from it (see the **label** command for details about the labelling process). If the **label** option is set, the source image is treated as being already labelled. Distances between pixels are measured, by default, using the true Euclidean metric. Faster and possibly acceptable results can be obtained with simpler metrics by specifying one of the options **diamond, square** or **octagon** (see the **dt** command to find out more about distance transforms and metrics).

Examples

```
zone
```

This command replaces the binary image in the current picture with the zones of influence associated with the connected foreground regions of the binary image.

```
zone 1 2 label diamond
```

This command outputs to picture 2 the zones of influence associated with the labelled regions in picture 1 using the 4-connected, *city block* metric.

Description

If the source picture is treated as a binary image (the default), a labelled image is obtained from it in exactly the same way as the **label** command. By default, the foreground regions are labelled. You can use the options **fg** and **bg** to control whether the foreground and/or background regions are labelled. Note that foreground regions are treated as being 8-connected, while background regions are 4-connected.

zone

By specifying the **label** option, you can surpress the labelling pass. This allows you to supply source images which have already been labelled, including images generated by any of the classification commands **(box, mindistance** and **likelihood**). Note that differently labelled regions are allowed to touch each other.

Each pixel in the output picture is assigned the label of the nearest labelled pixel. If more than one region is equidistant from a given pixel, the pixel is not labelled (set to zero).

Distances between pixels are calculated using one of several possible metrics. By default, the exact Euclidean metric is used. A simpler metric requiring less computation, can be selected with one of the options **diamond**, **square** or **octagon**. For more details about the different metrics and distance transforms, consult the documentation for the **dt** command.

Notes

see also: **dt, label, box, mindistance, likelihoood**

Defaults and Ranges

keys/options	defaults	range
[from]	current picture, held in the variable *select*	valid picture number
[to]	source picture	valid picture number
binary/label	treat source picture as a binary image	
fg/bg	label foreground regions	
circle/diamond/ square/octagon	use the Euclidean metric	

Appendices

Appendix A

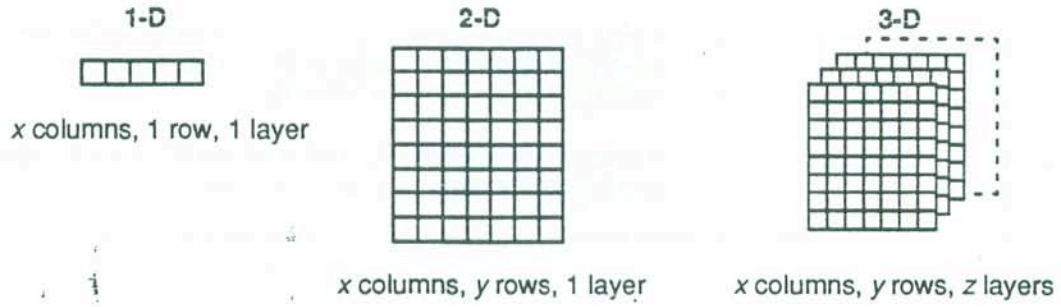
PICTURE TYPES

Overview

This appendix gives a summary of Semper picture types and explains some of the terms used in this manual. It details picture dimensions, class and form.

Picture dimensions

A Semper picture can have 1, 2 or 3 dimensions, as is illustrated in the following diagram:



Picture classes

Semper assigns a picture to a class according to its data type, as is shown in the following table:

No	Class	Description
1	*Image*	a sampled picture (most data fall into this class)
2	*Macro*	the text of a numbered macro
3	*Fourier*	the Fourier transform (*ft*) of an image
4	*Spectrum*	the intensity (modulus squared) of the *ft* of a picture
5	*Correlation*	the cross or auto-correlation function between pictures
6	*Undefined*	any data not otherwise classifiable
7	*Walsh*	the Walsh/Hadamard transform of a picture
8	*Plist*	a list of position coordinates with associated information
9	*Histogram*	the histogram (of the pixels) of a picture
10	*Lut*	a framestore look-up table

Picture Forms

Semper can store picture data in one of four different forms. Each form offers a different trade-off between storage and precision requirements. The following table describes the four types of picture storage. Note that the values of integer, fp and complex forms are, to some extent, machine dependent . The values given below may be considered as typical.

No	Form	Description
0	*Byte*	1 byte storage. 0 to 255, unsigned integers only. A compact but restrictive form of storage as negative values can easily arise in processing
1	*Integer*	2 byte storage. –32768 to 32767 or wider, integers only. *Integer* storage is less restrictive than *Byte* but still compact
2	*Fp*	4 byte storage usually. 1–35 to 1+35 or wider, either sign, integral or fractional (floating point). Typically one part in a million precision over the full range
3	*Complex*	8 byte storage usually. Pairs of *fp* values representing real and imaginary parts of a complex value

Further information

To see the dimension, class and form of a picture, for example picture number 2, type the following command:

```
examine 2
```

or type the command **examine all** for details of all pictures on a current device.

For further detail of Semper pictures, refer to the following manual:

Advanced Users' Guide

contained in the *Semper 6 Guide*.

Appendix B

SEMPER EXPRESSION SYNTAX

Overview

This manual frequently describes commands that require a *valid Semper expression.* This appendix defines the syntax of a valid expression.

You may use an expression wherever Semper requires a simple numerical or logical value. The following are valid components of an expression:

- numbers
- variables
- function calls

All of the above supply a numerical value. Note that a *logical* value is represented by the number 1 or 0, where 1=*true* and 0=*false*. If Semper finds a numerical value where it expects a logical value it interprets a zero value as *false* and a non-zero value as *true*. (For further detail of variables and functions, refer to the *Advanced Users' Guide* contained in the *Semper 6 Guide*).

Expression Operators

There are three types of expression: **arithmetical**, **relational** and **logical,** in which the following operators can be used:

- **arithmetical** ^ * / + – :
- **relational** < > = ~= <= >=
- **logical** ~ & |

Note that ^ specifies exponentiation (raising a number to a power).

The colon operator **:** is a special operator that you use to combine device and picture numbers into a single numerical value. For example, 1:45 represents the numerical value 1045 and refers to picture 45 on device 1. If you do not specify a number before the colon operator, the current device number is assumed, (the value is held in the variable *cd).*

The logical operators ~ & | specify the logical operations NOT, AND and OR respectively.

Operator priority

The order in which Semper evaluates nested expressions depends on the relative priorities of the operators used in each expression (however you can control the order of evaluation using brackets). The order of priorities is listed here, with equal priority operators on the same line:

High Priority	:
	^
	* /
	+ –
	< > = ~= <= >=
	~
	&
Low Priority	|

Functions

Function calls return a numerical value. Each call consists of a function name followed by one or more argument values enclosed in brackets (use commas to separate multiple arguments), for example:

exp(3) root(*x*) sin(*theta*) mod(3.6, –12.3)

The following functions are recognised by Semper:

sin(*x*)	sine (*x* in radians)
asin(*x*)	arcsin (result in radians), for mod(*x*) <= 1
cos(*x*)	cosine (*x* in radians)
acos(*x*)	arccosine (result in radians), for mod(*x*) <= 1
tan(*x*)	tangent (*x* in radians)
phase(*x*)	arctan (result in radians)
phase(*x,y*)	arg(*x+iy*) (result in radians), for mod(*x*) <= 1
exp(*x*)	exponent (e to the power of *x*)
ln(*x*)	ln (natural or Naperian log), for *x*>0
min(*x,y*)	lower of values *x,y*
max(*x,y*)	higher of values *x,y*
mod(*x*)	modulus (absolute value)
mod(*x,y*)	modulus(*x+iy*), mod(*x,y*) = root(*x*x + y*y*)
msq(..)	= mod(..) squared
root(*x*)	square root, with *x*>= 0
rem(*x,y*)	remainder when x is divided by y, rem(x,y) = x – fix(x/y)*y
round(*x*)	nearest integer to *x*
fix(*x*)	next integer towards zero from *x*
p(*x,y,z*)	value of pixel *x,y,z* of current picture
re(*x,y,z*)	real part of pixel *x,y,z*, same as *p(x,y,z)*
im(*x,y,z*)	imaginary part of pixel

and(*n,m*)	bitwise **and** of round(*x*) and round(*y*)
or(*n,m*)	bitwise **or** of round(*x*) and round(*y*)
not(*n*)	bitwise **not** of round(*x*)
rad(*x*)	angle in radians equivalent to *x* degrees
deg(*x*)	angle in degrees equivalent to *x* radians
ifelse(*x,y,z*)	returns *y* if *x* is non-zero and *z* if *x* is zero
set(*variable name*)	returns 1 if named variable is set; otherwise 0

Note that Semper does not confuse function calls with variables of the same name. For example, min(*x,y*) and *min* may be used quite independently.

Examples

Expressions provide a powerful and general way of combining and manipulating numerical data. A number of examples is given below to illustrate the range of possibilities:

56	–1.5	*x*	192*64
n+3	.5–*x*	*min**2.4	mod(*x,y*)5
(*a*+*b*)/2	*scale**(*r*+.2)	1/(*x***x*+a*a)	exp(–msq(*x,y*)/50)
(*x*–fix(*x*/*y*)**y*)	deg(*pi*)	sin(rad(60))	–set(*erase*)
x=y<=5	*a*<1|*a*>10	and(or(not(*a*),not(*b*),or(*a,b*))	

Appendix C

SEMPER KEYS AND OPTIONS

Overview

This appendix provides a summary of the following keys and options that can be used with Semper commands:

- *general* keys and options
- keys for display marking
- commonly used options
- subregion keys and options

General keys and options

The following keys and options are called *general* and are usually not listed under the syntax of a command. They can be used with any command (although they may be ignored if the context is inappropriate).

byte, integer, fp, complex — These options determine the form of a command output. By default, Semper commands produce output in the same format as the source picture, unless you override this by using one of these options

erase — The **erase** option erases display partitions before pictures are displayed in them. The default is **noerase**.

view — You can use the **view** option with any command that displays pictures to make a display picture visible. For example, you can use **view** with the **mark** command if the display that you want to mark is not initially visible, or with commands such as **erase** and **ramps**.

re, im — These options limit the annotation on a complex display to the real or imaginary part. By default, Semper annotates the real part and repeats the annotation on the imaginary part.

Mark keys

The following keys are used to mark a display. They are *general* keys but are listed under the syntax of a command in this manual, if a command uses display marking:

mark

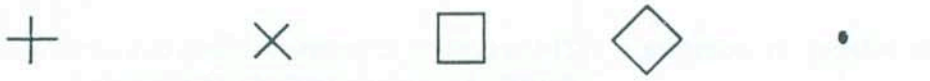
This key make take the values *yes* or *no* (1 or 0):
If *yes*, the current display picture is marked.
If *no*, display marking does not take place.
If you do not specify the key, **no** is assumed. You can also specify any valid display picture number with the **mark** key, in which case the specified display picture is marked. You can turn off display marking permanently by typing **mark=no**, and permanently turn on marking by typing **mark=yes**.

mkmode

The **mkmode** key determines the style of annotation when marking points. You can specify the following values:

1 upright cross (default)
2 diagonal cross
3 upright box
4 diagonal box
5 single pixel

The default is **mkmode 1.**



mkmode 1 **mkmode 2** **mkmode 3** **mkmode 4** **mkmode 5**

mksize

This key determines the size (radius) of the annotation. The total width of the mark is *2*mksize+1*. Note that **mksize** is ignored if **mkmode** is set to 5.

Other widely used options

The following options are commonly used and appear in the syntax description of many Semper commands:

preset	The **preset** option causes a command to use the current values of the variables *min* and *max*, instead of the actual pixel range, for scaling purposes. The default is **nopreset**.
verify	This option gives information about the action and results of a command at the console. The default depends on the particular command.
letter	This options adds lettering to a display that gives size, title and grey scale information for a picture. The default is to add lettering.
border	This option marks a border that outlines a display picture. The default is to mark a border.

Subregion keys and options

You can specify a subregion using the following keys and options:

- **size**
- **position**
- **left/right**, **top/bottom**, **near/far**
- **layer**

Use the **size** key to specify the dimensions of a subregion. **size** has three components which correspond to the *x*, *y* and *z* dimensions of the subregion.

If you do not specify a value for **size**, it defaults to the dimensions of the source picture. If you only specify one dimension, the *y* dimension defaults to the *x* dimension and the *z* dimension defaults to the source picture *z* dimension. For example:

```
size 300, 200
```

describes a subregion 300 pixels across by 200 down, and:

```
size 250
```

defines a subregion that is 250 pixels square.

Use the **position** key and the **left/right**, **top/bottom**, **near/far** options to specify the position of a subregion. By default, Semper positions a subregion so that its centre coincides with the source picture origin. Use the **left** or **right** option so that it is placed with its left or right side coinciding with

the left or right side of the source picture. The options **top/bottom** and **near/far** have a similar effect on the other two directions. For example:

```
size 100 bottom left
```

defines a 100 pixel square subregion at the bottom left of the source picture. The **position** key allows you to specify an offset for the subregion. The centre of a subregion is positioned exactly at the *x, y* position you specify if you do not use any of the subregion options, for example:

```
size 100 position 200, 350
```

defines a 100 square subregion centred on the position 200, 350.

Multi-layer subregions

You can specify a layer for a subregion using the **layer** key. For example:

```
size 300, 200 layer 5
```

defines the front layer of a picture with 5 layers 300 by 200 in size. You can also specify a range of layers, for example:

```
size 300, 200 layers 2,4
```

specifies a three layer subregion, starting at layer 2.

Appendix D

PARTICLE PARAMETERS

Overview

The **analyse** command records 25 different parameters for each particle it finds in a particle parameter list (*ppl*), held by Semper as a class *Plist* picture. Each parameter has a name and a variable associated with it. The *name* is used to specify or select individual or sets of parameters in later printout, sorting or display commands. The *variable* is used to store individual parameter values for further manipulation.

The table given below details the parameters recorded by **analyse**. An explanation of the individual parameters is given after the table.

Name	Variable	Parameter description
xref, yref	*xr, yr*	reference point
id	*pid*	particle identifier
parent	*pa*	parent identifier
holes	*h*	number of holes
background	*bg*	background flag
contact	*ec*	edge contact flag
xmin, xmax	*x1, x2*	limits – min, max *x*
ymin, ymax	*y1, y2*	limits – min, max *y*
hferet, vferet	*hf, vf*	horizontal and vertical feret diameters
aferet, bferet	*af, bf*	feret diameters – 45 degrees, 135 degrees
hproj, vproj	*hp, vp*	horizontal and vertical projections
perimeter	*p*	perimeter
area	*a*	area
xcen, ycen	*xc, yc*	centre of area
mmin, mmax	*m1, m2*	principal second moments of area – min, max
angle	*theta*	orientation
circularity	*c*	circularity

Reference point

The *reference point* gives the position of the right-most pixel of a particle at its bottom-most row and is therefore guaranteed to lie on the edge of a particle.

Particle identifier

The *particle identifier* provides a particle with a unique identifying number. Identifiers are assigned from 1 onwards in the order in which the particles are found during a picture scan.

Parent identifier

A *parent identifier* gives an identifier for the region surrounding a particle. Note that this parameter is to be used by later releases of Semper. Currently all parent identifiers are set to zero.

Number of holes

The *number of holes* means the number of background areas that are enclosed within a particle.

Background flag

A *background flag* indicates that the 'particle' is a background area (hole) enclosed inside another particle. This is to be used by later releases of Semper. Currently all background flags are set to zero.

Edge contact flag

The *edge contact flag* of a particle is set to 1 if the particle makes contact with an edge of an analysed picture, and to zero otherwise.

Limits

The *limits* of a particle are the minimum and maximum x and y values of all pixels belonging to a particle.

Feret diameters

The *feret diameters* of a particle are feret or *caliper* diameters in four different directions – horizontally, vertically, at 45 degrees anti-clockwise from the positive x axis and at 45 degrees clockwise from it. The diameters provide rough size indicators and can be used to detect elongation (if there is a large difference between minimum and maximum diameters) and orientation (using the largest diameter).

Projections

The *horizontal* and *vertical projections* of a particle record the total number of right-facing and upward facing edge pixels for an object, that is, the total projected edge length facing the relevant direction.

Perimeter

The *perimeter* is the distance around the outside of a particle.

Area

The *area* of a particle is simply the number of pixels in a particle.

Centre of area

The *centre of area* parameters provide the mean *x* and *y* values for all pixels belonging to a particle. In many cases this may be more convenient to use than the particle reference point, as the centre of area lies roughly at the centre of a particle. Note that it may not actually lie inside a particle if the particle is hollow or convoluted.

Principal second moments of area

A *second moment* is the mean square distance of all pixels about a line through the centre of area of a particle. The *principal second moments* recorded by **analyse** are the second moments with respect to a pair of mutually perpendicular axes in directions that achieve maximum and minimum moments. Note that their square roots provide a reasonable measure of a particle's mean external dimensions (actually a factor of 3 to 4 times the square root, depending on the shape of the particle), and their ratio a measure of elongation.

Orientation

The *orientation* of a particle is the angle in radians clockwise from the positive *x* axis to the axis giving the lowest second moment of area, that is, the *long* axis of a particle.

Circularity

The circularity of a particle measures, on a scale from 0 to 1, how circular a particle is. It uses the following equation:

*4 * π * area / (perimeter^2)*

Appendix E

ERROR MESSAGES

Overview

This appendix lists in numeric order the error messages given by Semper 6. It explains each error message and suggests possible causes.

Error messages

1 ***I/O error on device*** *number* ***at file*** *number* ***block*** *number*

Possible causes of error 1 include:

- inability to read/write a damaged or dirty tape
- hardware failure in tape drive/controller
- slight alignment differences between tape drives used to read and write tape

There isn't much you can do about this, as the offending block will already have been tried several times. You can set the variable *mtpass=yes*, which causes Semper to ignore the error while reading, and so allows you to continue processing with some data incorrectly transferred.

2 ***Unexpected EOF on device*** *device number*

Error 2 means that an end-of-tape mark has been reached unexpectedly on tape. Likely causes include:

- a malformed tape
- a hardware failure
- mistakes in a new Fortran routine of your own writing

3 ***Bad value for*** *keyword/variable*

Error 3 means that the named variable or keyword has an inappropriate value, for example, a negative radius for a circle. It may be caused indirectly by omitting a vital key for which there is no useful default, as this is usually equivalent to giving it a zero value . Note that only the first three letters of the keyword/variable appears in the message.

4 ***Abandoned***

Error 4 occurs when Semper acknowledges an abandon request from the terminal.

5 ***Bad size for picture*** *picture number*

Possible causes of error 5 include:

- zero or negative picture dimensions requested
- picture row length too great for internal row buffers (**show system** lists maximum lengths for each form)
- non-factorisable dimensions in operation such as **rotate** (that is, dimensions not factorisable into factors 2,3,4 and 5 with at least one factor 4. The command **show sizes** lists the acceptable sizes)
- dimensions not a power of two in an operation involving *Fourier* or *Walsh* transformation
- inappropriate picture dimensions, for example, a 2-D picture where 1-D is required.

6 ***Bad class for picture*** *picture number*

Error 6 means that you have tried to apply an operation to data to which it is not appropriate. Typical examples include:

- *Fourier* transforming a picture which is already a transform
- calling an *Image* picture as a macro (e.g. **@56**)
- displaying a macro, or doing almost anything except **list**ing or **edit**ing it
- histogram equalising with an *Image* rather than a *Histogram*
- loading a non-*Lut* picture as a look-up-table
- using another class where a *Plist* is needed, for example, as a mask boundary

Probably you have typed a wrong picture number; otherwise, you may have misunderstood how the command you are using works, and should ask **help** about it again (perhaps using **help/full...**).

7 ***Undefined label:*** *label*

Error 7 means that you (or a macro or program you are using) have tried to jump to the label indicated, but that the label cannot be found. The error context printout will tell you whether the problem is inside a program. Likely causes include:

- a mistyped name in the label or **jump** command
- trying to jump to a label outside the program
- trying to jump into a **for** loop
- trying to jump to a previous or subsequent line interactively

8 ***I/O error on device*** *number* ***at block*** *number ff*

Likely causes for error 8 include:

- inability to read/write information on a damaged/dirty disc
- hardware failure in a disc drive/controller
- writing to a file to which you have read access only (indicates a problem in the primitive routine *MCDC61*)

Little useful response is possible. No data loss will occur if the error arises during reading, but its extent is not usually very serious in any case, normally resulting in directories or picture data being backed-up to their state shortly before the error.

9 *Outside picture*

Likely causes for error 9 include:

- using or setting a pixel with coordinates that are out of range
- using a bad layer number
- using a subregion that does not overlap the picture at all
- setting a picture coordinate origin outside the picture
- mistakes in a new Fortran routine of your own writing

10 *Message depends on error condition*

Error 10 is not in fact reported by Semper. Commands that produce very specialised error messages of their own follow these by returning error 10 to the interpreter, which responds by returning to the terminal for fresh instructions as usual, but without printing any further message.

11 ***Disc full – can't open*** *picture number,* ***on device****, device number*

Error 11 means that there is insufficient free space left on a disc device for a new picture that you are creating. The things you might try in response are:

- deleting unwanted pictures from the device in question
- using a more compact form for your output picture (e.g. **lmean 50 byte**)
- storing other pictures in more compact forms (e.g. **copy 41 byte**)
- creating a new disc device, temporarily if necessary, with sufficient space (for example **assign new name '***newname***' size ..**)

It is *not* worth trying to **compress** the device; the error message means that there is insufficient space in total, not simply that it is fragmented.

12 ***No structure in picture*** *picture number*

Error 12 means that some kind of picture scaling operation is impossible, because the picture minimum and maximum values are equal. Likely causes include:

- displaying a picture (or subregion) in which all pixels are constant
- setting *min*, *max* equal when specifying **preset** (e.g. **scale preset range 1, 10** when *min* and *max* are both zero)
- using correlation functions to register two images one or both of which has all pixels equal

13 ***Can't set*** *variable*

Error 13 has two likely causes:

- you have tried to reset a protected variable such as *pi* (**help variables** tells you about these)
- you have tried to set too many variables simultaneously (**show system** tells you the maximum number supported by your installation). Use **unset** to clear those you don't really need

14 ***Unknown command:*** *command*

Error 14 means that your command (after any prefixed labels, conditionals and assignments have been stripped off) begins with a name that is not a currently defined command. Probably you mistyped it; if in doubt, try **show commands**, which lists all commands recognised by your installation.

15 ***Display device is write-only***

Error 15 means that you have tried to recover picture data from the display device in an installation where your display hardware is not capable of reading back such data. Note that trying to process a display picture *in situ* (e.g. **sharpen display**) in fact involves this kind of read-back, as the picture data must be recovered by your computer before the sharpened version can be calculated.

16 ***Unknown function:*** *function name*

Error 16 means that you have used an unrecognised function name in an expression, for example, **type six(theta)** instead of **type sin(theta)**. It can also be caused by mistyping: for example, **extract angle t(2** instead of **extract angle t*2** tries to treat *t* as a function name because of the bracket following it. **help functions** lists all functions recognised by Semper.

17 *Bad syntax*

Error 17 means that Semper cannot understand a command; possible reasons include:

- mistyping commands so that they are unrecognisable, e.g. **p x 2** or **[x=2** for **p x=2**
- omitting essential items from commands, e.g. **jump** without a label name
- including spurious additional items, e.g. **extract angle=.3** (which should have no = sign)
- typing **name#** without a following subscript variable

18 ***Expression stack overflow***

Error 18 means that you have used an expression that is too complicated for Semper. It is likely to occur in **calculate** commands (for which each intermediate result involves a whole picture row). To cure the problem, simply split your expression into two simpler ones.

19 *Can't unset variable*

Error 19 is reported if you try to **unset** one of the protected or fixed variables such as *pi* or *cd* (type **help variables** for a list).

20 *Bad syntax in expression*

Error 20 means that an expression you have used is wrongly constructed, e.g. **cos(), s+*6** or **root(2,3)**. **help expressions** explains the syntax of expressions if you are in doubt.

21 *Command line too long*

A likely cause of error 21 is that a macro expression has caused a command line to exceed the permitted length. Type **show system** to see the maximum length of a command line for your installation. Note that numbered macros are an old feature that will be removed in a later release. Converting the macro into a library program using the command **add** should remove the problem.

22 *Bad numerical operation*

Likely causes for error 22 include:

- division by zero, e.g. **type 1/x** when *x* is zero, or **calculate :sel/:2** when some pixels of picture 2 are zero
- taking the **ln** or square **root** of a negative number, e.g.**mark radius root**(t) when *t* is –2

23 *Bad position on device device number*

Error 23 means that you have referred to an inaccessible position on the tape device indicated, such as a file beyond the current end of tape, a negative file number, or a block number larger than the number of blocks in a file.

24 *RUN file already active*

Error 24 means that you have nested a **run** call from within a run file. If you want to call other programs from a run file load them into a program library file and call them using the **library** command.

25 *Unset variable: variable*

Likely causes of error 25 include:

- using an unset variable as a key value or in an expression, e.g. **If x>t** when *t* is unset
- providing no value for a key which you are required to set because no useful default can be provided
- neglecting to set variables needed by a command, e.g. **spc levels h3 list** when *h3* is unset
- syntax errors, e.g. **ask** '*value*', **x** for **ask** '*value*' **x** (in the former, *x* is treated as part of the prompt string

26 *Involves backward tape movement*

Error 26 means that you have tried to access a tape file before the current tape position, in an installation which cannot backspace or rewind tapes.

27 ***Output to** device number **not allowed***

Likely causes for error 27 include:

- trying to output data to tape with any command other than **copy**, e.g. **lmean 50 to 3:1** when device 3 is a tape
- trying to alter existing data on a tape, e.g. **lmean 3:2** or **select 3:2; p0=1** or **title 3:2**

28 ***Bad device or picture number:** number*

Likely causes for error 28 include:

- using a zero or negative picture number, e.g. **3:0**, when not **copy**ing to tape
- using a picture number greater than 999 for a disc or tape
- using a display picture number greater than the maximum for your installation
- using a zero or negative device number
- using a device number greater than the maximum for your installation

29 ***Bad medium for device** device number*

Error 29 means that you have tried to do something to a device or picture that is inappropriate to its storage medium (disc, tape, display, help library). Likely causes include:

- **rewind**ing a disc device
- **renumber**ing a tape or display picture
- **compress**ing a tape or display device, or asking for **directory** information about one
- doing anything to a help library other than **assign**ing and **deassign**ing it
- **mark**ing subregions on a disc or tape picture

30 ***Picture** picture number **does not exist***

Most commonly, error 30 means that you have referred to a disc or tape picture that does not exist. Try **examine all** or **examine device.**. to check what does exist, or **show device** to see how many pictures are on a tape device. Note that a display partition may exist without having a picture in it, so **show partition dis:3** can report details of the partition even when **examine dis:3** reports error 30.

31 ***Bad tab position***

Error 31 means that a tab setting used in a **type** or **ask** command, or in the value for a textual key, requests a character position that is negative or too far right, e.g. **type 'values ',#h,x,#v,y** when *h* is zero or *v* is 1e5.

Error number 32 is currently not in use

33 ***Display** display number **is undersampled***

Error 33 means that you have attempted to recover a picture from, or process *in situ*, a display picture which has been undersampled to make it fit its partition. The picture therefore no longer has all the pixels it ought to have.

34 ***Device** device number **is not assigned***

Error 34 means that you have tried to do something to a device number which is not currently assigned, and does not therefore refer to any particular disc, tape or display. **show devices** confirms the current list of assigned devices. Probably you have mistyped a picture number.

35 ***Device** device number **does not contain programs***

Error 35 means that you have used a command that expects to find programs on a specified device, for example, **list all device 3**, when device 3 is a picture disc or help library.

36 ***Bad FOR loop***

Likely causes for error 36 include:

- syntax errors in a **for** command, e.g. **for n=1**
- too many **for** loops active at once, e.g. **for..; for..; for..; for..; for..**
- bad loop increment values, e.g. **for n 1,2,0**

37 ***Disc directory full - can't open** picture number*

Error 37 means that insufficient space remains in the directory of a disc device for the recording of a new picture you are creating. It is likely only when the device is a **save** file, or when the directory has been deliberately created with a small number of slots to save space. But it often occurs if you have very large numbers of pictures within a device. You can ask **directory** about current directory slot usage. **compress**ing the device may help, as each empty segment of disc requires a directory slot to record it.

38 ***Insufficient display frames available***

Error 38 means that you are trying to display a multi-layer picture, probably full colour, for which each layer is output to a separate (successively numbered) display frame, and have run out of hardware display frames. If your installation provides four frames, for example, you can display full colour pictures in partition frames 1 to 3 and 2 to 4, but not in partitions which start with frames 3 or 4.

39 ***Incompatible re-use of picture** picture number*

Error 39 means that you are trying to process a display picture *in situ* in a way that requires two display pictures of different sizes to co-exist during the processing, for example, **extract display size 400** when the display is 300 square. (Differences in form do not cause problems, except that *Complex* pictures, for which real and imaginary parts are displayed side by side, are incompatible with non-*Complex* pictures).

40 *Display error*

Likely causes for error 40 include:

- hardware failure in the display device or interface
- using false or full colour look-up-tables in installations where the hardware does not support them
- using zoom factors or pan settings not supported by your display hardware

41 ***Write-protected device or picture*** *number*

Likely causes for error 41 include:

- altering a protected picture (processing it *in situ*, e.g. **lmean 51**, using the **p** command to change pixels, changing the title etc.)
- using an output picture that already exists and is protected, e.g. **lmean 50 to 51** when 51 is protected
- deleting a protected picture
- altering anything at all on a protected device

If in doubt about whether a picture is protected, ask **examine.** For devices, ask **show device** instead. Picture level protection can be removed or imposed via the **wp** command; device level protection is established when the device is **assign**ed and cannot be altered – deassign and then reassign the device.

42 ***Disc fragmented – can't open*** *picture number,* ***on device,*** *device number*

Picture files require contiguous free space on a disc. Use the **compress** command to compact the free space. Type **directory** for a list of the available free space on your current device.

43 ***Bad form for picture*** *picture number*

Error 43 means that you are trying to use or store data whose form (byte, integer, fp or complex) is unsuitable in some way, for example, applying an operation only meaningful for complex data to a non-*Complex* picture.

44 ***Assign/deassign failure on device*** *device number*

Error 44 is reported for any kind of problem arising during **assign**ing or **deassign**ing devices; common causes include:

- mistyping the name of a disc file you are **assign**ing
- **assign**ing without option **wp** a disc file you are not authorised to alter
- deleting (via **deassign delete**) a disc file you are not authorised to delete
- **assign**ing a **new** disc file for which space cannot be found
- **assign**ing a tape or display device which is being used by someone else

45 *Device table full*

Error 45 means that you have tried to assign too many devices at once. **show system** tells you the maximum number permitted in your installation, and **show devices** lists the devices you have assigned already. You need to **deassign** some other device before trying again. Note that the **save** command assigns a further device temporarily (for the save file), and so can also give rise to this error.

46 ***Picture** picture number **already exists***

Error 46 means that you have tried to **renumber** a picture to a new number that is not in fact currently unused.

47 ***Row length overflow in picture** picture number*

Error 47 usually means that Semper's internal row buffers are not large enough to process rows of the picture indicated in the data form that your command uses internally. This form is not under your control, and even when your source and output pictures are in byte form, the routine may in fact process pixels in *fp* form, and rows that fit the buffers in *byte* form may not do so in *fp* form. The only real solutions are breaking pictures up into smaller sub-pictures, or recompiling the entire system with longer row buffers.

48 ***Bad type for display** display number*

Error 48 means that the display indicated is of a type (2-D, 1-D, histogram, etc.) inappropriate for your command. Likely causes include:

- using the cursor in modes only possible for 2-D pictures (e.g. defining a closed curve) on a 1-D graph or histogram
- using a 1-D graph or histogram as source to a command, e.g. **copy dis:2 to 23** (the data cannot be recovered from these display types)

49 ***Unknown macro:** macro name*

Likely causes for error 49 include:

- mistyping the name of a macro, e.g. **@refion** for **@region**
- using the @ character accidentally

50 ***Bad block number: device** number, **block** number*

Error 50 does not usually happen in Semper, and is likely to indicate a mistake in Fortran code of your own writing. Specifically, it indicates that the low level routine DISC has been asked to read or write an out-of-range block number on the device indicated.

51 ***Unable to open any disc workspace***

Error 51 means that a command needing temporary work space on disc has been unable to find any or enough. Likely causes include:

- no disc devices assigned at all (**assign**)
- no disc device with sufficient contiguous space available (**compress**)
- all disc devices are write-protected (**wp**)

Only a few commands (amongst them **lmean, lvariance, lsd, sharpen** and **hp**) are in fact dependent on such work space.

52 ***Picture** picture number **has a malformed label***

Error 52 does not usually happen in Semper, but may occur following a disc hardware fault. It indicates that the label of the indicated picture, which contains all the crucial information about the picture (dimensions, form, etc.) is corrupt. The only useful response is to delete the picture using the command **delete .. malformed**.

53 ***Display partition** partition number **does not exist***

Error 53 means that you are using the indicated display picture or partition number without defining where the partition is stored and how large it is. If you have not simply mistyped the number, use the **partition** command before continuing.

54 ***Magnified region too large***

Error 54 can occur when you are magnifying a subregion with **display.** It means that you have magnified it so many times that it does not fit the display partition. Try again with a reduced value for **times**.

55 ***Too many histogram channels***

Error 55 means that a histogram you are creating has too many channels for the counts to fit Semper's internal buffers. Try again with a smaller value for the **channels** key.

56 ***Too many pictures open simultaneously***

Error 56 means that a command uses too many pictures at once. It may occur in a **calculate** command, e.g. **calculate :1+:2=:3+:4+:5+:6+:7+:8..**, in which case you must split the command into two simpler commands. Otherwise it may arise in new commands of your own writing. **show system** tells you the maximum number that may be open at once in your installation.

57 ***Illegal item:** item*

Error 57 means that your command is wrongly constructed in some way; likely causes include:

- you have used an illegal (possibly mistyped) name for a key or option, e.g.**extract anfle .3**
- you have omitted a required key name

58 *No pictures found*

Error 58 means that you have applied a command to a group of pictures (e.g. **examine 20,40** or **copy 1,10 ..** or **save 3:20,3:99**) when no pictures exist in the indicated range.

59 *Output must not overwrite source*

Error 59 means that the output from your command would be stored in the same place as the source, and that the processing operation being carried out would fail in such circumstances. Semper only reallocates the same storage if there is no change of picture number, dimensions and form, so you cure the problem by directing output somewhere else (for example, **extract 50 to 51** rather than simply **extract 50**).

60 *Conflicting keys and options: key/option and key/option*

Error 60 means that the two keys or options indicated are mutually incompatible, invoking strictly alternative modes – e.g. **ctf add** adds to the source, and **ctf multiply** multiplies by it, but **ctf add multiply** is faulted.

61 *Bad layer for picture picture number*

Error 61 does not normally occur in Semper, but may occur in new routines of your own writing if you accidentally refer to a picture layer that does not exist.

62 *Command does not process multi-layer pictures*

Error 62 means that the named command currently makes no provision for multi-layer pictures in any simple way (such as repeating the operation for each layer, or effecting a 3-D generalisation of the 2-D operation). This error may be expected to occur more rarely in later releases of Semper.

63 *Bad origin for picture picture number*

Likely causes for error 63 include:

- setting a picture coordinate origin outside the picture bounds (in new routines of your own writing)
- using a *Fourier* picture whose origin is neither at the centre nor at the centre of the left hand column

64 *Bad key/option: key/option*

Error 64 means that you have used an option inappropriate to the particular use of the command, e.g. **lut 2 red ..** when lut 2 is monochrome.

65 *All pixels zero*

A possible cause of error 65 is using a **fir** filter kernel (**fir with..**) with no non-zero values.

66 ***Bad kernel size***

Likely causes for error 66 include:

- using a **fir** filter kernel with too many points in each row (the limit is installation dependent, but is at least 21)
- using a **fir** filter kernel with too many points in total (the limit is the maximum number of fp values that fit a row buffer. See **show system**)
- offering a 2-D **fir** filter kernel for **separable** application

67 ***Picture** picture number **does not define a closed curve***

Error 67 means that the *Plist* picture indicated should define a closed curve and does not. Perhaps you have mistyped the picture number; for example, **mask with ..** requires a closed curve to define the boundary of the region inside/outside which the picture is to be masked. See **xwires.curve** for how to create such *Plists* if necessary.

68 ***Bad look-up table number:** number*

Error 68 indicates that you have used an out-of-range look-up-table number; negative, zero or greater than the maximum permitted for your installation (see **show system** for this limit).

69 ***Look-up table** number **does not exist***

Likely causes for error 69 include:

- **view**ing with a lut you have not created (use **lut** to create luts)
- using **lut** to recover or alter a lut you have not created

show luts lists the currently defined luts.

70 ***Bad display partition number:** partition number*

Error 70 means that you have used a display partition number that is illegal in some way. Likely causes include:

- using a negative or zero partition number
- omitting the display device number (e.g. typing an ambiguous 4 for **dis:4**)

show partitions lists the partitions currently defined.

71 ***Device** device number **is not a display device***

Error 71 means that you have tried to do something intended only for display devices (for example: **erase**) to the disc or tape indicated.

72 ***Bad type for Plist** picture number*

Likely causes for error 72 include:

- using the list *Plist* indicated when a curve is needed, e.g. **mark with ..**
- using the curve *Plist* indicated when a list is needed, e.g. **extract with ..**

73 *Bad device: partition number: number*

Likely causes for error 73 include:

- using a non-display device number when a display is needed, e.g. **mark 2:3** when 2 is a disc device
- using an out-of-range partition number (negative, zero or beyond the maximum your installation permits. See **show system**).

Error number 74 is currently not in use

75 *Bad display frame number: number*

Error 75 is caused by using an out-of-range frame number (negative, zero or beyond the maximum your installation permits). See **show devices**.

76 *Bad device number: number*

Error 76 means that the indicated device number is illegal (negative, zero or beyond the maximum permitted for your installation). See **show system**.

77 *Error message*

Commands that produce very specialised error messages of their own return error 77 and an error message which differs according to the context. You can see this message by using the **report error** command.

78 *3 points defining arc or circle are collinear*

Error 78 can arise in **xwires circle** or **xwires arc**, which seek to define a circle or a circular arc, if you mark three points in a straight line, as these would require an infinite radius.

79 *U and V not independent*

Error 79 means that the vectors (*u,u2*) and (*v,v2*), used by your command, are parallel and must not be so, for example, because they are supposed to be base vectors of a 2-D lattice.

80 *Maximum for number of input points has been exceeded*

Error 80 arises in **xwires** if so many points are marked (**list** mode) or generated (**sampling** key specified in **curve** or **graph** mode) that internal workspace tables overflow. The only useful response is to repeat the operation with a smaller number of points.

81 ***Picture is not a 1-D graph***

Error 81 means that you have used **xwires graph** on a display which is not a 1-D graph; probably you have simply mistyped the display number. (**xwires graph** only operates on an existing graph; if you have none initially, use something like **create 1 size 500,1 value 0; min=0 max=30; display preset** to create one).

82 ***First point beyond right edge of graph***

Error 82 means that the first point you marked in a **xwires graph** command was unsatisfactory. Draw graphs from the left hand side to the right.

83 ***Bad zoom factor:*** *zoom factor*

Error 83 means that you have used an illegal zoom factor for display viewing – particularly a negative or zero value.

84 ***Command with too many keys:*** *command*

85 ***Command with too many options:*** *command*

86 ***Command with too many open specifications:*** *command*

Errors 84, 85 and 86 should not normally occur; they indicate a fault in the System Generation program *SEMGEN*, which should ensure than no verb descriptor can have more keys, options or open requests than can be accommodated by the interpreter's internal tables. Contact *Synoptics* for advice.

87 ***Label for picture*** *picture number* ***not accessible***

Unless it is an indirect consequence of disc hardware failure, error 87 only occurs on tape devices in installations unable to backspace their tapes, and therefore unable to read a tape picture label more than once during a session. In such circumstances, even commands like **copy tape:n to 1** lead to the error when Semper attempts to transfer the title from source to output. (**show devices** reports a *nobackspace* flag if your installation cannot backspace tapes).

88 ***Sub-region specified with more than one layer***

Error 88 means that you have attempted to use a rotated, resampled or skewed subregion with more than one layer. (Semper currently only supports 2-D operations on such regions).

89 ***Display sub-region outside border limits***

Error 89 means that you have referred to a display (picture, partition or frame) subregion that is entirely outside the display. Likely causes include:

- mistyping the value of a **position** key
- confusing one partition with another (be explicit, e.g. **erase fs:4 ..** if in doubt)

show partition n reports the size and position of partition *n*.

90 ***Device** device number **is not a picture storage device***

Error 90 means that you have tried to apply a picture processing or inspection operation, or simply **examine** etc., to a device that contains text (e.g. a *Help Library*) rather than pictures. Likely causes of error 90 include:

- mistyping a device number
- using **examine** on a *Help Library* (use **help/topics** instead)

(**show devices** tells you about all the devices you have currently assigned.)

91 ***Arithmetic underflow detected***

Error 91 means that an arithmetic underflow has been detected. This means that the result of a calculation is too small to be represented accurately. If you are using a PC, the value zero will be substituted as the result of the calculation. If you are using a VAX, this error is only produced if Semper has been compiled with Underflow checking switched on.
Likely causes include:

- multiplying together pictures with very small (but non-zero) values
- dividing a very small valued picture by very large valued one
- taking too large a negative power, for example, **a=-1e10 b=2^a**
- using a form *fp* picture you have **created** but not yet filled with any particular values
- using a half-processed output picture following an abandoned operation

92 ***Bad floating-point value detected***

If you are using a *PC*, error 92 means that the Numeric Data Processor has found an unrepresentable value (infinity, 0/0 etc.), and has substituted the value zero instead. Likely causes include:

- using a form *fp* picture you have **created** but not yet filled with any particular values
- using a half-processed output picture following an abandoned operation
- taking too high a power, e.g. **a=1e10 b=2^a**

If you are using a *VAX*, error 92 means that Semper has met an illegal floating-point value when processing a picture. Likely causes include:

- using a form *fp* or *complex* picture you have created but not yet filled with any particular values
- using a half-processed output picture following an abandoned operation

Otherwise, error 92 means that a calculation has used or created an unrepresentable value (infinity, 0/0, etc.). Likely causes include:

- using a form *fp* picture you have **created** but not yet filled with any particular values
- using a half-processed output picture following an abandoned operation

93 *Arithmetic overflow detected*

Error 93 means that an arithmetic overflow has been detected. This means that the result of a calculation is too large to be represented accurately. If you are using a PC, the value zero will be substituted as the result of the calculation. If you are using a VAX, this error is only produced if Semper has been compiled with Overflow checking switched on. Likely causes include:

- multiplying together pictures with very large values
- dividing a very large valued picture by very small valued one
- using a form *fp* picture you have **created** but not yet filled with any particular values
- using a part-processed output picture following an abandoned operation
- if you are using a Sprynt system, storing to a smaller data class when the values do not fit (e.g. **fp** to **byte** when the range exceeds 0 to 255).

94 *Divide by zero detected*

Error 94 means that an attempt to divide by zero has been detected. This error is rare, as Semper spots the problem in advance in most cases, and reports error 95 instead. This error may occur in locally written code.

95 *Attempt to divide by zero*

Error 95 means that the divisor in an arithmetic operation was found to be zero. For example:**a=0 b=1/a** or **calculate :2/:3** where picture 3 contains some zero pixels.

96 *Attempt to take square root of negative value*

Error 96 means that the argument to **root** was found to be negative. For example: **a=-1 b=root(a)** or **calculate ln(:9)** where picture 9 contains some negative pixels.

97 **Argument to LN negative or zero**

Error 97 means that the argument to **ln** was not positive. For example **a=-1 b=ln(a)** or **calculate root(:9)** where picture 9 contains some negative or zero pixels.

98 **Argument to ACOS or ASIN out of range -1 to 1**

Error 98 means that the argument to ACOS or ASIN did not lie in the range -1 to 1.For example: **a=-pi b=acos(a)** or **calculate asin(:4)** where picture 4 contains some pixels with a value greater than 1 or less than -1.

99 *Attempt to take fractional power of zero or negative value*

Error 99 means that an attempt to raise a negative value to a non-integer power was detected. For example: **a=-pi b=1.5 c=a^b** or **calculate (:4^.5)** where picture 4 contains some negative pixels

100 *No free directory slots on device device number*

Error 100 means that the directory on a program library has filled. Try **compress**ing the device if **directory** indicates deleted slots. Otherwise create a new program library with more slots (use the **slots** keyword in **assign new**) and copy the programs across to the new library.

101 *Insufficient free space on device device number*

Error 101 means that the text area on a program library has filled. Try **compress**ing the device if **directory** indicates deleted space. Otherwise create a new program library with more space (use the **size** keyword in **assign new**) and copy the programs across to the new library.

102 *Program index overflow – reduce complexity*

Error 102 indicates that a program has more **for**, **loop** commands and labels than can be buffered internally. Either restructure the program to remove unnecessary labels, or split out part of it as a new program which can then be invoked from the remainder of the original.

103 *Local variables not allowed in FOR loop*

Error 103 means that you have a **local** statement directly inside a **for** loop. **local** statements are allowed inside programs called from within a **for** loop, but must be outside of any **for** loops within that program.

104 *Too many local variables, can't save variable*

Error 104 means that the local variable table has overflowed (see **show system** for the size of the table). Local variables are used when keys and options are quoted in a command, when variables are used as **for** loop variables, when explicit **local** variables are declared and also occasionally in code within some processing commands.

105 *LOOP doesn't match active FOR loop*

Error 105 means that the variable name specified in a **loop** command does not match the loop variable name of the inner-most active **for** loop. Likely causes for this error are:

- having a **for** loop in a macro accessed from a program
- having a **loop** command with the wrong loop variable name in a conditional expression.

(You should use **next** or **break** to conditionally affect **for** loops)

106 *FOR with no matching LOOP*

Error 106 means the interpreter cannot find a matching **loop** for a **for** command. This is more likely in a program, as you will be prompted for terminating **loop**s interactively. Other possible causes are:

- using the wrong loop variable name, for example: **for i 1,10 ... loop j**
- having a **for** loop in a macro accessed from a program

107 *RUN not allowed from within a program or FOR loop*

Error 107 means you have a **run** command within a program or **for** loop. The **run** command is used to select input from a new source, as a side-effect the current command line is lost. Any **for** loop or active program would then be left in limbo. Include the contents of the run file as a program and use **library** instead.

108 *Program nest level exceeded*

Error 108 means you have exceeded the maximum level of nesting of **library** commands that are allowed. One possible cause of this is unterminated recursion, either directly (*lib xxx* within program *xxx*) or indirectly (*lib xxx* within program *yyy* and *lib yyy* within program *xxx*). Look at the error context messages to find out the likely cause. See **show system** for the local maximum level of nesting.

109 *FOR loop nest level exceeded*

Error 109 means you have exceeded the maximum level of nesting for a **for** loop that is allowed. See **show system** for the local maximum level of nesting.

110 *NEXT or BREAK doesn't match any active FOR loops*

Error 110 means that a **next** or **break** command has been encountered with an index variable that is not in use by any active **for** loop, or there are no active **for** loops at all.

111 *LOOP with no FOR*

Error 111 means a **loop** command has been encountered when there was no active **for** loops.

112 *Label label name is inside a FOR loop*

Error 112 means that you have tried to **jump** to a label that is within a **for** loop that is not currently active.

113 *Program device device number is in use*

Error 113 means that you have attempted to **deassign** or **reinitialise** a device that has an active program on it. This error will only occur within a program. Look at the error context to determine which program is active on the device.

114 *No help library assigned*

Error 114 means that no help library has been assigned, which means that no help information can be provided on specified topics. The command **help** by itself prints some text which explains how to use the command and describes other Semper commands that can provide information about your Semper session.

115 ***No help for specified topics***

Error 115 means that no topics matching the keywords following the **help** command were found in any of the assigned help libraries. Type **help/topics** for a list of available topics.

116 ***Device or picture number out-of-range***

Error 116 means that the value of a numerical expression either before or after the colon **':'** operator is out of range. Device numbers must lie in the range 1 to the maximum allowed for the system (type **show system** to find out what is the maximum). Picture numbers must lie in the range 1 to 999.

117 ***Help libraries can only be created by the Help Manager***

Error 117 means that you are trying to create a new help library within Semper. Help libraries can only be created outside Semper using the *Help Manager* utility (usually called by **helpman**).

118 ***Display device,*** *device number*, ***is already assigned***

Error 118 occurs when you attempt to re**assign** a display device that is already assigned. If you want to change the characteristics of a display you must **deassign** it first. Currently the display device is forced to be device 1, but this will change in future versions.

119 ***Invalid or unknown file type***

Error 119 usually occurs when an attempt is made to assign a file that is not a Semper disc file, for example, attempting to assign a text file.

120 ***Reassignment of new disc files only allowed in full version***

This message only occurs in a demonstration version of Semper software. A more limited range of Semper operations are allowed in this context.

121 ***Device*** *device number* ***is already assigned***

Error 121 occurs when you assign a device that is already in use. Type **show devices** to list the currently assigned devices.

122 ***Value must be given for*** *keyword/variable*

Error 122 means that some of the options or keywords given to a command require that another keyword or variable should also be given a value.

123 ***Unable to create workspace device (retry count exceeded)***

Error 123 means that Semper has been unable to create a unique workspace area after 20 retries. Possible reasons include:

- system clock failure
- too many Semper users are trying to create workspace devices at the same time (networked/Multiuser systems only)
- the primitive routine *WAITS* is not performing correctly
- the primitive routine *MCTIME* is not performing correctly
- the primitive routine *MCDC61* is not performing correctly

124 ***Device** device number **is not a text file***

This error means that a picture device, display, tape device or help library has been used in a context that requires a program library device.

125 ***Program** program name **not found***

This error means that the specified program is not available in any of the currently assigned program library devices. Use **show programs** to produce a list of available programs.

126 ***Program** program name **already exists***

This means that an attempt has been made to **copy** or **rename** a program onto an existing program. If this is what you really want to do, delete the existing program first.

127 ***Attempt to delete active program** program name*

This error means that you have tried to delete a program while the program is still in use. This can happen if one program invokes another program that tries to delete the first one.

128 ***Program** program name **has a corrupted index***

Error 128 should not normally happen, as it means that the program library index is corrupt or has been written back incorrectly. Possible causes include system crashes when adding a new program or incompatible versions of Semper on the same host machine.

129 ***Fortran i/o error** number **on unit** number **file** filename*

Error 129 can be produced by any command that accesses the file input/output system. Look up the i/o error number in your local Fortran IOSTAT or run-time i/o error tables. On some systems Semper also produces a more meaningful interpretation of the error number as a supplementary message.

130 *File filename* ***not found***

Error 130 means that the indicated file could not be found and/or opened for input. On some systems the local PATH environment string is used to locate input files, and this can result in unreadable versions of files being found before the intended file. In these conditions give the complete local pathname for the file.

131 ***Unexpected end-of-file found in*** *filename*

Error 131 occurs when a data file ends unexpectedly. This can happen when using the commands **read**, **input** etc if the original file was incorrectly written or in the wrong format.

132 *Picture number* ***has no title***

Error 132 means that a command which requires a title for a picture was unable to locate one.

133 ***Program name*** *name* ***is invalid***

Error 133 means that a program name contains some invalid characters. Valid characters are *a* to *z*, *0* to *9* and *$*.

134 ***Too many files open***

Error 134 means that the local Semper i/o system has run out of file handles. This is usually caused by opening an excessive number of files.

135 *File filename* ***already exists***

Error 135 occurs when a command that outputs a file finds that the file already exists. In general these commands require the option **new** to overwrite a file.

136 ***File name incorrectly specified***

Error 136 means that you have incorrectly specified a filename. Check for overlong names or invalid characters.

137 ***Internal file name too long***

This error occurs when an internal buffer has overflowed when trying to construct a file name. Possible causes include:

- An error in the specification of the PATH in the local environment
- The current directory is at the end of a very long chain of directories
- The user specifying a very long file name component

138 ***No match found***

This error means that the **examine** command cannot find a match in the directory for the range, text, class or form combination that you have specified.

139 Command line buffer overflow

This error message means that an 'immediate' command, such as **textfield execute**, has caused the delayed command line buffer to overflow. This error is usually caused by a user-written routine making excessive use of the Semper function OBEYCL without returning to the command interpreter.

140 Security device missing

This error message means that the required security device is missing or inoperable. Check that the device has not been accidentally removed.

141 Wrong level security device

This error message indicates that the security device does not match the software release version. Ensure that the correct device is connected.

142 , 143, 144 Security failure

This message means that a security problem has occurred (for example, a security device malfunction or violation). If you cannot locate the source of the problem please send a Customer Report form to *Synoptics* or contact your technical support service.

145 LP lp number **already opened**

This error indicates a syntax error in the command syntax file. There is more than one open statement referring to the specified LP number. Modify the syntax entry and rebuild Semper.

146 **Attempt to reference LP** lp number **before opening**

This error indicates a syntax error in the command syntax file. An open statment for a new image references an unopened LP in the comparision field (e.g. open (lp2,new,lp1) where there is no open for LP1). Modify the syntax entry and rebuild Semper.

147 **Picture** picture number **has bad value** value **in label field** field name

This error should not normally happen in Semper. It is a less severe form of error 52, and indicates that one or more crucial fields in an existing picture label have corrupt values. The first corrupt field found and the erroneous value are reported in the error message. As with error 52 the only really useful response is to delete the picture.

148 **Picture** picture number **new value** value **in label field** field name **is invalid**

This error indicates an attempt to write a picture label that has one or more crucial fields with values that are out of range (e.g. dimensions with zero size, invalid picture form). This can happen when attempting to import external data files that are in the wrong format or which have an incorrectly produced Semper label in them. If this is the case the image may still be recoverable by indentifying the field Semper is complaining about (given in the error message) and correcting the value – either by correcting the program sourcing the file or by patching the file with a suitable text or disc editing utility.

149 **Attempt to write a malformed label for picture** picture number

Error 149 indicates an attempt to write a totally corrupt picture label to disc. This can happen when attempting to import external data files that are in the wrong format. It can also happen when user written code overwrites the label data before writing it out to disc.

150 *Chord buffer overflow*

Error 150 means that there are too many particles intersected by a picture row. This could be caused by too much noise in the source image. Some additional processing to clean up the source image or adjustment of the threshold values given to **analyse** might help to reduce the number of spurious particles. If, however, all the particles are genuine, the last resort is to break down the source image into several smaller pictures.

151 *Too many particles*

Error 151 means that the particle parameter list is not big enough to hold the results for all the particles found by **analyse**. The size of the particle parameter list is limited by the row buffer size. The **area** key can be used to screen out particles less than a certain size. **analyse** can also be made to output the results for only those particles which fall within a specified subregion of the source picture (using the 2-D subregion keys when invoking **analyse**).

Error number 152 is currently not in use

153 *Too many items to label in segmented picture*

Error 153 will occur if a byte-formatted segmented picture is requested and there is too much detail in the source image. No more than 255 particles can be recorded in a byte-formatted segmented picture. Depending on the complexity of the particles, this limit may be reduced during the scan through the source image.

154 *Bad pixel value in segmented picture*

Error.154 means that a negative pixel value has been encountered in a segmented picture. A segmented picture should contain zero (background) and positive (particle id) pixel values.

155 *Particle not found*

Error 155 means that the specified particle id was not found in the particle parameter list.

156 ***Excessively large curve***

Error 156 means that the coordinate limits of the curve supplied to **pcurve** exceed the range –32000 to 32000.

157 ***Picture is not a 2-D image***

Error 157 means that the operation to be carried out with the specified display picture requires a 2-D image.

158 *No particles selected*

Error 158 means that none of the particles described in the particle parameter list satisfy the constraints given by the **if** and **unless** keys.

159 ***Zero pixel area***

Error 159 means that there were no pixel positions generated inside the specified closed curve. This means that results that depend on summing values at all the pixel positions, for example, total intensity, are not defined.

160 ***Particle not present in segmented picture***

Error 160 means that there are no pixels in the segmented picture set to the specified particle id.

161 ***Event queue routine error***

Error 161 is an internal error that indicates problems in the low level event interface routines. If the error persists please submit a Report Form to *Synoptics (see Appendix I, Customer Report Form).*

162 ***Device** device number **already quoted***

Error 162 means that a command that expects a list of devices (for example, **order**) has found a particular device used more than once in the list.

163 ***Look-up table** number **not currently active***

This message means that a command requires an active lookup table (for example, **lut keys**) and did not find one.

164 ***CTF wave requires full-plane Fourier pictures***

This message means that the command **ctf** with the option **wave** was given a half-plane Fourier picture as a source.

165 ***Attempt to use invalid LP number*** *lp number*

Error 165 means that a call to a kernel function (e.g. SEMROW) has been made with an LP number that is out of the valid range 1 to NLPS.

This usually indicates an error in locally written code, for instance passing the wrong variable as the LP number.

166 ***Attempt to use unopened LP number*** *lp number*

Error 166 means that a call to a kernel function (e.g. SEMROW) has been made with an LP number that does not refer to an open picture.

This usually indicates an error in locally written code, for instance passing the wrong variable as the LP number or failing to open a picture before accessing it. Check that any implicit OPENs that the local code expects are present in the command descriptor.

Error numbers 167 to 191 are currently not in use

192 FORTRAN subscript out of range (program fault)

This error indicates that a *subscript out of range* error occurred in a FORTRAN routine. If the problem is not caused by a routine of your own writing, please send a Customer Report form to *Synoptics*.

193 FORTRAN adjustable array dimension error (program fault)

This error message occurs when an adjustable array dimension, passed to a FORTRAN routine, is zero or negative. For example:

```
CALL SUB(A,0)
.
.
.
SUBROUTINE SUB(A,N)
INTEGER N
REAL A(N)
```

If this error has not been caused by a user-written routine, please send a Customer Report Form to *Synoptics*.

Error numbers 194 to 249 are currently not in use

250 ***No UIF error – status set TRUE with no error code***

You should not normally get this error but you might get it when performing operations on elements with the variable *eno* set incorrectly. Has the **create** option been omitted?

251 *UIF is not initialised*

Error 251 means that you have issued a user interface command before giving the **uif enable** command. This command initialises internal variables.

252 *Maximum number of windows exceeded*

Error 252 means that that there are too many windows. A window is a *panel*– so you are using too many panels. Use the **uif status** command for a list of the number of allowed panels.

253 *Maximum number of panels exceeded*

Error 253 means that that there are too many panels. Use the **uif status** command to see a list of the available number of panels. Note that pull-down and pop-up menus use up (hidden) panels..

254 *Maximum number of scrolling areas exceeded*

Reserved for future use.

255 *Maximum number of cells exceeded*

Error 255 means that that there are too many cells. Use the **uif status** command to list the available number of cells.

256 *Maximum number of menus exceeded*

Error 256 means that that there are too many menus. Use the **uif status** command to list the available number of menus.

257 *Maximum number of textfields exceeded*

Error 257 means that that there are too many textfields. Use the **uif status** command to list the available number of textfields.

258 *Operation cannot be performed while panel is showing*

Error 258 means that you are trying to perform an operation when a panel is visible. This may be caused by:

- an attempt to add a new element to the panel
- an attempt to alter any of the characteristics of the panel

259 *Operation cannot be performed while scrolling area is showing*

Reserved for future use.

260 *Operation cannot be performed while panel is not showing*

Error 260 means that the operation cannot be performed while the panel is showing, because the operation would cause the size of the panel to change.

261 *Error setting variable value*

Error 261 occurs when the user interface system is setting a value to a variable. This is most likely if you have used too many variables (try **unset**ting some of them). If not, this indicates an internal error. Contact *Synoptics* for advice.

262 *Error getting variable value*

An internal error, please report this to *Synoptics*.

263 *Error creating window*

An internal error, please report this to *Synoptics*.

264 *Error clearing window*

An internal error, please report this to *Synoptics*.

265 *Error obtaining size of window device*

An internal error, please report this to *Synoptics*.

266 *Error showing window*

An internal error, please report this to *Synoptics*.

267 *Error destroying window*

An internal error, please report this to *Synoptics*.

268 *Error hiding window*

An internal error, please report this to *Synoptics*.

269 *Error moving window*

An internal error, please report this to *Synoptics*.

270 *Error creating panel*

An internal error, please report this to *Synoptics*.

271 *Error destroying panel*

An internal error, please report this to *Synoptics*.

272 *Error hiding panel*

An internal error, please report this to *Synoptics*.

273 ***Error showing panel***

Error 273 means that the panel is too big to display on the device (framestore/host screen) or the panel is located outside the display limits. Very often, this is because **justification** is incorrectly set.

274 ***Error moving panel***

Error 274 means that an attempt was made to move the panel outside the device limits.

275 ***Error naming panel***

An internal error, please report this to *Synoptics*.

276 ***Error setting panel/element colours***

You are trying to set a colour that your hardware cannot support.

277 ***Error clearing device***

An internal error, please report this to *Synoptics*.

278 ***Attempt to perform operation when mandatory panel showing***

Error 278 means that a *mandatory* panel is showing but you tried to perform some other user interface related action, for example showing another panel or popping up a menu. Remember that when a mandatory panel is showing interactions are only allowed with it or elements on it.

279 ***Invalid panel identifier***

Error 279 means that you have used an identifier for a panel which the user interface system knows nothing about. Are you specifying the **Id** key?

280 ***Invalid scrolling area identifier***

Reserved for future use.

281 ***Error showing scrolling area***

Reserved for future use.

282 ***Invalid position***

Error 282 means that an attempt has been made to position the panel outside the limits of the device (framestore/host display).

283 ***String too short for copy/concatenate operation***

An internal error, please report this to *Synoptics*.

284 ***UIF initialisation failed***

Error 284 means that the user interface system was unable to initialise. Have you already initialised the system? It is advisable to exit Semper and then re-start it.

285 ***Invalid device requested***

Error 285 means that an attempt was made to use an unknown device for a panel. Is the variable *cdi* set correctly? Permitted values for *cdi* are 0, 1 and 2.

286 ***Invalid element type***

An internal error, please report this to *Synoptics*.

287 ***Invalid action***

An internal error, please report this to *Synoptics*.

288 ***Invalid element position/size***

Error 288 means that the specified element (menu, cell, textfield) was too big or was badly positioned on its panel.

289 ***Invalid cell identifier***

Error 289 means that you have used an identifier for a cell which the user interface system knows nothing about. Have you passed the right value with the **id** key? Is the variable *eno* set correctly? Have you omitted the **create** option?

290 ***Invalid cell highlighting type***

An internal error, please report this to *Synoptics*.

291 ***Invalid textfield identifier***

Error 291 means that you have used an identifier for a textfield which the user interface system knows nothing about. Have you passed the right value with the **id** key? Is the variable *eno* set correctly? Have you omitted the **create** option?

292 ***Invalid textfield length***

Error 292 means that you have given an incorrect length for a textfield. Textfield lengths should be greater than zero and less than the amount of (dynamic) memory available. Type **uif status** for a list of the available memory.

293 ***Invalid numeric textfield range specifier***

Reserved for future use.

294 ***Numeric textfield contents out of range***

Reserved for future use.

295 ***Unable to convert numeric textfield contents to a number***

Reserved for future use.

296 ***Unable to convert number to a string***

Error 296 means that Semper was unable to convert the string given into a number.

297 **Invalid menu identifier**

Error 297 means that you have used an identifier for a menu which the user interface system knows nothing about. Have you passed the right value with the **id** key? Is the variable *eno* set correctly? Have you omitted the **create** option?

298 ***Invalid menu type***

An internal error, please report this to *Synoptics*.

299 ***Invalid menu style***

An internal error, please report this to *Synoptics*.

300 ***Menu panel has not been created***

An internal error, please report this to *Synoptics*.

301 ***Action string is too long to fit application input buffer***

Error 301 means that the string that you specified for an action is too long to fit into Semper's command line buffer. An action might be using the **begins** key action on a cell, or the click of a **left** mouse button, or the execution of a textfield. Try breaking the command down into smaller parts.

302 ***Framestore access error***

Error 302 means that while the user interface system was accessing the framestore, the framestore reported an error. Check for framestore hardware failure.

303 ***UIF is not running***

An internal error, please report this to *Synoptics*.

304 ***Window of requested size/position will not fit device***

Error 273 means that the panel is too big to display on the device (framestore/host screen) or the panel is located outside the display limits. Very often, this is because **justification** is incorrectly set.

305 ***UIF termination failed***

An internal error, please report this to *Synoptics*.

306 *Cursor position stack is empty*

Error 306 means that **device restore** has been used before **device save** has been used (or **restore** has been used more times than **saved** is used).

307 ***Cursor position stack is full***

Error 307 means that you have given too many **device save** commands. The cursor position stack is of a fixed size, typically about four positions. Have you omitted to restore a position somewhere?

308 ***Dynamic memory system not initialised***

An internal error, please report this to *Synoptics*.

309 ***Dynamic memory system already initialised***

An internal error, please report this to *Synoptics*.

310 ***Invalid block size of dynamic memory requested***

An internal error, please report this to *Synoptics*.

311 ***Invalid logical index to dynamic memory used***

An internal error, please report this to *Synoptics*.

312 ***Dynamic memory logical index table full***

An internal error, please report this to *Synoptics*.

313 ***Dynamic memory exhausted***

Error 313 means that all the memory reserved for storing names of objects (panels, elements), actions, etc. has been used. **uif status** will show how much you have left. The amount of memory available is fixed and installation dependent.

314 *Invalid positioning point*

An internal error, please report this to *Synoptics*.

315 *Invalid mouse button number*

An internal error, please report this to *Synoptics*.

316 *Display is not assigned for windows on framestore*

Error 316 means that you have tried to show a panel on a display that has not yet been assigned.

317 *Invalid element identifier*

An internal error, please report this to *Synoptics*.

318 *Invalid panel/scrolling area/element identifier*

An internal error, please report this to *Synoptics*.

Error numbers 319 to 332 are currently not in use

333 *The Semper UIF is not installed in this system*

UIF is not installed in this system and you have tried to use a UIF facility or command. The Semper system is either a BlackBox version of Semper, or has had the User InterFace code explicitly removed to reduce code size and memory requirements. Note that it is not possible to run the TUTOR interface without having UIF in Semper.

Error numbers 334 to 499 are currently not in use

500 *Source and covariance sizes differ*

The number of layers of source picture must correspond with the number of columns of the covariance matrix.

501 *Too many classification regions*

The number of classification regions that can be processed by the classification commands (**box**, **mindistance** and **likelihood**) is dependent on the length of a Semper row buffer. The more layers a picture has, the fewer classification regions it may have.

502 *Too many layers*

The number of layers that can be processed by the classification commands depends on both the length of a Semper row buffer and on the width of the picture. The wider a picture is, the fewer layers it may have.

503 *Threshold must be in the range 0.0 .. 100.0*

Thresholds to the **likelihood** command, maximum likelihood, are given in terms of percentages. The percentage gives the number of pixels in the class which will be classified. For this reason the percentage must be less than 100.0.

504 *Not enough thresholds*

When more than one threshold is given there must be as many thresholds as classes.

505 *Probabilities must be positive, less than 1.0*

Probabilities must be in the range 0.0 (never) to 1.0 (always).

506 *Sum of probabilities greater than 1.0*

The sum of probabilities for all classes (in the **likelihood** command) must sum to less than 1.0.

507 *Too many channels, maximum (square) is: number*

In order to achieve a respectable performance the 2-D histogram is built up in memory. As with other Semper commands the maximum number of channels is proportional to the length of a Semper row buffer.

508 *Thresholds must be positive*

Since thresholds are in terms of standard deviations it is not sensible to give negative thresholds.

509 *Unable to calculate eigenmatrix*

The maximum iteration count was reached in calculating the eigenvectors. This should not arise with covariance matrices calculated with the **covar** command. If the covariance matrix has been determined by some other method ensure it is symmetric, as the solution (*Jacobi's* method) relies upon this.

510 *Invalid polynomial order, largest is: number*

The maximum order polynomial is restricted by the (implementation dependent) size of a Semper row buffer. The polynomial order must be greater than zero.

511 *Control points must be plists*

The control points must be supplied in *Plist* class pictures (perhaps use the **reclass** command to do this).

512 ***Image and map plists must be the same length***

The map and image plists (control points) must be of the same length. Otherwise the **warp** command cannot perform point matching.

513 ***Too many control points, maximum is:*** *number*

The **warp** command can only handle a certain number of control points. This should, in normal circumstances, be sufficient to solve the maximum order of polynomial allowed. Use the **cut** command to reduce the size of the list.

514 ***Not enough control points (or singular set)***

When solving the equations for the coefficients not enough points are in the control data set to solve the equations. Alternatively, the set of equations is ill-conditioned, so it is not possible to solve them. Try adding more control points.

515 ***Number of classes and probabilities disagree***

The number of probabilities given must be equal to the number of classes.

516 ***Class has zero determinant – no inverse possible***

The covariance matrix is singular and it is not possible to evaluate its inverse. This, in turn, means that the maximum likelihood method cannot be used.

Error numbers 517 to 899 are currently not in use

900 to 949 *Message depends on error condition*

This is a non-specific error generated by a user-supplied command. In general, any non-standard errors generated by routines not supplied by *Synoptics* should be in the range 900 to 949.

Error numbers 950 to 996 are currently not in use

997 QUIT in response to page prompt

This error means that a *QUIT* option was selected in response to the page prompt. Strictly speaking, this is not an error in that it generates no error message or traceback and diagnostic information, but it is a condition which will cause commands and programs to abort. For this reason, it can be trapped like a normal error. Note that you can disable this condition using the **page...noquit** command to disable the *QUIT* facility.

998 Internal error *number* ***status*** *number* ***routine*** *name*

This is a non-specific internal Semper error. It is used to flag problems that may occur at a low level within Semper. If you encounter this error, please make a note of the information that it reports and contact *Synoptics*.

999 ***FATAL error*** *number* ***status*** *number* ***routine*** *name*

This error is a "fatal" error and means that Semper cannot continue and will shut itself down. This occurs if Semper encounters a condition which means that the session can no longer continue, for example, if commands can no longer be read from the terminal. Semper attempts to close the session in an orderly fashion, as if you had typed **stop**. If you see this error, please make a note of the information that it reports and contact *Synoptics*.

Please note that error numbers **900** to **949** are reserved for Fortran errors from user-written routines. Refer to the manual:

Fortran Programmers' Guide

for details of how to implement these errors.

Appendix F

PROTECTED & FIXED VARIABLES

Overview

This appendix describes the small number of Semper variables that have special meanings and properties. Variables have a name and an associated numerical value. They are used to hold and manipulate values for various purposes. Semper maintains a list of the variables that are set at any given time. A variable that is not on this list has no value at all and is called *unset*. Some special Semper variables on this list are called *protected* or *fixed* variables and are described below.

Protected variables

The following variables are *protected* variables, which means that you cannot change their values by assignment:

ric	release identification code (for example, 6.2)
yes	=1, used as logical value *true* and for answering questions
no	=0, used as logical value *false* and for answering questions
pi	= π = 3.1415927
select	the current picture number
display	the current display number
rc	error number after a command is executed
cframe	current display frame number
clut	current look-up-table number

Fixed variables

The following variables are *fixed* variables, which means that you can change their values by assignment but cannot unset them:

min, max, mean, me2, sd	last determined pixel data range, mean, standard deviation
trap	used in user controlled error handling
cd	current picture device number
fs	current display device number

Appendix G

PIXEL CONNECTIVITY

Overview

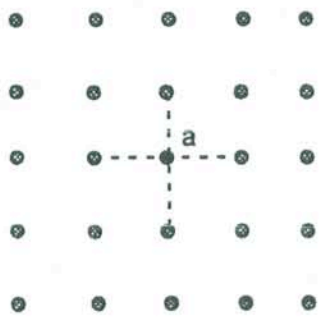
This appendix explains the concept of *pixel connectivity* that is used by the morphology commands **analyse**, **erode**, **dilate** and **median**. Pixel connectivity determines how pixels are grouped together to form objects.

4 and 8 connectivity

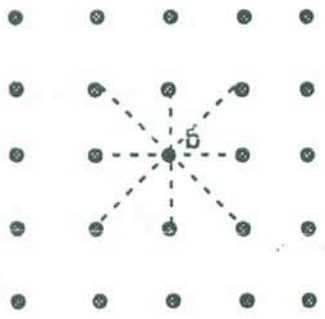
There are two kinds of pixel connectivity used:

- 4-connectivity
- 8-connectivity

In 4-connectivity, pixels are treated as if they are connected to their neighbours horizontally and vertically, but not diagonally. In 8-connectivity, diagonal neighbours are also treated as connected as shown in the diagram below. Pixel ***a*** is linked by 4-connectivity to four of its neighbours. Pixel ***b*** is linked by 8-connectivity to all eight of its neighbours.



4-connectivity



8-connectivity

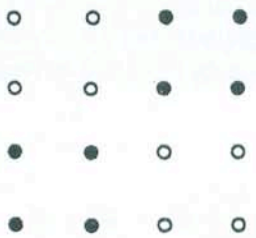
For example, the following group of pixels forms a single region using 8-connectivity but forms four separate regions using 4-connectivity.

pixel links using 8- connectivity

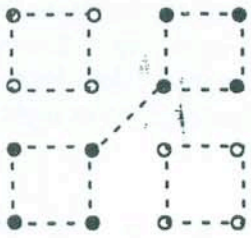
pixel links using 4- connectivity

Object and background connectivity

8-connectivity gives smoother curves and better rendering of edges, however Semper does not use it for both shapes and background as this is topologically inconsistent. For example, both object and background are 8-connected across the centre of the configuration given below. Semper therefore uses 8-connectivity to link objects and 4-connectivity for the background to separate the shape from the background. Object pixels are shown in black, background pixels in white.



original configuration



8-connectivity used to link object, 4-connectivity for background

This type of object and background separation is used, for example:

- by the **analyse** command to distinguish a particle from its surroundings
- by the **dilate separately** command to shrink the background to a 4-connected skeleton to maintain separation of objects
- by **erode skeleton** to produce an 8-connected skeleton

Appendix H

ILLUMINATION

Overview

The **sheet** and **solid** commands make use of an illumination or lighting model to determine the light intensity at every point of the shaded surface representation. There are four components in the lighting model:

- ambient lighting
- depth contrast
- forward lighting
- directed lighting

} diffuse and specular reflections

This appendix describes each of these components and presents the lighting equation which specifies exactly how these components are combined.

Ambient lighting

Ambient light is non-directional and is applied uniformly at all points of a surface that it illuminates. The amount of ambient light is specified by the **ambient** key (default=40). It is used to light all parts of an object so that the object stands out from the background.

Depth contrast

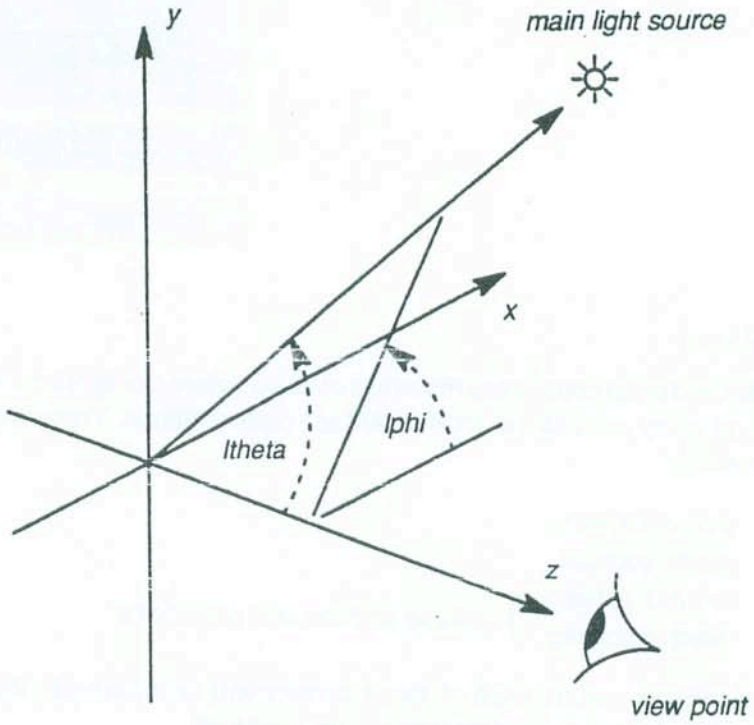
Depth contrast (or depth cueing) makes parts of an object that are further away from you less bright. This conveys some sense of depth for an object. The value that you specify with the **dcontrast** key is the difference in brightness between the front and rear of the object (default=30).

Forward lighting

This is a source of light that shines along the viewing direction. It helps to provide lighting on regions of a surface that are not illuminated from the main light source. If a surface is turned away from the eye, there can be no contribution from this light source, and in any case, the surface is not visible, so all lighting is suppressed. The intensity of the forward light source is specified with the **forward** key (default=42).

Directed lighting

The main light source is the only light source whose direction can be controlled. The keys **ltheta** and **lphi** specify the direction with respect to the viewing direction. The light source is inclined at an angle **ltheta** from the positive *z* axis, with an *x–y* azimuth of **lphi** anti-clockwise from the positive *x* axis. The axes mentioned here lie in the projected coordinate space, with the axes centred on the source origin and with the eye looking along the negative *z*-axis.



By default the light shines from your right (**ltheta**=pi/4, **lphi**=0). The intensity of the main light source is specified with the **main** key (default=154).

Diffuse and specular reflection

The light from the two light sources (forward and main) is reflected from a surface as two components. The **sdr** key specifies how the incident light intensity is to be split between diffuse and specular reflections (default=0.4). *Diffuse reflection* models the scattering of light at a surface in all directions and depends only on the incident angle of the light on the surface. *Specular reflection* models the reflection of light from a polished or mirror-like surface, where the reflected intensity is greatest when the surface normal is midway between the lighting and eye directions, and it falls off rapidly at other angles. It is specular reflection that generates well defined highlights on an object. The value of **sdr** is a measure of surface reflectivity – zero models a perfectly rough surface and large values (3 or more) model a highly polished surface.

Lighting equation

surface intensity = ***ambient***
+ ***dcontrast*** * *z/(zmax–zmin)*
+ ***forward*** * *(d* * ***n.e*** *+ s* * *sr(**n.e**))*
+ ***main*** * *(d* * ***n.l*** *+ s* * *sr(**n.m**))*

where: *z* = surface *z* value
zmax = maximum projected *z* value
zmin = minimum projected *z* value

n = surface normal
e = eye direction
l = main light source direction
m = 'mirror' normal = $\frac{e + l}{|e + l|}$

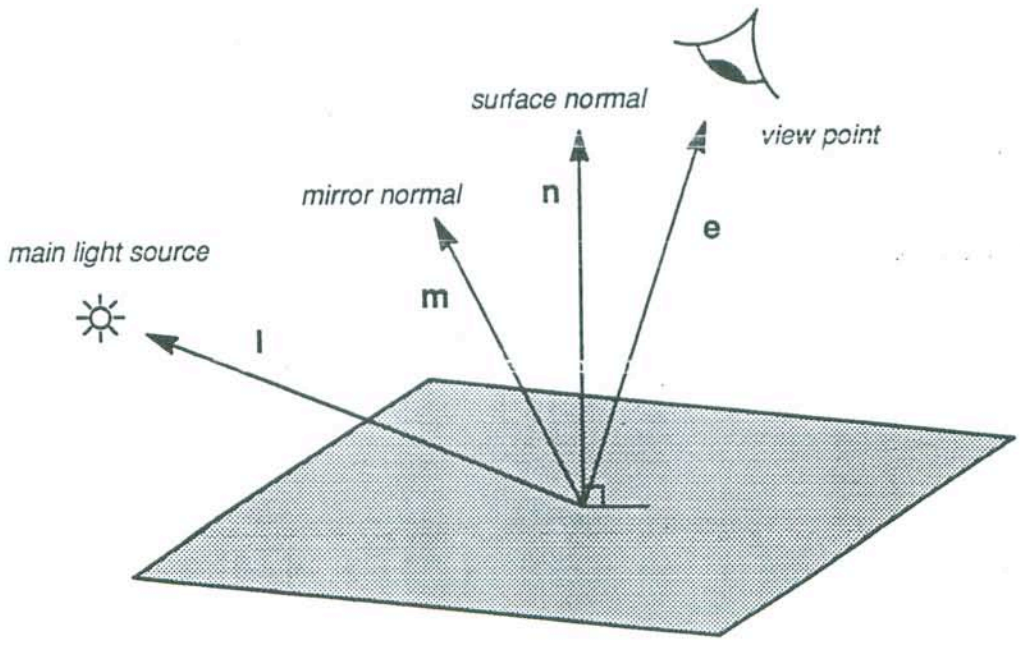
d = diffuse reflection factor = $\frac{1}{1 + sdr}$

s = specular reflection factor = $\frac{sdr}{1 + sdr}$

$sr(x)$ = specular reflection function = $\frac{0.03x}{1.03 - x}$

Notes:

1. n, e, l, m are all unit vectors.
2. If $n.e < 0$ the surface points away from the eye, so the surface intensity is zero.
3. If $n.l < 0$ the main light source does not illuminate the surface, so its contribution is zero.



Appendix I

CUSTOMER REPORT FORM

Using the report form

Please fill in and detach the form given overleaf if you find a fault with the following:

- software
- hardware
- documentation

Also use this form to record technical enquiries, enhancement requests, suggestions or comments. Send the form to the appropriate address given below:

Synoptics Ltd
271 Cambridge Science Park
Milton Road
Cambridge
CB4 4WE
UK
Tel: (0223) 423223
Fax: (0223) 420020

Synoptics Ltd
400 Amherst Street
Nashua
New Hampshire 03063
USA

Tel: 603 881 7035
Fax: 603 881 7078

Appendix J

ASCII KEY CODES

Overview

This appendix describes the ASCII key codes that are used by Semper. Each keystroke from the keyboard is converted to an internal character code (see the commands **event** and **inkey**). Semper uses the standard ASCII character sequence for the keycodes, with some additions for special keystrokes such as function keys and cursor keys. The ASCII codes and special keycodes are shown in the tables given below.

ASCII keycodes

space	32	0	48	@	64	P	80	‘	96	p	112
!	33	1	49	A	65	Q	81	a	97	q	113
"	34	2	50	B	66	R	82	b	98	r	114
#	35	3	51	C	67	S	83	c	99	s	115
$	36	4	52	D	68	T	84	d	100	t	116
%	37	5	53	E	69	U	85	e	101	u	117
&	38	6	54	F	70	V	86	f	102	v	118
'	39	7	55	G	71	W	87	g	103	w	119
(	40	8	56	H	72	X	88	h	104	x	120
)	41	9	57	I	73	Y	89	i	105	y	121
*	42	:	58	J	74	Z	90	j	106	z	122
+	43	;	59	K	75	[	91	k	107	{	123
,	44	<	60	L	76	\	92	l	108	|	124
–	45	=	61	M	77	]	93	m	109	}	125
.	46	>	62	N	78	^	94	n	110	~	126
/	47	?	63	O	79	_	95	o	111		

Special keycodes

9	Tab	257	Backspace	513	Cursor up
13	Return	258	Delete line	514	Cursor down
27	Escape	259	Insert/replace mode	515	Cursor left
		260	Start of line (home)	516	Cursor down
		261	End of line		
		262	Refresh line		

768 – 1023 Function key= 768 + key number