ESD-TR-89-118

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

SKETCH 4B
AN IMAGE UNDERSTANDING OPERATING SYSTEM

14 JUNE 1989

Prepared for the Defense Advanced Research Projects Agency under Air Force Contract F19628-85-C-0002.

LINCOLN MANUAL 163

AD A209677

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

SKETCH 4B
AN IMAGE UNDERSTANDING OPERATING SYSTEM

R. WALTON
J. VERLY
P. VAN HOVE
Group 21

LINCOLN MANUAL 163

14 JUNE 1989

Prepared for the Defense Advanced Research Projects Agency under Air Force Contract F19628-85-C-0002.

LEXINGTON MASSACHUSETTS

The report is based on studies performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. The work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-85-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Hugh L. Southall

Hugh L. Southall, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

SKETCH 4B

AN IMAGE UNDERSTANDING OPERATING SYSTEM

by

Robert Walton

Jacques Verly

Patrick Van Hove

April 1989

MIT Lincoln Laboratory

1. SKETCH LICENSE REQUIREMENTS.
2. SKETCH MANUAL.
3. SKETCH DEMONSTRATION PROGRAMS.
4. SOURCE FILES OF A REPRESENTATIVE SKETCH PACKAGE.

SKETCH LICENSE REQUIREMENTS

April 1989

WARNING

TO RECEIVE SKETCH YOU MUST HAVE THE FOLLOWING:

1. PERMISSION FROM MIT.
2. BERKELEY UNIX ON VAX OR SUN3.
3. WESTERN ELECTRIC DEVICE INDEPENDENT TROFF
   (UNLESS YOU DO NOT WANT TO PRINT DOCUMENTATION).
4. FRANZ LISP FROM FRANZ INC. IF SUN3
   (NOT NECESSARY IF VAX).

SKETCH MANUAL

VERSION 4B

April 1989

by

Robert Walton

Jacques Verly

Patrick Van Hove

MIT Lincoln Laboratory

CHAPTERS

APPENDIXES

CHAPTER 1

INTRODUCTION

1. PURPOSE. SKETCH is an image understanding operating system designed for use by the serious programmer who is trying to construct and debug complex AI Image Understanding programs. The emphasis in SKETCH is on being small, efficient, and flexible, and on promoting modularity of the final program. SKETCH is not intended to be an image processing system for non-programmers.

SKETCH is also useful, in the hands of an experienced programmer, for evaluating the performance of algorithms, complex or simple, against large data sets, and for storing resulting images and tables on disk for rapid review and reference.

2. REQUIREMENTS. SKETCH strives to meet the following requirements:

2.1. SKETCH IS LISP BASED. An interpretive language is desirable for control, because the programmer productivity of an interpretive language is at least 50% better than that of a compiled language.

Little use of disk should be made during the computation, for speed reasons.

The best standard language for these purposes is LISP.

2.2. SKETCH USES FORTRAN AND C FOR SIGNAL PROCESSING. No LISP, including COMMON LISP in most of its current implementations, seems to be able to handle numeric computations in a cost effective manner. Therefore, array oriented algorithms need to be written in FORTRAN or C.

2.3. SKETCH INTERFACES WITH EXISTING DATA. Tapes in existing data libraries usually do not have to be reformated.

2.4. SKETCH PROCESSES MODERATE AMOUNTS OF DATA. Development of AI Image Understanding algorithms requires testing against many thousands of images. Although this is usually not practical, because the CPU time requirements are excessive, it is at least practical to test against hundreds of small images each day, given proper support. This involves batch runs, storing results on disk for rapid review and reference, and interactively repeating alternative computations on any input image that requires further analysis.

2.5. SKETCH INTERFACES WITH EXISTING AI PACKAGES. Existing AI software packages, like OPS5, PEARL, etc., should be integrable with SKETCH. SKETCH should not prejudge the representation of symbolic data, because different representations have different efficiencies in different applications.

2.6. SKETCH PROMOTES MODULARITY. Users of the SKETCH should generally be able to write signal processing modules without regard to interfaces other than those defined by the SKETCH.

3. MAJOR SKETCH COMPONENTS. SKETCH supports several general purpose object data types that facilitate communications between user subroutines in the same way that the floating point number types do: by providing easy to use standards that everyone can use without much thought. Specifically provided are SKETCH array, catalog, and display data types, and extra support for the LISP S-expression data type when it is used for messages to human users. SKETCH also provides an object package with specific support for defining semantic networks and storing them in catalogs (i.e., files).

3.1. THE ARRAY PACKAGE. SKETCH tries to do for array computations what the floating point number did for simple arithmetic: provide a single array data type everyone can share, and reduce user bookkeeping operations to a minimum.

SKETCH arrays support element values not supported in other systems. Block floating point elements are supported: that is, elements that are stored as integers, with all elements of the same array being multiplied by the same power of two. Block floating point arrays are useful to compact array storage on disk or in MOS memory, and have been useful in the past to speed computation on computers with slow floating point hardware. Missing values are also supported for array elements, even in block floating point and on computers without IEEE floating point.

More precisely, SKETCH supports arrays with 8-, 16-, and 32- bit signed or unsigned block floating point numbers, 1-bit unsigned integers, and 32- or 64- bit floating point numbers. Any array with signed numeric values may store missing values.

SKETCH allows a subroutine that has received an array to request that the array be put in the format of the subroutine's choice. The subroutine can ask for the array to have single precision floating point elements, for example, and if the array does not already have these, it will be converted to have them. In image processing work, it is also often desirable to add extra rows and columns around the edges of a 2D array by mirroring the rows and columns near the edges, and this can be requested by the subroutine at the same time.

Thus the programmer is relieved from the burden of having to present **arrays** in the right format to each subroutine. There is no debilitating speed penalty for conversion of array element types and provision of boundary extension values, as this is accomplished by working with data in blocks, rather than one element at a time.

Similarly, because arrays are allocated in a garbage collected memory, the programmer is relieved from managing memory, or allocating output arrays for a subroutine he is calling. Each subroutine merely allocates its own output arrays with the element format it prefers, and returns these arrays as values to its caller.

In SKETCH, arrays are not just collections of elements, but are in fact a vector of elements plus a linear map of array subscripts to vector subscripts. Two arrays may share the same vector, and as any linear subscript map is allowed, one array may be a window into the other, or a transpose of the other.

SKETCH also solves the problem of writing functions that deal with arrays of any dimension; for example, elementwise addition of arrays of any dimension. SKETCH makes all arrays 6 dimensional, with unused dimensions set to size 1. An elementwise array addition program checks all its arrays for identical dimension sizes, and then executes 6 nested loops, with the outermost loops iterating only once for unused dimensions. The 6 nested loops are embedded in a macro, so the fact that there are as many as 6 loops is generally invisible to the programmer.

SKETCH arrays have several facilities aiding input/output. First, the array control information is stored separately from the array elements on disk. The control information is stored as ASCII LISP S-expressions, human readable and editable, in files called catalogs (catalogs are discussed in more detail below). The array elements are stored in binary form, in separate files called array caches, and are pointed to by the control information, which contains the file name and offset of where the elements are stored.

As a consequence of this arrangement, array control information may be copied between catalog files without copying the binary elements. Different catalogs may be created without duplicating array elements.

Arrays have a property list like LISP symbols, permitting the user to add identification and other information to arrays. This information is stored with the array control information in catalogs, and may be used to select arrays from a catalog.

When an array, i.e., its control information, is read from a catalog, the array in MOS memory has no elements, but continues to point at the elements on disk. If the elements of an array are wanted in a particular format by some subroutine, and the elements are not in MOS memory but are on disk, then the elements are read from disk. Before altering elements of an array, a program indicates that it wants to write the array, and the array is marked as no longer having valid elements on disk. But if the array elements are not altered, the array continues to remember where its elements are stored on disk.

When an array is written into a catalog file, the array elements will be written into an appropriate array cache file, if an only if these elements do not already exist somewhere on disk.

Thus catalogs containing arrays can be copied and edited without copying array elements unnecessarily.

A general underlying idea of SKETCH is that the elements of an array can be in many different formats in many different places. SKETCH, as it now exists, makes less use of this notion than it might, but uses it enough to work well. Future versions designed for parallel computers would necessarily make extensive use of the notion, because in addition to changing the element format of an array, and adding mirrored edges to image arrays, different subroutines would also want arrays to be laid out in different ways in parallel memory.

3.2. THE OBJECT PACKAGE. The object package is used to store information about arrays and other objects. It combines significant features of the LISP defstruct facility, LISP property lists, and objects in the SMALLTALK language. Objects are LISP values which have a type and attributes. The attributes play the role of properties in a property list, defstruct slots, or SMALLTALK messages.

The storage of objects and access to their attributes can be optimized after the manner of defstruct. The method for doing this is extensible, and the SKETCH object system should be integrable with the data storage systems of other AI tools, e.g. PEARL.

Currently the SKETCH object package is well integrated with the C language structure defining facility.

An important feature of SKETCH is that attribute labels and object types do not have to be predeclared, nor is it necessary to specify in advance all the attributes that can be attached to a SKETCH object or all the types of SKETCH objects. Input data can create attribute labels and object types on the fly, as can interpreted code. With special syntax, compiled code can do this too. In this respect, SKETCH objects are similar to LISP property lists. The SKETCH data cataloging facility makes important use of this feature.

It is possible to have access to an attribute of a SKETCH object trigger a function, and to pass extra arguments to that function. By this means the basic capabilities of SMALLTALK objects are supported. Operations not having to do with attribute access can also be defined on objects. These are generic in that they have different definitions when applied to different object types.

The SKETCH objects package permits both a function and a macro to be supplied for operations on objects and attributes. The macro is used to obtain efficiency when there is sufficient information about object types available at macro expansion time. The function performs the operation when such information is not available until later.

The SKETCH objects package also facilitates I/O of recursive semantic networks by providing a system for naming objects with analogs of symbol print names, and a system for forward referencing named objects pointed at by other objects being read into memory. In a catalog a reference to a named object is represented by just the type and name of the object. When a forward reference to an object is read, a place holder, or stub, is allocated for the object. Later, when the full object is read, the stub is filled in with the rest of the object attributes.

3.3. THE CATALOG PACKAGE. The catalog package stores information in the file system concerning arrays and other LISP values. A catalog is just a sequence of objects stored in an ASCII file.

Catalog entries are LISP values that are read and evaluated to create objects. Because they are evaluated, they can represent objects by giving algorithms for computing them, rather than just by providing a direct representation of the object.

To write an object in a catalog, one unevaluates the object and prints the resulting expression. Unevaluation is a standard objects package operation that can be defined according to object type. The unevaluate-print-read-evaluate mechanism of data storage and retrieval is a very powerful mechanism for representing complex objects in catalogs.

Catalogs can include other catalogs. A special include entry can be placed in one catalog to cause the contents of a second catalog to appear to replace the include entry in the first catalog.

Catalogs can be defined as applying filters to other catalogs. A filter is a function of one variable. If one catalog is a filter of a second catalog by a particular function, the function is applied to each object in the second catalog to make the corresponding object of the first catalog. This is done incrementally whenever an object is to be read from the first catalog. By returning the special value *please-ignore*, the function can cause objects from the second catalog to be deleted, in the sense they will be skipped when objects are read from the first catalog.

An index for a catalog may be built and saved on disk, so random access in the catalog is fast.

It is generally easy to write programs which read existing datasets and produce SKETCH catalogs listing the arrays in these datasets. These cataloged array objects can contain whatever parameters describe the data. In particular, they may include new attributes not declared to SKETCH code.

3.4. THE DISPLAY PACKAGE. The display package implements display objects and the means of displaying them on a monitor. Display objects are basically memories that describe 2D arrays of pixels. Included is an intensity array which typically has one 8-bit unsigned integer code per pixel. Display objects reference color maps that map pixel intensity codes to colors. Display objects can additionally have up to 32 bitgraph planes, each with one bit per pixel. Pixels with a '1' bit in a bitgraph plane are overlaid with a color code determined by the plane. Each bitgraph plane is individually erasable, permitting somewhat dynamic displays.

Bitgraph planes are used to store text and vectors. A display can also store text and vectors in an S-expression based format that can be used to redraw this portion of the display in a different resolution.

The display package emphasizes display device independence and the precomputation of displays for rapid browsing. Display objects can be stored in catalogs, after the manner of any SKETCH object some of whose attributes are SKETCH arrays.

3.5. THE TOP LEVEL PACKAGE (TO BE IMPLEMENTED). The top level package will contain functions commonly used to control SKETCH runs. Computations of objects from other objects are defined by tables so that when an object is required it can be computed in the most efficient possible manner. Information about generating displays is similarly defined by tables. Given these tables, a user can quickly customize interactive jobs that compute and display objects, and batch jobs that precompute objects and displays for later use. The user can also quickly modify the form of the display and the objects displayed.

Detailed CPU timing statistics will also be automatically recorded.

Currently the top level package does not exist.

4. MANUAL CONVENTIONS. SKETCH is a set of packages, each of which is documented in its own manual chapter (or appendix). The last section of a package chapter is the GLOSSARY, which describes all the global names defined in the package. Also described are a few technical terms used in the package documentation. Names defined in the glossary are italicized wherever they appear in the manual. There is an index of all glossary names in Appendix A.

The sections of the package chapter before the GLOSSARY are called the tutorial for the package. The glossaries by themselves are complete reference documentation: the package tutorials are not generally complete. If there is no tutorial, it is recommended that one first read the demonstration program listings for the package that appear after the manual, looking up the new names encountered in the glossary as you go.

Just before a package glossary there is often a section titled HITLIST which lists known problems with the package which we would like to fix, plus enhancements we might like to make.

See APPENDIX D for more details on writing package chapters.

5. GETTING STARTED. To get started invoke the *sketch* program as a UNIX command. This program is in the SKETCH root directory (you must ask where this root directory is on your system). It is a lisp environment with all the features of the FRANZ *lisp*(1) program, plus the additional features of SKETCH. For example, if you invoke *sketch* and at the prompt '->' you type—

(*print-array* (*an-array has-sizes* '(10 10) *by-expression* '(*sum* X Y)))

then SKETCH will print out a 10X10 array with each element equal to the sum of the subscripts that reference the element.

SKETCH should be learned by trying things out as you read the manual. In lieu of this, you may consult the demonstration listings at the end of the manual. Demonstration programs are run as if they had been typed into *sketch*, one line at a time, and the resulting display was made into a listing. Such demonstration programs are used to debug programs initially, to recheck programs after changes, and to act as examples for new users. Demonstration programs with names of the form xxx_xdemo.l may be found in package subdirectories of the SKETCH root directory.

The SKETCH compiler, named *sketchcom*, is used just like FRANZ *liszt*(1). *Sketchcom* is also in the SKETCH root directory.

SKETCH C code can be compiled with the normal C compiler, provided you *#include* the appropriate package *.h* files. These are in the package subdirectories of the SKETCH root directory, and you should compile with UNIX commands such as—

cc -O -c -I<SKETCH-root-directory> <file1>.c ...

which permit the SKETCH *.h* files to be found.

Each package has a file named <xxx/xxx_defs.h> relative to the SKETCH root directory. This *.h* file includes everything that C language code needs to use the package. Here xxx is the package prefix, the same prefix as on all global names defined by the package. E.g., if you use *sar_array*, defined by the ARRAYS package, *#include <sar/sar_defs.h>*.

The ARRAYS package *.h* file, *<sar/sar_defs.h>*, includes the *.h* files of all packages appearing before it in this manual. Since most programs do not use C global names that are defined in packages after the ARRAYS package in this manual, one can often just include this ARRAYS package *.h* file and nothing else.

There is also a sophisticated and fairly easy to use SKETCH file-making facility built on top of the UNIX *make* program. See APPENDIX C for details.

CHAPTER 2

LISP TUTORIAL

1. APOLOGY Sorry, but we have not yet finished converting the very old SKETCH1 version of this chapter to something correct for current SKETCH.

CHAPTER 3

FRANZ EXTENSIONS

1. FRANZ EXTENSIONS. This package consists of a set of miscellaneous functions that extend the capabilities of FRANZ LISP in many different directions. Tables on this and the following pages briefly describe the functions and global variables defined by this package. All these are defined in more detail in the glossary, but those marked with a dagger (†) are also mentioned in the tutorial sections before the glossary.

2. PRETTY PRINTING. The philosophy of SKETCH is that messages intended for people can be organized as LISP lists. The messages are usually like sentences with the left parenthesis '(' serving in place of initial capitalization and the right parenthesis ')' serving as the period. Paragraphs are just lists of sentences, with an extra '(' serving as paragraph beginning and an extra ')' as paragraph end.

The *pretty-print* function prints arbitrarily complex lists, and is the heart of the system for outputting messages. Unlike most other programming languages, this system does formatting almost automatically, relieving the programmer of a very substantial

ARITHMETIC	
(**ceiling** 'n_number)	Computes the smallest integer that is not less than n_number.
(**floor** 'n_number)	Computes the largest integer that is not larger than n_number.
pi **sqrt-pi**	The constant pi and its square root.
(**round** 'n_number)	Computes the nearest integer to a given number.

ENVIRONMENT	
is-compiler	Non-*nil* in a SKETCH compiler environment, and *nil* in a SKETCH evaluator environment.
SFE_LINT †	C Macro. 1 if macro expansion is being done for lint, 0 if for the C compiler.
SFE_VAX	C Macro. 1 if compilation is for a DEC VAX. 0 if not.
SFE_MC68000	C Macro. 1 if compilation is for a Motorola 68000. 0 if not.

ERROR CHECKING AND HANDLING	
(**assert** 'g_condition ['g_message])	LISP macro. Evaluates g_condition, and if it is *nil*, calls *error* with g_message.
(**ccheck** 'g_value) †	A LISP function to check whether a C function that has just been called has signaled an error by calling *sfe_error*. If yes, reads the error message stored by *sfe_error* and passes it to *error*. If no returns g_value.
(**error** 'l_message) †	Signals that an error has occurred, taking as a single argument an error message, l_message, which is a list to be *pretty-print*'ed.
(**error-trace** 's_switch)	Sets a switch that if on causes the system to continuously keep records that allow the detailed state of the stack to be printed if an error should occur. Unfortunately, this record keeping can be quite consumptive of CPU cycles.
exit-on-error	If non-*nil*, causes the program to exit on an *error*.
sfe_assert (g_test, t_message) † **sfe_assert1** (g_test, t_message, ...) **sfe_assert2** (g_test, t_message, ...) **sfe_assert3** (g_test, t_message, ...) **sfe_assert4** (g_test, t_message, ...) **sfe_assert5** (g_test, t_message, ...)	A C Macro. Evaluates g_test, and if false (0), calls *sfe_error* with t_message and the other arguments (...) to signal the error, and then calls *sfe_return* to take an error return from the current C function.
sfe_check () †	A C macro. Checks whether a C function just called signaled an error by calling *sfe_error*. If yes, calls *sfe_return* to take an error return from the current C function.
sfe_error (t_message, ...) †	A C function. Called with a message to signal an error. T_message and the other arguments (...) are as for *sprintf*. Sets an error switch that is read and reset by *ccheck*, and stores the error message in a buffer for *ccheck* to read and pass to *error*.
sfe_iserror () †	A C macro that returns true (non-zero) if the error switch set by *sfe_error* is on.
sfe_return; †	A C macro. Returns from the current C function. For use if an error has occurred.

FILE HANDLING	
(**demo** 's_input-file [*t*] ['s_output-file [*t*]])	Reads from s_input-file and produces output as if the lines of s_input-file were typed in. The output may be redirected to s_output-file.
(**search-path** '(s_directory ...) 's_file ['s_mode])	Searches for and returns the full pathname of a file given a list of directories, the user supplied partial pathname s_file, and an access mode, s_mode, which is '*r*, '*w*, or '*a* to denote read, write, or append.
(**split-filename 's_filename**)	Separates the directory part of s_filename from the basename part, returning the two element list: (directory-name base-name).
(**stringopen** 't_string 'x_size † 's_mode ['t_name])	Opens a port that makes t_string into a file. X_size is the number of bytes in t_string, and s_mode is '*r*, t_string are not treated specially, and '*rs*, '*ws*, or '*as* for read, write, or append where NUL's are specially treated as the end-of-file.
(**use-ptport** 'p_port)	Indicates when output sent to p_port is also being sent to *ptport*. Useful for making C code that uses printf work right with *demo*.

LIST HANDLING	
(**copy-list** 'l_list ['x_length 'g_fill])	Copies l_list, without recursively copying sublists (unlike *copy*). Can optionally fill the resulting list to a specific x_length with elements equal to g_fill, or truncate the result to x_length.
(**equal-filled-lists** 'l_list-1 'l_list-2 'g_fill)	Tests equality of lists, filling the shorter list with elements equal to g_fill, if the lists are not of equal length.
(**list-depth** 'g_value)	Computes the nesting depth of sublists of g_value. Returns 0 if g_value is not a list.
(**list-length** 'g_list ['u_predicate])	Computes the length of g_list, verifies that the list terminates with a *nil*, and can optionally check whether the list elements satisfy u_predicate. Returns the list length if all is well, or -1 otherwise.

LOADING AND DUMPING	
(**cload** '([s_discipline] s_function ...) † '(s_file [s_library]))	Loads C language s_file.*o* file containing definitions of s_function ... with calling discipline s_discipline. S_library specifies C *.o* file libraries to be searched after loading s_file.*o*.
(**dumplisp** s_file) †	Dumps the current evaluator or compiler environment into a file, s_file, which becomes a new evaluator or compiler.

MEMORY MANAGEMENT	
(**carray** 'a_array)	Returns address of first element of a_array as a *fixnum*, so that can be passed to a C function.
gc-history ***gc-history-count*** ***gc-count***	Variables that hold records of garbage collector activity.
(**purearray** ...) (***purearray** ...)	Like the LISP *array* macro or **array* function, but allocates an array whose elements are ignored by the garbage collector. This speeds up garbage collection.
(**puresegment** 's_type 'x_size)	Returns the first of x_size contiguous LISP objects of *typep* type s_type. Like *segment*, but the allocated elements are ignored by the garbage collector. This speeds up garbage collection.

MEMORY REFERENCE	
(**copy-setf-function** 's_symbol 's_source)	Makes s_symbol have the same *setf* behavior as s_source.
(**defsetf** s_function ...)	Defines the *setf* behavior of s_function.
(**dpb** 'x_value #oPPSS 'x_number)	Returns x_number with the field specified by #oPPSS (see *ldb* below) replaced by x_value.
(**has-setf-function** 's_symbol)	Returns non-*nil* if s_symbol has a *setf* behavior.
(**ldb** #oPPSS 'x_number)	Returns the bit field obtained by right shifting x_number by PP bits and masking off the low order SS bits. PP and SS are octal numbers.
(**vrefi-double** 'V_vector 'x_index)	Accesses the x_index+1'st *flonum* stored in the immediate vector V_vector.
(**vsize-long** 'V_vector) (**vsize-double** 'V_vector)	Returns the number of 32 bit long *fixnum*'s or 64 bit double *flonum*'s stored in the immediate vector, V_vector.

PRINTING AND PRETTY PRINTING	
float-format	The C format in which LISP *flonum*'s are printed. Defaults to "%.6g" in SKETCH.
line-length	The line length in columns for pretty printing. Defaults to 80.
(**pretty-format** 'g_value ['x_level])	Returns the format of g_value for pretty printing. Such a format gives detailed instructions for controlling optional carriage return insertion.
(**pretty-print** 'g_value † ['p_port ['x_margin ['s_string ['x_repeat ['x_right-margin]]]]])	Prints g_value in a pretty format by inserting carriage returns and tabs. In detail, first *pretty-format*'s the value and then *pretty-print-format*'s the resulting format.
(**pretty-print-format** 'g_format ['p_port ['x_margin ['s_string ['x_repeat ['x_right-margin]]]]])	Pretty prints a format, g_format, returned by *pretty-format*.
(**pretty-tab** 'x_margin ['p_port ['s_string ['x_repeat]]])	Tabs to the x_margin+1'st column. Permits special line headers to indicate indentation of tracing or similar matters.
(**print-size** 'g_value ['x_maximum])	Computes the number of characters that would be outputted by *print*'ing g_value.

TIMING	
(fdelay 'f_time)	Delays until f_time. F_time is as measured by *ftime*.
(ftime)	Returns a finely measured time. The error of measurement is 1/60'th second or less. The time returned is measured in seconds from midnight, Jan. 1, 1970. GMT.
ptime-counts-per-second	The number of ticks per second for the value returned by the *ptime* function.
(xtime 'g_expression)	Measures the CPU time in seconds taken by the evaluation of g_expression, exclusive of garbage collection time.

TOP LEVEL	
(**argv-shift** ['x_number])	Removes x_number arguments from the beginning of the list of UNIX command line arguments which are individually returnable by *argv*.
top-level-init ***top-level-exit*** ***top-level-prompt*** ***top-level-read*** ***top-level-eval*** ***top-level-print*** ***top-level-times*** ***top-level-print-times***	Global variables which are set to the functions that perform various parts of the top level algorithm. May be reset to control that algorithm.
top-level-init-started ***top-level-init-times*** ***top-level-saved-times*** ***top-level-saved-print-times***	Global variables used by the top level to save information.
(status **top-level-rc-files**) †	The list of places to look for a parameter file to be read during initialization of a SKETCH evaluator or compiler.
(status **top-level-switches**) †	The list of switches (*-E* and *-I*) that will be recognized and processed at the beginning of the UNIX argument list to a SKETCH evaluator or compiler.
top-level-threshold-time †	The minimum time in seconds that must be consumed by evaluating an expression read by the top level before timing statistics for evaluating the expression will be printed out.
+ † ++ † +++ †	The last (+), next-to-last (++), and next-to-next-to-last (+++) expression read by the top level.
* † ** † *** †	The last (*), next-to-last (**), and next-to-next-to-last (***) result of evaluating an expression read by the top level.

burden. Also, SKETCH *pretty-print* is somewhat more sophisticated than most other LISP pretty printers.

As an example, consider the LISP expression—

(*pretty-print* '(cannot open ,file for writing))

which is intended to output an error message in the case a file cannot be opened for output. There is no need for the programmer to worry about line feeds in long error messages, such as when the file has a very long name. The *pretty-print* function will insert line feeds for him. However, the programmer must put line feeds after messages, using the *terpri* function, as *pretty-print* does not do this.

The normal FRANZ LISP *error* function, which signals that an error has occurred and outputs an error message, has been modified to take a single list argument as the error message which is to be pretty printed. E.g.—

(*error* '(cannot open ,file for writing))

Although LISP does have formatted print routines similar to those of C and FORTRAN, their use is avoided in SKETCH, because they do not automatically insert carriage returns or indent for readability.

3. RC FILES. Whenever any version of a SKETCH evaluator or of a SKETCH compiler is loaded, it searches for files in the list returned by—

(*status top-level-rc-files*)

and applies the LISP *load* function to the first such file found. The usual default values for these lists are—

(*sketch.rc ../sketch.rc ../../sketch.rc ~/sketch.rc*)

for the SKETCH evaluator and—

(*sketchcom.rc ../sketchcom.rc ../../sketchcom.rc ~/sketchcom.rc*)

for the SKETCH compiler.

4. SKETCH SWITCHES. When a SKETCH evaluator or compiler is loaded, and after any *top-level-rc-files* are loaded, the following argument flags are processed. Any

-*I* file-name

arguments cause the file-name to be loaded by the LISP *load* function. Any

-*E* "expression"

arguments cause the expression to be read by the LISP *read* function and evaluated by the LISP *eval* function. Any errors occurring during these loadings and evaluations will terminate the SKETCH program. The arguments so processed must be at the beginning of the argument list, and will be removed from the argument list. The rest of the argument list may then be accessed as if these removed arguments had never existed.

5. TOP LEVEL VARIABLES. The top level reads an input expression, evaluates it, and prints the resulting value. The global variables +, ++, and +++ are set respectively to the last, next-to-last, and next-to-next-to-last expressions read. The global variables *, **, and *** are set respectively to the results of evaluating the last, next-to-last, and next-to-next-to-last expressions read.

If evaluation of an expression takes more than **top-level-threshold-time** seconds (including time for garbage collections), then after the evaluation result is printed, a message indicating how long evaluation took and how much of that time was spent garbage collecting is printed. In the message the phrase *compute-time* refers to CPU time not spent garbage collecting, while the phrase *gc-time* refers to CPU time spent running the normal FRANZ LISP garbage collector. **top-level-threshold-time** defaults to 1 second.

6. STRING FILES. SKETCH supports the use of character strings in memory as files. This permits output to be prepared for displays without having to first write the output on disk. It is also used for passing error messages from C to LISP. This facility is implemented by the *stringopen* function described in the glossary.

7. DUMPLISP. The FRANZ LISP *dumplisp* function has been extended so that it will correctly dump a SKETCH evaluator or compiler environment. The resulting file can be executed as a new variant of the *sketch* evaluator or compiler.

When a SKETCH compiler is called without arguments (other than *−E* or *−I*), it will read and evaluate its standard input, just like any LISP environment. Statements in this input may load files and then call *dumplisp*.

8. LOADING C AND FORTRAN FILES. Loading C and FORTRAN *.o* files should be done with the *cload* function, which is described in the glossary. This function allows C and FORTRAN *.o* files to be reloaded into the current LISP environment. It does this by taking as an argument a list of all the global function names and initialized variables in the *.o* file, and removing these from the symbol table before reloading the file.

To fully understand argument passing, it is necessary to read the section on foreign functions in the FRANZ LISP manual chapter on functions. However, the following will suffice for many purposes. Numeric arguments and values will be appropriately passed without problems. Functions with the *c-function* discipline return integers which become LISP *fixnum*'s, and functions with the *double-c-function* discipline return floating point numbers that become LISP *flonum*'s. Lists, symbols, character strings, hunks, and vectors may be passed as arguments, and will be passed as pointers equal to their respective addresses. Except for character strings, these are all structures defined in the SKETCH ATOMS package.

[C functions that return LISP values are currently difficult to write because FRANZ lacks a discipline for them. This should be fixed.]

9. PASSING ERROR MESSAGES FROM C TO LISP. The *sfe_error* function can be called from C to record an error message and set an error switch. This function takes the same arguments as *printf*. The message is written into a string file (see above), and is later read by the LISP *read* function and passed to the LISP *error* function. Thus the message must be a valid representation of a LISP value. An example is-

sfe_error ("(cannot open %s for writing)",
sat_sformat (file_name));

where the *sat_sformat* function from the ATOMS package reformats the character string file_name, if necessary, so that it is a valid LISP symbol (e.g. #play becomes |#play|).

Upon returning to LISP from C the error switch is checked by the *ccheck* function. If set, it is cleared, the error message is read using *read*, and the *error* function is called with the LISP value read. The form for employing *ccheck* is usually

(*ccheck* (_some_C_function ...))

in which *ccheck*, if it finds the error switch off, returns the value of its argument, which is the value returned by _some_C_function.

There are a variety of C utility functions for working with the error handling facility just described. The *sfe_assert* macro makes a test and calls *sfe_error* if the test fails.

E.g.—

sfe_assert (count > 0, "(count argument is <= 0)");

In the failure case, *sfe_assert* also returns from the current function. It does this by executing the *sfe_return* macro, which defaults to *return* (0), but which can be redefined by the programmer if it is necessary for the current C function to clean up on an error return, or if 0 is incompatible with the data type of the value returned by the function. *Sfe_assert* is heavily used to test for errors in C functions.

It is moderately rare for SKETCH C functions to call each other. When they do, the caller may have to check the error switch upon return from the called function. This is done with the *sfe_iserror* macro. The *sfe_check* macro combines this test with an call to *sfe_return* if the error switch is on. E.g.—

```
my_function (...);
sfe_check();
```

10. DEFINING FUNCTIONS AND GLOBAL VARIABLES FOR LINT. It is important to lint C functions to find errors. When doing so, all functions callable in C code outside the file in which they are defined should be given public definitions sufficient to specify the types of their arguments. This is done by including code such as-

```
#if SFE_LINT
        #ifndef PPP_MMMM_C
                /* ARGSUSED */
                some_function (argument_1, ...)
                        type_1 argument_1; ... { returns (0); }
#               endif
#       endif
```

in the *.h* file of the package that defines the function. The statement

#define PPP_MMMM_C

must also be included before any *#include* statements in the file ppp_mmmm.c that gives the normal definition of the function.

This code works as follows. If a file other than ppp_mmmm.c is being linted, *SFE_LINT* will be 1 and PPP_MMMM_C will be undefined. Therefore, the definition of some_function given in the *.h* file will actually be used by *lint*. If ppp_mmmm.c is being linted, this definition will be suppressed by the *#ifndef* PPP_MMMM_C. If a file is being compiled instead of linted, this definition will be suppressed because *SFE_LINT* will be 0.

The function definition accessed by *lint* needs to declare the type of each argument and the type of the value returned. The body of this function definition should consist only of a valid return statement if the function returns a value. If the function returns no value, the body should be empty. The special line-

/ ARGSUSED */*

must be placed before this function definition to keep *lint* from complaining that the arguments are not used in the function body.

A similar thing must be done for global variables. To keep *lint* happy, these must be given an explicit *extern* in the normal part of the *.h* file, and then redefined without the *extern* inside *#if SFE_LINT* and *#ifndef* PPP_MMMM_C.

11. HITLIST

(1) Provide library directories and searching for *autoload*.

Make *cload* handle composite files consisting of many *.o* files linked together. These will load faster in *autoload* situations.

(2) Add general-c-function discipline to return lisp values from C functions.

(3) Speed up pretty-print.

(4) Possibly add argument processing facility.

(5) Possibly add new reader/printer that uses expression syntax and operator hierarchy.

(6) Add abbreviation handler.

(7) Make *error* set *prinlength* and *prinlevel* to reasonable values.

12. GLOSSARY.

(**argv-shift** ['x_number]) [LISP Function]

SIDE EFFECT: Remove x_number arguments from the beginning of the command line arguments returnable by *argv*. Specifically, remove the arguments returned by (*argv* 1) through (*argv* x_number). (*argv* 0) is left untouched. X_number defaults to 1.

(**assert** 'g_condition ['g_message]) [LISP Macro]

SIDE EFFECT: Evaluates g_condition, and if false evaluates (*error* g_message). G_message defaults to '(g_condition is false). Note that g_message is evaluated only if given and g_condition is false.

(**carray** 'a_array) [LISP Macro]

RETURNS: An integer equal to the address of the first data word of the array. This can be passed to a c-function as the address of the beginning of the array.

(**ccheck** 'g_value) [LISP Function]

WHERE: 'g_value is usually a call to a C or FORTRAN function loaded by *cload*: e.g., as in-

(*ccheck* (_sar_copy x y))

RETURNS: G_value.

SIDE EFFECT: A check is made to see if a C function has called *sfe_error* since the last call to *ccheck*. If the answer is yes, *error* is called with the LISP expression read from the character string generated by the call to *sfe_error*.

(**ceiling** n_number) [LISP Function]

RETURNS: The smallest integer greater than or equal to n_number.

(**check-list** 'g_list ['u_predicate]) [LISP Function]

RETURNS: -1 if g_list is not a normal *nil*-terminated list each element of which satisfies u_predicate, if that is given. Otherwise returns the number of elements in g_list (0, 1, ...).

(**cload** '([s_discipline] s_function ...)
's_file) [LISP Function]

(**cload** '([s_discipline] s_function ...)
'(s_file [s_library])) [LISP Function]

WHERE: S_file.o (s_file with the extension *.o* added) is an object file of some foreign language, most likely C but maybe FORTRAN or PASCAL. This file is assumed to contain the functions with global load names s_function .... In order to allow reloading of this file, these global names should be an exhaustive list of all global functions defined in the file. Note that if the names are those of C global functions, they must being with _, as the load names of all global C functions have _ prefixed by the C compiler.

S_discipline is one of the FRANZ function disciplines or the symbol *constant*, which refers to initialized global data. The default is *c-function*, which refers to a C language function that returns an integer. Some other possibilities are *double-c-function* which is a C function returning a real number, *lisp-c-function* which is a C function returning a lisp value, *integer-function* which is a FORTRAN function returning an integer, and *real-function* which is a FORTRAN function returning a real. See the FRANZ LISP documentation on functions for other disciplines and a precise explanation of the calling linkages.

The first argument is a list of s_discipline's and s_function's, with each s_discipline applying to all the functions following it and an implicit *c-function* at the beginning of the list. No function name may be the same as a discipline. The possible disciplines are listed in the global constant ***function-disciplines***.

S_library is passed as a character string to the UNIX loader (ld) as the library to be searched for undefined globals. It may also be a list of more than one library: e.g.

'|-lm -lV foo.a|.

SIDE EFFECT: Loads s_file.o, searching the directories in the list (*status load-search-path*) just as the *load* function does. Defines the function definitions of s_function ... to refer to the entry points of the same names in the files.

Does nothing if the file is already loaded (this *cload* has already been executed, and the file found by searching directories does not have a more recent modification time than the version of the file that was previously loaded.

BUG: If a C or FORTRAN function is referred to by other C or FORTRAN functions, then reloading the first function will leave these other C or FORTRAN functions referring to the old version of the function, and not the newly reloaded version. This can be corrected only by subsequently reloading all the functions that refer to the reloaded function.

NOTE: You should not use initialized global variables in your programs, as it is impossible to reload the files containing them. One can get out of this problem partially by listing the global variable name in the *cload* function call as if it were the name of a function. But then one has the problem that functions loaded before the global variable was reloaded will still refer to the old global variable, and not the newly reloaded one. This is OK only if the global variable is really a constant. Such a variable should be given the discipline *constant*.

"**compiler**" [SKETCH Term]

MEANS: A SKETCH environment built on top of the *liszt* program which compiles SKETCH code, but does not have all the apparatus to evaluate arbitrary SKETCH functions. A compiler is as opposed to an *evaluator*.

computer-format [LISP Global Constant]

VALUE: The type of the computer, from the point of view of the data formats it uses. Thus all DEC vax's have the type *dec*, all Motorola 68000's have the type *motorola*, most all IBM computers have the type *ibm*, and most all INTEL computers have the type *intel*. Note that *ibm* and *motorola* use the same integer formats but different floating point formats. Ditto for *dec* and *intel*. Note that all computers use the same formats for arrays with 1-bit or 8-bit elements (this format is determined by IO devices, and is IBM compatible).

(**copy-list** 'l_list ['x_length 'g_fill]) [LISP Function]

RETURNS: A copy of l_list. Only the top level list cells are copied, unlike the *copy* function (which copies list cells recursively). If the last element of l_list is dotted, so is the last element of the returned value. If x_length is given, the result will have exactly x_length list cells. If l_list is too short for this, cells containing g_fill will be added (and the result will be dotted if l_list is). If l_list is too long, it will be truncated (and will not be dotted even if l_list is).

(**copy-setf-function** 's_symbol 's_source) [LISP Function]

EQUIVALENT TO: (*defsetf* s_symbol ...) where ... was whatever appeared in a previous (*defsetf* s_source ...). If s_source has no current *setf* expansion function, s_symbol will be set to have no *setf* expansion function.

(copy-string 't_string) [LISP Function]

RETURNS: A copy of t_string that does not share memory with t_string or any other string.

(defcache s_function (g_size s_equal s_cache) l_arguments . l_body) [LISP Macro]

WHERE: G_size defaults to 10, s_equal to *eq*, and s_cache to *s_function-*cache**.

SIDE EFFECT: Defines s_function after the manner of *defun* to be a function that looks items up in a cache and maps them onto values. The first argument to s_function is the item to be looked up, and the function returns the value **found**.

The cache is maintained in the global variable s_cache, which is declared after the manner of *defvar*. The size of the cache, the number of items remembered, is g_size. The most recently used g_size item/value pairs are retained in the cache, and the other items are discarded. The function used to test for equality between items is s_equal.

If the item is not found in the cache, its value is computed by the function body, l_body, whose last expression produces the value. The new item/value pair is added to the cache. L_arguments is a normal *defun* argument list for the function, and arguments other than the first may be used by the function body to compute the value.

(defsetf s_function (s_expression s_value) g_statement ...) [LISP Macro]

SIDE EFFECT: Defines a lambda function like *defun* with two arguments named s_expression and s_value, and with a body g_statement .... However, this lambda function is not named s_function, but is rather attached to the property list of s_function in such a way that whenever the *setf* macro is called by an expression of the form—

(*setf* (s_function ...) g_value)

then the *setf* macro will call

(*funcall* <lambda-function> '(s_function ...) 'g_value))

to produce the macro expansion of *setf*. Thus g_statement ... should return the *setf* expansion given that the s_expression argument is bound to (s_function ...) and the s_value argument is bound to g_value.

IMPLEMENTATION: FRANZ actually implements this, but does not document it.

(demo ['s_input-file [*t*] ['s_output-file [*t*]]]) [LISP Function]

SIDE EFFECT: Does a read-eval-print loop reading from s_input-file and printing into s_output-file. All expressions *read* from *piport* are also printed, each followed by an end of line. Atoms read by *ratom* and characters read by *readc* or *tyi* are similarly printed (but not followed by an end of line).

Prompts are printed, and in general the behavior of the standard top level is faithfully simulated with this redirected input and output.

Calls to *break* cause the equivalent of control-D to be typed and the read-eval-print loop to be resumed. Calls to *exit* terminate *demo* only: not the program that called *demo*.

The *t* switch following s_input-file causes the program to wait after printing each prompt for a control-D to be typed on the standard input. If an expression is typed instead, it is evaluated and printed, another prompt is typed, and the program waits again.

The *t* switch following s_output-file causes output to go into both s_output-file and the standard output.

DEFAULTS: Output goes by default to the standard output.

If no arguments are given, those from the last call to demo are used.

NOTE: While *demo* is running, *poport*, *piport*, and *ptport* are changed to input from s_input-file and output to the standard output or s_output-file as appropriate. Thus other functions can *read* and *print*. Also, the *read*, *ratom*, *readc*, and *tyi* functions are modified to print what they read if they read it from *piport*.

NOTE: The *demo* read-eval-print loop is the same as the SKETCH top level read-eval-print loop, operating in a slightly different mode. In particular, functions such as **top-level-prompt**, are used.

BUG: The characters *read* from *piport* are not individually printed, but only the results returned by *read*. Thus comments are lost and new lines are inserted after every expression *read*, even if this is inappropriate. On the other hand, if **top-level-print** is a pretty printer, the print alignment of expressions *read* can be much improved.

Similarly new lines and comments are lost when using *ratom*.

BUG: *Untyi* does not work, and it is suggested that *tyipeek* be used instead.

BUG: *Exec* works and the standard output from the command it executes is captured in s_output-file, but the standard error output from the command is not captured in s_output-file, and goes to the standard error output no matter what.

BUG: If you use the call—

(*demo* s_input-file *t* 's_output-file *t*)

and type an expression in place of ^D, the expression will not be printed in s_output-file, but the value it evaluates to will be.

(dpb 'x_value #oPPSS 'x_number) [LISP Function]

RETURNS: x_number with the field specified by #oPPSS (see *ldb*) replaced by x_value.

(dumplisp s_file) [LISP macro]

SIDE EFFECT: Dumps the current LISP environment into the file named s_file. Either an evaluator or a compiler environment can be dumped. S_file becomes a program that can be invoked as a UNIX command to restart the environment.

(environment ...) [LISP Macro]
(environment-maclisp ...) [LISP Macro]
(environment-lmlisp ...) [LISP Macro]

in-environment [LISP Global Variable]

CHANGES: These now maintain the global variable **in-environment** which is t when *load* is called by a *files* clause in one of the various *environment* statements, and *nil* otherwise.

(equal-filled-lists 'l_list-1 'l_list-2 'g_fill) [LISP Function]

RETURNS: *t* if l_list-1 equals l_list-2, and *nil* otherwise. However, for the purposes of this comparison, the two lists are made of equal length by filling the shorter out with elements equal to g_fill. Neither list may be dotted.

(error 'l_message) [LISP Function]
(error 's/t_message ['g_data_1 ['g_data_2]]) [LISP Function]

EXTENSION: May be called with a single argument which is a list explaining the error. This will be pretty printed. Use of this feature allows complex error explanations without worrying about printed line lengths.

SIDE EFFECT: Signals an exception, as per the chapter on EXCEPTION HANDLING in the Franz Manual. The error type will be 'ER%err, the unique id will be 1, and the error will not be continuable. If an *errset* is active, *nil* will be returned from the *errset* call.

If s/t_message is a string or a symbol, it will become the error message string (it will be made into a symbol for that purpose, if it is a string), and g_data_1 and g_data_2, if present, will become the error data. If L_message is not a string or symbol, the error message will be '|| and l_message will become the error data (in this case g_data_1 and g_data_2 may not be given).

It is expected that when printing an error with error message equal to '||, the first and only error datum will be pretty-print'ed, whereas when printing any other error, the error message and all data will be patom'ed.

BUGS: More than two error data should be allowed.

(error-trace 's_switch) [LISP Function]

SIDE EFFECT: Turns error tracing on if s_switch is non-*nil*, or off if s_switch is *nil*. If on, error tracing causes information to be created in the stack during normal execution that allows a detailed trace to be printed upon an error. Unfortunately, creating this information is costly: LISP bound programs typically run 2.5 times slower with error tracing on than with error tracing off. The default is for error tracing to be on.

IMPLEMENTATION: Currently error tracing is implemented by (**rset t*) and (*sstatus translink nil*).

"evaluator" [SKETCH Term]

MEANS: A SKETCH environment built on top of the *lisp* program which can evaluate any SKETCH function call, but does not compile SKETCH code. An evaluator is as opposed to a *compiler*.

exit-on-error [LISP Global Variable]

VALUE: If non-*nil* causes any error (routed trough *ER%tpl*) to exit from the current program using the value of **exit-on-error** as the exit code. The default value of **exit-on-error** is *nil*, and the recommended non-*nil* value is 2.

(fdelay 'f_time) [LISP Function]

SIDE EFFECT: Delay until f_time, which is measured in seconds since 00:00:00 GMT, Jan 1, 1970. F_time may have the same resolution as the value returned by *ftime*. If the delay is over a second, the CPU will be given up to other users during most of the delay.

(filestat-atime ...) [LISP Function]
(filestat-ctime ...) [LISP Function]
(filestat-dev ...) [LISP Function]
(filestat-gid ...) [LISP Function]
(filestat-ino ...) [LISP Function]
(filestat-mode ...) [LISP Function]
(filestat-mtime ...) [LISP Function]
(filestat-nlink ...) [LISP Function]
(filestat-rdev ...) [LISP Function]
(filestat-size ...) [LISP Function]
(filestat-type ...) [LISP Function]
(filestat-uid ...) [LISP Function]

USE: Use these function names instead of *filestat:atime* etc. so that code will work in those versions of SKETCH based on Franz LISP with packages (Opus 42 and later), as well at those without packages (Opus 38).

EQUIVALENT TO: The Franz Opus 38 functions *filestat:atime*, etc.

float-format [LISP Global Variable]

VALUE: The C language *printf* format used by *patom* etc. to print *flonum*'s. Default value is "%.16g" in normal FRANZ LISP, but is "%.6g" in SKETCH.

(**floor** n_number) [LISP Function]

RETURNS: The largest integer less than or equal to n_number.

(**ftime**) [LISP Function]

RETURNS: The time in seconds since 00:00:00: GMT. Jan. 1, 1970, as a *flonum* with an accuracy of at least 1/60 second.

gc-history [LISP Global Variable]
gc-history-length [LISP Global Variable]
gc-count [LISP Global Variable]
gc-errors [LISP Global Variable]
gc-dumpfile [LISP Global Variable]

VALUE: **gc-history** is a list of messages summarizing the first **gc-history-length** garbage collections since the process started. The default value of **gc-history-length** is 20.

gc-count is the number of garbage collections since the current process started.

gc-errors is the number of garbage collector errors that have occurred since the beginning of time (the count is not zeroed by *dumplisp* and reload). After the first garbage collection that has errors, a *dumplisp* is done to the file named **gc-dumpfile** (before **gc-history** is updated). The default value of **gc-dumpfile** is *'gc-error-dump*.

NOTE: The **gc-history** messages are of two types: *compute* messages specify am amount of CPU time used by non-garbage collection computation between two garbage collections; and *gc* messages specify an amount of CPU time used by one garbage collection. The messages are in the order that the actions occur, but **gc-history** may be slightly delayed relative to the current state of the process.

The *gc* messages list the number of pages allocated to each of several types of data: e.g. *fixnum*'s. One of these numbers may be of the form N + M: the N is the number of pages that was allocated before garbage collection, and the M is the number of fresh pages allocated by the garbage collector to try to avoid another collection for a while. The data type for which the number of pages has this N + M form is the data type whose exhaustion caused the garbage collection.

(gentemp) [LISP Function]

USE: Use *gentemp* instead of *gensym* in macros, because the latter will foul up when macros are expanded for execution at the top level during load in versions of Franz that have packages.

The problem is that *gensym* produces macros such as—

((*lambda* #:g00052) (*sum* #:g00052 5))

where the two unintered generated symbols are not *eq* when read.

EQUIVALENT TO: (*intern* (*gensym* '*T*)). The equivalence is exact in SKETCH based on Franz Opus 38, and near in SKETCH based on Franz Opus 42 and later Franz's. In the later case see the Franz documentation for the exact definition.

(**has-setf-function** 's_symbol) [LISP Function]

RETURNS: Non-*nil* if s_symbol has a *setf* expansion function defined for it by *defsetf*.

NOTE: The *setf* expansion functions of some symbols are defined only when they are needed. E.g., *caaaar* may be so handled.

is-compiler [LISP Global Variable]

VALUE: Non-*nil* if the current environment is a compiler, and *nil* if the current environment is an evaluator.

(**ldb** #oPPSS 'x_number) [LISP Function]

RETURNS: The bit field obtained by right shifting x_number by PP bits and masking off the low order SS bits. PP and SS are octal numbers.

(**list-depth** 'g_value) [LISP Function]

RETURNS: The depth of the list nesting in g_value. Atoms (and hunks) including *nil* have depth 0. Dotted lists are handled.

pi [LISP Global Constant]
sqrt-pi [LISP Global Constant]

VALUE: The indicated constant floating point number. *Sqrt-pi* is the square root of *pi*.

(port-string 'p_port) [LISP Function]

RETURNS: The string associated with p_port by *stringopen* if p_port was created by *stringopen*, or *nil* if p_port was not created by *stringopen* and therefore has no associated string. Note that for the string returned to be valid, a NUL must have been written at its end.

(pretty-format 'g_value ['x_level]) [LISP Macro]
pretty-format-hook [LISP Global Variable]
(*setf* (*get* 's_symbol **'pretty-format)** [LISP Property]
'(*character* s_prefix x_prefix-size))
(*setf* (*get* 's_symbol **'pretty-format)** [LISP Property]
'(*breaks* s_break x_count [*] ...))
prinlevel [LISP Global Variable]
prinlength [LISP Global Variable]

RETURNS: The pretty-print format of g_value. This format contains a specification of how to print g_value that is more precise than g_value itself.

X_level is the number of parentheses that will finally surround the pretty printed version of g_value. If it is equal to or greater than *prinlevel*, the elements of a composite g_value should not be printed. At most *prinlength* elements should be printed in any case.

FORMAT SYNTAX: A pretty print format may be a list or an atom. If it is an atom, it is to be *print*'ed as is. If it is a list, then it contains a list of items which are themselves pretty-print formats, plus other information that controls the insertion of carriage returns during the printing process.

The syntax of the *reverse* of a pretty print format list is—

([x_prefix-size s_prefix] [s_break g_item] ... [x_postfix-size s_postfix])

Note that for efficiency reasons the actual pretty print format is the *reverse* of this list.

The s_prefix and s_postfix are symbols that are *patom*'ed before and after the g_item's. For efficiency x_prefix-size and x_postfix-size are also given: these are just the number of characters that will be printed by *patom*'ing s_prefix and s_postfix, respectively. Either the prefix information or the postfix information may be omitted if there is no prefix or posfix.

If possible the prefix, g_items, and s_postfix will all be printed on one line, with a single space separating each pair of g_items, but no space after the prefix or before the postfix.

If everything will not fit on one line, there are several cases. In describing these we will refer to the 'g_item + string following an s_break'. This is the g_item following the s_break, plus any subsequent pairs of the form '+ g_item-2' following that. That is, the longest following list of g_item's separated by s_break's that are +, including these s_breaks.

In the simplest case, the first s_break is / or //. The item margin is then set to the first column of the printed prefix, plus 3 columns. A // s_break will always return to the item margin. A / s_break will return to the item margin if necessary to avoid line overflow while printing the following g_item + string. A + s_break will return to the item margin plus 3 columns if necessary to avoid such line overflow.

In the more complex case, the first s_break is +. The first item will then be printed immediately after the prefix, and the item margin will be set to one column after the end of the first item's printout. After this a + s_break will return to the item margin if necessary to avoid line overflow while printing the following g_item + string. When the first s_break that is not a + is encountered, a decision will be made about reseting the item margin. If necessary to conserve horizontal space, the item margin will be reset to the starting column of the prefix plus 3 columns, and a carriage return will be inserted. After this point, whether the line margin is reset or not, printing precedes as in the simpler case above.

PRETTY-FORMAT-HOOK: *Pretty-format* is a macro that works by handling symbols, numbers, and strings inline, and executing—

(*funcall *pretty-format-hook* g_value x_level)*

to handle anything else. A series of functions are be written in the form—

```
(defvar *my-format-hook* (prog1 *pretty-format-hook*
                             (setq *pretty-format-hook*
                                   'my-format-hook)))

(defun my-format-hook (the-argument the-level)
   (or (progn ...)
       (funcall *my-format-hook*
              the-argument the-level)))
```

Each of these functions processes the-argument if it is able to (in the *progn* block), and returns a non-*nil* format. Or the function returns *nil* if it is not able to process the-argument. The last function defined, the one named in **pretty-format-hook**, has the first crack at the-argument.

PRETTY-FORMAT PROPERTY: If a symbol has a *pretty-format* property on its property list, the default **pretty-format-hook** routine will take special action. This property may have one of two forms.

If it has the form—

(*character* s_prefix x_prefix-size)

then the symbol is like the *quote* function, a function of one argument with a special printed representation consisting of s_prefix followed by the argument. Thus *quote* has the *pretty-format* property—

(*character* |'| 1)

X_prefix-size is just the *print-size* of s_prefix, and is included to improve efficiency by eliminating the recomputation of this size every time it is needed.

If the *pretty-format* property has the form—

(*breaks* s_break-1 x_count-1 s_break-2 x_count-2 ... [*] ...)

then the symbol is treated as a function of many arguments, like *setq* or *do*. In the format of a list beginning with the symbol, the first x_count-1 s_break's will equal s_break-1, the next x_count-2 s_break's will equal s_break-2, etc. A star (*) before an s_break indicates that when the *breaks* list is exhausted, it is to repeat beginning with the s_break just after the star. For example, the *pretty-format* property for *setq* is—

(*breaks* + 1 * / 1 + 1)

and for *do* is—

(*breaks* + 3 * // 1)

BUG: Does not handle strings or symbols containing embedded line feeds correctly.

(pretty-print 'g_value ['p_port ['x_margin ['s_string ['x_repeat ['x_right_margin]]]]]) [LISP Function]

line-length [LISP Global Variable]

EQUIVALENT TO: (print 'g_value ['p_port]) but uses the line length in the global variable **line-length**, and uses indentation. Each line begins with x_repeat s_strings followed by space till the column equals x_margin. At least 'x_right-margin spaces must be left at the end of the last line (into which to put left parentheses for lists containing 'g_value).

DEFAULTS: The default p_port is *poport*, the default x_margin is the current column as found by *nwritn*, the default s_string is '||, the default x_repeat is 1, and the default x_right-margin is 0.

Line-length defaults to 76, which allows for various things like *diff*(1) listings using the first few columns of a line, and terminals or editors using or abusing the last column.

NOTE: If the line is not long enough to hold a sensible representation of some part of g_value, the indent may be moved back to the beginning of the line. The lines where the indent has been moved back are bracketed by comment lines in the form—

; <<<N
; N>>>

where N is the current depth of parentheses.

RETURNS: Number of carriage returns printed.

BUG: Does not handle strings or symbols containing embedded line feeds correctly.

(pretty-print-format 'g_format ['p_port ['x_margin ['s_string ['x_repeat ['x_right_margin]]]]]) [LISP Function]

EQUIVALENT TO: *Pretty-print* but takes as input the result g_format of applying *pretty-format* to the item to be *pretty-print*'ed.

WARNING: If g_format is a list, it is destroyed.

(pretty-tab 'x_margin ['p_port ['s_string ['x_repeat]]]) [LISP Function]

SIDE EFFECT: Spaces until the current column equals x_margin. If the current column is initially > x_margin, a *terpri* is done first. Whenever spacing is begun at the beginning of a line, x_repeat s_string's are printed before spacing is done. The default p_port is *poport*, the default s_string is '||, and the default x_repeat is 1.

RETURNS: Number of carriage returns printed.

(print-size 'g_value ['x_maximum]) [LISP Function]

RETURNS: The number of characters needed to *print* g_value, or x_maximum, whichever is smaller.

BUG: A line feed in a string value counts as one character.

ptime-counts-per-second [LISP Global Constant]

VALUE: The number of ticks per second of the *ptime* function clocks.

(puresegment 's_type 'x_size) [LISP Function]

EQUIVALENT TO: (*segment* 's_type 'x_size) but the resulting segment is allocated to pure memory and is never garbage collected.

(round n_number) [LISP Function]

RETURNS: The nearest integer to n_number.

(search-path '(s_directory ...) 's_file ['s_mode]) [LISP Function]

WHERE: 'S_mode is either '*r* for "read", '*w* for "write", or '*a* for "append", and defaults to '*r*.

RETURNS: An interned symbol naming a file which has a name of the form s_directory/s_file. For the *r* s_mode, s_directory is the first symbol in the list '(s_directory ...) for which the file s_directory/s_file exists, but *nil* is returned if no such file exists. For the *w* s_mode, s_directory is just the first symbol on the list '(s_directory ...), for which the directory s_directory/s_subdirectory exists, where s_subdirectory is the directory part of s_file (see *split-filename*). For the *a* s_mode, s_directory is as for the *r* s_mode if some file exists, and otherwise as for the *w* s_mode.

No check is made for file readability, writability, or creatability: just existence.

For s_directory equal to the symbol |.|, the name s_file is used in place of ./s_file. If s_file begins with a slash (/) or tilde (~), no search is done, but the existence of the file or subdirectory is checked for, and *nil* returned if the file or subdirectory does not exist.

sfe_assert (g_test, t_message) [C Macro]
sfe_assert1 (g_test, t_message, g_argument_1) [C Macro]
sfe_assert2 (g_test, t_message, g_argument_1, g_argument_2) [C Macro]
sfe_assert3 (g_test, t_message, g_argument_1, ..., g_argument_3) [C Macro]
sfe_assert4 (g_test, t_message, g_argument_1, ..., g_argument_4) [C Macro]
sfe_assert5 (g_test, t_message, g_argument_1, ..., g_argument_5) [C Macro]

SIDE EFFECT: Evaluate g_test and if false (zero) call

sfe_error (t_message, g_argument_1, ...)

and then execute *sfe_return*, which is a macro that defaults to *return* (0).

The digit at the end of *sfe_assert* counts the number of g_argument's. The C macro preprocessor will complain if it is wrong.

sfe_check () [C Macro]

SIDE EFFECT: If the error flag set by *sfe_error* is on, executes the *sfe_return* macro, which by default does a *return* (0).

sfe_error (t_format, g_argument, ...) [C Function]

SIDE EFFECT: An error flag is set, and the arguments cause an error message string to be produced after the manner of *sprintf*.

When control returns to LISP, the *ccheck* function will see the error flag, reset it, read the error message string using *read*, and send the value read to *error*.

The error flag can also be tested using the C macros *sfe_check* and *sfe_iserror*.

The error message must be a *read*able LISP value.

sfe_iserror () [C Macro]

RETURNS: True if the error flag set by *sfe_error* is on. False otherwise.

SFE_LINT [C Macro]

VALUE: 1 if lint is running, 0 if not.

sfe_return [C Macro]

DEFAULT: *return* (0)

USE: Executed by *sfe_assert* and *sfe_check* in order to return from the current function upon detecting an error. Can be changed if 0 is not acceptable as a return value.

SFE_VAX [C Macro]
SFE_68XXX [C Macro]
SFE_BSD [C Macro]
SFE_SUN [C Macro]
SFE_FRANZ [C Macro]
SFE_SKETCH [C Macro]

VALUE: If a SKETCH is running on a DEC VAX processor, *SFE_VAX* is non-zero and equals the type number of the processor: e.g. 780, 785, etc. If a SKETCH is running on a MOTOROLA 68000 processor, *SFE_68XXX* is non-zero and equals the type number of the processor: e.g. 68010, 68020, etc.

If a SKETCH is running under a Berkeley Software Distribution version of UNIX, *SFE_BSD* is non-zero and equals the version number of the distribution. If a SKETCH is running under a SUN Microsystems version of UNIX, *SFE_SUN* is non-zero and equals the version number of the distribution.

If a SKETCH is using Franz LISP, *SFE_FRANZ* is non-zero and equals the version number of the Franz LISP being used.

SFE_SKETCH is always non-zero and equals the version number of SKETCH that is being used.

Software version numbers are written as a version number followed by six digits: the first 3 for the minor version number, and the next three for the micro version number. E.g., 4.3 is written 4003000, while 42.16.1 is written 42016001. Where versions are denoted as 'a', 'b', 'c', etc., these are represented by 1, 2, 3.

The macros just described are always defined: they equal zero if their hardware or software is not being used.

sketch-version [LISP Global Constant]
franz-version [LISP Global Constant]

VALUE: These are the version numbers of the SKETCH and FRANZ LISP that are being used. They are floating point numbers, generally, with the major version number as the integer part, and each minor verson number as three decimal places. Thus FRANZ version 42.16.1 is represented by 42.016001. Minor versions 'a', 'b', 'c', etc. are represented by 1, 2, 3, ... (and given 3 decimal places). Thus SKETCH version 4b is represented as 4.002.

(**split-filename** 's_file) [LISP Function]

RETURNS: The pair (s_directory s_basename) such that s_file is equivalent to—

s_directory/s_basename

If there is no slash / in s_file, s_directory equals '|.|. Slashes are removed from the end of s_directory. Thus an s_directory value of '|| means the root directory.

(**stringopen** 't_string 'x_size 's/t_mode ['t_name]) [LISP Function]

WHERE: X_size is the number of bytes in t_string. Currently the best way to get such a string is to call (*puresegment 'string* x_pages) where x_size = 512 * x_pages. T_name is the name of the port, and defaults to "stringfile".

RETURNS: If s/t_mode is 'r (or "r"), returns a port which when read from will read the string. An end of file will occur after the x_size'th byte. An end of file will not occur before a NUL byte: the NUL byte will be read.

If s/t_mode is 'rs (or "rs"), behaves as for 'r, but an end of file will occur just before the first NUL byte in the string, if there is one, or after the x_size'th byte, if there is no NUL.

If s/t_mode is 'w (or "w"), returns a port which when written will write into the string. An end of volume is returned when trying to write beyond the x_size'th byte.

If s/t_mode is 'a (or "a"), behaves just as for 'w, but sets the initial position of the port to the first NUL byte in the t_string, or just after the end of t_string if there are no NUL bytes.

If s/t_mode is 'ws or 'as (or "ws" or "as") behaves like 'w or 'a, but writes NUL's into the part of the string to be written, and arranges for an end of volume just before the last byte of the string (which is NUL). Ensures there is a NUL in the string. If 'as is used with a string that has no NUL, the last byte of the string is set to NUL.

The *close, drain, terpri, fseek,* and *nwritn* functions can be used on these ports in the normal way.

The string may be used as a normal string after the port is closed, as long as the string is NUL terminated.

NOTE: *Fseek* can be use to determine the location of the current position in one of these ports relative to the beginning of the string, and to reset that position.

Nwritn can be used to determine the number of characters between the current position and any previous line feed in the string, or the beginning of the string if there is no previous line feed.

top-level-init [LISP Global Variable]
top-level-init-started [LISP Global Variable]
top-level-exit [LISP Global Variable]
top-level-prompt [LISP Global Variable]
top-level-read [LISP Global Variable]
top-level-eval [LISP Global Variable]
top-level-print [LISP Global Variable]
top-level-times [LISP Global Variable]
top-level-print-times [LISP Global Variable]
top-level-init-times [LISP Global Variable]
top-level-saved-times [LISP Global Variable]
top-level-saved-print-times [LISP Global Variable]
top-level-threshold-time [LISP Global Variable]
+ [LISP Global Variable]
++ [LISP Global Variable]
+++ [LISP Global Variable]
***** [LISP Global Variable]
****** [LISP Global Variable]
******* [LISP Global Variable]

VALUE: The top level executes—

```
(setq *top-level-init-started* t)

(funcall *top-level-init*)
(find and load the (status top-level-rc-files))
(remove and process any -I and -E options at the
        the beginning of the argument list)

(setq *top-level-saved-times* *top-level-times*)
(setq *top-level-saved-print-times* *top-level-print-times*)

(setq *top-level-init-times* (funcall *top-level-saved-times*))

(if (and *is-compiler* (there are more arguments))
        (process arguments as for liszt compiler and exit))
```

just before printing the first prompt after loading. It never executes this again during the current process, but will re-execute it when the process is first dumped by ***dumplisp*** and then the resulting file is *exec*'ed.

top-level-init-started is *nil* from the beginning of loading until the execution of the above.

The top level then begins the read-eval-print loop, which is roughly-

```
(let ((prompt *top-level-prompt*)
      (read *top-level-read*)
      (times *top-level-times*)
      (eval *top-level-eval*)
      (print *top-level-print*)
      (print-times *top-level-print-times*)
      expression value pre-eval-times post-eval-times)
     (funcall prompt p_port)
     (setq expression (funcall read p_port g_eof-value))
     ... check for read errors and end of file ...
     (setq pre-eval-times (funcall times))
     (setq value (funcall eval expression)
     (setq post-eval-times (funcall times))
     (setq +++ ++
           ++ +
           + expression)
     (setq *** **
           ** *
           * value)
     (funcall print value p_port)
     (funcall print-times post-eval-times pre-eval-times p_port)))
```

Here the various functions used are saved at the beginning of the read-eval-print loop iteration, so any changes made to them will not become effective till the next iteration. Changes to these functions should be synchronized by including them in a single *progn* which is read all at once by the reader.

Any call to the *exit* function results in executing

```
(funcall *top-level-exit*)
(funcall *top-level-saved-print-times*
        (funcall *top-level-saved-times*)
        *top-level-init-times* poport)
```

before the normal exit actions are taken. Here again the values of **top-level-times** and **top-level-print-times** have been saved when **top-level-init-times** was set, to avoid incompatibilities when these variables are reset.

BREAK AND TRACE: *Break* and *trace* also use—

top-level-prompt
top-level-read
top-level-eval
top-level-print

In addition the top level values of the global variables **line-length**, *prinlength*, and *prinlevel* are temporarily restored every time **top-level-print** is called.

DEFAULT VALUES:

top-level-init	A no-operation function.
top-level-times	A function that retuns— '(,@(*ptime*) ,(number gc's)).
top-level-print-times	A function that prints execution times if the total time is larger than **top-level-threshold-time** seconds.
top-level-threshold-time	1.0
top-level-prompt	A function that prints "—> ".
top-level-read	*read*
top-level-eval	*eval*
top-level-print	*pretty-print*

(sstatus **top-level-rc-files** (s_rc-file ...)) [LISP Function]
(status **top-level-rc-files**) [LISP Function]

SIDE EFFECT: Whenever a LISP environment is loaded by UNIX, then after the function specified by **top-level-init** is called, the list of files s_rc-file ... is examined to find the first file that exists, and that file is loaded. If none of the files exists, no action is taken. The *load-search-path* is *not* used to locate the files.

DEFAULT VALUE: In *sketch* the default value is—

(sketch.rc ../sketch.rc ../../sketch.rc ~/sketch.rc)

and in the compiler, *sketchcom*, the default is (*sketchcom.rc ../sketchcom.rc ../../sketchcom.rc ~/sketchcom.rc*).

(sstatus **top-level-switches** (s_switch ...)) [LISP Function]
(status **top-level-switches**) [LISP Function]

SIDE EFFECT: Whenever the argument list to the current LISP environment begins with one of the switches in the s_switch list, that switch and any following arguments it requires are processed after the **top-level-init** function is called and any *top-level-rc-files* file loaded. To be processed the switch must be one of the following-

-I file-name	The file is *load*'ed, using *load-search-path* to find the file.
-E "expression"	The expression is read and evaluated.

Arguments processed by this mechanism are removed from the list of UNIX command arguments by using the *argv-shift* function. Errors encountered while processing these arguments will cause the program to exit with an exit code of 2.

DEFAULT VALUE:

(-I -E).

(**use-ptport** 'p_port) [LISP Function]

USE ONLY WHEN: Writing C or FORTRAN functions that do their own printing. Helps obey the conventions involving *ptport* (which is used by *demo*).

RETURNS: *t* if output sent to p_port is also to be sent to *ptport* according to the standard conventions for using *ptport*.

EXAMPLE:

```
(ccheck (_xxx_print port ...))
(if (use-ptport port) (ccheck (_xxx_print ptport ...)))
```

(**vrefi-double** 'V_vector 'x_index) [LISP Special Function]

EQUIVALENT TO: *Vrefi-long* but for flonum's, i.e. double precision floating point numbers. Can be *setf*. A flonum index is in 8 byte units (0 is the beginning of V_vector).

(**vsize-long** V_vector) [LISP Function]
(**vsize-double** V_vector) [LISP Function]

RETURNS: The number of *a-long*'s or *a-double*'s in a vector.

(**xtime** 'g_expression) [LISP Function]

RETURNS: The time in seconds taken on the average to execute g_expression. To compute this, g_expression is executed many times, until several seconds have passed. The time reported is the average time in seconds of one execution, excluding time taken by the garbage collector.

CHAPTER 4

ATOMS

1. INTRODUCTION. A SKETCH atom is a number, pointer, or small structure that is passed as an argument or return value by copying the atom itself, rather than by copying a pointer to the atom. (It is not to be confused with a LISP atom, which is any object that is not a non-empty list or hunk, and is an anachronism to be avoided.)

The atoms required by SKETCH are numbers and pointers to LISP values. Most of this ATOMS package is concerned with interfacing these atoms to the C language.

2. NUMERIC ATOMS. The numeric atom types supported by SKETCH are listed in Table 4.1. All the types but *a-ubit* and *an-lbit* have C type names: e.g. *char*. The C types *uchar*, *ushort*, and *ulong* are standard SKETCH-defined abbreviations for *unsigned char*, *unsigned short*, and *unsigned long*, respectively.

All the types except *int* and *unsigned* have associated SKETCH types (see the SKETCH OBJECTS package chapter), such as *a-float*, and can therefore be designated as the element types of SKETCH arrays (see the SKETCH ARRAYS Package).

All the types store numbers, except the *an-lbit* type, which stores *nil* or *t*, represented in C as an off bit or an on bit respectively.

Several of the types can be used for arguments to C functions. These have an associated Argument Prefix: e.g. *f_* for *double* precision floating point. The *x_* prefix for *int*'s and the *f_* prefix for *double*'s were chosen to be the same as the prefixes for the corresponding LISP argument types, *fixnum* and *flonum*.

Most of these types have C macros or global variables equal to the minimum and maximum numeric values storable in variables of the type (of course 0 is the minimum value for unsigned integers). For example,

if (x > *SAT_CMAXIMUM*) x = *SAT_CMAXIMUM;*
else if (x < SAT_CMINIMUM) x = SAT_CMINIMUM;

clips a value stored in a variable x if that value is outside the range of a *char* variable.

The signed number types have a special value which denotes a missing value. E.g. *SAT_CMISSING* is stored in a *char* variable to denote a missing value. In LISP missing values are denoted by *nil*. Thus if the value of a SKETCH array element with type *a-char* is returned to LISP, the missing value will be returned as *nil*.

However, C routines that return integers or floating point numbers to LISP cannot return *nil*, which is not a number. They must return either a *fixnum* or *flonum*, and use the values stored in the LISP Global Constants *_SAT_IMISSING* or *_SAT_DMISSING*, respectively, to denote the missing value. These LISP constants are equal numerically to the corresponding C constants *SAT_IMISSING* and *SAT_DMISSING*.

TABLE 4.1

SKETCH NUMERIC ATOM TYPES

C Type SKETCH Type	Arg. Prefix	Meaning	Size In Bits	C Minimum C Maximum	Value LISP Missing
— an-lbit		1-bit true/false logical bit	1		
— a-ubit		1-bit unsigned integer	1	0 1	
uchar a-uchar		8-bit unsigned integer	8	0 *SAT_UCMAXIMUM*	
char a-char		8-bit signed integer	8	*SAT_CMINIMUM* *SAT_CMAXIMUM*	*SAT_CMISSING* —
ushort a-ushort		16-bit unsigned integer	16	0 *SAT_USMAXIMUM*	
short a-short		16-bit signed integer	16	*SAT_SMINIMUM* *SAT_SMAXIMUM*	*SAT_SMISSING* —
ulong a-ulong	ulx_	32-bit unsigned integer	32	0 *SAT_ULMAXIMUM*	
long a-long	lx_	32-bit signed integer	32	*SAT_LMINIMUM* *SAT_LMAXIMUM*	*SAT_LMISSING* —
unsigned —	ux_	32-bit unsigned integer	32	0 *SAT_UMAXIMUM*	
int —	x_	32-bit signed integer	32	*SAT_IMINIMUM* *SAT_IMAXIMUM*	*SAT_IMISSING* *_SAT_IMISSING*
float a-float		32-bit floating point number	32	*SAT_FMINIMUM* *SAT_FMAXIMUM*	*SAT_FMISSING* —
double a-double	f_	64-bit floating point number	64	*SAT_DMINIMUM* *SAT_DMAXIMUM*	*SAT_DMISSING* *_SAT_DMISSING*

3. NUMERIC FUNCTIONS. The ATOMS Package contains some miscellaneous C functions and macros for handling numbers, such as *sat_round* for converting a floating point number to a *long* integer after scaling and rounding. These are listed with a brief explanation in Table 4.2. See the glossary for details.

4. FOREVER IN C. The macro *forever* is defined to be '*for (;;)*' as an aid to writing more readable code.

TABLE 4.2 C LANGUAGE NUMERIC FUNCTIONS AND MACROS	
sat_ceiling (f_number, x_exp)	The smallest *long* integer that is not less than f_number times 2^{x_exp}.
sat_floor (f_number, x_exp)	The largest *long* integer that is not greater than f_number times 2^{x_exp}.
sat_log (f_number)	The logarithm base 2 of the smallest power of 2 greater than or equal to the absolute value of f_number, or *SAT_IMISSING* if f_number is 0.
sat_mad (lx_m1, lx_m2, lx_a, lx_d)	(lx_m1*lx_m2+lx_a)/lx_d. The product and sum are stored internally as 64-bit integers.
sat_mas (lx_m1, lx_m2, ux_a0, x_a1, x_shift)	(lx_m1*lx_m2+2^{32}*x_a1+ux_a0) << x_shift. The product and sum are stored internally as 64-bit integers. If x_shift < 0, then the shift is a right shift by −x_shift.
sat_rmas (lx_m1, lx_m2) **sat_rset** (x_shift) **sat_rdeclare**	Some macros that permit a set of *sat_mas* operations to be done with a common x_shift value, and with common ux_a0 and x_a1 values which round the shifted result if x_shift < 0.
sat_round (f_number, x_exp)	Converts f_number to a *long* integer by multiplying it by 2^{x_exp} and then rounding.

5. YES, NO, AND EXCEPTION IN C. To represent the concepts of yes, no, and there-was-an-exceptional-case, the following macros have been defined as aids to writing readable code—

SAT_NO (comment)	0
SAT_YES (comment)	1
SAT_EXCEPTION (comment)	−1

The comment can be any legal C macro argument (it must not contain commas).

Some examples—

return (SAT_YES (everything was done OK));

return (SAT_NO (we could not find the body));

return (SAT_EXCEPTION (the number is too big to compute with));

6. LEFT-TO-RIGHT AND RIGHT-TO-LEFT COMPUTERS. A left-to-right computer stores integers with the highest order byte at the lowest address. Thus when bytes are printed left to right in order of increasing address, the highest order byte is printed leftmost, as people are accustomed to seeing it.

A right-to-left computer stores integers with the lowest order byte at the lowest address. Thus when bytes are printed right to left in order of increasing address, the highest order byte is printed leftmost, as people are accustomed to seeing it.

IBM, Motorola, and related computers are generally left-to-right. DEC and INTEL computers are generally right-to-left.

No matter what type the computer is, bit arrays are stored as if the computer was left-to-right. This is because I/O devices, such as frame buffers, standardly use this method of storage.

In C the macro *SAT_LEFT_TO_RIGHT* is defined as 1 if the computer is a left-to-right computer, and 0 if it is right-to-left. The LISP global variable ***left-to-right*** is similarly defined. Lastly, there is a utility function, *integer-to-bytes*, to convert an integer into a list of bytes according to the machine type.

7. LISP VALUES IN C. LISP values are designated in SKETCH as having the ***sat_lvalue*** C type and the *g_* argument prefix. All LISP values are pointers to LISP objects: even ***fixnum***'s are represented by a pointer to an integer stored in garbage collectible memory. A LISP value (i.e. the object it points at) can be of many different subtypes: e.g. ***list***'s, ***fixnum***'s, ***symbol***'s, etc. Table 4.3 lists these subtypes, the C names for the elements of the subtypes, and the C function usable to create a new LISP object of the given subtype.

Each subtype is known in three different ways: to LISP it is known by a symbol returned by the LISP ***typep*** function; to SKETCH it is known by ***a-type*** value returned by the ***has-type*** macro (see the SKETCH OBJECTS package); and to C it is known by the ***sob_type*** value returned by the ***sob_ltype*** macro (also see the SKETCH OBJECTS package).

Whenever C code allocates a new LISP object, the garbage collector may be called, and will destroy any previous LISP objects that is not referencible by starting from the global variables or the LISP local variables on the LISP (not C) stack. If C code allocates two LISP objects before returning to LISP, it must store the LISP value pointing to the first object in some place where it will be referencible. A good place is inside some other LISP object that is referencible, such as one passed as an argument to the C function by LISP code, or a global variable. C local and global variables are not referencible.

8. STRINGS. LISP values that are strings are in fact C ***char **** pointers to NUL terminated C strings. The argument prefix *t_* is used for both LISP and C string arguments.

TABLE 4.3: PART I LISP VALUES		
typep Type *has-type* Type *sob_ltype* Type	**Expression**	**Meaning**
fixnum **a-fixnum** SOB_FIXNUM	g_value->*sat_lint*	The *int* value of a *fixnum*.
	sat_nfixnum (x_n)	Creates a new *fixnum* = x_n.
	sat_nsfixnum (x_n)	Ditto but requires that $-128 \leq$ x_n < 255 and is more efficient.
flonum **a-flonum** SOB_FLONUM	g_value->*sat_ldouble*	The *double* value of a *flonum*.
	sat_nflonum (f_n)	Creates a new *flonum* = f_n.
string **a-string** SOB_STRING	& g_value->*sat_lchar*	The *char* * value of a *string*. The string is a normal NUL terminated C string.
list **a-list** SOB_LIST	g_value->*sat_lfirst*	The *first* element of a list (i.e. the *car*).
	g_value->*sat_lrest*	The *rest1* of a list (i.e. the *cdr*).
	sat_nlist (g_first, g_rest)	Creates a new list with given first element and rest of list.
	sat_nil	The *nil* value.
hunk0 **hunk1** **hunk2** **hunk3** **hunk4** **hunk5** **hunk6** **a-hunk** SOB_HUNK	g_value-> *sat_hvalue*[x_index]	The x_index+1st element of the hunk: i.e. the (*cxr* x_index g_value) value.
	g_value->*sat_lfirst*	The *first* element of the hunk: i.e. the *car* of the hunk.
	g_value->*sat_lrest*	The *rest1* element of the hunk: i.e. the *cdr* of the hunk.
	sat_nhunk (x_size)	Creates a new hunk with x_size elements all set to *nil*.
	sat_empty	If a hunk is supposed to have x_size elements, it actually has more elements if x_size is not a power of 2, and the extra elements are set to *sat_empty*.

TABLE 4.3: PART II LISP VALUES		
typep Type *has-type* Type *sob_ltype* Type	**Expression**	**Meaning**
symbol a-symbol SOB_SYMBOL	g_value->*sat_svalue*	The value element of the symbol g_value.
	g_value->*sat_splist*	The property list element (head) of the symbol g_value.
	g_value->*sat_sfunction*	The function definition element of the symbol g_value.
	g_value->*sat_slink*	The hash table link element of the symbol g_value.
	g_value->*sat_spname*	The print name element of the symbol g_value.
	sat_nsymbol (t_string)	Returns the existing symbol with the print name t_string, if one exists and is in the hash table. Otherwise creates a new symbol with print name t_string and puts it in the hash table.
	sat_nil	The symbol *nil*.
	sat_t	The symbol *t*.
	sat_cnil	The value stored in the value element of an unbound symbol.

TABLE 4.3: PART III

LISP VALUES

typep Type *has-type* Type *sob_ltype* Type	**Expression**	**Meaning**
vectori an-immediate-vector SOB_IVECTOR	g_value->*sat_vchar*[x_index]	The x_index+1st *char* in an immediate vector.
	g_value->*sat_vuchar*[x_index]	Ditto for *uchar*.
	g_value->*sat_vshort*[x_index]	Ditto for *short*.
	g_value->*sat_vushort*[x_index]	Ditto for *ushort*.
	g_value->*sat_vlong*[x_index]	Ditto for *long*.
	g_value->*sat_vulong*[x_index]	Ditto for *ulong*.
	g_value->*sat_vfloat*[x_index]	Ditto for *float*.
	g_value->*sat_vdouble*[x_index]	Ditto for *double*.
	g_value->*sat_vprop*	The property list element of an immediate vector.
	g_value->*sat_vsize*	The size of an immediate vector in bytes.
	sat_nivector (x_size)	Creates a new immediate vector with x_size bytes.
vector a-lisp-vector SOB_LVECTOR	g_value->*sat_vvalue*[x_index]	The x_index+1'st element of the LISP vector g_value.
	g_value->*sat_vplist*	The property list element of the LISP vector g_value.
	g_value->*sat_vsize*	The size of the LISP vector g_value in bytes.
	sat_nlvector (x_size)	Creates a new LISP vector with x_size elements.

TABLE 4.3: PART IV LISP VALUES		
typep Type *has-type* Type *sob_ltype* Type	Expression	Meaning
array **a-lisp-array** **SOB_LARRAY**	g_value->*sat_afunction*	The function element of a LISP array.
	g_value->*sat_aaux*	The aux element of a LISP array.
	g_value->*sat_adata*	The data element of a LISP array.
	g_value->*sat_alength*	The length element of a LISP array.
	g_value->*sat_adelta*	The delta element of a LISP array.
port **a-port** **SOB_PORT**	g_value->*sat_lport*	The C port, or *FILE* * value, associated with a LISP port.
value **a-value** **SOB_VALUE**	g_value->*sat_lvalue*	The value of a LISP *value* object.

9. FORMATING READABLE STRINGS. There are several C functions for printing strings in a format that can be read by the LISP *read* function. For example, an arbitrary file name string can be printed to be read as a symbol by the LISP reader via a call such as—

printf ("(cannot open the file %s)", *sat_sformat* (filename));

where filename is a C *char* * string. If filename were to equal—

"/usr/foo/fancy"

then the printf would print—

(cannot open the file /usr/foo/fancy)

but if filename were to equal—

"#play"

then the printf would print—

(cannot open the file |#play|)

Sat_tformat is a similar function for printing a string so it will be read as a string by the LISP reader.

Because these functions return a pointer to a static character string buffer allocated inside the function, two calls to one of these functions cannot be used inside one call to *printf*. See the GLOSSARY for details.

10. HITLIST. Empty for the moment.

11. GLOSSARY.

f_ [Argument Prefix]

DENOTATION: In C, denotes arguments of *double float* (*double*) type. In LISP, denotes arguments of *flonum* type.

forever [C Macro]

EQUIVALENT TO: *for (;;)*.

(**integer-to-bytes** 'x_integer) [Lisp Function]

RETURNS: A list of the 4 consecutive bytes (*fixnum*'s from 0 through 255) that would be stored consecutively in memory to represent the integer. The list has an order on right to left machines, such as VAX'es, which is the opposite of its order on left to right machines, such as 68000's.

left-to-right [LISP Global Variable]

VALUE: Non-*nil* if computer stores bytes in an integer from left to right (high order to low order, like IBM and MOTOROLA). *Nil* if the bytes are stored from right to left (low order to high order, like DEC and INTEL).

lx_ [Argument Prefix]
ulx_ [Argument Prefix]

DENOTATION: In C, denotes arguments of *long int* (*long*) or *unsigned long int* (*ulong*) type.

PI [C Macro]

VALUE: The constant π.

g_larray->**sat_afunction** [C Macro]
g_larray->**sat_aaux** [C Macro]
g_larray->**sat_adata** [C Macro]
g_larray->**sat_alength** [C Macro]
g_larray->**sat_adelta** [C Macro]

WHERE: G_larray must be a LISP *array*.

VALUE: The various parts of g_larray: function, aux, data, length, and delta. *Sat_adata* is a pointer to the array data, which is a block of contiguous memory in a page with the appropriate data type for the array elements (*fixnum*, *flonum*, or *value*).

WHEN ASSIGNED: Changes the part of g_larray.

sat_ceiling (f_number, x_exponent) [C Function]

VALUE: A *long* equal to the smallest integer greater than or equal to f_number times $2^{x\_exponent}$.

SAT_CMAXIMUM [C Constant]
SAT_CMINIMUM [C Constant]
SAT_CMISSING [C Constant]
SAT_UCMAXIMUM [C Constant]
SAT_SMAXIMUM [C Constant]
SAT_SMINIMUM [C Constant]
SAT_SMISSING [C Constant]
SAT_USMAXIMUM [C Constant]
SAT_LMAXIMUM [C Constant]
SAT_LMINIMUM [C Constant]
SAT_LMISSING [C Constant]
SAT_ULMAXIMUM [C Constant]
SAT_IMAXIMUM [C Constant]
SAT_IMINIMUM [C Constant]
SAT_IMISSING [C Constant]
SAT_UMAXIMUM [C Constant]
SAT_FMAXIMUM [C Global Variable]
SAT_FMINIMUM [C Global Variable]
SAT_FMISSING [C Global Variable]
SAT_DMAXIMUM [C Global Variable]
SAT_DMINIMUM [C Global Variable]
SAT_DMISSING [C Global Variable]

sat_cmissing (x_number) [C Macro]
sat_smissing (x_number) [C Macro]
sat_lmissing (x_number) [C Macro]
sat_imissing (x_number) [C Macro]
sat_fmissing (f_number) [C Macro]
sat_dmissing (f_number) [C Macro]

VALUES: The constants and variables are the largest value, smallest value, and missing value for various data types according to the table below. For types with a missing value, the missing value is not part of the range from *SAT_...MIMINIM* through *SAT_...MAXIMUM* inclusive.

SAT_IMAXIMUM SAT_IMINIMUM SAT_IMISSING	*int*	SAT_UMAXIMUM	*unsigned*
SAT_LMAXIMUM SAT_LMINIMUM SAT_LMISSING	*long*	SAT_ULMAXIMUM	*ulong*
SAT_SMAXIMUM SAT_SMINIMUM SAT_SMISSING	*short*	SAT_USMAXIMUM	*ushort*
SAT_CMAXIMUM SAT_CMINIMUM SAT_CMISSING	*char*	SAT_UCMAXIMUM	*uchar*
SAT_FMAXIMUM SAT_FMINIMUM SAT_FMISSING	*float*	SAT_DMAXIMUM SAT_DMINIMUM SAT_DMISSING	*double*

WARNING: On some IEEE hardware,

$$SAT\_FMISSING == SAT\_FMISSING$$
and
$$SAT\_DMISSING == SAT\_DMISSING$$

are both *false*. Therefore the macros *sat_fmissing* and *sat_dmissing* have been provided to test for missing values.

WARNING: Some C compilers cannot convert *SAT_ULMAXIMUM* to a double precision floating point number properly: they insist on going through an *int* as an intermediate step and get -1.0 as a result. To ensure proper results use—

sat_ultod (*SAT_ULMAXIMUM*).

RETURNS: The macros *sat_cmissing*, ..., *sat_dmissing* return true if and only if the number they are testing is a missing value of the given type.

The tests for a particular type of missing value may be made on a copy of the missing value held in a variable of some other type, provided that the other type is large enough to hold all values of the missing value's type. E.g. *SAT_CMISSING* may be copied into a *double* variable and tested there by *sat_cmissing*. Similarly *SAT_FMISSING* may be copied into a *double* variable before being tested.

sat_cnil [C Constant]

VALUE: A *sat_lvalue* specially used as the value of unbound symbols and in other places where the LISP interpreter needs to distinguish a missing value from *nil*.

_SAT_DMISSING [LISP Global Constant]

VALUE: The *flonum* used to denote the *double* missing value by C and FORTRAN code.

sat_empty [C Constant]

VALUE: A *sat_lvalue* which is specially used as a value for unused elements at the end of a LISP *hunk*. E.g., a 3 element *hunk* is actually represented by a 4 element *hunk* (rounding the length up to a power of 2) whose last element is equal to *sat_empty*.

sat_floor (f_number, x_exponent) [C Function]

VALUE: A *long* equal to the largest integer less than or equal to f_number times $2^{x\_exponent}$.

g_hunk->**sat_hvalue**[x_index] [C Macro]
g_hunk->**sat_lfirst** [C Macro]
g_hunk->**sat_lrest** [C Macro]

WHERE: G_hunk must be a LISP *hunk* or the value *sat_nil*.

VALUE: *Sat_hvalue* [x_index] is the x_index+1'th element of g_hunk. Hunks can be used like dotted pairs, with *sat_lfirst* and *sat_lrest* accessing *car* and *cdr* of the hunk. These are the first two elements of the hunk, but the order of these first two elements is implementation dependent. *Sat_nil* may be treated like a hunk if only the first two elements are to be read; both these will equal *sat_nil*.

WHEN ASSIGNED: Changes the element of g_hunk. G_hunk must not be *sat_nil*.

_SAT_IMISSING [LISP Global Constant]

VALUE: The *fixnum* used to denote the *int* missing value by C and FORTRAN code.

& g_string->**sat_lchar** [C Macro]

VALUE: The *char* * value of a *string* g_string. This string ends with a NUL character, as per C conventions. Remember the '&'; g_string->*sat_lchar* is just the first character.

g_number->**sat_ldouble** [C Macro]

VALUE: The *double* value of a *flonum* g_number.

SAT_LEFT_TO_RIGHT [C Macro]

VALUE: 1 if the high order byte of an *int* has a lower address than the low order byte of an *int*, so that printing the bytes from left to right as addresses ascend will print the high order byte first. 0 otherwise, in which case printing from right to left will print the high order byte first.

g_list->**sat_lfirst** [C Macro]
g_list->**sat_lrest** [C Macro]

SEE ALSO: *Sat_hvalue*.

WHERE: G_list must be a dotted pair (*list* value) or the value *sat_nil*.

VALUE: *Sat_lfirst* is the first element of g_list, and *sat_lrest* is the rest of g_list after the first element. If g_list is *sat_nil*, both these return the value *sat_nil*.

WHEN ASSIGNED: Changes the first element or the rest of g_list. G_list must not be *sat_nil*.

g_number->**sat_lint** [C Macro]

VALUE: The *int* value of a *fixnum* g_number.

sat_log (f_number) [C Function]

RETURNS: The logarithm base 2 of the smallest power of 2 which is greater than or equal to the absolute value of f_number, or SAT_IMISSING if f_number is 0.

g_port->**sat_lport** [C Macro]

VALUE: The *FILE ** port associated with a LISP *port* object.

sat_lvalue [C Type]
g_ [Argument Prefix]

VALUE: A lisp value. The prefix *g_* is used in the documentation of C functions to denote such a value.

g_value->**sat_lvalue** [C Structure Element]

VALUE: The value of a LISP *value* object.

sat_mad (lx_multiplicand, lx_multiplier, lx_addend, lx_divisor) [C macro]

WHERE: All arguments are automatically cast to longs.

RETURNS: (multiplicand * multiplier + addend) / divisor as a long.

NOTE: The numerator is computed as a 64 bit signed quantity and then divided to produce a 32 bit long quotient.

sat_mas (lx_multiplicand, lx_multiplier, ux_addend0, x_addend1, x_shift) [C macro]

WHERE: Multiplicand, and multiplier are automatically cast to longs.

RETURNS: (multiplicand * multiplier + addend0 + (addend1 << 32)) << shift as a long.

NOTE: The quantity to be shifted is computed with 64 bit signed arithmetic, and truncated to 32 bits after shifting. A negative << shift is equivalent to >> -shift.

sat_nfixnum (x_number) [C Function]
sat_nsfixnum (x_number) [C Macro]

RETURNS: A *sat_lvalue* equal to a new LISP *fixnum* with *sat_lint* value x_number. Note, however, that if x_number is near 0 the *fixnum* returned will be one of a small table of constant *fixnum*'s whose *sat_lint*'s cannot be changed.

Sat_nsfixnum may be used for greater efficiency in place of *sat_nfixnum* when it is certain that x_number is in the range from -128 through 255 inclusive.

SIDE EFFECT: May call the garbage collector when creating a new *fixnum*.

sat_nflonum (f_number) [C Function]

RETURNS: A *sat_lvalue* equal to a new LISP *flonum* with *sat_ldouble* value f_number.

SIDE EFFECT: May call the garbage collector when creating a new *flonum*.

sat_nhunk (x_size) [C Function]

RETURNS: A *sat_lvalue* equal to a new LISP *hunk* with at least x_size elements. Actually, the hunk is a power of two elements in size (128 elements is the maximum). The first x_size elements are set to *sat_nil*, and the rest to *sat_empty*.

SIDE EFFECT: May call the garbage collector when creating a new *hunk*.

sat_nil [C Constant]

VALUE: The LISP symbol *nil*.

sat_nivector (x_size) [C Function]

RETURNS: A *sat_lvalue* equal to a new LISP immediate vector (*vectori*) object with x_size *bytes*. Note the size is in bytes. The *sat_vprop* of the immediate vector will be *sat_nil*.

SIDE EFFECT: May call the garbage collector when creating a new immediate vector.

sat_nlist (g_first g_rest) [C Function]

RETURNS: A *sat_lvalue* equal to a new LISP *list* with g_first as the first element and g_rest as the rest of the list.

SIDE EFFECT: May call the garbage collector when creating a new *list*.

sat_nlvector (x_size) [C Function]

RETURNS: A *sat_lvalue* equal to a new LISP *vector* with x_size *elements*. The *sat_vprop* of the vector will be *sat_nil*.

SIDE EFFECT: May call the garbage collector when creating a new *vector*.

sat_nsymbol (t_string) [C Macro]

RETURNS: A *sat_lvalue* equal to the LISP *symbol* in the symbol table whose print name is t_string. If this symbol does not already exist, a new symbol is created.

SIDE EFFECT: May call the garbage collector when creating a new *symbol*.

sat_rmasN (lx_multiplier, lx_multiplicand) [C Macro]
sat_rsetN (x_shift) [C Macro]
sat_rdeclareN; [C Macro]

WHERE: N is either nothing, or is one of the digits 1, 2, 3, or 4.

SIDE EFFECT: *Sat_rdeclare*N declares the variables *shift*N, *round*N*0*, and *round*N*1*; *sat_rset*N sets these variables; and *sat_rmas*N uses them.

RETURNS: *Sat_rmas*N returns lx_multiplicand * lx_multiplier left shifted by x_shift. The 64 bit product is computed and shifted, before being truncated to a 32 bit *long*. If x_shift is negative, the product is right shifted by - x_shift with rounding induced by *round*N*0* and *round*N*1* having been set to the proper values.

Sat_rmasN(x,y) expands to—

sat_mas (x, y, *round*N*0*, *round*N*1*, *shift*N).

sat_round (f_number, x_exponent) [C Function]

VALUE: A *long* equal to f_number times $2^{\text{x\_exponent}}$ rounded to the nearest integer.

sat_snformat (t_string, x_count) [C Function]
sat_sformat (t_string) [C Function]
sat_tnformat (t_string, x_count) [C Function]
sat_tformat (t_string) [C Function]

WHERE: X_count is the maximum length of t_string in case the latter is *not* NUL terminated.

RETURNS: A string (*char* * pointer to a static area inside the routine) that is the same as t_string reformatted for input to LISP as a symbol (for *sat_snformat* or *sat_sformat*) or as a string (for *sat_tnformat* or *sat_tformat*). For symbols not containing any special characters, t_string is returned as is (or more precisely, a copy of t_string in the static area is returned). In all other cases, t_string is surrounded by quotes (| or "), and a backslash is prepended to any quote or \ characters.

WARNING: If output would be longer than 4000 characters, exclusive of surrounding |'s or "'s, then the end of the part of the output inside the |'s or "'s may be truncated.

WARNING: The same static area is used by all calls to these functions, which may result in strange effects unless the caller finishes with the result of one call before making another call. Thus the call—

printf ("%s = %s", *sat_sformat* (x), *sat_tformat* (y));

will not work, and should be replaced by something like

```
char temp [1001];
temp[1000] = 0;
...
printf ("%s = %s", strncpy (temp, sat_sformat (x), 1000),
                   sat_tformat (y));
```

NOTE: These functions are contained in the file *sat_csform.c*, which is written so it does not have any *#include* statements, and can be moved to any location and used independently of the rest of SKETCH. A declaration such as—

```
extern char * sat_sformat(), * sat_snformat(),
            * sat_tformat(), * sat_tnformat();
```

will be required in SKETCH-independent code that calls functions in this file.

g_symbol->**sat_svalue** [C Macro]
g_symbol->**sat_splist** [C Macro]
g_symbol->**sat_sfunction** [C Macro]
g_symbol->**sat_slink** [C Macro]
g_symbol->**sat_spname** [C Macro]

WHERE: G_symbol must be a LISP *symbol*.

VALUE: The various parts of the symbol object: value, property list (plist), function definition (function), and print name (pname).

Sat_slink exists in the current version of FRANZ and chains to the next entry in a hash table queue. The last entry has *sat_cnil* as a link value.

WHEN ASSIGNED: Changes the part of the symbol object. The link and print name should not normally be changed. G_symbol should not be *sat_nil* or *sat_t*.

sat_t [C Constant]

VALUE: The LISP symbol *t*.

sat_ultod (ul_x) [C Macro]

RETURNS: Ul_x converted to a double precision floating point number. This is necessary because some C compilers do not do it right: they convert to *int* as an intermediate step, and thus get false results like SAT_ULMAXIMUM == -1.0.

g_ivector->**sat_vchar**[x_index] [C Macro]
g_ivector->**sat_vuchar**[x_index] [C Macro]
g_ivector->**sat_vshort**[x_index] [C Macro]
g_ivector->**sat_vushort**[x_index] [C Macro]
g_ivector->**sat_vlong**[x_index] [C Macro]
g_ivector->**sat_vulong**[x_index] [C Macro]
g_ivector->**sat_vfloat**[x_index] [C Macro]
g_ivector->**sat_vdouble**[x_index] [C Macro]
g_ivector->**sat_vprop** [C Macro]
g_ivector->**sat_vsize** [C Macro]

WHERE: G_ivector must be a LISP immediate vector (*vectori*).

VALUE:

Sat_vchar [x_index] is the x_index+1'th *char* of g_ivector;
sat_vuchar [x_index] is the x_index+1'th *uchar* of g_ivector;
sat_vshort [x_index] is the x_index+1'th *short* of g_ivector;
sat_vushort [x_index] is the x_index+1'th *ushort* of g_ivector;
sat_vlong [x_index] is the x_index+1'th *long* of g_ivector;
sat_vulong [x_index] is the x_index+1'th *ulong* of g_ivector;
sat_vfloat [x_index] is the x_index+1'th *float* of g_ivector; and
sat_vdouble [x_index] is the x_index+1'th *double* of g_ivector.

Sat_vprop is the *sat_lvalue* property list of g_ivector, and *sat_vsize* is the *int* size of g_ivector in bytes.

WHEN ASSIGNED: Changes the element of g_ivector.

g_lvector->**sat_vvalue**[x_index] [C Macro]
g_lvector->**sat_vprop** [C Macro]
g_lvector->**sat_vsize** [C Macro]

WHERE: G_vector must be a LISP *vector*.

VALUE: *Sat_vvalue* [x_index] is the *sat_lvalue* x_index+1'th element of g_lvector. *Sat_vprop* is the *sat_lvalue* property list of g_lvector, and *sat_vsize* is the *int* size of g_lvector in bytes.

WHEN ASSIGNED: Changes the element of g_lvector.

SAT_YES (<comment>) [C Macro]
SAT_NO (<comment>) [C Macro]
SAT_EXCEPTION (<comment>) [C Macro]

WHERE: <comment> is any C macro argument (e.g., it must not contain commas outside parentheses).

VALUES: *SAT_YES* (<comment>) equals 1, *SAT_NO* (<comment>) equals 0, and *SAT_EXCEPTION* (<comment>) equals -1. The <comment> is ignored.

uchar [C Type]

EQUIVALENT TO: *Unsigned char.*

ulong [C Type]

EQUIVALENT TO: *Unsigned long.*

ushort [C Type]

EQUIVALENT TO: *Unsigned short.*

x_ [Argument Prefix]
ux_ [Argument Prefix]

DENOTATION: In C, denotes arguments of *int* or *unsigned int* (*unsigned*) type. In LISP, *x_* denotes arguments of *fixnum* type.

CHAPTER 5

OBJECTS

1. OBJECTS. A SKETCH object has a type and a list of attributes. Each attribute has a label and a value. The types of SKETCH objects have names beginning with 'a-' or 'an-'. The attribute labels of SKETCH objects have names beginning with 'has-' or 'is-', or, in general, with any auxiliary verb or preposition followed by a hyphen.

A SKETCH object may be represented by an expression that evaluates to the object, such as—

(a-man has-weight 174 has-height 70).

The 'a-man' macro called by this expression is the same as the name of the object type, and the argument list consists of attribute label/value pairs. with each label (e.g. 'has-weight') followed by its value (e.g. '174').

SKETCH types are themselves SKETCH objects whose type is the SKETCH type *a-type*. SKETCH attribute labels are themselves SKETCH objects whose type is the SKETCH type *an-attribute*. Thus the existence of the above object implies the existence of other objects such as—

(*a-type has-name* 'a-man ...)

(*an-attribute has-name* 'has-weight)

(*an-attribute has-name* 'has-height)

and these in turn imply the existence of—

(*a-type has-name* 'a-type ...)

(*a-type has-name* 'an-attribute ...)

(*an-attribute has-name* 'has-name).

2. MAKING OBJECTS. A SKETCH object can be made by evaluating an expression that represents it, such as—

(a-man has-weight 174 has-height 70).

Symbols naming the type and attribute labels are used in this expression, along with the values of the attributes. In this expression, all the attribute labels and values are evaluated, so that the expression gives the same result as—

(a-man has-weight (*plus* 100 74) has-height (*difference* 72 2)).

Use is made of the facts that the symbol 'a-man' is defined as a macro which creates objects of type 'a-man', and that the symbols 'has-weight' and 'has-height' evaluate to *an-attribute* SKETCH objects that serve as attribute labels.

The object that results from evaluating one of these expressions can be bound to a variable, as in—

(*setq* george (a-man has-weight 174 has-height 70)).

This actually stores a pointer to the a-man object in the variable george, and we will describe in more detail what this means at the end of the next section.

It is also possible to use one object as a prototype to supply default values for the attributes of a new object. Writing—

(a-man george has-weight 169 has-age 57)

uses george as such a prototype, and makes the object represented by—

(a-man has-weight 169 has-age 57 has-height 70).

The prototype, if present, is the first thing after the type, a-man, in the expression making the new object.

3. GETTING AND SETTING ATTRIBUTES. Attributes can be gotten by expressions such as—

(has-weight george),

which, given the above definition of george, has the value '174', or—

(has-height george),

which has the value '70'. The type of an object can be gotten as if it were the object's *has-type* attribute, via—

(*has-type* george),

which has the value—

(*a-type has-name* 'a-man ...).

Objects with *has-name* attributes often print as just their names, so if you print out this last object you may get just 'a-man'.

The LISP *setf* macro can be used to change attributes, as in—

(*setf* (has-weight george) 185),

after which george will equal—

(a-man has-weight 185 has-height 70)

New attributes can be defined for an object, as in—

(*setf* (has-age george) 34),

after which george will equal—

(a-man has-weight 185 has-height 70 has-age 34).

If an attempt is made to get an attribute that an object does not have, *nil* will be returned, as in—

(has-waist-size george).

This is not an error. Setting an attribute to the value *nil* generally makes the attribute disappear ('generally' means that exceptions are rare, and noted in documentation). Thus after—

(*setf* (has-weight george) *nil*)

george will equal—

(a-man has-height 70 has-age 34).

In general, saying that an object does not have an attribute, and saying that it has the attribute value *nil*, are two ways of saying the same thing.

If the attribute to be gotten is not known till *eval* time, the *get-attribute* function may be used to get the attribute. Examples are—

(*setq* x (*an-attribute has-name* 'has-height))
(*get-attribute* x george),

in which the second expression evaluates to 70, and—

(*setf* (*get-attribute* x george) 85),

which changes george to—

(a-man has-height 85 has-age 34)

In the above examples, george is just a variable that is always evaluated. If one had executed—

(*setq* y george)

first, one could use y and george interchangeably above.

When two variables, such as y and george, are both bound to the same object, they in fact both contain equal pointers to the object. Any change to the object will appear to effect both variables. Thus if george equals—

(a-man has-height 85 has-age 34),

so will y, and after—

(*setf* (has-age y) 35),

both y and george will equal—

(a-man has-height 85 has-age 35).

The type of an object cannot be changed:

(*setf* (*has-type* george) ...)

is in error.

4. NAMES. If an object has a *has-name* attribute that has a non-*nil* value, that value must be a symbol, and that symbol will be set equal to the object. For example, evaluation of the expression—

(a-man *has-name* 'Bill has-weight 143 has-height 68)

will make an object and set the variable Bill equal to that object.

When the *print* function is asked to print an object with a *has-name* attribute, the value of this attribute will be printed as the complete representation of the object. Thus —

(*print* Bill)

will print just 'Bill'. Other forms of printing objects with *has-name* attributes are available, and are described below (see PRINTING AND UNEVALUATING OBJECTS).

After an object with a name is made, it can be referenced by an expression that appears to make a new object with the same *has-name* but no other attributes. In this case, evaluating the expression—

(a-man *has-name* 'Bill)

will not make a new object, but will instead return the already made object that is the value of the variable Bill.

An object that has a type and a *has-name* attribute, but has no other non-*nil* attributes, is called a stub. In general, an attempt to make a stub for an object that already exists will not make a new object, but will merely return the existing object.

The order of making stubs and objects can be reversed. If the stub is made first, an attempt to make the object will not make a new object. Instead, it will fill in the attributes of the stub, and return that stub, which will no longer be a stub any more. Thus the code—

(a-man *has-name* 'Bill has-wife (a-woman *has-name* 'Jill))
(a-woman *has-name* 'Jill has-husband (a-man *has-name* 'Bill))

will work, making only two objects, and setting the variables Bill and Jill. This code would give the same result if we reversed the order of its two statements. The value of—

(has-wife Bill)

is the same as the value of the variable Jill, while the value of—

(has-husband Jill)

is the same as the value of the variable Bill.

If an object with a *has-name* is to be made, and another object with the same name exists before hand, and if neither object is a stub, then the two objects are tested for equality of their attributes (using the *compare-object* function that ignores hidden attributes: see the GLOSSARY). If there is equality, a new object is not made, and the old object is returned as the result of the expression that might have made the new object. If there is no equality, an error is signaled. Thus a named object may be made many times if it is always made the same way.

The notion of a name may be generalized to use attributes other than ***has-name*** to denote an object. Such generalized naming is referred to as 'indexing', and is discussed later in more detail. Indexing also includes placing objects on hidden cross-reference lists that may be used to retrieve the object.

We have discussed 'making' objects in SKETCH, and not 'creating' them. In SKETCH, 'creating' an object is a suboperation of 'making' the object, and does not include any indexing.

5. DYNAMIC TYPE AND ATTRIBUTE CREATION. New types and attributes can be created by expressions such as—

(*a-type has-name* 'a-man)

(*an-attribute has-name* 'has-weight)

(*an-attribute has-name* 'has-height)

(*an-attribute has-name* 'has-age).

However, types and attributes mentioned in data and interpreted code, but not in compiled code, need not be created before they are used. Instead, they may be given names that begin with one of several specific prefixes, in which case they will be created automatically when they are used.

For types, the prefixes are *a-* and *an-*. For attributes, the standard prefixes are *has-*, *is-*, and *isnt-*, and any auxiliary verb or preposition followed by a hyphen may be added to this list as needed (see *define-object-name-prefix* in the GLOSSARY).

For example, evaluating—

(a-man *has-name* 'George has-age 53 has-wife (a-woman *has-name* 'Jill))

when a-man, has-age, has-wife, and a-woman are unbound variables will automatically cause the expressions—

(*a-type has-name* 'a-man)

(*an-attribute has-name* 'has-age)

(*an-attribute has-name* 'has-wife)

(*a-type has-name* 'a-woman)

to be evaluated.

Thus data bases stored in files may use types and attributes previously unknown to the program.

Types and attributes explicitly mentioned in compiled code, however, should be made before they are used. This may be done by executing expressions such as—

```
(eval-when (compile load eval)
           (a-type has-name 'a-man)
           (an-attribute has-name 'has-age)
           (an-attribute has-name 'has-wife)
           (a-type has-name 'a-woman)).
```

The *eval-when* is necessary to ensure that the types and attributes are created both in the compiler and at eval time.

If an object with a non-*nil has-name* attribute is made in the compiler environment, the name of that object will automatically be declared to be *special*, thus permitting reference to it in code. Objects made in the compiler environment should also be made in the evaluation environment, so the code will reference the right object. The *eval-when* (*compile load eval*) in the above example does just this.

Often the *declare-hunk-type* or *declare-vector-type* macros described in the next section are used to create types and attributes, instead of the more direct methods just described.

6. BASIC TYPES. In SKETCH one builds types on top of one another. Generally, one starts with a basic type that is made by an expression such as—

```
(declare-hunk-type an-event
                   has-password *event-password*
                   has-name has-start-time
                   is-read-init-private
                   has-stop-time
                   is-hidden is-private
                   has-previous-event).
```

Declare-hunk-type is a macro whose arguments are generally not evaluated (like *declare*). However, there is a similar function, *define-hunk-type*, whose arguments are evaluated. Both the macro and the function make one *a-type* object with the given name (e.g. an-event), and several *an-attribute* objects with given names (e.g. has-name, has-start-time, has-stop-time, and has-previous-event).

An-attribute-descriptor objects are also created for each attribute label, and *an-operation-descriptor* objects are made for each operation (e.g. *make-object*, *object-is*, *uneval-object*, *format-object*) that is to be defined in a type specific manner. See the sections below and the GLOSSARY for details of making these objects.

The above call defines a new type: an-event. The attributes of this type that are known to the compiler are—

has-name has-start-time has-stop-time has-previous-event.

These attributes are packed into objects of the new type, and are efficiently accessed (the objects are actually hunks, and the access is by indexing elements of the hunks). Other attributes may be set and gotten for an-event object, but these will be stored in a property list where their access will be slower.

By default, attributes can be initialized and read, but not written (i.e. not *setf*). The *is-read-init-private* keyword signifies that subsequent attributes can also be written if the password, in this case the symbol *event-password*, is included, as in—

(*setf* (has-stop-time x *event-password*) y).

We will use the fact that has-stop-time can be written with a password in the section on THE FORMAT-OBJECT OPERATION below.

The *is-private* keyword signifies that subsequent attributes cannot be initialized, but can be read or written if the password is include, as in—

(has-previous-event x *event-password*)

and—

(*setf* (has-previous-event x *event-password*) y).

Assuming that code in one program package does not use the password of another package, a private attribute may be protected from incorrect access by code outside the package that defined the attribute.

Other keywords that play a role similar to *is-read-init-private* and *is-private* are *is-read-init*, which is the default and disallows writing the attribute but allows reading and initializing it; *is-read-init-write* which allows reading, initializing, and writing; and *is-read-private* which allows reading but not initializing, and allows writing, but only with a password.

By default, attributes are printed out when the object is printed, and are included in the result of unevaluating the object (see PRINTING AND UNEVALUATING OBJECTS below). The *is-hidden* keyword signifies that subsequent attributes are not to be printed or appear in the unevaluated object. Such hidden attributes are often used for cross-reference lists between objects. These cross-reference lists can be very bulky to print, and should not be transmitted between different memory loads (which is the purpose of unevaluated objects).

Hidden attributes are also ignored when testing two objects for equality, as is done when two objects with the same name are made (see NAMES above, and *compare-object* in the GLOSSARY):

The *is-visible* keyword is the opposite of *is-hidden*, and signifies that subsequent attributes are to be printed, appear in the unevaluated object, and be considered during tests for object equality. In the OTHER ATTRIBUTE SWITCHES section below, we describe how an attribute can be made hidden in some ways and visible in others.

The *declare-vector-type* macro is similar to *declare-hunk-type* but defines objects that are LISP immediate vectors (see the FRANZ LISP manual) and C structures. A typical use might be—

(*declare-vector-type* an-event
has-password *event-password*
a-value has-name
a-long has-start-time
is-read-init-private
a-long has-stop-time
is-hidden is-private
a-value has-previous-event).

In a *declare-vector-type* call, the data type of the attributes can be declared to be be a C numeric type, such as *char*, *long*, or *float*, rather than just a LISP value. This is done by including type names such as *a-char*, *a-long*, and *a-float* in front of the attribute labels for the attributes that are to have the given type. The type name *a-value* refers to LISP values, and is the default at the beginning of the attribute list. The first element of the vector stores the type of the object, as a LISP value. The property list of the vector (see the FRANZ LISP manual) is a hunk that stores a copy of all the LISP values stored in the vector, so that the garbage collector will know about these values.

Both *declare-hunk-type* and *declare-vector-type* expand into an—

(*eval* (*compile load eval*) ...)

form, so they will be effective at all times. If appropriate extra arguments are given to these macros, and if the global variable **C-definition-port** is set to a port when either of these macros is called (e.g. loaded or compiled), then C structure definitions are written into this port so that C code can access the information in the object.

For example, the declaration—

```
(declare-vector-type (an-event ev_event ev_)
  has-password *event-password*
  a-value (has-name nil ev_name)
  a-float (has-start-time nil ev_start)
  is-read-init-private
  a-float (has-stop-time nil ev_stop)
  is-hidden is-private
  an-event (has-previous-event nil ev_previous))
```

will output the C structure definitions—

```
typedef struct ev_struct * ev_event;
struct ev_struct {
        union { int SOB_VSIZE [1];
                sat_lvalue * SOB_VPLIST [1];
                sob_type SOB_VTYPE; } SOB_VFIRST;
#       define ev_type SOB_VFIRST.SOB_VTYPE
#       define ev_plist SOB_VFIRST.SOB_VPLIST[-1][0]
#       define ev_vsize SOB_VFIRST.SOB_VSIZE[-2]
        sat_lvalue ev_name;
        float ev_start;
        float ev_stop;
        ev_event ev_previous;
        };
#define ev_alloc(x,y) struct ev_struct (x) [y]
```

See the GLOSSARY entries on *declare-hunk-type* and *declare-vector-type* for more information.

7. CHECKING TYPES. It is often necessary to check whether an object is of a particular type. This can be most efficiently done by the *object-is* function, as in—

(*object-is* an-event x),

which evaluates to non-*nil* if x is an-event. If it is necessary to discover the type of an object, this may be done less efficiently by the *has-type* function, as in—

(*has-type* x),

which evaluates to—

(*a-type has-name* an-event ...)

if x is an-event.

8. PRINTING AND UNEVALUATING OBJECTS. Printing objects is best done by the *pretty-print* function, as in—

(*pretty-print* x).

This function contrives to insert line feeds as necessary to make the object fit within lines. No part of the object is to the left of the initial print position, and every attribute value is indented with respect to its label. The number of line feeds inserted is returned by this function.

If an object x has an attribute with value y, and if y has a *has-name* attribute with value z, then when x is *pretty-print*'ed, z will be printed in place of y. Thus y is represented by its name. However, this will not be done for x itself, which will always be printed as a type and list of attributes.

The top level printer uses pretty-print to print evaluation results, unless a result has a non-*nil has-name* attribute value which is not identical to the expression evaluated to get the result, in which case just the *has-name* value is printed. Thus after evaluating—

(a-man *has-name* 'Bill has-wife (a-woman *has-name* 'Jill))
(a-woman *has-name* 'Jill has-husband (a-man *has-name* 'Bill)),

evaluating 'Bill' at the top level prints—

(a-man *has-name* 'Bill has-wife Jill),

while evaluating—

(has-husband Jill)

prints just 'Bill'.

One cannot copy the printed representation of an object into a file, read back the file, and get the object again. This sort of thing can be done for some LISP values, but not for SKETCH objects. However, the *uneval-object* function will transform any SKETCH object into a LISP object that has this print-re-read ability, and which, when evaluated, will yield the SKETCH object. Thus the code—

(*setq* y (*uneval-object* x))
(*pretty-print* y some-output-port)
(*setq* z (*read* corresponding-input-port))
(*setq* w (*eval* z))

will generally cause w to equal x (and z to equal y).

Here, also, if some attribute value of x is an object with a *has-name* attribute, that attribute of x will be represented in y by just its type and name, as in—

(a-man *has-name* 'Bill).

So an equal object of the same name must be made to exist in the environment that evaluates z. However, x itself will not be represented by its name, if it has one, but will always be represented as a type and list of attributes.

Thus the unevaluation of 'Bill' above is—

(a-man *has-name* 'Bill has-wife (a-woman *has-name* 'Jill)),

while the unevaluation of 'Jill' is—

(a-woman *has-name* 'Jill has-husband (a-man *has-name* 'Bill)).

9. OPERATIONS. Operations can be defined which are like functions that have different definitions depending upon the type of their first argument. Operations can also have both a macro definition, used at macro expansion time if the type of the first argument can be deduced at that time, and a function definition, used at evaluation time, if the type of the first argument is not known at macro expansion time.

To define an operation called "move-forward" we write—

(*eval-when* (*eval load compile*)
(*an-operation has-name* 'move-forward)).

To define how it will be applied to an-event object we write—

(*eval-when* (*eval load compile*)
(*an-operation-descriptor has-name* '*move-event-forward-descriptor*
has-type an-event
has-operation move-forward
has-function 'move-event-forward-function
has-macro 'move-event-forward-macro
has-parameters <some-parameter>)).

Now the call—

(move-forward x y)

were x is an-event will evaluate the same as—

(*funcall* 'move-event-forward-function
move-event-forward-descriptor move-forward
x y).

Move-event-forward-function can access the *has-parameters* attribute of *move-event-forward-descriptor* if it wants to. This can allow one function to serve for several related operations.

Move-event-forward-function will be used instead of move-event-forward-macro because the type of x is not known at macro expansion time. However, the call—

(move-forward (an-event x) y)

will be macro expanded to—

(move-event-forward-macro #.*move-event-forward-descriptor* #.move-event
(an-event x) y)

which will expand in turn. Note that the first two arguments are not expressions, but rather *an-operation-descriptor* object and *an-operation* object (the '#.' instructs the LISP reader to both read and evaluate the next expression, and return the result of the evaluation as the thing read). The *has-parameters* attribute of the former could be accessed by move-event-forward-macro.

Macro arguments like the first two to move-event-forward-macro are called 'pre-evaluated'. Such arguments are actual values, rather than expressions which evaluate to values at some later time. Pre-evaluated macro arguments provide parametric information to macros efficiently. However, pre-evaluated arguments must not be used in the expansion of the macro, unless the macro expands to a call on another macro that also accepts pre-evaluated arguments.

If move-event-forward-macro were not given (the *has-macro* attribute of *move-event-forward-descriptor* was omitted), then—

(move-forward (an-event x) y)

would be macro expanded to—

(move-event-forward-function *move-event-forward-descriptor* move-event
(an-event x) y).

Omitting move-event-forward-function (the *has-function* attribute of *move-event-forward-descriptor*), is not permitted.

By defining another operation descriptor, such as—

```
(eval-when (eval load compile)
        (an-operation-descriptor has-name '*move-truck-forward-descriptor*
                                 has-type a-truck
                                 has-operation move-forward
                                 has-function 'move-truck-forward-function
                                 has-macro 'move-truck-forward-macro
                                 has-parameters <some-parameter>)),
```

the move-forward operation could be defined differently on events and trucks.

10. PARENT OPERATIONS. It is possible to redefine an operation in such a way that the new definition uses the old definition. Suppose we have defined the move-forward operation as above, and write—

```
(eval-when (eval load compile)
        (an-operation-descriptor has-name '*newer-move-event-forward-descriptor*
                                 has-type an-event
                                 has-operation move-forward
                                 has-function 'newer-move-event-forward-function
                                 has-macro 'newer-move-event-forward-macro
                                 has-parameters <some-parameter>)).
```

Now the call—

```
(move-forward x y)
```

were x is an-event will evaluate the same as—

```
(funcall 'newer-move-event-forward-function
        *newer-move-event-forward-descriptor* move-forward
        x y)
```

However the previous definition of move-forward has not been lost. Whenever *an-operation-descriptor* with particular *has-descriptor-type* and *has-descriptor-operation* attributes is made, the most recently made operation descriptor with the same *has-descriptor-type* and *has-descriptor-operation*, if any, becomes the parent of the new descriptor. In our case, the parent operation can be executed by the call—

```
(execute-parent-operation *newer-move-event-forward-descriptor*
                          move-forward x y),
```

which will evaluate the same as—

```
(funcall 'move-event-forward-function
        *move-event-forward-descriptor* move-forward
        x y).
```

Similarly the call—

```
(execute-parent-operation *newer-move-event-forward-descriptor*
                          move-forward (an-event x) y)
```

will macro expand to—

(move-event-forward-macro #.*move-event-forward-descriptor* #.move-forward
(an-event x) y).

There can be a problems with reloading a code file into an environment into which the file has previously been loaded, such as after fixing bugs in the file during debugging, if the file contains attribute descriptor definitions such as that in the *eval-when* above. Normally, any newly made descriptor is added to all the previously existing descriptors, so the new version of the descriptor and the old version would both be active, with the old version being an ancestor of the new. However, if the descriptor has a *has-name* attribute, remaking it will merely return the old descriptor in place of the new descriptor, without making any new active descriptor. This is what should happen, so descriptors should be named. They are usually named anyway, to facilitate their use in ***execute-parent-operation*** calls.

But now a different problem appears: the reloaded descriptor must be identical with the previously loaded descriptor to prevent an error (see NAMES above). Thus one cannot fix a bug in the descriptor definition without reloading from scratch.

11. CREATE-OBJECT OPERATIONS. Often the creation of an object of a particular type should be accompanied by checks on the attribute values of the object. These may be performed by a special create function for the object.

First note that the ***create-object*** operation is invoked by calls such as—

(***create-object*** (*list* an-event has-start-time 1100
has-stop-time 1330)
nil)

in which the first argument is a list which represents the object, and the second argument is a prototype object, which is missing (i.e. *nil*) in this case. The list which represents the object is called an 'abnormal object'. It has the object type as its first element, and the object's attribute label/value pairs as its remaining elements. The prototype object, were it present, would be used to supply default values for attributes not specified in the abnormal object.

Now given the *declare-hunk-type* definition of an-event above, we may evaluate—

(***eval-when*** (***compile load eval***)
(***an-operation-descriptor***
has-name '*create-event-descriptor*
has-descriptor-operation create-object
has-descriptor-type an-event
has-function 'create-event)),

and thereby introduce a new function, create-event, to take over the job of creating an-event objects. This function might be written as—

```
(defun create-event (the-operation-descriptor the-operation
                                               the-object the-prototype
                                               &aux the-event)
     (setq the-event (create-parent-object *create-event-descriptor*
                                               the-object the-prototype))
     (cond ((not (object-is-a-stub (an-event the-event)))
            (assert (fixp (has-start-time (an-event the-event)))
                    '(has-start-time attribute is not a fixnum))
            (assert (fixp (has-stop-time (an-event the-event)))
                    '(has-stop-time attribute is not a fixnum))
            (assert (not (lessp (has-stop-time (an-event the-event))
                                (has-start-time (an-event the-event))))
                    '(has-stop-time attribute is less than has-start-time attribute))))
     the-event).
```

This function first uses the object creation facility provided by the parent descriptor of *create-event-descriptor*: that is, by the descriptor for the *create-object* operation on an-event type objects that existed just before *create-event-descriptor* was made. This parent is invoked by the call—

(*create-parent-object* *create-event-descriptor* the-object the-prototype),

which is almost equivalent to—

(*execute-parent-operation* *create-event-descriptor* create-object
the-object the-prototype),

but differs in that it does not try to extract the type of the-object by executing—

(*has-type* the-object),

but uses—

(*first* the-object)

instead, because the-object is not an-event object, but rather an abnormal object.

Our function then checks the attribute values, and returns the object created. We must not check the attributes in the case when the object created is a stub (see NAMES above).

Note that we write '(an-event the-event)' instead of simply 'the-event' whenever we reference an attribute of the-event. The compiler uses the extra information that the-event is an-event to compile much more efficient code for accessing the event. In fact, the code that is compiled for element references executes in about 1 microsecond in this case, whereas if the information is omitted the compiled code might take more than 100 microseconds.

12. MAKE-OBJECT OPERATIONS AND INDEXING. The act of making an object is different from creating it. Making an object first creates it, and then indexes it. We can add a make function special to an-event by writing—

```
(eval-when (compile load eval)
          (an-operation-descriptor
           has-name '*make-event-descriptor*
           has-descriptor-operation make-object
           has-descriptor-type an-event
           has-function 'make-event))
```

to introduce the new function make-event for making an-event objects. The make-event function could be defined as follows—

```
(defvar *event-list*) ; List of all events sorted by has-start-time.

(defun make-event (the-operation-descriptor the-operation
                                           the-object the-prototype
                                           &aux the-event)
      (setq the-event (make-parent-object *make-event-descriptor*
                                         the-object the-prototype))
      (cond ((not (has-start-time (an-event the-event))))
            ((or (null *event-list*)
                 (lessp
                  (has-start-time (an-event the-event))
                  (has-start-time (an-event (first *event-list*)))))
             (push the-event *event-list*))
            (t
             (do ((the-list *event-list* (rest1 the-list)))
                 ((or (null (rest1 the-list))
                      (lessp
                       (has-start-time (an-event the-event))
                       (has-start-time (an-event (second the-list)))))
                  (setf (has-previous-event the-event *event-password*)
                        (first the-list))
                  (if (rest1 the-list)
                     (setf (has-previous-event (second the-list)
                                               *event-password*)
                           the-event))
                  (setf (rest1 the-list)
                        `(,the-event . ,(rest1 the-list)))))))
      the-event).
```

This function first uses the object making facility provided by the parent descriptor of *make-event-descriptor*. This facility is invoked by the call to *make-parent-object* which behaves like *create-parent-object* (see last section: the-object is an abnormal object here too). Our function then indexes the new event, by setting its has-previous-event attribute to the nearest previous event, if any, and by *push*'ing it into the *event-list*. However, this indexing is not done if the newly created event is a stub, which would be true if and only if its has-start-time is *nil* (because of the checks made by the create-event function above).

The reason why the has-previous-event attribute is hidden (see the section above on BASIC TYPES) should now be clear. If the has-previous-event attribute were to be printed when an-event object is printed, its value would be another event object, which when printed would contain another has-previous-event attribute, which would print yet another an-event object, and so on recursively. Also, if an-event object is copied from one memory load to another. the has-previous-event list in the target memory might be different from that in the source memory. So the has-previous-event attribute should not be copied, but should be recomputed when the object arrives in the target memory.

13. STANDARD OPERATIONS. Table 5.1 is a synopsis of all the operations that are known to the object system. All but the ones that index descriptors are standardly defined for all SKETCH objects by *declare-hunk-type*, *declare-vector-type*, or the SKETCH dynamic type creation mechanism (see BASIC TYPES and DYNAMIC TYPE AND ATTRIBUTE CREATION above). *Compare-object* and *uneval-object* are also defined for LISP objects, such as numbers and lists.

14. ATTRIBUTE DESCRIPTORS. There are a number of different operations associated with a given attribute and a given type—

(1) Get the value of the attribute from an object of the given type.

(2) Set the value of the attribute for an object of the given type.

(3) Inspect and optionally change an initial value of the attribute for an object of the given type which is being made.

(4) Provide the default value of the attribute for an object of the given type which is being made.

(5) Determine whether the attribute is to appear in a pretty-printed version of an object of the given type, and optionally format the attribute value in a special manner when it is to be part of such a pretty-printing.

(6) Determine whether the attribute is to appear in an unevaluated version of an object of the given type, and optionally unevaluate the attribute value in a special manner when it is to be part of such an unevaluation.

(7) Determine whether the attribute's values are to be compared when objects of the given type are compared, and optionally compare the attribute's values in a special manner when such objects are compared.

TABLE 5.1 STANDARD OBJECT OPERATIONS	
make-object	Makes an object. First applies *has-init-* functions and macros to attribute values destined for the object, and finds default values for attributes not specified. Then creates the object, and lastly indexes the object.
create-object	Creates an object. Does no indexing. Does not use default values or *has-init-* functions or macros.
object-is	Tests objects to see if they are of a given type.
object-is-a-stub	Tests objects to see if they are a stub.
compare-object	Tests objects for equality of all non-hidden attributes.
move-object	Sets all the attributes of the second object to the values of the attributes of the first object, and then discards the first object (it cannot be further used again).
uneval-object	For an object, returns a LISP object that will evaluate to the object, and which can be printed and reread without being changed.
format-object	For an object, returns a format that can be *pretty-print-format*'ed to *pretty-print* the object.
index-operation-descriptor	Records the existence of a new operation descriptor for a type (see *an-operation-descriptor* in the GLOSSARY).
index-attribute-descriptor	Records the existence of a new attribute descriptor for a type (see *an-attribute-descriptor* in the GLOSSARY).

(8) Determine whether the attribute value is to be tested for *nil* when an object of the given type is tested to see if it is a stub, and optionally perform this test in a special manner when such an object is tested.

Rather than have *an-operation-descriptor* for each of these 8 operations, we have *an-attribute-descriptor* which provides information for all 8 operations. The attribute descriptor in turn references *an-attribute-function-table* which has functions and macros for the first 3 of the above operations. Specifically, the attribute function table has the attributes—

has-get-function	*has-get-macro*
has-set-function	*has-set-macro*
has-init-function	*has-init-macro*

which play the same roles as the *has-function* and *has-macro* attributes of *an-operation-descriptor*.

For the fourth operation above, default value specification, the attribute descriptor does not provide a function or macro. Instead it provides an expression which is evaluated when an initial value is needed.

For the last 4 operations the attribute descriptor contains a switch, which behaves something like *an-operation-descriptor has-function* attribute. The switches can also take the values *yes* or *no*, whose meaning depends upon the type of switch. For example, a *no* value for the *has-pretty-format* switch that controls pretty-printing means the attribute is not to be included when its containing object is pretty-printed; a *yes* value means it is to be included; a *nil* value expresses no opinion on inclusion (if no one expresses an opinion, *yes* is assumed); and any other value is taken to be a function that is called in place of *pretty-format* to format the attribute value for pretty-printing.

Below we will discuss the get operation, default value, and pretty-format switch in more detail. See the GLOSSARY for details on the set and init operations and the uneval, compare, and is-a-stub switches. All the operations and switches mentioned in this section are reviewed in Table 5.2.

15. HAS-GET-FUNCTION'S. The following attribute descriptor definition supplies special functions to get and set the has-duration attribute of an-event—

```
(eval-when (compile load eval)
        (an-attribute-descriptor
            has-name '*get-event-has-duration-descriptor*
            has-descriptor-attribute has-duration
            has-descriptor-type an-event
            has-functions
                (an-attribute-function-table
                 has-get-function 'event-duration-get-function
                 has-set-function 'never-set-function
                 has-init-function 'never-init-function)))

(defun event-duration-get-function (the-descriptor the-attribute the-object)
    (difference (has-stop-time (an-event the-object))
                (has-start-time (an-event the-object)))).
```

This definition specifies that if x is an-event,

(has-duration x)

will be computed by calling—

(*funcall* 'event-duration-get-function *get-event-has-duration-descriptor* has-duration x).

This call will return the difference of the stop and start times for x.

The first argument to event-duration-get-function is the descriptor just made above, the one that triggered the call to event-duration-get-function. This descriptor is not used

TABLE 5.2 ATTRIBUTE OPERATIONS, VALUES, AND SWITCHES	
Attribute of An-Attribute-Descriptor **Attribute of An-Attribute-Function-Table**	**Use**
has-functions **has-get-function** **has-get-macro**	Get the value of the attribute from an object of the given type.
has-functions **has-set-function** **has-set-macro**	Set the value of the attribute for an object of the given type.
has-functions **has-init-function** **has-init-macro**	Inspect and optionally change an initial value of the attribute for an object of the given type which is being made.
has-default-value	Find the default value of the attribute for an object of the given type which is being made.
has-format-switch	Determine whether the attribute is to appear in a pretty-printed version of an object of the given type, and optionally format the attribute value in a special manner when it is to be part of such a pretty-printing.
has-uneval-switch	Determine whether the attribute is to appear in an unevaluated version of an object of the given type, and optionally unevaluate the attribute value in a special manner when it is to be part of such an unevaluation.
has-compare-switch	Determine whether the attribute's values are to be compared when objects of the given type are compared, and optionally compare the attribute's values in a special manner when such objects are compared.
has-is-a-stub-switch	Determine whether the attribute value is to be tested for *nil* when an object of the given type is tested to see if it is a stub, and optionally perform this test in a special manner when such an object is tested.

in the above example, but in general it may be used to allow one function to get many different attributes. The descriptor *has-parameters* attribute can be used as a parameter by this function.

This *has-parameters* attribute is an integral part of the descriptor so that it can be efficiently accessed. Other attributes special to an application may be defined for the descriptor, but they will not be accessed as efficiently (as they will be placed on the descriptor's property list).

The event-duration-get-function is the *has-get-function* of the attribute descriptor defined above. The *has-set-function* of the same descriptor specifies that—

(*setf* (has-duration x) y)

will be computed by calling—

(*funcall 'never-set-function* y *get-event-has-duration-descriptor* has-duration x).

Never-set-function is a standard function supplied by SKETCH which will print an error message saying that has-duration can never be set for objects of an-event type. *Never-init-function* is similar, and prohibits has-duration from being initialized when an-event is made.

For details on *has-get-function*'s, *has-set-function*'s, and *has-init-function*'s. see *an-attribute-function-table* in the GLOSSARY.

16. HAS-GET-MACRO'S. The expression—

(has-duration (an-event x))

will expand into a *funcall* to event-duration-get-function, given the above definitions. It would be nice to allow event-duration-get-function to be a macro, so it could produce more efficient in-line code. But this is not always possible, because an expression such as—

(has-duration x)

does not know the type of x at compile time, and therefore must expand into something that does not locate the event-duration-get-function until eval time. Since compiled code cannot call macros at eval time, event-duration-get-function cannot be a macro.

However, if we change the attribute descriptor definition to add a *has-get-macro*, as in—

```
(eval-when (compile load eval)
        (an-attribute-descriptor
           has-name '*get-event-has-duration-descriptor*
           has-descriptor-attribute has-duration
           has-descriptor-type an-event
           has-functions
              (an-attribute-function-table
               has-get-function 'event-duration-get-function
               has-get-macro 'event-duration-get-macro
               has-set-function 'never-set-function
               has-init-function 'never-init-function))),
```

then—

(has-duration (an-event x))

will expand into—

```
(event-duration-get-macro #.*get-event-has-duration-descriptor* #.has-duration
                          (an-event x)),
```

where the first two macro arguments are pre-evaluated (see OPERATIONS above). Event-duration-get-macro should be a macro, and may be defined by—

```
(defmacro event-duration-get-macro (the-descriptor the-attribute the-object)
          `(let ((x ,the-object))
               (difference (has-stop-time (an-event x))
                           (has-start-time (an-event x))))).
```

Event-duration-get-function must still exist, and will be called by the expansion of—

(has-duration x),

which does not specify the type of x at macro expansion time.

17. DEFAULT VALUES. Default values may be specified when types are defined by *declare-hunk-type* or *declare-vector-type*, as in—

```
(declare-hunk-type an-event
                   has-password *event-password*
                   has-name (has-start-time 0)
                   is-read-init-private
                   (has-stop-time *default-duration*)
                   is-hidden is-private
                   has-previous-event),
```

where the default values are '0' and '*default-duration*'. These default values become the *has-default-value* attributes of appropriate attribute descriptors.

Default values are expressions which are evaluated when needed. They may refer to global variables, such as '*default-duration*', but not to local variables. Also, if one wants *nil* to be a default value, one must use the non-*nil* expression '*nil* (with a quote) as a default value expression.

When an object is made, a search is made for default values declared for attributes and associated with the type of the object being made (see SEARCHING FOR DESCRIPTORS below). Default values are found as the *has-default-value* attribute of *an-attribute-descriptor*'s whose *has-descriptor-type* is the type of the object being made (or an ancestor of that type: see the *has-parent* attribute of *a-type* object in the GLOSSARY), and whose *has-descriptor-attribute* is the attribute which has the default value (neither *has-descriptor-type* or *has-descriptor-attribute* may be *nil*). If a non-*nil* default value is found, it is an expression which is evaluated to produce an initial value for an attribute in an object being made.

Default values are inherited. If a *has-default-value* attribute is initialized to *nil* when *an-attribute-descriptor* is made, the attribute will be reset to the value of the parent descriptor's *has-default-value* attribute.

18. PRETTY-FORMAT SWITCHES. A *format-object* operation is provided for each SKETCH object type to perform the duities of the *pretty-format* macro for objects of that type. The default *format-object* operations provided by *declare-hunk-type* and *declare-vector-type* will only include in the resulting format attributes actually stored in the object which is to be pretty-printed. Also, only attributes with non-*nil* values are included.

These default *format-object* operations search attribute descriptors for *has-format-switch*'s in the same way as default *make-object* operations search for *has-default-value*'s. The format switch found for an attribute is used to control inclusion of the attribute value in the object's format, and may also control the formatting of the attribute value.

(1) If the format switch is *no*, the attribute value is not included in the object format.

(2) If the format switch is *yes*, the attribute value is included, and the *pretty-format* macro is used to format the attribute value for inclusion.

(3) If the format switch is a symbol other than *yes*, *no*, or *nil*, that symbol is used as a function called with the same arguments as *pretty-format* to format the attribute value. If the function returns a non-*nil* value, the attribute is included in the object format, and the function's return value is taken to be the format of the attribute value. If the function returns *nil*, the attribute value is not included.

(4) A *nil* format switch, which is the same as no format switch being provided by any attribute descriptor, is taken as equivalent to a *yes* format switch.

(5) Note that if an attribute has a *nil* value, it is not included in the object format, and no check of the attribute's format switch is mode.

19. OTHER ATTRIBUTE SWITCHES. Attribute descriptors have three other kinds of switches.

The *has-uneval-switch* attribute is just like the *has-format-switch* attribute, except it is for the *uneval-object* operation rather than for the *format-object* operation.

The *has-compare-switch* attribute is similar, but it is for the *compare-object* operation. Here a *no* means not to test the attribute values when comparing two objects, while a *yes* or *nil* means to test the values with the *equal* function. Any other symbol as the switch value means to test by calling the symbol as a replacement for the *equal* function. The symbol *compare-object-function* may be used just so to cause attribute values to be themselves compared piece by piece, after the manner of *compare-object*. *Equal*, on the other hand, will consider SKETCH objects to be different if they do not occupy the same position in memory, even if the objects have identical parts.

The *has-is-a-stub* attribute is similar, but tests attribute values for *nil* in order to determine whether an object is a stub (see NAMES above).

A hidden attribute, as described in the BASIC TYPES section above, has *no* as the value of its *has-format-switch*, *has-uneval-switch*, and *has-compare-switch*. A visible attribute has *nil* as the value of these switches. By chosing different values for these switches, an attribute may be made partly visible and partly hidden. See *declare-hunk-type* and *declare-vector-type* in the glossary for how to specify values for these switches.

20. THE FORMAT-OBJECT OPERATION. The *format-object* operation does the work of *pretty-format* for SKETCH objects. It is common to want to adjust the pretty-printed version of the object beyond what can be done with format switches. This is usually done by creating a new copy of the object and tinkering with the attribute values in the copy. For example, to pretty-print an-event objects with a has-duration attribute in place of has-stop-time, the following might be used—

```
(eval-when (compile load eval)
          (an-operation-descriptor
           has-name '*format-event-descriptor*
           has-descriptor-operation format-object
           has-descriptor-type an-event
           has-function 'format-event))

(defun format-event (the-operation-descriptor the-operation
                                              the-object
                                              &optional the-level
                                              &aux the-event)
      (setq the-event (create-parent-object *create-event-descriptor*
                                            (list an-event) the-object))
      (cond ((has-start-time (an-event the-event))
             (setf (get-parent-attribute
                    *get-event-has-duration-descriptor* has-duration
                    (an-event the-event))
                   (diff (has-stop-time (an-event the-event))
                         (has-start-time (an-event the-event))))
             (setf (has-stop-time (an-event the-event) *event-password*)
                   nil)))
      (execute-parent-operation *format-event-descriptor* format-object
                                (an-event the-event) the-level)).
```

This function creates a copy of the-object, called the-event. If the-event is not a stub, then the function stores an actual has-duration attribute value in the-event, so that that attribute will print, and sets the actual has-stop-time attribute of the-event to *nil*, so that attribute will not print. Lastly, the function formats the-event using *execute-parent-operation*.

In order to set the actual has-duration attribute of the-event, the function must use the parent of *get-event-has-duration-descriptor*, which was first introduced in the HAS-GET-FUNCTION'S section above. The *get-parent-attribute* macro aids in this.

21. SEARCHING FOR DESCRIPTORS. When an operation, such as *make-object*, is to be performed for a particular type, a search for *an-operation-descriptor* is made. All the descriptors searched must have their *has-descriptor-operation* attribute equal to the operation to be performed.

First, an operation descriptor whose *has-descriptor-type* is the particular type of the object being operated on (or an ancestor of that type: see *a-type* in the GLOSSARY) is searched for. Such an operation descriptor is specific to the particular type of object being operated on. The search is made in most-recently-made-first order.

Then, if no such specific descriptor is found, a global operation descriptor valid for all types is searched for. Such a global descriptor has *nil* as its *has-descriptor-type*. Again the search is made in most-recently-made-first order.

The descriptor that is found is passed as an argument to the function or macro designated by that descriptor.

When a function, such as the *has-get-function*, is required for a particular type/attribute pair, a search is first made among all attribute descriptors whose *has-descriptor-attribute* equals the attribute. As for operations, the search begins in most-recently-made-first order with descriptors whose *has-descriptor-type* equals the type of the object whose attribute is being gotten (or an ancestor of that type), and then continues, again in most-recently-made-first order, to descriptors whose *has-descriptor-type* is *nil*. However, in this case, if no descriptor is found, the search continues further to descriptors whose *has-descriptor-attribute* is *nil*, but whose *has-descriptor-type* is the type of the object whose attribute is being gotten (or an ancestor of that type). Again, this last search is made in most-recently-made-first order.

Another difference in the attribute descriptor case is that not all the descriptors searched will have a *has-get-function*. Those without such a function cannot be used, and are ignored.

The descriptor that supplies the *has-get-function* is passed as an argument to that function.

The searches for *has-set-function*'s and *has-init-function*'s are similar. Searches for macros (e.g. *has-get-macro*), will be satisfied by a descriptor that has either a macro or a function. If only the function is present (e.g. only *has-get-function* and not *has-get-macro*), it will be used as if a macro existed that simply called the function.

Execute-parent-operation and *get-parent-attribute* continue searches from the point where they left off. If a first search for *an-operation-descriptor* found a descriptor D, then—

(*execute-parent-operation* D ...)

continues the search from the point where it left off. *Get-parent-attribute* can be used with *self* to continue searches for setting attributes.

Make-parent-object and *create-parent-object* are used in place of *execute-parent-object* for the *make-object* and *create-object* operations, because these latter operations do not have a first argument which is an object of the type to be used in searching for the descriptor. Rather the first argument is a list whose first element is that type. Similarly *parent-object-is* is used for the *object-is* operation, whose first argument is the type itself. Other non-standard operations can be created with help from the *find-operation-descriptor* macro (see the GLOSSARY).

Searching for switches and default values is similar to searching for *has-get-function*'s, except that one searches instead for non-*nil* *has-default-value*'s, *has-format-switch*'es, *has-uneval-switch*'es, *has-compare-switch*'es, or *has-is-a-stub-switch*'es.

22. GENERALIZED INDEXING. Suppose we want to create a data base containing people defined by—

```
(declare-hunk-type a-person

                   has-height has-weight has-age ...

                   has-is-a-stub-switch 'yes
                   has-name

                   has-is-a-stub-switch 'no
                   has-social-security-number).
```

We want each person to be uniquely defined by his social security number, but not by his name. Looking back at the previous section on NAMES, we see that we want to define the concept of a-person stub to be a-person object with all attributes missing (*nil*) except for has-social-security-number. This is done by the two *has-is-a-stub-switch* lines in the above definition. The first ensures that the *has-name* attribute of a-person must be *nil* in a stub: by default it would not have to be *nil*. The second allows the has-social-security-number attribute of a-person in a stub to be non-*nil*.

The next step is to define a special *make-object* function for a-person objects—

```
(eval-when (eval load compile)
        (an-operation-descriptor has-descriptor-type a-person
                                 has-descriptor-operation make-object
                                 has-function 'make-person-function))

(defun make-person-function (the-operation-descriptor the-operation
                             the-abnormal-object the-prototype
                             &aux the-new-object the-social-security-number the-previous-object)
  (setq the-new-object
        (create-object (process-attributes the-abnormal-object the-prototype)
                       the-prototype))
  (setq the-social-security-number
        (has-social-security-number (a-person the-new-object)))
  (assert (and the-social-security-number
               (symbolp the-social-security-number))
          '(has-social-security-number attribute is not a non-nil symbol))
  (setq the-previous-object
        (and (boundp the-social-security-number)
             (symeval the-social-security-number)))
  (assert (or (not the-previous-object)
              (object-is a-person the-previous-object))
          `(previous value of ,the-social-security-number is not a-person))

  (cond ((not the-previous-object)
         (set the-social-security-number the-new-object))
        ((object-is-a-stub (a-person the-new-object))
         the-previous-object)
        ((object-is-a-stub (a-person the-previous-object))
         (move-object the-new-object the-previous-object)
         the-previous-object)
        ((compare-object (a-person the-new-object)
                         (a-person the-previous-object))
         the-previous-object)
        (t (error '(object made is not equal to previous
                    person with same social security number)))))
```

The first step in the make-person-function is to apply *process-attributes* to the abnormal version of the object being made (see abnormal objects in CREATE-OBJECT OPERATIONS above). E.g.–

```
(make-person-function <an-attribute-descriptor D> make-object
                      (list a-person has-social-security-number '|40-90-000|
                                     has-name 'George has-height 72 has-weight 200)
                      nil)
```

is a typical call to make-person-function, in which the third argument is an abnormal object. The *process-attributes* function alters the abnormal object by filling in default values of attributes not specified (using the *has-default-value* attributes of attribute-descriptors: see DEFAULT VALUES above). It also applies *has-init-functions*, if any, to non-default attribute values.

The modified abnormal object is passed to *create-object* to create the a-person object. The has-social-security-number is extracted from the resulting object, and checked to be sure it is a legal non-*nil* symbol (in FRANZ lisp, 40-90-000 is read as a symbol, even if not surrounded by vertical bars).

The social security number is supposed to be a unique identifier for the person, so in this example we simply set the symbol value of the social security number to the person object. Thus if there was a previous object with the same social security number, it can be found as the value of the social security number.

The rest of the function is as follows—

(1) If there is no previous object, we set the social security number symbol value to the new object, and return the new object.

(2) Otherwise, if the new object is a stub, we return the previous object.

(3) Otherwise, if the previous object is a stub, we move the new object into the previous object and return the latter.

(4) Otherwise, we compare the new and previous objects, and return the latter if the two objects are equal.

(5) Otherwise, we signal an error.

We also need to provide special *format-object* and *uneval-object* operations to output stubs such as—

(a-person has-social-security-number '|40-90-000|)

in place of the full object when the object is an attribute value of another object. The situation where the stub should be output is recognized by a non-zero level argument to *format-object*, or a non-*nil* index-switch argument to *uneval-object* (see the GLOSSARY).

23. C TYPES. The OBJECTS package contains some C language support code that is a continuation of the ATOMS package. This continuation makes use of ***a-type*** objects, which is why this code is not included in the ATOMS package to begin with (the ATOMS package does not depend upon the OBJECTS package).

C code can obtain a pointer to any object with a *has-name* attribute by calling—

sob_nobject ("<name>")

In particular, the system has already done this for basic *a-type* objects, such as *a-char*, and stored the results away in C global variables, such as *SOB_CHAR*. The following table lists all the global C variables set to *a-type* objects in this manner—

C Global Variable	SKETCH a-type Object	C Global Variable	SKETCH a-type Object
SOB_ATTRIBUTE	*an-attribute*	SOB_LONG	*a-long*
SOB_BIGNUM	*a-bignum*	SOB_LVECTOR	*a-lisp-vector*
SOB_BINARY	*a-binary-function*	SOB_NONLISP	*a-non-lisp-value*
SOB_CHAR	*a-char*	SOB_PORT	*a-port*
SOB_DOUBLE	*a-double*	SOB_SHORT	*a-short*
SOB_FIXNUM	*a-fixnum*	SOB_STRING	*a-string*
SOB_FLOAT	*a-float*	SOB_SYMBOL	*a-symbol*
SOB_FLONUM	*a-flonum*	SOB_TYPE	*a-type*
SOB_HUNK	*a-hunk*	SOB_UBIT	*a-ubit*
SOB_INT	*an-int*	SOB_UCHAR	*a-uchar*
SOB_IVECTOR	*an-immediate-vector*	SOB_ULONG	*a-ulong*
SOB_LARRAY	*a-lisp-array*	SOB_UNSIGNED	*an-unsigned*
SOB_LBIT	*an-lbit*	SOB_USHORT	*a-ushort*
SOB_LIST	*a-list*	SOB_VALUE	*a-value*

The C data type *sob_type* is defined to be a pointer to *a-type* object. The C data type *sob_attribute* is defined to be a pointer to *an-attribute* object.

There is a C equivalent of the *has-lisp-type* function: *sob_ltype*. This is actually a fast macro. An example of its use is—

if (*sob_ltype* (x) == *SOB_FIXNUM*) ... .

There is also a C function, *sob_tsize*, to get the *has-size* attribute of *a-type* object (the number of bits taken by a datum of the given type when it is an array element).

To allow code that deals with different types of numbers to use the C *case* statement (which can only test integers known at compile time, and cannot test pointers), there is a function, *sob_tcase*, that returns an integer code for numeric types. E.g.,

sob_tcase (*SOB_CHAR*)

returns an integer equal to the C manifest constant *SOB_CCASE*. The following is a table of the codes returned—

Ty_type Value	Code Returned	Numeric Type
SOB_UBIT	SOB_UBCASE	unsigned 1 bit integer
SOB_CHAR	SOB_CCASE	signed 8 bit integer
SOB_UCHAR	SOB_UCCASE	unsigned 8 bit integer
SOB_SHORT	SOB_SCASE	signed 16 bit integer
SOB_USHORT	SOB_USCASE	unsigned 16 bit integer
SOB_LONG *SOB_INT*	SOB_LCASE	signed 32 bit integer
SOB_ULONG *SOB_UNSIGNED*	SOB_ULCASE	unsigned 32 bit integer
SOB_FLOAT	SOB_FCASE	signed 32 bit floating point number
SOB_DOUBLE	SOB_DCASE	unsigned 64 bit floating point number

There is another function, *sob_tmissing*, what will return the missing value for particular numeric type, given the *sob_case* code of that type. The missing value returned is always a *double*. For example,

sob_tmissing (*SOB_CCASE*)

returns *SOB_CMISSING* cast to a *double*. For unsigned integers, *sob_tmissing* returns some value that can never be taken by the integer.

24. HITLIST.

(1) Possibly add inheritance under the constraint that the underlying format of related objects is the same.

(2) Possibly make compare-object really test equality of fixnums with flonums.

(3) Find out why (get-attribute xxx yyy) (type of yyy not available at compile time) is so slow and try to fix it. Also make compiled version more compact. Maybe use C code?

(4) Possibly make *has-vector-C-element-type* attribute be non-hidden.

(5) Consider not allocating separate hunk part of vector object with 2 element hunk part until property list is set (use constant hunk part in the meantime).

(6) Possibly return component length as part of pretty-print format, so better judgements can be made using prinlength.

(7) Possibly implement different levels of verbosity in printing.

(8) Possibly make sob_unbound handle abbreviations.

(9) Possibly disable special consideration of *has-name* by default *declare-hunk-type* and *declare-vector-type* provided *format-object* and *uneval-object* functions if its *has-is-a-stub-switch* is not *no*.

(10) Possibly outlaw allowing the parentheses to be omitted from (s_attribute), on the grounds that misspelled options (e.g. is-read instead of is-read-private) are mistaken for attributes.

25. GLOSSARY.

a-bignum	[SKETCH Type]
a-binary-function	[SKETCH Type]
a-char	[SKETCH Type]
a-double	[SKETCH Type]
a-fixnum	[SKETCH Type]
a-float	[SKETCH Type]
a-flonum	[SKETCH Type]
a-hunk	[SKETCH Type]
an-immediate-vector	[SKETCH Type]
an-int	[SKETCH Type]
an-lbit	[SKETCH Type]
a-lisp-array	[SKETCH Type]
a-lisp-vector	[SKETCH Type]
a-list	[SKETCH Type]
a-long	[SKETCH Type]
a-non-lisp-value	[SKETCH Type]
a-port	[SKETCH Type]
a-short	[SKETCH Type]
a-string	[SKETCH Type]
a-symbol	[SKETCH Type]
a-ubit	[SKETCH Type]
a-uchar	[SKETCH Type]
a-ulong	[SKETCH Type]
an-unsigned	[SKETCH Type]
a-ushort	[SKETCH Type]
a-value	[SKETCH Type]

USE: These are types of C and LISP values according to the following table—

a-bignum	LISP *bignum*: large integer.
a-binary-function	LISP *binary*: compiled function.
a-char	C *char*: 8 bit signed integer.
a-double	C *double*: 64 bit floating point number.
a-fixnum	LISP *fixnum*: small integer.
a-float	C *float*: 32 bit floating point number.
a-flonum	LISP *flonum*.
a-hunk	LISP *hunk0*, *hunk1*, ..., or *hunk6*.
an-immediate-vector	LISP *vectori*.
an-int	C *int*: 32 bit signed integer.
an-lbit	C 1 bit unsigned integer for which the value 0 denotes *nil* and the value 1 denotes *t*.
a-lisp-array	LISP *array*.
a-lisp-vector	LISP *vector*.
a-list	LISP *list*.
a-long	C *long*: 32 bit signed integer.
a-non-lisp-value	LISP *other*.
a-port	LISP I/O *port*.
a-short	C *short*: 16 bit signed integer.
a-string	LISP *string*.
a-symbol	LISP *symbol*.
a-ubit	C 1 bit unsigned integer.
a-uchar	C *uchar*: 8 bit unsigned integer.
a-ulong	C *ulong*: 32 bit unsigned integer.
an-unsigned	C *unsigned*: 32 bit unsigned integer.
a-ushort	C *ushort*: 16 bit unsigned integer.
a-value	LISP *value*.

"abnormal object" [SKETCH Term]

USE: An 'abnormal object' is a list which represents an object. The first list element is the type of the object, and the rest of the elements are attribute label/value pairs. The object type is itself ***a-type*** object, and not the symbol naming the type. Similarly the attribute labels are ***an-attribute*** objects, and not symbols.

An example of an abnormal object is—

(*list* a-person has-weight 99 has-age 13) (**abnormal-object-for-macro** '(*list* s_type s_attribute g_value ...)) [LISP Function]

RETURNS: The list–

(s_type s_attribute g_value ...)

if the argument is formatted as indicated and the type and all the attributes are represented by their names. In this case *get* may be applied to get g_value's from the returned list.

Otherwise returns *nil* (and does NOT call *error*).

(**an-attribute** *has-name* 's_name) [SKETCH Type Macro]
an-attribute [SKETCH Type]
at_ [SKETCH Argument Prefix]

USE: *An-attribute* serves as a label for an attribute value of a SKETCH object. See s_*attribute*.

Whenever an attribute is gotten, set, initialized, pretty-printed, unevaluated, or compared, the attribute and the type of the object being referenced are used together to find *an-attribute-descriptor* that specifies functions, macros, and parameters to do these tasks. See *an-attribute-descriptor*.

ARGUMENT PREFIX: Attribute arguments are indicated by the prefix *at_*.

HAS-NAME: Each attribute MUST have a name which is a symbol. By convention, this name should begin with an auxiliary verb or a preposition followed by a hyphen: e.g. *has-parent* and *has-parameters*. See *has-name*.

INDEXING: Whenever an attribute is indexed, the name of the attribute has its function definition set if it was previously *nil*. This is also done for stubs, as a stub may be a completely defined attribute.

(**an-attribute-descriptor** [*has-descriptor-type* 'ty_type] [SKETCH Type Macro]
[*has-descriptor-attribute* 'at_attribute]
[*has-functions* 'aft_attribute-function-table]
[*has-parameters* 'g_parameters]
[*has-info* 'g_info]
[*has-default-value* 'g_default-value]
[*has-is-a-stub-switch* 's_is-a-stub-switch]
[*has-compare-switch* 's_compare-switch]
[*has-format-switch* 's_format-switch]
[*has-uneval-switch* 's_uneval-switch])
an-attribute-descriptor [SKETCH Type]
atd_ [SKETCH Argument Prefix]
(**has-parent** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-descriptor-type** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-descriptor-attribute** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-functions** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-parameters** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-info** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-default-value** 'atd_descriptor) [SKETCH Attribute Macro]

(**has-is-a-stub-switch** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-compare-switch** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-format-switch** 'atd_descriptor) [SKETCH Attribute Macro]
(**has-uneval-switch** 'atd_descriptor) [SKETCH Attribute Macro]

USE ONLY WHEN: Defining non-standard SKETCH types and attributes.

USE: *An-attribute-descriptor* describes how a particular attribute, at_attribute, is stored in objects of a particular type, ty_type. At_attribute may be *nil* to indicate that the descriptor applies to all attributes of objects of type ty_type. Ty_type may be *nil* to indicate that the descriptor applies to at_attribute for all objects, regardless of type. At_attribute and ty_type may not both be *nil*.

See SEARCH ORDER below to find which attribute descriptor is used when several have the same at_attribute and ty_type.

ARGUMENT PREFIX: Attribute descriptor arguments are indicated by the prefix *atd_*.

HAS-DESCRIPTOR-ATTRIBUTE: At_attribute: *an-attribute* or *nil*.

HAS-DESCRIPTOR-TYPE: Ty_type: *a-type* or *nil*.

HAS-FUNCTIONS: *An-attribute-function-table* or *nil*. The functions and macros in this table are used to get values from, set values into, and check initial values of the attribute.

When two attribute descriptors are compared by *compare-object*, their *has-functions* attributes are compared by *compare-object* instead of by *equal*. This permits *an-attribute-descriptor* with a *has-name* to be repeatedly defined as long as all the definitions are the same, even if the *has-functions* attribute value does not itself have a name (see *stub*).

HAS-PARAMETERS: Any LISP value. Used by the *has-functions* functions and macros. Can be *setf*.

HAS-INFO: Just like *has-parameters* but is not visible: is not printed or represented in the unevaluated attribute descriptor. Useful for cross reference lists.

HAS-DEFAULT-VALUE: An evaluatable LISP expression. Evaluated when an object is made to provide a default value for the attribute. Cannot refer to local variables.

If initialized to *nil*, will be set to the default value of the parent of this descriptor (see below), if any.

HAS-IS-A-STUB-SWITCH: A symbol. If *no*, this attribute is not tested to see if it has any particular value (such as *nil*) in order to verify that an object is a stub. If *yes*, the attribute is tested by the *not* function, and must be *nil* if the object is a stub. If *nil*, no opinion on testing the attribute is expressed (if everyone expresses no opinion, the result is the equivalent of *yes*). If some other symbol, then this is the name of a function which is called in place of *not* to test the value of the attribute to see if it is acceptable for a stub.

If initialized to *nil*, will be set to the is-a-stub switch of the parent of this descriptor (see below), if any.

HAS-COMPARE-SWITCH: A symbol. If *no*, this attribute is not tested when two objects are compared for equality. If *yes*, the attribute values for the two objects are tested by the *equal* function if they are numbers, strings, or lists, and by the *eq* function otherwise. If *nil*, no opinion on testing the attribute is expressed (if everyone expresses no opinion, the result is the equivalent of *yes*). If some other symbol, then this is the name of a function which is called in place of *equal* or *eq* to test the two value of the attribute to see if they are equal.

If initialized to *nil*, will be set to the compare switch of the parent of this descriptor (see below), if any.

HAS-FORMAT-SWITCH: A symbol. If *no*, pretty-printing this attribute is suppressed. If *yes*, pretty-printing is required, and *pretty-format* is called to format the value of the attribute for printing. If *nil*, no opinion on printing is expressed (if everyone expresses no opinion, the result is the equivalent of *yes*). If some other symbol, then this is the name of a function which is called in place of *pretty-format* to format the value of the attribute. However, should this function return *nil*, the attribute will not be printed.

If initialized to *nil*, will be set to the format switch of the parent of this descriptor (see below), if any.

HAS-UNEVAL-SWITCH: A symbol. If *no*, inclusion of this attribute in the results of *uneval-object* is suppressed. If *yes*, inclusion is required. If *nil*, no opinion on inclusion is expressed (if everyone expresses no opinion, the result is the equivalent of *yes*). If some other symbol, then this is the name of a function which is called in place of *uneval-object* to unevaluated the value of the attribute. However, should this function return *nil*, the attribute will not be included in the results. The function may return '*nil* to force inclusion of the attribute with the value *nil*.

If initialized to *nil*, will be set to the uneval switch of the parent of this descriptor (see below), if any.

HAS-PARENT: *An-attribute-descriptor* or *nil*. Automatically set (may not be initialized or *setf*) to the last attribute descriptor indexed before this one which has the same at_attribute and either the same ty_type, or a type that is an ancestor of ty_type. The parent of an attribute descriptor, the parent's parent, the parent's parent's parent, etc. are said to be ancestors of the attribute descriptor.

SEARCH ORDER: When an attribute, at_attribute, is gotten from or set into an object of type ty_type, or set to an initial value when the object is made, *an-attribute-descriptor* must be found, and the appropriate *has-functions* function selected to perform the get, set, or init. A search is made of three groups of descriptors. Each group consists of all descriptors with particular values of their *has-descriptor-attribute* and *has-descriptor-type* attributes as follows—

group	*has-descriptor-attribute*	*has-descriptor-type*
1	at_attribute	ty_type or an ancestor of ty_type
2	at_attribute	*nil*
3	*nil*	ty_type or an ancestor of ty_type

The search examines each of the three groups in order. Each group is examined by examining all descriptors in the group in most-recently-made-first order. This can be done by examining first the most recently made descriptor in each group, called the head of the group, and then examining the head descriptor's parent, that parent's parent, and so forth, until all ancestors of the group head have been examined.

The search stops when a descriptor is found whose *has-get-function*, *has-set-function*, or *has-init-function* is non-*nil*.

When an object of type ty_type is made, a similar search is made for non-*nil* *has-default-value*'s. When an object is tested by *object-is-a-stub* or *compare-object*, a similar search is made for non-*nil* *has-is-a-stub-switch*'s or *has-compare-switch*'s. When an object is *pretty-print*'ed or *uneval-object*'ed, a similar search is made for non-*nil* *has-format-switch*'s or *has-uneval-switch*'s.

Sometimes a search does not begin at the beginning, but instead begins just after a particular descriptor in the order, thus in effect continuing the previous search which found that descriptor.

ORDER OF MAKING: The assumption is made that all attribute descriptors with the same *has-descriptor-type* and *has-descriptor-attribute* are made in the same order in both the compile and evaluation environments. The *get-attribute-descriptor* function and all the macros that use it depend upon this assumption.

It is an error to make a descriptor with a non-*nil* ty_type if ty_type is the ancestor of any other type. Thus all the descriptors for a type must be made before the type is made a parent of another type.

NIL DEFAULT VALUES: To set a *has-default-value* attribute to an expression which evaluates to *nil*, one must use

(*an-attribute-descriptor* ... *has-default-value* "*nil* ...)

This is necessary only if a non-*nil* default value from some ancestor of the descriptor must be overridden. If there are no non-*nil* default values, the default value will be *nil*.

INDEXING: *An-attribute-descriptor* may be indexed by a *has-name* attribute in the normal way. It is also referenced by a variety of indices which enable the search described above. Just before indexing,

(*execute-found-operation*
 (*find-operation-descriptor nil index-attribute-descriptor* ty_type)
 index-attribute-descriptor atd_descriptor ty_type)

is executed if the call to *find-operation* returns non-*nil*. The value returned by this operation replaces atd_descriptor as the value to be returned by the descriptor making operation. This value is the descriptor that is indexed, provided it is not a stub and has never been indexed before. The returned value may be a stub, or any attribute descriptor, either previously indexed, or never before indexed.

(an-attribute-function-table [SKETCH Type Macro]
 [*has-get-function* 's_get-function
  [*has-get-macro* 's_get-macro]]
 [*has-set-function* 's_set-function
  [*has-set-macro* 's_set-macro]]
 [*has-init-function* 's_init-function
  [*has-init-macro* 's_init-macro]])
an-attribute-function-table [SKETCH Type]
aft_ [Argument Prefix]
(**has-get-function** 'aft_table) [SKETCH Attribute Macro]
(**has-get-macro** 'aft_table) [SKETCH Attribute Macro]
(**has-set-function** 'aft_table) [SKETCH Attribute Macro]
(**has-set-macro** 'aft_table) [SKETCH Attribute Macro]
(**has-init-function** 'aft_table) [SKETCH Attribute Macro]
(**has-init-macro** 'aft_table) [SKETCH Attribute Macro]

USE ONLY WHEN: Defining non-standard SKETCH types and attributes.

ARGUMENT PREFIX: *An-attribute-function-table* arguments are indicated by the prefix *aft_*.

USE: *An-attribute-function-table* provides a set of functions and macros to access an attribute, at_attribute, of an object, ob_object, of a particular type, ty_type. Functions and macros are provided to get or set the attribute value, and to inspect and change initial values of the attribute.

Let s_attribute be the name of at_attribute, and s_type be the name of ty_type. Let atd_descriptor be the attribute descriptor which is to be used to access the attribute, and whose *has-functions* equals the attribute function table that is to supply the functions and macros. Then the function table has the following components, all of which are functions callable by *funcall* or macros callable by expansions of other macros.

HAS-GET-FUNCTION: Calls of the form—

(s_attribute ob_object ...)

and—

(*get-attribute* at_attribute ob_object ...)

may be computed by the call—

(*funcall* s_get-function atd_descriptor at_attribute ob_object ...)

HAS-GET-MACRO: Calls of the form—

(s_attribute (s_type ob_object ...) ...)

and—

(*get-attribute* s_attribute (s_type ob_object ...) ...)

may be expanded by macros into—

(s_get-macro atd_descriptor at_attribute (s_type ob_object ...) ...)

where at_attribute and atd_descriptor are pre-evaluated.

HAS-SET-FUNCTION: Calls of the form—

(*setf* (s_attribute ob_object ...) g_value)

and—

(*setf* (*get-attribute* at_attribute ob_object ...) g_value)

may be computed by the call—

(*funcall* s_set-function g_value atd_descriptor at_attribute
ob_object ...)

HAS-SET-MACRO: Calls of the form—

(*setf* (s_attribute (s_type ob_object ...) ...) g_value)

and—

(*setf* (*get-attribute* s_attribute (s_type ob_object ...) ...)
g_value)

may be expanded by macros into—

(s_set-macro g_value atd_descriptor at_attribute
(s_type ob_object ...) ...)

where at_attribute and atd_descriptor are pre-evaluated.

HAS-INIT-FUNCTION: Calls to make an object of type ty_type invoke either a *has-init-function* or a *has-init-macro* on all explicitly given initial attribute values for which these functions or macros are available. However, such calls are never made for default attribute values.

The call to a *has-init-function* has the form—

(*funcall* s_init-function g_value atd_descriptor at_attribute
ty_type)

The value returned by this call is used as the value to assign to the attribute.

HAS-INIT-MACRO: The call to a *has-init-macro* (see HAS-INIT-FUNCTION above) has the form—

(s_init-macro g_value atd_descriptor at_attribute ty_type)

where ty_type, at_attribute, and atd_descriptor are pre-evaluated. The value returned by this expression when it is evaluated is used as the value to assign to the attribute.

CONSTRAINT: If a macro attribute of *an-attribute-function-table* is non-*nil*, the associated function attribute must also be non-*nil*. The reverse is not true: if a macro is needed, and only a function is found, the function will be used in place of the macro.

(**an-operation** ***has-name*** 's_name [SKETCH Type Macro]
has-index-subscript 'x_index-subscript)
an-operation [SKETCH Type]
op_ [SKETCH Argument Prefix]
operation-index-size [LISP Global Variable]
(**has-index-subscript** 'op_operation) [SKETCH Attribute Macro]

USE: *An-operation* serves as a name for an operation that may be performed on different types of SKETCH object in a manner depending upon the type of the object operated on. See s_*operation*.

Whenever an operation is performed upon an object of a given type, the operation and type together are used to select *an-operation-descriptor* that supplies a function and additional parameters to that function to perform the operation. A macro may also be supplied to expand the operation more efficiently at compile time.

ARGUMENT PREFIX: Operation arguments are indicated by the prefix *op_*.

HAS-NAME: Each operation MUST have a name which is a symbol. By convention, this name usually contains an active verb: e.g., *make-object* and *format-object*. See ***has-name***.

HAS-INDEX-SUBSCRIPT: Each type has an operation index table associated with it which is used to more rapidly look up operation descriptors associated with the type. Operation objects can be assigned an integer, their ***has-index-subscript***, which is their subscript in these tables. Assigning such a subscript speeds up execution of the operation.

No two operations may have the same subscript, unless the two operations are never defined for the same type. On the other hand, if every operation were assigned a different subscript, the index tables would be exceptionally large.

If the ***has-index-subscript*** attribute is not initialized, its value becomes *nil*, and the operation is not speeded up. If the attribute is initialized to an integer, that integer is used as a subscript. If the attribute is initialized to a non-*nil*, non-integer value, the value of the global variable **operation-index-size** is becomes the actual value of the *has-index-subscript* attribute, and that variable is incremented by 1. The value of this variable is the size of any newly allocated index table. In any case, the value of this variable is maintained at one larger than the maximum of all operation *has-index-subscript* attributes.

Previously allocated tables are not increased in size, and therefore subscript assignment may not speed new operations added to old types. Index tables are not defined for a type until the first definition of an operation descriptor for the type is made. Thus it is desirable to define all operations used with a type before defining any operation descriptors for the type.

INDEXING: Whenever *an-operation* is indexed, the name of the operation has its function definition set if it was previously *nil*. This is also done for stubs, as a stub may be a completely defined operation. See s_*operation*.

Replacement of non-*nil*, non-integer *has-index-subscript* attributes and updating **operation-index-size** is also done at operation indexing time.

(**an-operation-descriptor** [*has-descriptor-type* 'ty_type] [SKETCH Type Macro]
has-descriptor-operation 'op_operation
has-function 's_operation-function
[*has-macro* 's_operation-macro]
[*has-parameters* 'g_parameters]
[*has-info* 'g_info])

an-operation-descriptor [SKETCH Type]
opd_ [SKETCH Argument Prefix]
(**has-parent** 'opd_descriptor) [SKETCH Attribute Macro]
(**has-descriptor-type** 'opd_descriptor) [SKETCH Attribute Macro]
(**has-descriptor-operation** 'opd_descriptor) [SKETCH Attribute Macro]
(**has-function** 'opd_descriptor) [SKETCH Attribute Macro]
(**has-macro** 'opd_descriptor) [SKETCH Attribute Macro]
(**has-parameters** 'opd_descriptor) [SKETCH Attribute Macro]
(**has-info** 'opd_descriptor) [SKETCH Attribute Macro]

USE ONLY WHEN: Defining non-standard SKETCH types and operations.

USE: *An-operation-descriptor* describes how a particular operation, op_operation, is executed for objects of a particular type, ty_type. Ty_type may be *nil* to indicate that the descriptor applies to op_operation for all objects, regardless of type. Op_operation may not be *nil*.

ARGUMENT PREFIX: Operation descriptor arguments are indicated by the prefix *opd_*.

HAS-DESCRIPTOR-OPERATION: Op_operation: *an-operation*.

HAS-DESCRIPTOR-TYPE: Ty_type: *a-type* or *nil*.

HAS-FUNCTION: A non-*nil* symbol. Calls such as those of the form—

(s_operation ob_object ...),

(*execute-operation* op_operation ob_object ...)

and

(*execute-found-operation* opd_descriptor op_operation ob_object ...)

may be computed by the call—

(*funcall* s_operation-function opd_descriptor op_operation ob_object ...)

HAS-MACRO: Calls such as those of the form—

(s_operation (s_type ob_object ...) ...)

(*execute-operation* op_operation (s_type ob_object ...) ...)

or

(*execute-found-operation* opd_descriptor op_operation
(s_type ob_object ...) ...)

may be expanded by macros into—

(s_operation-macro opd_descriptor op_operation (s_type ob_object ...) ...)

In this last call op_operation and opd_descriptor are pre-evaluated.

HAS-PARAMETERS: Any LISP value. Used by the *has-function* function and *has-macro* macro. Can be *setf*.

HAS-INFO: Just like *has-parameters* but is not visible: is not printed or represented in the unevaluated operation descriptor. Useful for cross reference lists.

HAS-PARENT: *An-operation-descriptor* or *nil*. Automatically set (may not be initialized or *setf*) to the last operation descriptor indexed before this one which has the same op_operation and either the same ty_type, or a type that is an ancestor of ty_type. The parent of an operation descriptor, the parent's parent, the parent's parent's parent, etc. are said to be ancestors of the operation descriptor.

SEARCH ORDER: When an operation, op_operation, is to be executed for an object of type ty_type, *an-operation-descriptor* must be found. A search is made of two groups of descriptors. Each group consists of all descriptors with particular values of their *has-descriptor-operation* and *has-descriptor-type* operations as follows—

group	*has-descriptor-operation*	*has-descriptor-type*
1	op_operation	ty_type or an ancestor of ty_type
2	op_operation	*nil*

The search examines each of the two groups in order. Each group is examined by examining all descriptors in the group in most-recently-made-first order. This can be done by examining first the most recently made descriptor in each group, called the head of the group, and then examining the head descriptor's parent, that parent's parent, and so forth, until all ancestors of the group head have been examined.

The search stops when a descriptor is found (all operation descriptors must have non-*nil has-function* attributes, so any operation descriptor examined will do).

Sometimes the search does not begin at the beginning, but instead begins just after a particular descriptor in the order, in effect resuming the previous search which found that descriptor.

ORDER OF MAKING: The assumption is made that all operation descriptors with the same *has-descriptor-type* and *has-descriptor-operation* are made in the same order in both the compile and evaluation environments. The *get-operation-descriptor* function and all the macros that use it depend upon this assumption.

It is an error to make *an-operation-descriptor* with a non-*nil* ty_type if ty_type is the ancestor of any other type. Thus all the operation descriptors for a type must be made before the type is made a parent of another type.

INDEXING: *An-operation-descriptor* may be indexed by a *has-name* operation in the normal way. It is also referenced by a variety of indices which enable the search described above. Just before indexing,

(*execute-found-operation*
(*find-operation-descriptor nil index-operation-descriptor* ty_type)
index-operation-descriptor opd_descriptor ty_type)

is executed if the call to *find-operation-descriptor* returns non-*nil*. The value returned by this operation replaces opd_descriptor as the value to be returned by the descriptor making operation. This value is the descriptor that is indexed, provided it is not a stub and has never been indexed before. The returned value may be a stub, or any operation descriptor, either previously indexed, or never before indexed.

(**s_attribute** 'ob_object ...) [SKETCH Attribute Macro]
(**s_attribute** (s_type ob_object ...) ...) [SKETCH Attribute Macro]

WHERE: S_attribute is the name of some SKETCH attribute, at_attribute: e.g. *has-name, has-functions*.

RETURNS: The value of the attribute labeled at_attribute for ob_object. If the attribute has never been assigned a value for the object, *nil* is returned.

WHEN SETF: The value of the attribute is changed.

EFFICIENCY: The form with (s_type ob_object ...) rather than just ob_object is often more efficient when compiled, because s_type tells the compiler the type of ob_object.

NOTE: By default, attributes can be initialized but not *setf*. Attributes that are otherwise are marked as such in documentation.

Attributes can be declared to have non-standard behaviors for certain types of object, and such behaviors will be documented.

EQUIVALENT TO: (*get-attribute* s_attribute ob_object ...)

However, it is permissible to override this definition by setting the function definition of s_attribute. The default macro definition of s_attribute will not replace an existing definition.

NOTE: All symbols beginning with an auxiliary verb (has, is, do, etc.) or preposition followed by a hyphen should name SKETCH attributes, and all SKETCH attributes should have names beginning with such prefixes.

(**a-type** *has-name* 's_name [SKETCH Type Macro]
[*has-size* 'x_size]
[*has-parameters* 'g_parameters]
[*has-info* 'g_info]
[*has-parent* 'ty_parent])
a-type [SKETCH Type]
ty_ [SKETCH Argument Prefix]
(**has-attribute-descriptors** 'ty_type) [SKETCH Attribute Macro]
(**has-operation-descriptors** 'ty_type) [SKETCH Attribute Macro]
(**has-allocation-count** 'ty_type) [SKETCH Attribute Macro]
(**has-children** 'ty_type) [SKETCH Attribute Macro]
(**has-size** 'ty_type) [SKETCH Attribute Macro]
(**has-parameters** 'ty_type) [SKETCH Attribute Macro]
(**has-info** 'ty_type) [SKETCH Attribute Macro]
(**has-parent** 'ty_type) [SKETCH Attribute Macro]

USE: *A-type* is the type of SKETCH objects. E.g., (*has-type an-attribute*) is *eq a-type* and so is (*has-type a-type*).

ARGUMENT PREFIX: Arguments with the *ty_* prefix are *a-type* values.

HAS-NAME: Type objects *MUST* have a name beginning with *a-* or *an-*: e.g., *an-operation*. See *has-name*.

HAS-SIZE: The size in bits of a datum of this type when it is an element of a vector or array. *Nil* if unknown or not useful.

HAS-PARAMETERS: Parameters for use by the operations on objects of the type. May be *setf*.

HAS-INFO: Just like ***has-parameters*** but is not visible: is not printed or represented in the unevaluated type object. Useful for cross reference lists.

HAS-PARENT: Another type used in place of this type if this type does not have some operation or attribute descriptor that is needed. E.g., this type inherits operations such as *create-object* from its parent.

A type's parent, its parent's parent, etc. are the ancestors of a type. A type inherits all the attribute descriptors and operation descriptors of its ancestors (see *an-attribute-descriptor* and *an-operation-descriptor*).

HAS-ALLOCATION-COUNT: The number of objects of this type that have been created.

If *nil*, this count is not maintained. Code that does not maintain this count can be slightly more efficient than code which does.

Defaults to *nil*. If initialized to any non-*nil* value, the value 0 will be substituted for the initial value. If initialized to a non-*nil* value in the compiler environment, must be initialized to a non-*nil* value in the evaluator environment.

Can be *setf*.

HAS-ATTRIBUTE-DESCRIPTORS: A list of the heads of all the attribute descriptor groups for this type (these are the groups labeled by 1 in the documentation of *an-attribute-descriptor*). Cannot be initialized.

HAS-OPERATION-DESCRIPTORS: A list of the heads of all the operation descriptor groups for this type (these are the groups labeled by 1 in the documentation of *an-operation-descriptor*). Cannot be initialized.

HAS-CHILDREN: A list of all the other types whose parents are this type. Cannot be initialized.

ORDER OF MAKING: All attribute descriptors and operation descriptors for a type must be made before the type is made a parent of any other type.

INDEXING: Whenever a type is indexed, the name of the type has its function definition set if it was previously *nil*. This is also done for stubs, as sometimes a stub is a completely defined type (it will cease to be a stub when operation or attribute descriptors are associated with it). See s_*type*.

(**a-vector-element-C-type** *has-parent-type* 'ty_parent-type] [LISP Function]
[*has-C-type-format* (g_C-type-format-part-1 ...)]
[*has-C-type-repeat-format* (g_C-type-repeat-format-part-1 ...)]
has-size x_size
has-alignment x_alignment
[*has-initial-value* g_initial-value]
has-get-function s_get-function
has-get-macro s_get-macro
has-set-function s_set-function
has-set-macro s_set-macro
[*has-parameters* g_parameters]
[*has-info* g_info])

a-vector-element-C-type [SKETCH type]
veCty_ [Argument Prefix]
(**has-parent-type** 'veCty_type) [SKETCH Attribute Macro]
(**has-C-type-format** 'veCty_type) [SKETCH Attribute Macro]
(**has-C-type-repeat-format** 'veCty_type) [SKETCH Attribute Macro]
(**has-size** 'veCty_type) [SKETCH Attribute Macro]
(**has-alignment** 'veCty_type) [SKETCH Attribute Macro]
(**has-initial-value** 'veCty_type) [SKETCH Attribute Macro]
(**has-get-function** 'veCty_type) [SKETCH Attribute Macro]
(**has-get-macro** 'veCty_type) [SKETCH Attribute Macro]
(**has-set-function** 'veCty_type) [SKETCH Attribute Macro]
(**has-set-macro** 'veCty_type) [SKETCH Attribute Macro]
(**has-paramters** 'veCty_type) [SKETCH Attribute Macro]
(**has-info** 'veCty_type) [SKETCH Attribute Macro]

USED ONLY WHEN: Defining new C data types for inclusion in *declare-vector-type* defined objects.

USE: *A-vector-element-C-type* defines a C data type that can be used for elements of *declare-vector-type* defined objects. Such a C data type corresponds to *a-type* object which has a *has-vector-element-C-type* attribute that is *a-vector-element-C-type* with descriptive information. The *a-type* object is used to denote the resulting type. Examples are *a-char* and *a-short*.

ARGUMENT PREFIX: Arguments with the *veCty_* prefix must be *a-vector-element-C-type* values.

HAS-PARENT-TYPE: This is the *a-type* object used to denote the *a-vector-element-C-type* object. The latter is the *has-vector-element-C-type* attribute of the former.

HAS-C-TYPE-FORMAT:

HAS-C-TYPE-REPEAT-FORMAT: '(g_type-format-part-1 ...) is a list of elements which may be *patom*'ed to declare a variable. The symbol *NAME* in this list is replaced by the name of the variable. No semicolon or carriage return should be included.

An example is—

(|short |*NAME*)

for *a-short*.

Has-C-type-repeat-format is the same thing, but is used in case there is an x_repeat-count (see *declare-vector-type*). The symbol *REPEAT* is replaced by the repeat count. For example—

(|xxx_alloc (| *NAME* |, | *REPEAT* |)|)

for a structure allocated by the xxx_alloc macro with a required repeat count (see C CODE SIDE EFFECTS under *declare-vector-type*).

An expression which is itself a list may be an element of these format lists, in which case the expression is not *patom*'ed, but rather is *eval*'ed with *NAME* bound to the variable name and *REPEAT* to the repeat count. and the result is *patom*'ed.

The symbol REPEAT is also recognized in *has-C-type-format* and replaced by 1. If *has-C-type-repeat-format* is *nil*, *has-C-type-format* will be used in its place.

The *has-C-type-format* and *has-C-type-repeat-format* are currently unused if the *has-size* attribute is less than 8. as the C language requires special treatment for fields.

HAS-SIZE: X_size is the number of bits in the element. This must be an integer above 0.

HAS-ALIGNMENT: X_alignment is a number which must exactly divide the displacement of the element in bits within any vector. It must equal 1, 2, 4, 8, 16, 32, or 64. X_size must be an exact multiple of x_alignment.

If x_size is less than 8, then it must equal x_alignment. If x_size is 8 or greater, then so must be x_alignment.

HAS-INITIAL-VALUE: The initial value, g_initial-value, may be stored into the vector using the s_set-function to get a default initial value appropriate for the element.

HAS-GET-FUNCTION:

HAS-GET-MACRO: S_get-function and s_get-macro may be used to return the value of the element in a vector. The calls are—

(*funcall* s_get-function veCt_type x_index V_vector ...)

and

(s_get-macro veCt_type x_index V_vector ...)

where veCt_type is the vector-element-C-type object in which s_get-function or s_set-macro was found, x_index is the displacement of the element within V_vector measured in units of x_alignment bits, V_vector is the immediate vector containing the element. and ... are any extra arguments that might be of use in selecting part of the element, instead of the whole element. The veCt_type argument to this macro is pre-evaluated.

HAS-SET-FUNCTION:

HAS-SET-MACRO: S_set-function and s_set-macro may be used to return the value of the element in a vector. The calls are—

(*funcall* s_set-function g_value veCt_type x_index V_vector ...)

and

(s_set-macro g_value veCt_type x_index V_vector ...)

where g_value is the value to be stored, veCt_type is the vector-element-C-type object in which s_set-function or s_set-macro was found, x_index is the displacement of the element within V_vector measured in units of x_alignment bits, V_vector is the immediate vector containing the element, and ... are any extra arguments that might be of use in selecting part of the element, instead of the whole element. The veCt_type argument to this macro is pre-evaluated.

HAS-PARAMETERS:

HAS-INFO: These attributes are parameters for s_get-function, s_get-macro, s_set-function, and s_set-macro. The *has-parameters* attribute is visible, and the *has-info* attribute hidden: otherwise there is no difference.

Both of these attributes can be *setf*.

NOTE: When included in *declare-vector-type* objects, an element may be included in both the vector and the hunk part of the object. If this is done, it is not permitted to set parts of the element, although parts can be read. It is important, in this case, that the the copy of the element stored in the hunk be read-only, like a number, as it may be shared: no attempt is made to copy it.

(**compare-object** 'ob_object-1 'ob_object-2) [SKETCH Operation Macro]
(**compare-object-function** 'ob_object-1 'ob_object-2) [LISP Function]
compare-object [SKETCH Operation]

RETURNS: Non-*nil* if ob_object-1 equals ob_object-1. Otherwise *nil*.

Compare-object and *compare-object-function* do the same things, except the first is a macro with in-line optimizations, and the second is a function with no optimizations. However, the second can be used as a *has-compare-switch* value, and in other places where only a function will do.

Compare-object is the same as *equal* when comparing numbers, strings, symbols, ports, and lists. Other objects are equal if the object types are *eq* and all the object attributes compare equal according to their *has-compare-switch* values. If no explicit *has-compare-switch* value is given for an attribute, or the value is given as *yes*, then the two attribute values are tested by *equal*. If the switch has the value *no* the attributes are not tested at all. If the switch has some other symbol as a value, that symbol is used in place of *equal* to test the attribute values for equality. See HAS-COMPARE-SWITCH under *an-attribute-descriptor*.

Non-standard equality tests may also be defined for any object type.

WARNING: *Equal* considers a *fixnum* and a *flonum* to be unequal, even if they have the same value. E.g., 1 does not *equal* 1.0.

EFFICIENCY: This macro compiles more efficient code if ob_object-1 or ob_object-2 is either a literal or an expression of the form—

(s_type ...)

whose type s_type is specified at compile time.

The code to test equality of numbers, strings, symbols, lists, and ports is compiled in-line.

(create-object '(ty_type at_attribute g_value ...) ['ob_prototype]) [SKETCH Operation Macro]

(create-parent-object 'opd_descriptor '(ty_type at_attribute g_value ...) ['ob_prototype]) [LISP Macro]

create-object [SKETCH Operation]

USE ONLY WHEN: *Create-parent-object* is used only when defining non-standard SKETCH types.

EQUIVALENT TO: *Make-object* and *make-parent-object*, except that

(1) Stubs are not handled.

(2) Objects returned are not indexed.

(3) Default values are not added to the at_attribute/g_value list if ob_prototype is *nil*.

(4) *Has-init-function*'s and *has-init-macro*'s are not invoked.

It is possible to get the effects mentioned in (3) and (4) by applying *process-attributes* or *process-attributes-for-macro* to the first argument before calling *create-object*.

(declare-hunk-type (s_type [s_C-type s_C-prefix]) [LISP Macro]
[s_attribute-visibility]
[*has-is-a-stub-switch* s_is-a-stub-switch]
[*has-compare-switch* s_compare-switch]
[*has-format-switch* s_format-switch]
[*has-uneval-switch* s_uneval-switch]
[s_attribute-protection] [*has-password* s_password]
[*has-allocation-count* g_allocation-count]
s_attribute-1
(s_attribute-2 g_default-value-2 [s_C-attribute-name-2])
...)

(define-hunk-type (*list* 'ty_type ['s_C-type 's_C-prefix]) [LISP Function]
['at_attribute-visibility]
[*has-is-a-stub-switch* 's_is-a-stub-switch]
[*has-compare-switch* 's_compare-switch]
[*has-format-switch* 's_format-switch]
[*has-uneval-switch* 's_uneval-switch]
['at_attribute-protection] [*has-password* 's_password]
[*has-allocation-count* 'g_allocation-count]
'at_attribute-1
(*list* 'at_attribute-2 'g_default-value-2 ['s_C-attribute-name-2])
...)

C-definition-code-port [LISP Global Variable]

WHERE: *Declare-hunk-type* and *define-hunk-type* take substantially the same arguments and do the same thing, except that the first is a macro that does not evaluate its arguments, and the second is a function that does. For the macro, types and attributes are specified by their symbol names, whereas for the function, the types and attributes themselves may be given. The function will accept stubs of types and attributes, and will also accept symbols naming types and attributes, making the stubs itself.

If s_C-type and s_C-prefix are omitted, (s_type) may be abbreviated to s_type and (*list* 'ty_type) may be abbreviated to 'ty_type. Similarly s_attribute-1 abbreviates (s_attribute-1), and 'at_attribute-1 abbreviates (*list* 'at_attribute-1).

For *define-hunk-type*, in what follows, s_type, s_attribute-visibility, s_attribute-protection, s_attribute-1, and s_attribute-2 are the names of ty_type, at_attribute-visibility, at_attribute-protection, at_attribute-1, and at_attribute-2.

The arguments consist of options (s_attribute-visibility, s_is-a-stub-switch, s_compare-switch, s_format-switch, s_uneval-switch, s_attribute-protection, and s_password, and g_allocation-count) and attributes. The options may be listed in any order, and may be repeated. Each option applies to all attributes following it, and supercedes any previous option of the same kind. G_allocation-count is an exception, and should appear at most once.

S_attribute-visibility specifies the print and uneval characteristics of attributes following it. The possible values of s_attribute-visibility are—

is-visible	Include in all *compare-object* tests, *pretty-print*'s, and *uneval-object*'s. This is the default effective at the beginning of the argument list. Equivalent to *has-compare-switch nil, has-format-switch nil*, and *has-uneval-switch nil*.
is-hidden	Do not include in *compare-object* tests, *uneval-object*'s, or *pretty-print*'s. Equivalent to *has-compare-switch no, has-format-switch no*, and *has-uneval-switch no*.

It is also possible to specify the compare, format, and uneval switches more explicitly using the *has-compare-switch*, *has-format-switch*, and *has-uneval-switch* options. These are useful for specifying function names for these switches: see *an-attribute-descriptor*.

S_attribute-protection specifies the protection of all the attributes following it, and is one of—

is-read-init	Readable by everyone, but not writable. Can be initialized. This is the default effective at the beginning of the argument list.
is-read-init-write	Readable and writable by everyone. Can be initialized.
is-private	Readable and writable only by calls of the form (s_attribute ob_object s_password ...) that contain s_password as the second argument. Cannot be initialized (i.e., will always be initially set to the default value).
is-read-private	Readable by everyone, but writable only by calls that contain s_password as the-second argument. Cannot be initialized (i.e., will always be initially set to the default value).
is-read-init-private	Readable by everyone, but writable only by calls that contain s_password as the-second argument. Can be initialized.

The *has-password* option specifies the password, s_password, to be used by all private attributes following this option.

S_password will be made into a global constant, and should follow the naming conventions for such (have *'s at the beginning and end).

The default value of g_allocation-count is non-*nil*, which enables maintenance of the *has-allocation-count* attribute of s_type. This attribute counts the number of objects of type s_type which have been created. A value of *nil* disables maintenance of the attribute.

An attribute may be specified either by a single s_attribute label, or by a list (s_attribute g_default-value [s_C-attribute-name]) in which the default value is an expression not referencing local variables which is to be evaluated and used as the value of the s_attribute attribute whenever a new object of type s_type is made and no explicit value is given for the attribute. Giving a default value of *nil* is the same as giving no default value at all.

S_C-type, s_C-prefix and s_attribute-name-2 are symbols used to generate C code: see C CODE SIDE EFFECTS below.

RETURNS: The *a-type* object made.

SIDE EFFECTS: Makes *a-type* named s_type whose objects are hunks with attribute elements s_attribute-1 .... *An-attribute* objects named s_attribute-1 ... are also made if they do not previously exist.

An-attribute-descriptor is made for s_type and each s_attribute-1. *An-operation-descriptor* is made for s_type and the each of the following operations: *make-object*, *create-object*, *object-is*, *object-is-a-stub*, *compare-object*, *move-object*, *uneval-object*, *format-object*.

The attribute elements specified in the call to *declare-hunk-type* are elements of hunks and are very quickly accessible by indexing (*cxr*). In addition, any other attributes not specified in the call to *declare-hunk-type* or *define-hunk-type* may be set for an object of type s_type, but these will be put on a property list for the object, and will not be accessed as efficiently.

A (*defvar* s_password 's_password) is generated for each password. It is important that a password evaluate to itself, so that s_password can be used as an argument to both macro and function calls.

C CODE SIDE EFFECTS: If the global variable **C-definition-code-port** is non-*nil*, if s_C-prefix is non-*nil*, and if *in-environment* is *nil* (we are not being loaded by an *environment* statement), then a structure definition will be written into **C-definition-code-port** (which must be *a-port*). This structure definition will have the form—

typedef struct <s_C-prefix>*struct* * <s_C-type>;
struct <s_C-prefix>*struct* {
    sat_lvalue <s_C-prefix>plist;
    sob_type <s_C-prefix>type;
    ...
    sat_lvalue <s_C-attribute-name-2>;
    ... };
#define <s_C-prefix>*alloc*(x,y) *struct* <s_C-prefix>*struct* (x) [y]

where the exact order of the structure element definitions will be implementation dependent.

NOTE: *Declare-hunk-type* expands into a call to *define-hunk-type* nested inside an *eval-when* (*compile eval load*).

NOTE: Attribute and operation descriptors can be made for a hunk type after the call to *declare-hunk-type* or *define-hunk-type* that makes the type. If a new descriptor is for an attribute or operation declared by the execution of *declare-hunk-type* or *define-hunk-type*, the descriptors declared by that execution will become ancestors of the new descriptor.

NOTE: A call to *declare-hunk-type* or *define-hunk-type* may be repeated more than once. Only the first call will make or change anything. Subsequent calls will merely test that they are essentially identical to the first call, and complain if they are not.

NOTE: The *compare-object*, *object-is-a-stub*, *format-object*, and *uneval-object* functions defined by *declare-hunk-type* or *define-hunk-type* use only the attributes actually stored in the objects, and get these attributes using the functions and macros defined by *declare-hunk-type* or *define-hunk-type*. Attribute descriptors not defined by *declare-hunk-type* or *define-hunk-type* are ignored for the purposes of getting these attributes. However, these latter attribute descriptors are not ignored for purposes of getting the necessary switchs: *has-compare-switch*, *has-is-a-stub-switch*, *has-format-switch*, and *has-uneval-switch*.

All attribute descriptors that provide switches for an object of a given type should be defined before the first object of that type is created. This is because optimizing information for performing operations such as *format-object* is computed at that time.

IMPLEMENTATION: The current implementation (which is subject to change) uses hunks that have two more elements than the number of attributes. The first two elements are used as the first cell of a disembodied property list. The object type is stored in the *first* element of the hunk, and a pointer to the first attribute label on the property list is stored in the *rest1* element of the hunk. The attributes s_attribute-1, ... are assigned to hunk elements with indices 2, 3, ..., in the order in which the attributes appear as arguments to *declare-hunk-type*.

Attributes made for a hunk type after the execution of *declare-hunk-type* or *define-hunk-type* will not be assigned to elements of the hunk, but will be put on the property list.

(declare-vector-type (s_type [s_C-type s_C-prefix]) [LISP Macro]

[*has-allocation-count* g_allocation-count]
[*has-C-type-vector-element-name* s_C-type-vector-element-name]
[*has-C-plist-vector-element-name* s_C-plist-vector-element-name]
[*has-C-vsize-vector-element-name* s_C-vsize-vector-element-name]
[*has-pointer-C-type* s_pointer-C-type]
[*has-allocate-C-type* s_allocate-C-type]

[s_attribute-type] [s_attribute-location] [s_attribute-visibility]
[*has-is-a-stub-switch* s_is-a-stub-switch]
[*has-compare-switch* s_compare-switch]
[*has-format-switch* s_format-switch]
[*has-uneval-switch* s_uneval-switch]
[s_attribute-protection] [*has-password* s_password]

[x_repeat-count] s_attribute-1
[x_repeat-count] (s_attribute-2 g_default-value-2 [s_C-attribute-name-2])
...)

(define-vector-type (*list* 's_type ['s_C-type 's_C-prefix]) [LISP Function]

[*has-C-type-vector-element-name* 's_C-type-vector-element-name]
[*has-C-plist-vector-element-name* 's_C-plist-vector-element-name]
[*has-C-vsize-vector-element-name* 's_C-vsize-vector-element-name]
[*has-pointer-C-type* 'ty_pointer-C-type]
[*has-allocate-C-type* 'ty_allocate-C-type]

['ty_attribute-type] ['at_attribute-location] ['at_attribute-visibility]
[*has-is-a-stub-switch* 's_is-a-stub-switch]
[*has-compare-switch* 's_compare-switch]
[*has-format-switch* 's_format-switch]
[*has-uneval-switch* 's_uneval-switch]
['at_attribute-protection] [*has-password* 's_password]
[*has-allocation-count* 'g_allocation-count]

['x_repeat-count] 'at_attribute-1
['x_repeat-count] (list 'at_attribute-2 'g_default-value-2
['s_C-attribute-name-2])
...)

C-definition-code-port [LISP Global Variable]
(has-vector-type 'ty_type) [SKETCH Attribute Macro]

WHERE: *Declare-vector-type* and *define-vector-type* take substantially the same arguments and do the same thing, except that the first is a macro that does not evaluate its arguments, and the second is a function that does. For the macro, types and attributes are specified by their symbol names, whereas for the function, the types and attributes themselves may be given. The function will accept stubs of types and attributes, and will also accept symbols naming types and attributes, making the stubs itself.

If s_C-type and s_C-prefix are omitted, (s_type) may be abbreviated to s_type, and (*list* 'ty_type) may be abbreviated to 'ty_type. Similarly s_attribute-1 abbreviates (s_attribute-1), and 'at_attribute-1 abbreviates (*list* 'at_attribute-1).

For ***define-vector-type***, in what follows, s_type, s_pointer-C-type, s_allocate-C-type, s_attribute-type, s_attribute-location, s_attribute-visibility, s_attribute-protection, s_attribute-1, and s_attribute-2 are the names of ty_type, ty_pointer-C-type, ty_allocate-C-type, ty_attribute-type, at_attribute-location, at_attribute-visibility, at_attribute-protection, at_attribute-1, and at_attribute-2.

The arguments consist of options (s_C-type-vector-element-name, s_C-plist-vector-element-name, s_C-vsize-vector-element-name, s_pointer-C-type, s_allocate-C-type, s_attribute-type, s_attribute-location, s_attribute-visibility, s_is-a-stub-switch, s_compare-switch, s_format-switch, s_uneval-switch, s_attribute-protection, s_password, and g_allocation-count) and attributes. Most options may be listed in any order, and may be repeated. Each option applies to all s_attribute's following it, and supercedes any previous option of the same kind. S_C-type-vector-element-name, s_C-plist-vector-element-name, s_C-vsize-vector-element-name, s_pointer-C-type, s_allocate-C-type, and g_allocation-count are exceptions, should appear at most once, and must appear before other arguments.

Declare-vector-type and *define-vector-type* define a type named s_type such that objects of that type are represented by the combination of an immediate vector and a hunk. Attributes may be assigned to either the vector or the hunk or both.

Attributes that hold pointers may be of type *a-value*. A-value attributes are usually assigned to both the vector and the hunk. They are assigned to the vector so they will be readily available to C functions, which are given a pointer to the vector when called with the object as a parameter, and they are assigned to the hunk so the garbage collector will know about them.

Other attributes, such as numeric ones, are commonly assigned only to the vector, and are stored as numbers proper, and not as pointers. However, the value ***nil*** may also be stored for signed numeric types by representing it as a special missing value.

S_attribute-type specifies the type of the attributes, and may be one of the following—

Number of Bits				
1	**8**	**16**	**32**	**64**
an-lbit *a-ubit*	*a-uchar* *a-char*	*a-ushort* *a-short*	*a-ulong* *a-long* *an-int* *an-unsigned* *a-float* *a-value*	*a-double*

When a value of one of these types is stored in the immediate vector, it is packed according to the associated C type (the C type associated with *a-value* is *sat_lvalue*, and the C type associated with either *an-lbit* or *a-ubit* is *unsigned:1*).

A-value is the default attribute type effective at the beginning of the argument list.

Additional types can be allowed as vector elements by defining *a-vector-element-C-type* object describing them. Also see s_pointer-C-type and s_allocate-C-type below (under C CODE SIDE EFFECTS). Types for which this has not been done can still be used as *in-hunk* elements (see next paragraph).

S_attribute-location is one of the following—

in-vector	Assign to the immediate vector only.
in-hunk	Assign to the hunk only.
in-one	Assign *a-value* attributes to the hunk and other attributes to the immediate vector.
in-default	Assign *a-value* attributes to both the immediate vector and the hunk, and other attributes to the immediate vector only. This is the default attribute location effective at the beginning of the argument list.
in-both	Assign attributes to both the immediate vector and the hunk.

S_attribute-visibility specifies the print and uneval characteristics of attributes following it. The possible values of s_attribute-visibility are—

is-visible	Include in all *compare-object* tests, *pretty-print*'s, and *uneval-object*'s. This is the default effective at the beginning of the argument list. Equivalent to *has-compare-switch nil*, *has-format-switch nil*, and *has-uneval-switch nil*.
is-hidden	Do not include in *compare-object* tests, *uneval-object*'s, or *pretty-print*'s. Equivalent to *has-compare-switch no*, *has-format-switch no*, and *has-uneval-switch no*.

It is also possible to specify the compare, print, and uneval switches more explicitly using the *has-compare-switch*, *has-format-switch*, and *has-uneval-switch* options. These are useful for specifying function names for these switches: see *an-attribute-descriptor*.

The is-a-stub switch can similarly be specified explicitly by *has-is-a-stub-switch*. This switch defaults to *nil* and is not affected by s_attribute-visibility.

S_attribute-protection specifies the protection of all the attributes following it, and is one of—

is-read-init	Readable by everyone, but not writable. Can be initialized. This is the default effective at the beginning of the argument list.
is-read-init-write	Readable and writable by everyone. Can be initialized.
is-private	Readable and writable only by references of the form (s_attribute ob_object s_password ...) that contain s_password as the second argument. Cannot be initialized (i.e. is always set to the default value on initialization).
is-read-private	Readable by everyone, but writable only by calls that contain s_password as the second argument. Cannot be initialized (i.e. is always set to the default value on initialization).
is-read-init-private	Readable by everyone, but writable only by calls that contain s_password as the second argument. Can be initialized.

The *has-password* option specifies the password, s_password, to be used by all private attributes following this option.

S_password will be made into a global constant, and should follow the naming conventions for such (have *'s at the beginning and end).

The default value of g_allocation-count is non-*nil*, which enables maintenance of the ***has-allocation-count*** attribute of s_type. This attribute counts the number of objects of type s_type that have been created. A value of *nil* disables maintenance of the attribute.

An attribute may be specified either by a single s_attribute label, or by a list (s_attribute g_default-value [s_C-attribute-name]) in which the default value is an expression not referencing local variables which is to be evaluated and used as the value of the s_attribute attribute whenever a new object of type s_type is made and no explicit value is given for the attribute. Giving a default value of *nil* is the same as giving no default value at all.

An attribute may have a repetition count, x_repeat-count, provided the attribute is in the vector but not the hunk. The C code version of the attribute will get the dimension specifier '[x_repeat-count]', and will therefore be repeated in the vector x_repeat-count times.

If the attribute is gotten a list of x_repeat-count element values will be returned, and such a list may be written to the attribute. If an extra argument x_N is supplied to the attribute access expression, the x_N+1'st element of the x_repeat-count elements will be accessed. E.g.–

(s_attribute V_object x_N)

accesses the x_N+1's s_attribute element of V_object.

S_C-type, s_C-prefix, and s_C-attribute-name are symbols used to generate C code: see C CODE SIDE EFFECTS below.

Elements less than 8 bits long may have an x_repeat-count, but any s_C-attribute-name will refer to only the first of the sequence of x_repeat-count elements, as C does not support vector indexing of such elements.

RETURNS: The *a-type* object made.

SIDE EFFECTS: Makes *a-type* named s_type whose objects are immediate vectors with attribute elements s_attribute-1 .... The property list of these vectors begins with a hunk that contains additional information about the object.

An-attribute objects named s_attribute-1 ... are also made if they do not previously exist.

An-attribute-descriptor is made for s_type and each s_attribute-1. *An-operation-descriptor* is made for s_type and each of the following operations: *make-object*, *create-object*, *object-is*, *object-is-a-stub*, *compare-object*, *move-object*, *uneval-object*, *format-object*.

The attribute elements specified in the call to *declare-vector-type* are elements of vectors and hunks, and are quickly accessible by indexing (see *vrefi*-xxx and *cxr*). In addition, any other attributes not specified in the call to *declare-vector-type* or *define-vector-type* may be set for an object of type s_type, but these will be put on a property list for the object, and will not be accessed as efficiently.

A (*defvar* s_password 's_password) is generated for each password. It is important that a password evaluate to itself, so that s_password can be used as an argument to both macro and function calls.

C CODE SIDE EFFECTS: If the global variable **C-definition-code-port** is non-*nil*, if s_C-prefix is non-*nil*, and if *in-environment* is *nil* (we are not being loaded by an *environment* statement) then a structure definition will be written into **C-definition-code-port** (which must be *a-port*). This structure definition will have the form—

```
typedef struct <s_C-prefix>struct * <s_C-type>;
struct <s_C-prefix>struct {
        union {
                int SOB_VSIZE [1];
                sat_lvalue * SOB_VPLIST [1];
                sob_type SOB_VTYPE; } SOB_VFIRST;
#       define <s_C-type-vector-element-name> SOB_VFIRST.SOB_VTYPE
#       define <s_C-plist-vector-element-name> SOB_VFIRST.SOB_VPLIST[-1][0]
#       define <s_C-vsize-vector-element-name> SOB_VFIRST.SOB_VSIZE[-2]
        ...
        <C-attribute-type-1> <s_C-attribute-name-1>;
        ...
#       define <s_C-attribute-name-2> SOB_VFIRST.SOB_VPLIST[-1] [<x2>]
        ...
        };
#define <s_C-prefix>alloc(x,y) struct <s_C-prefix>struct (x) [y]
```

The default value of s_C-type-vector-element-name is <s_C-prefix>type if s_C-prefix is non-*nil*, or *nil* otherwise. If s_C-type-vector-element-name is non-*nil*, it is the C structure element name of the first element of the C accessible vector which is defined to be *a-type* value designating the type of the vector. If s_C-type-vector-element-name is *nil*, this value will not be included as the first element of the vector (it can still be found by LISP via the property list element of the vector).

If s_C-type-vector-element-name is *nil*, the *SOB_VFIRST union* and all *#define*'s using it will be omitted. This means that C code will not be able the access the type or vsize of the vector or any part of the hunk.

S_C-plist-vector-element-name and s_C-vsize-vector-element-name are the names of the C structure elements that may be used to access the object property list and object vector size in bytes. They default to <s_C-prefix>plist and <s_C-prefix>vsize, respectively. If s_C-plist-vector-element-name is *nil*, its definition will be omitted (it will also be omitted if s_C-type-vector-element-name is *nil*). Similarly for s_C-vsize-vector-element-name.

If s_pointer-C-type is non-*nil*, it is taken as the name of *a-type* which can be used as an s_attribute-type for elements which are pointers to objects of type s_type. The default value of s_pointer-C-type is s_type. S_pointer-C-type may be used in the current declaration: i.e. the type may be defined in terms of itself. The associated C data type is s_C-type, which must be non-*nil*, or s_pointer-C-type will be ignored.

If s_allocate-C-type is non-*nil*, it is taken as the name of *a-type* which can be used as an s_attribute-type for vector elements which are direct inclusions of objects of type s_type. The <s_C-prefix>alloc macro is used in C for such inclusions. The default value of s_allocate-C-type is *nil*. It will be ignored if s_C-prefix is *nil*. S_allocate-C-type cannot be used in the current declaration.

<C-attribute-type-1> is an appropriate C data type, such as *long*, *short*, *uchar*, or *sat_lvalue*. S_C-attribute-name-2 is assumed here to be an attribute included in the hunk only, at position <x2> in the hunk.

Because the exact form of storage of a vector is subject to change, the definitions of SOB_VPLIST, SOB_VFIRST, SOB_VTYPE, SOB_VSIZE, and <s_C-prefix>vsize may be withdrawn. The other definitions may change though their usage should not.

If an attribute is present in both the vector and the hunk, only a way of accessing the vector attribute is provided.

With these definitions one can use statements such as—

```
<s_C-type> y = ...;
```

One should not set attributes from C code if they occur in both the vector and the hunk.

<S_c-prefix>alloc should not be used if any reference to the hunk part of the resulting object is required, as it does not

allocate the hunk.

HAS-VECTOR-TYPE: Ty_allocate-C-type and ty_pointer-C-type, if given, are assigned the *has-vector-type* attribute value ty_type. If ty_type is the same as ty_pointer-C-type, it is given itself as its *has-vector-type* attribute.

NOTE: *Declare-vector-type* expands into a call to *define-vector-type* nested inside an *eval-when* (*compile eval load*).

NOTE: Attribute and operation descriptors can be made for a vector type after the call to *declare-vector-type* or *define-vector-type* that makes the type. If a new descriptor is for an attribute or operation declared by the execution of *declare-vector-type* or *define-vector-type*, the descriptors declared by that execution will become ancestors of the new descriptor.

IMPLEMENTATION: The current implementation (which is subject to change) represents the object by an immediate vector whose property (see *vprop*) is a hunk. The *first* element of the hunk is set to the object type, and the *rest1* element is set to the property list for the object. The rest of the hunk elements are the attributes assigned to the hunk in the order of their appearance in the call to *declare-vector-type* or *define-vector-type*.

The first 4 bytes of the immediate vector optionally hold a pointer to the object type (a copy of the *first* element of the hunk). The remaining bytes hold the attributes assigned to the vector, in the order of their appearance in the call to *declare-vector-type* or *define-vector-type*. Each attribute is aligned by inserting zero padding so that its displacement within the vector is an exact multiple of its length. 1-bit attributes are assigned from the high order bits to the low order bits within one byte.

Attributes made for a vector type after the execution of *declare-vector-type* or *define-vector-type* will not be assigned to elements of the vector or hunk, but will be put on the property list.

(define-attribute 's_name) [LISP Function]

USE ONLY WHEN: Using *define-object-name-prefix*.

EQUIVALENT TO: (*an-attribute has-name* s_name)

(define-object-name-prefix 's_prefix 's_function) [LISP Function]

USE ONLY WHEN: Adding a new attribute name prefix for attribute names that will be used in data bases.

SIDE EFFECT: Specifies that whenever the value of an unbound symbol beginning with s_prefix is gotten, s_function will be called with the symbol as its only argument in order to bind the symbol. Similarly, if the symbol has a *nil* function definition and is called or *setf*, s_function is called (the *cmacro* property of the function should also be *nil*, or it may be used as the function definition).

If **s_function** is *nil*, no function will be called for the prefix.

NOTE: The default object name prefixes include—

a-	define-type
an-	define-type
has-	define-attribute
do-	define-attribute
dont-	define-attribute
is-	define-attribute
isnt-	define-attribute

As a general rule, any auxiliary verb (has, have, do, is, should, ...) or preposition (to, by, ...) followed by a hyphen may be declared a prefix for attribute names.

NOTE: In order to make s_prefix indicate that a symbol is *a-type* name (like *a-* and *an-*), s_function should be *define-type*. In order to make s_prefix indicate that a symbol is *an-attribute* name (like *has-*), s_function should be *define-attribute*.

(define-type 's_name) [LISP Function]

USE ONLY WHEN: Using *define-object-name-prefix*.

EQUIVALENT TO: (*a-type has-name* s_name).

(equal-property-lists 'l_list-1 'l_list-2) [LISP Function]

WHERE: Both l_list-1 and l_list-2 are assumed to have an even number of elements and be organized as attribute label/value pairs, where no attribute label appears twice. It is assumed that no attribute value is *nil* in either list.

RETURNS: Non-*nil* if the attributes of l_list-1 and l_list-2 are *equal*. Note that the attribute labels are compared using *eq* instead of *equal*.

(**equal-property-lists-with-switches** 'l_list-1 'l_list-2 'l_info) [LISP Function]

USE ONLY WHEN: Building new object subpackages: like those of *declare-hunk-type* or *declare-vector-type*.

WHERE: L_list-1 and l_list-2 are property lists each with an even number of elements and no attribute whose value is *nil*. L_info is the value returned by—

(*get-switch-info* ty_type ... #'*get-compare-switch*)

and is used to quickly find the value of—

(*get-compare-switch* at_attribute ty_type)

for any **at_attribute** that can be in either l_list-1 or l_list-2.

WARNING: The types of the arguments are not checked.

RETURNS: *Nil* if l_list-1 and l_list-2 have an unequal attribute, and *t* otherwise. Equality of the two values of the attribute labeled at_attribute-1 is tested according to the value of—

(*get-compare-switch* at_attribute ty_type)

as computed using the third argument to *equal-property-lists-with-switches*. If this switch is *yes* or *nil*, the two values are tested with *equal*. If it is *no*, the two values are presumed equal no matter what their actual values are: i.e. the test for equality is skipped over. If it is any other value, its is called in place of the *equal* function to test the two values for equality.

SIDE EFFECT: If an attribute is not found in the third element of l_info, it is found by calling *get-switch-from-info* which adds the attribute to the third element of l_info.

(**execute-operation** 'op_operation 'ob_object ...) [LISP Macro]
(**execute-found-operation** 'opd_descriptor 'op_operation ...) [LISP Macro]
(**execute-parent-operation** 'opd_descriptor 'op_operation 'ob_object ...) [LISP Macro]
(**lexpr-execute-found-operation** 'opd_descriptor 'op_operation ...) [LISP Macro]
(**lexpr-execute-parent-operation** 'opd_descriptor 'op_operation 'ob_object ...) [LISP Macro]

USE ONLY WHEN: *Execute-found-operation*, *execute-parent-operation*, *lexpr-execute-found-operation*, and *lexpr-execute-parent-operation* are used only when defining non-standard SKETCH operations.

WHERE: The last argument to *lexpr-execute-found-operation* or *lexpr-execute-parent-operation* is treated as the last argument to *lexpr-funcall*, namely, as a list of the remaining arguments necessary to make a call to *execute-found-operation* or *execute-parent-operation*.

Op_operation and opd_descriptor may be pre-evaluated.

RETURNS: The value of the operation op_operation applied to the the arguments with opd_descriptor and op_operation omitted. Except for (*lexpr-*)*execute-found-operation*, the argument ob_object is necessary to provide a type used in finding the operation descriptor needed to execute the operation.

EFFICIENCY: These macros may find the operation descriptor required to execute the operation at macro expansion time, and produce much more efficient compiled code, if op_operation is the name, s_operation, of *an-operation*, if ob_object is an expression of the form

(s_type ...)

where s_type is the name of *a-type*, and if opd_descriptor is pre-evaluated or is the name, s_descriptor, of an operation descriptor.

If the descriptor can be determined at macro expansion time, and a *has-macro* attribute is available from the descriptor, then that macro can be invoked to get even further efficiency. However, this macro cannot be invoked by the *lexpr* forms of the above macros.

DESCRIPTOR SEARCH: *Execute-found-operation* performs the operation using the given opd_descriptor, which was presumably found by calling *find-operation-descriptor*.

Execute-operation searches for an appropriate descriptor by calling—

(*find-operation-descriptor nil* op_operation (*has-type* ob_object)).

Execute-parent-operation searches for an appropriate descriptor, starting with the parent of opd_descriptor, by calling—

(*find-operation-descriptor* opd_descriptor op_operation
(*has-type* ob_object)).

EQUIVALENT TO:

(*execute-operation* ...)

is equivalent to—

(*execute-parent-operation nil* ...).

(**find-get-attribute-descriptor** 'atd_descriptor 'at_attribute 'ty_type) [LISP Function]

(**find-set-attribute-descriptor** 'atd_descriptor 'at_attribute 'ty_type) [LISP Function]

(**find-get-attribute-descriptor-for-macro** 'g_descriptor 'g_attribute 'g_type) [LISP Function]

(**find-set-attribute-descriptor-for-macro** 'g_descriptor 'g_attribute 'g_type) [LISP Function]

WHERE: For the *for-macro* functions the arguments are macro expansion time expressions which will evaluate at eval time into arguments for the non-*for-macro* versions of these functions.

RETURNS: *An-attribute-descriptor* with a non-*nil has-get-function* (for *find-get-attribute-descriptor*) or *has-set-function* (for *find-set-attribute-descriptor*) in its has-functions table. This descriptor is found by searching using at_attribute and ty_type, starting the search with the parent of atd_descriptor. If atd_descriptor is *nil*, all applicable descriptors are searched. *Nil* is returned if the descriptor cannot be found.

It is permissible for the at_attribute or ty_type arguments to be *nil*, indicating an absence of information. In this case it may not be possible to complete the search, and *nil* will be returned to indicate this fact.

The *for-macro* versions of these functions assume that they are running at macro expansion time and have been passed the expressions which are to be evaluated at eval time in a call to the associated non-*for-macro* function. The *for-macro* functions attempt to deduce at macro expansion time what the result will be at eval time, and return that result if they can make the deduction. They return *nil* if they cannot make the deduction. They apply *object-expression-is* to their type, attribute, and attribute descriptor arguments, and thus understand arguments which are names of descriptors, attributes, or types, arguments which are pre-evaluated, and arguments of the form—

(*has-type* (s_type ...)).

NOTE: If ty_type is a *a-type* stub, the call—

(*define-type-stub* ty_type)

is made to fill it in.

(**find-operation-descriptor** 'opd_descriptor 'op_operation 'ty_type) [LISP Macro]
(**find-operation-descriptor-for-macro** 'opd_descriptor 'op_operation 'ty_type) [LISP Function]

RETURNS: *An-operation-descriptor* found by searching using op_operation and ty_type, starting the search with the parent of opd_descriptor. If opd_descriptor is *nil*, all applicable descriptors are searched. *Nil* is returned if the descriptor cannot be found.

Ty_type or op_operation may be *nil* to indicate lack of information. In this case it may not be possible to find the descriptor, and *nil* will be returned, even though a descriptor could be found if both ty_type and op_operation were known.

The *for-macro* version of this function assumes that it is running at macro expansion time and has been passed the expressions which are to be evaluated at eval time in a call to the associated non-*for-macro* function. The *for-macro* function attempts to deduce at macro expansion time what the result will be at eval time, and return that result if it can make the deduction. It returns *nil* if it cannot make the deduction. It applies *object-expression-is* to its type, operation, and operation descriptor arguments, and thus understands arguments which are names of descriptors, operations, or types, arguments which are pre-evaluated, and arguments of the form—

(*has-type* (s_type ...)).

(**format-object** 'ob_object 'x_level) [LISP Macro]

WHERE: Where x_level (0, 1, 2, ...) is the depth of parentheses or brackets within which ob_object is being printed.

RETURNS: A LISP value which when *pretty-print-format*'ed will print a representation of ob_object on an appropriate number of lines with appropriate indentation. The format of the returned value is discussed under *pretty-print-format*.

All SKETCH objects indexed by their *has-name* attribute will be represented by their index, the symbol which is the value of that attribute. If x_level is 0, an exception will be made, and ob_object will be represented as a type and **attribute list** in the usual way.

Other kinds of indexing may or may not behave similarly.

EFFICIENCY: This macro compiles more efficient code if ob_object is an expression of the form—

(s_type ...)

whose type s_type is specified at compile time.

HAS-FORMAT-SWITCH: If an attribute, at_attribute, of an object of type ty_type has a non-*nil* value of—

(*get-format-switch* ty_type at_attribute),

then in any call to *format-object*, this value will control inclusion of the attribute in the format returned. If the switch is *no*, the attribute will not be included. If the switch is *yes* or *nil*, the attribute will be included, and its value will be formatted by calling *pretty-format*. If the switch is another symbol, that symbol will be taken as the name of a function to be called in place of *pretty-format* to format the attribute value. The value returned by this function will be used as the format of the attribute value inside the object format, unless this returned value is *nil*, in which case the attribute will not be included in the object format. See HAS-FORMAT-SWITCH under *an-attribute-descriptor*.

(**get-attribute** 'at_attribute 'ob_object ...) [LISP Macro]
(**get-found-attribute** 'atd_descriptor 'at_attribute 'ob_object ...) [LISP Macro]
(**get-parent-attribute** 'atd_descriptor 'at_attribute 'ob_object ...) [LISP Macro]
(**lexpr-get-found-attribute** 'atd_descriptor 'at_attribute 'ob_object ...) [LISP Macro]
(**lexpr-get-parent-attribute** 'atd_descriptor 'at_attribute 'ob_object ...) [LISP Macro]

USE ONLY WHEN: *Get-found-attribute*, *get-parent-attribute*, *lexpr-get-found-attribute*, and *lexpr-get-parent-attribute* are only used when defining non-standard SKETCH attributes.

WHERE: The last argument to *lexpr-get-found-attribute* and *lexpr-get-parent-attribute* is treated as the last argument to *lexpr-funcall*, namely as a list of the remaining arguments necessary to make a call to *get-found-attribute* or *get-parent-attribute*.

At_attribute and atd_descriptor may be pre-evaluated.

RETURNS: The value of the attribute labeled at_attribute for ob_object. If the attribute has never been assigned a value for the object, *nil* is returned. See DESCRIPTOR SEARCH below to determine which attribute descriptor is used to get the attribute value.

WHEN SETF: The value of the attribute is changed. See DESCRIPTOR SEARCH below to determine which attribute descriptor is used to set the attribute value.

EFFICIENCY: These macros may find the attribute descriptor they need to get or set the attribute value at macro expansion time, and be much more efficient when compiled, if at_attribute is the name s_attribute of *an-attribute*, ob_object is an expression of the form—

(s_type ...)

where s_type is the name of a type, and atd_descriptor is the name, s_descriptor, of a descriptor, or is pre-evaluated.

If the descriptor can be determined at macro expansion time, then the *has-get-macro* or *has-set-macro* macros for the attribute may be invoked if they are defined, to get even further efficiency (see *an-attribute-function-table*). However these latter macros cannot be invoked by the *lexpr* forms of the above macros.

DESCRIPTOR SEARCH: *Get-found-attribute* gets the attribute using the given atd_descriptor. This descriptor must have an associated *has-get-function*.

Get-attribute searches for an appropriate descriptor by calling—

(*find-get-attribute-descriptor nil* at_attribute (*has-type* ob_object)).

Get-parent-attribute searches for an appropriate descriptor, starting with the parent of the atd_descriptor argument, by calling—

(*find-get-attribute-descriptor* atd_descriptor at_attribute
(*has-type* ob_object)).

If an attribute is being set, instead of gotten, *find-set-attribute-descriptor* is used instead of *find-get-attribute-descriptor* to find descriptors.

(**get-attribute-descriptor** 'atd_descriptor) [LISP Function]

USE ONLY WHEN: Defining non-standard SKETCH types and attributes.

RETURNS: An expression which evaluates to atd_descriptor in the eval environment. This is non-trivial when *get-attribute-descriptor* is called at compile time.

If atd_descriptor has a non-*nil* *has-name* attribute, that attribute is returned as the value of *get-attribute-descriptor* at compile time.

WARNING: The assumption is made that all the descriptors with the same *has-descriptor-type* and *has-descriptor-attribute* as atd_descriptor are made in the same order in both the compile and evaluation environments.

WARNING: Although *get-attribute-descriptor* performs the same function as a macro, it cannot be called like a macro. The reason it is not a macro is that some dumb macro expanders exist which will recursively expand top level macros but will not expand arguments to functions, and would end up trying to output attribute descriptors as literals in program binaries.

(**get-default-value** 'at_attribute 'ty_type ['atd_descriptor]) [LISP Function]
(**get-is-a-stub-switch** 'at_attribute 'ty_type ['atd_descriptor]) [LISP Function]
(**get-compare-switch** 'at_attribute 'ty_type ['atd_descriptor]) [LISP Function]
(**get-format-switch** 'at_attribute 'ty_type ['atd_descriptor]) [LISP Function]
(**get-uneval-switch** 'at_attribute 'ty_type ['atd_descriptor]) [LISP Function]

USE ONLY WHEN: Defining non-standard SKETCH types and attributes.

RETURNS: The first non-*nil* *has-default-value*, *has-is-a-stub-switch*, or *has-compare-switch* *has-format-switch*, or *has-uneval-switch* obtained by searching the attribute descriptors in the order indicated under *an-attribute-descriptor*. Returns *nil* if no non-*nil* value found.

If the atd_descriptor argument is present and non-*nil*, only descriptors after atd_descriptor in the search order are searched.

(**get-operation-descriptor** 'opd_descriptor) [LISP Function]

USE ONLY WHEN: Defining non-standard SKETCH types and operations.

RETURNS: An expression which evaluates to opd_descriptor in the eval environment. This is non-trivial when *get-operation-descriptor* is called at compile time.

If opd_descriptor has a non-*nil* *has-name* attribute value, that value is returned as the value of *get-operation-descriptor* at compile time.

WARNING: The assumption is made that all the operation descriptors with the same *has-descriptor-type* and *has-descriptor-operation* as opd_descriptor are made in the same order in both the compile and evaluation environments.

WARNING: Although *get-operation-descriptor* performs the same function as a macro, it cannot be called like a macro. The reason it is not a macro is that some dumb macro expanders exist which will recursively expand top level macros but will not expand arguments to functions, and would end up trying to output attribute descriptors as literals in program binaries.

(**get-switch-from-info** 'at_attribute 'l_info s_switch) [LISP Macro]

USE ONLY WHEN: Building new object subpackages: like those of *declare-hunk-type* or *declare-vector-type*.

WHERE: L_info is a result returned by *get-switch-info* and has the form—

(ty_type (s_switch-1 ...) ((at_attribute s_switch-2) ...) s_default-switch)

and s_switch is one of the symbols—

has-is-a-stub-switch
has-compare-switch
has-format-switch
has-uneval-switch

Note that s_switch is an unevaluated argument.

RETURNS: The switch value obtained by searching the third element of l_info, the list—

((at_attribute s_switch-2) ...)

for a match to at_attribute. If not found, the switch is retrieved by invoking an abbreviated form of the *get-xxx-switch* function (where s_switch equals has-xxx) which bypasses the search for group 1 attribute descriptors (those with both type and attribute specified: see SEARCH ORDER under *an-attribute-descriptor*) and uses the fourth element of l_info in place of the search for group 3 descriptors (those with the given type but any attribute). The switch found by this method, even if *nil*, is pushed onto the third element of l_info, so it will be found the next time it is used.

All this is done with in-line code for speed.

(**get-switch-info** 'ty_type '(atd_descriptor ...) 'u_get-switch-function) [LISP Function]

USE ONLY WHEN: Building new object subpackages: like those of *declare-hunk-type* or *declare-vector-type*.

WHERE: U_get-switch-function is #'*get-uneval-switch*, #'*get-format-switch* or some similar function.

WARNING: The types of the arguments are not checked.

RETURNS: A list (called the switch info) of the form—

(ty_type (s_switch-1 ...) ((at_attribute s_switch-2) ...) s_default-switch)

The sublist (s_switch-1 ...) corresponds to the list—

(atd_descriptor ...)

with s_switch equal to—

(*funcall* u_get-switch-function (*has-descriptor-attribute* atd_descriptor) ty_type).

The sublist—

((at_attribute s_switch-2) ...)

is made by taking for each descriptor D in the list—

(*has-attribute-descriptors* ty_type),

the attribute A equal to—

(*has-descriptor-attribute* D),

and the pair P equal to—

'(,A ,(*funcall* u_get-switch-function A ty_type)),

and including P in the output list if the attribute A is not the attribute of any element of the

(atd_descriptor ...)

list.

The s_default-switch is the value of the switch found by using u_get-switch-function to search for a switch ignoring group 1 and 2 attribute descriptors (see SEARCH ORDER under *an-attribute-descriptor*).

(**has-lisp-type** g_value) [LISP Function]

RETURNS: The SKETCH type corresponding to the LISP type of g_value, according to the following table—

LISP TYPE	SKETCH TYPE	LISP TYPE	SKETCH TYPE
fixnum	**a-fixnum**	*binary*	**a-binary-function**
bignum	**a-bignum**	*value*	**a-value**
flonum	**a-flonum**	*hunk0*	**a-hunk**
string	**a-string**	*hunk1*	**a-hunk**
symbol	**a-symbol**	*hunk2*	**a-hunk**
port	**a-port**	*hunk3*	**a-hunk**
list	**a-list**	*hunk4*	**a-hunk**
vector	**a-lisp-vector**	*hunk5*	**a-hunk**
vectori	**an-immediate-vector**	*hunk6*	**a-hunk**
array	**a-lisp-array**		

NOTE: The type returned is not the type of g_value as a SKETCH object, but rather its type as a LISP object. Thus—

(*has-lisp-type an-attribute*)

would be something like *a-hunk*, depending on implementation, whereas—

(*has-type an-attribute*)

would always be *a-type*.

(has-name 'ob_object) [SKETCH Attribute]
make-name-function [LISP Function Name]
make-name-macro [LISP Macro Name]

VALUE: (*has-name* ob_object) is the name of ob_object. This is a symbol whose value is always equal ob_object, provided the symbol is not *nil*.

WHEN SETF: (*has-name* ob_object) cannot be *setf*.

INDEXING: For most object types, an object made by the call—

(s_type *has-name* 's_name ...)

is indexed by the symbol s_name's being set equal to the object.

STUBS: A SKETCH object with its *has-name* attribute a non-*nil* symbol, but no other attribute non-*nil*, is called a stub.

If a stub is made by a call such as—

(s_type *has-name* 's_name),

and an object with the same *has-name* already exists, the stub is discarded, and the previously existing object returned as the result of making the object.

If an object which is not a stub is made, and a stub already exists with the same *has-name*, then the attributes of the existing stub are set to those of the newly created object, the newly created object is discarded, and the existing object, now no longer a stub, is returned as the result of the call making the object.

If an object which is not a stub is made, and another object not a stub already exists with the same *has-name*, then the two objects are tested for equality by *compare-object*, and, if equal, the new object is discarded, and the existing object returned as the result of the call making the object. It is an error if the objects are not equal.

This behavior may be modified for some types of objects, in which case the modified behavior is documented.

COMPILE TIME DECLARATIONS: Making an object with a non-*nil* symbol *has-name* attribute at compile time causes the symbol to be declared to be special.

NON-SYMBOL VALUES: *Has-name* attribute values must normally be symbols.

IMPLEMENTATION: *Has-name* indexing is implemented by *make-name-function* and *make-name-macro* which are used as the *make-object an-operation-descriptor has-function* and *has-macro* values for basic data types (e.g. those defined by *declare-hunk-type*).

The default *has-is-a-stub-switch* value for *has-name* is *no*, while the default *has-set-function* makes setting a *has-name* attribute illegal, and the default *has-init-function* makes it illegal to initialize a *has-name* attribute to a non-symbol value.

(**has-size** 'ty_type) [SKETCH Attribute]

VALUE: The length in bits of a datum of type ty_type from the point of view of the C language. E.g., (*has-size a-value*) is 32. Used in allocating arrays of objects of the given type.

(**has-type** 'ob_object) [LISP Function]
(**has-type** 'g_object) [LISP Function]
has-type [SKETCH Attribute]

USE: The *has-type* attribute value of a SKETCH object, ob_object, is the type of ob_object, and specifies the format of the object.

The *has-type* function applied to any LISP object, g_object, which is not a SKETCH object, will return the value returned by *has-lisp-type*.

INITIALIZATION: It is illegal to initialize the *has-type* attribute in the way that other attributes are initialized.

WHEN SETF: It is illegal to *setf* the *has-type* attribute.

(**is-typed-expression** 'g_expression) [LISP Macro]

USE ONLY WHEN: Defining non-standard SKETCH types.

RETURNS: The type named by s_type if g_expression has the form—

(s_type ...)

where s_type is the name of *a-type*. Otherwise returns *nil*.

"**make**" [SKETCH Term]
"**create**" [SKETCH Term]
"**index**" [SKETCH Term]

USE: Creating an object and indexing an object are part of making an object.

To make an object is to—

(1) Apply initial value functions or macros to all attribute values provided by the user.

(2) Find default values for all attributes for which values were not provided by the user, but for which non-*nil* default values were provided for the type of the object being created (or one of the ancestors of this type).

(3) Create the object.

(4) Index the object. That is, place the object in cross reference lists; and do processing related to stubs (see *has-name*).

(**make-object** '(ty_type at_attribute g_value ...) ['ob_prototype]) [SKETCH Operation Macro]
(**make-parent-object** 'opd_descriptor '(ty_type at_attribute g_value ...) ['ob_prototype]) [LISP Macro]
make-object [SKETCH Operation]

USE ONLY WHEN: *Make-parent-object* is used only when defining non-standard SKETCH types.

WHERE: Here ... is a list of attribute label/value pairs, like 'at_attribute g_value'. The entire first argument is called an 'abnormal object' because it represents an object as a list, the first of whose elements is the type of the object, and the rest of whose elements are attribute label/value pairs.

Ob_prototype defaults to *nil*.

Ob_prototype, if non-*nil*, must have type ty_type.

Opd_descriptor may be pre-evaluated.

RETURNS: An object of type ty_type with attribute at_attribute set to g_value, and other attributes specified similarly by the .... If ob_prototype is non-*nil*, it provides default values for all unspecified attributes. Otherwise, default values are provided by attribute default values (see HAS-DEFAULT-VALUE under *an-attribute-descriptor*). Attribute init functions or macros (see HAS-INIT-FUNCTION and HAS-INIT-MACRO under *an-attribute-function-table*) are applied to explicitly given attribute values (not those that are default values). The object returned is both created and indexed (see "*make*" and "*stubs*").

DEFAULT MAKE: The default behavior of make is usually provided by the *make-name-function* and *make-name-macro*, which consider objects with not attribute but a *has-name* to be stubs. See this function and macro in the glossary.

EFFICIENCY: For *make-object* to produce efficient code during compilation, the first argument should have the form—

(*list* s_type s_attribute g_value ...)

where ty_type is represented by its name s_type, and each attribute label is represented by its name s_attribute. Furthermore, ob_prototype must either be the *nil* expression, or an expression of the form—

(s_type ...)

(which promises that ob_prototype is a non-*nil* object of type named by s_type). Then much of the work of *make-object* is done at macro expansion (i.e. compile) time.

Otherwise all the work will be done at eval time.

PROCESSING ATTRIBUTES: At some point during the make operation, *create-object* is called to create the object (if it is not a stub). Just before this is done, the abnormal object is processed by either *process-attributes* or *process-attributes-for-macro*. These functions handle default values and *has-init-function*'s for attributes. The default make function and macro, *make-name-function* and *make-name-macro*, call these functions and *create-object*.

EQUIVALENT TO: *Make-parent-object* is equivalent to—

```
(execute-found-operation (find-operation make-object
                                          opd_descriptor
                                          ty_type)
                         make-object
                         (list ty_type at_attribute g_value ...) ob_prototype)
```

where the *find-operation* is performed at macro expansion time if possible, and steps are taken to avoid evaluating ty_type twice.

Make-object is equivalent to *make-parent-object* with a *nil* opd_descriptor.

(**merge-property-lists** 'l_list-1 'l_list-2) [LISP Function]

WHERE: Both l_list-1 and l_list-2 are assumed to have an even number of elements and be organized as attribute label/value pairs, where no attribute label appears twice.

RETURNS: L_list-1 with any properties on l_list-2 which are not on l_list-1 appended to l_list-1. Properties with a *nil* value are removed from l_list-1, but do serve to suppress appending of l_list-2 properties of the same name. L_list-1 is destroyed.

(**move-object** 'ob_object-1 'ob_object-2) [SKETCH Operation Macro]
move-object [SKETCH Operation]

RETURNS: Ob_object-2 after modifying it.

SIDE EFFECT: Moves ob_object-1 into ob_object-2. This means ob_object-2 will get exactly the same attribute values as ob_object-1.

WARNING: It is assumed that ob_object-1 will be discarded immediately after the move. Thus any property lists that are part of ob_object-1 may be moved to ob_object-2 without creating new list elements, for example.

EFFICIENCY: This macro compiles more efficient code if ob_object-1 is an expression of the form—

(s_type ...)

whose type s_type is specified at compile time, but not if ob_object-2 is of that form.

(**never-set-function** 'g_value 'atd_descriptor 'at_attribute 'ob_object ...) [LISP Function]
(**never-init-function** 'g_value 'atd_descriptor 'at_attribute 'ty_type) [LISP Function]

USE ONLY WHEN: Defining non-standard SKETCH types and attributes.

SIDE EFFECT: Calls *error* with a message that at_attribute cannot be *setf* or init'ed for an object of type (*has-type* ob_object) or ty_type.

USE: Usable as the *has-set-function* or *has-init-function* value for *an-attribute-function-table*.

(**null-property-list-with-switches** 'l_list 'l_info) [LISP Function]

USE ONLY WHEN: Building new object subpackages: like those of *declare-hunk-type* or *declare-vector-type*.

WHERE: L_list is a property list with an even number of elements and no attribute whose value is *nil*. L_info is the value returned by—

(*get-switch-info* ty_type ... #'*get-is-a-stub-switch*)

and is used to quickly find the value of—

(*get-is-a-stub-switch* at_attribute ty_type)

for any at_attribute that can be in l_list.

WARNING: The types of the arguments are not checked.

RETURNS: *Nil* if l_list has all absent attributes, and *t* otherwise. Whether or not the value of the attribute labeled at_attribute is absent is tested according to the value of—

(*get-is-a-stub-switch* at_attribute ty_type)

as computed using the third argument to *null-property-list-with-switches*. If this switch is *yes* or *nil*, the value is tested with *not*, and thus is absent only if it is *nil* (i.e., missing from the property list). If it is *no*, the value is presumed absent, no matter what it is: i.e. the test is skipped over. If it is any other value, it is called in place of the *not* function to test the value for absence.

SIDE EFFECT: If an attribute is not found in the third element of l_info, it is found by calling *get-switch-from-info* which adds the attribute to the third element of l_info.

(**object-expression-is** 'ty_type 'g_expression) [LISP Macro]

RETURNS: The object that g_expression will evaluate to at eval time, if this can be determined at compile time, and if this object is of type ty_type. Or, if g_expression is an object of type ty_type, returns g_expression (as in pre-evaluated arguments to macros). Otherwise returns *nil*.

If g_expression is a symbol which is the *has-name* of an object of type ty_type, that object is returned. If the symbol is unbound, it will first be automatically bound by the *define-object-name-prefix* facility, if possible.

As a special case, if g_expression has the form—

(*has-type* (s_type ...))

and ty_type is *a-type*, then g_expression is replaced by s_type.

(**object-is** 'ty_type 'ob_object) [SKETCH Operation Macro]
(**parent-object-is** 'opd_descriptor 'ty_type 'ob_object) [LISP Macro]

USE ONLY WHEN: *Parent-object-is* should be used only when defining non-standard SKETCH types.

WHERE: Opd_descriptor may be pre-evaluated.

RETURNS: Non-*nil* if ob_object has the type ty_type. Otherwise *nil*.

EFFICIENCY: This is much more efficient than—

(*eq* ty_type (*has-type* ob_object)),

when ty_type is specified by its name, s_type, at compile time, or when opd_descriptor is given and is either pre-evaluated or is a descriptor name.

EQUIVALENT TO:

```
(execute-found-operation object-is
                         (find-operation-descriptor opd_descriptor object-is
                                                    ty_type)
                         ty_type ob_object)
```

where opd_descriptor is *nil* for *object-is*, the call to *find-operation* is performed at macro expansion time if possible, and steps are taken to avoid evaluating ty_type twice.

(object-is-a-stub 'ob_object) [SKETCH Operation Macro]
object-is-a-stub [SKETCH Operation]

RETURNS: Non-*nil* if ob_object is a stub. Otherwise *nil*.

The standard test for an object being a stub is to test each attribute as specified by the attribute's its *has-is-a-stub-switch* value: see HAS-IS-A-STUB-SWITCH under *an-attribute-descriptor*. If this switch is *nil* or *yes*, the attribute is tested by the *not* function, and is therefore required to be *nil*. If this switch is *no*, the attribute is not tested at all. If this switch is some other symbol, that symbol is used in place of *not* to test the attribute value. All attributes must pass their tests for the object to be a stub.

Standardly *has-name* attributes have the *no has-is-a-stub-switch* value, and the value of this switch for all other attributes is not specified (*nil*).

(object-symeval 's_symbol) [LISP Macro]

RETURNS: The value of the symbol if it has one or can be bound by the *define-object-name-prefix* facility; *nil* if the symbol is unbound and cannot be bound by that facility.

(s_operation 'ob_object ...) [SKETCH Operation Macro]
(s_operation (s_type ob_object ...) ...) [SKETCH Operation Macro]

WHERE: S_operation is the name of some SKETCH operation, op_operation: e.g. *make-object*, *format-object*.

RETURNS: The value of executing op_operation on the arguments.

EFFICIENCY: The form with (s_type ob_object ...) rather than just ob_object is often more efficient when compiled, because s_type tells the compiler the type of ob_object.

NOTE: Some operations do not take an object as their first argument. E.g. *make-object* and *create-object* take a list whose first element is the type used to control the operation behavior, while *object-is* takes that type directly as the first argument.

EQUIVALENT TO:

(*execute-operation* s_operation 'ob_object ...).

However, it is permissible to override this definition by setting the function definition of s_operation. The default macro definition will not replace an existing definition.

(patom ...) [LISP Function]

EQUIVALENT TO: Normal LISP *patom*, except that objects which have a *has-name* attribute are represented by the value of that attribute.

"pre-evaluated" [SKETCH Term]

USE: An argument to a macro is said to be pre-evaluated if it is the intended argument value itself, as opposed to an expression which is to be evaluated at some later time to the intended value. Thus for a number argument, this would be the number itself, which can also serve as an expression that evaluates to itself. Other pre-evaluated arguments, however, are not expressions that evaluate to themselves, and cannot be passed to code that expects expressions and not values.

Attributes, attribute descriptors, operations, and operation descriptors are often passed to macros as pre-evaluated arguments. One must be careful not to output these in the expansion of the macro, unless that expansion also calls a macro that expects pre-evaluated arguments.

(pretty-print 'ob_object ...) [LISP Function]

EQUIVALENT TO: For SKETCH Objects (those for which *has-type* differs from *has-lisp-type*) *pretty-print* uses *format-object*. Also, lists in the format of abnormal objects, either using symbols to name the type and attributes, or using the type and attribute objects themselves, are formated like objects: the attribute values are indented with respect to the attribute labels.

(print ...) [LISP Function]

EQUIVALENT TO: Normal LISP *print*, except that objects which have a *has-name* attribute are represented by the value of that attribute.

(process-attributes '(ty_type at_attribute g_value ...) 'ob_prototype) [LISP Function]

(process-attributes-for-macro '(*list* s_type s_attribute g_value ...) 'g_prototype) [LISP Function]

USE ONLY WHEN: Defining non-standard SKETCH *make-object* operations.

WHERE: In the arguments to *process-attributes-for-macro*, s_type, s_attribute, g_value, and g_prototype are expressions which will evaluate at eval time to the corresponding components of the arguments to *process-attributes*.

The first argument to *process-attributes* is an abnormal object: see *make-object*.

RETURNS: *Process-attributes* returns the list—

(ty_type at_attribute g_value ...)

with default values appended for the missing attributes that, depending upon ty_type, should have default values, and *has-init-function*'s called for explicitly given attributes that, depending upon ty_type, have these functions defined. The default values are not appended if ob_prototype is not *nil*.

Process-attributes-for-macro is similar but returns the list—

(*list* ty_type at_attribute g_value ...)

and applies *has-init-macro*'s instead of calling *has-init-function*'s.

Process-attributes obtains default values by *eval*'ing default value expressions associated with the at_attribute's and ty_type. It applies *has-init-function*'s s_init-function by calling—

(*funcall* s_init-function g_value atd_descriptor at_attribute ty_type)

for each explicitly given g_value (not for default values) which has such a function associated with its at_attribute and ty_type. Here atd_descriptor is the **attribute** descriptor that contributed s_init-function.

Process-attributes-for-macro inserts unevaluated default expressions into the returned list. It replaces each unevaluated explicitly given g_value by the result of applying the *has-init-macro* s_init-macro to the form—

'(,s_init-macro ,g_value ,atd_descriptor ,at_attribute ,ty_type)

if there is an associated *has-init-macro*, or by the form—

'(,s_init-function ,g_value (*get-attribute* ,atd_descriptor)
,(*has-name* at_attribute) ,(*has-name* ty_type))

if there is only an s_init-function. Note that in the application of s_init-macro the arguments at_attribute, atd_descriptor, and ty_type are preevaluated, whereas g_value is unevaluated.

Calls to init functions or macros are not made for default values or for attributes for which there is no *has-init-function* associated with at_attribute and ty_type.

In the case of *process-attributes-for-macro*, s_type, s_attribute, and g_value in the abnormal object list are yet unevaluated, and it is not possible to call init functions or macros or to append default values unless s_type and all the attribute labels s_attribute are represented by their names. If this is not the case, or if g_prototype is non-*nil* and does not have the form—

(s_type ...),

process-attributes-for-macro will return *nil*, but not call *error*.

Lastly, error checking is done on the list and prototype. *Process-attributes* calls *error* if it discovers that the first element of the list is not a type, or the even numbered elements are not attributes, or the list length is not odd. It also calls error if the prototype is not *nil* and does not have a type equal to the first element of the list. *Process-attributes-for-macro* simply returns *nil* if there is any problem with the list.

SIDE EFFECT: The results returned are copies of the input lists, and the input lists are not changed.

read-write-password-attribute-functions [LISP Global Variable]
read-private-password-attribute-functions [LISP Global Variable]
private-password-attribute-functions [LISP Global Variable]

USE ONLY WHEN: Defining non-standard SKETCH types and attributes.

USE: These are named *an-attribute-function-table*'s which may be used to define *an-attribute-descriptor* that makes an attribute handle passwords. The password must be a symbol whose value equals itself, and must be the *has-parameters* attribute of the attribute descriptor whose *has-functions* attribute is one of the above function tables.

Thus a typical use is—

(*an-attribute-descriptor has-descriptor-attribute* at_attribute
has-descriptor-type ty_type
*has-functions *read-private-password-attribute-functions**
has-parameters s_password)

SIDE EFFECT: In some cases a password argument must be used with the attribute. When required, the password argument must be the first extra argument to the attribute, as in any of the following—

(s_attribute ob_object s_password ...)

(*get-attribute* at_attribute ob_object s_password ...)

(*get-parent-attribute* atd_descriptor at_attribute ob_object
s_password ...)

The password must be used to read the attribute if the function table is **private-password-attribute-functions**.

The password must be used to write the attribute if the function table is **private-password-attribute-functions** or **read-private-password-attribute-functions**.

In other cases the password is optional: it may be used or omitted.

In the case where a password must be used and is not, *error* will be called with a message that at_attribute is private and cannot be gotten or *setf* in objects of the type of ob_object.

In all cases, if the attribute read or write is allowed, the parent get or set attribute descriptor will be used, and any password present will be removed from the extra argument list and *not* passed to the parent.

(remove-abnormal-attributes [*do-return-really-nil*] [LISP Function]
'(ty_type at_attribute-1 g_value-1 ...)
'at_attribute-11 'at_attribute-12 ...)

(get-abnormal-attributes [*do-return-really-nil*] [LISP Function]
'(ty_type at_attribute-1 g_value-1 ...)
'at_attribute-11 'at_attribute-12 ...)

USE ONLY WHEN: Writing create and make functions which have abnormal object arguments.

RETURNS: A list of the values of the attributes at_attribute-11, at_attribute-12, ... found in the abnormal object

'(ty_type at_attribute-1 g_value-1 ...)

If the *do-return-really-nil* switch is present, the value returned for an attribute which has a *nil* value in the abnormal object is the symbol *really-nil*, whereas the value returned for an attribute with no abnormal value is *nil*. Without the *do-return-really-nil* switch, *nil* is returned in both cases.

SIDE EFFECT: *Remove-abnormal-attributes* removes the attributes it gets from the abnormal object. *Get-abnormal-attributes* does not.

"SKETCH object" [SKETCH Term]
ob_ [SKETCH Argument Prefix]

USE: A SKETCH object is one whose SKETCH *has-type* value is a type defined by *declare-hunk-type* or *declare-vector-type*. Note that types that appear in data but have not been declared to the program are implicitly declared in one of these ways, and are SKETCH object types.

LISP numbers, strings, symbols, lists, and ports are not SKETCH objects.

ARGUMENT PREFIX: SKETCH object arguments are indicated by the *ob_* argument prefix. This is less general than the *g_* prefix, which includes both SKETCH objects and other LISP objects such as numbers and lists.

sob_attribute [C Type]
at_ [Argument Prefix]

VALUE: A lisp value which is a pointer to a SKETCH *an-attribute* object.

SOB_ATTRIBUTE [C Global Variable]
SOB_BIGNUM [C Global Variable]
SOB_BINARY [C Global Variable]
SOB_CHAR [C Global Variable]
SOB_DOUBLE [C Global Variable]
SOB_FIXNUM [C Global Variable]
SOB_FLOAT [C Global Variable]
SOB_FLONUM [C Global Variable]
SOB_HUNK [C Global Variable]
SOB_INT [C Global Variable]
SOB_IVECTOR [C Global Variable]
SOB_LARRAY [C Global Variable]
SOB_LBIT [C Global Variable]
SOB_LIST [C Global Variable]
SOB_LONG [C Global Variable]
SOB_LVECTOR [C Global Variable]
SOB_NONLISP [C Global Variable]
SOB_PORT [C Global Variable]
SOB_SHORT [C Global Variable]
SOB_STRING [C Global Variable]
SOB_SYMBOL [C Global Variable]
SOB_TYPE [C Global Variable]
SOB_UBIT [C Global Variable]
SOB_UCHAR [C Global Variable]
SOB_ULONG [C Global Variable]
SOB_UNSIGNED [C Global Variable]
SOB_USHORT [C Global Variable]
SOB_VALUE [C Global Variable]

VALUE: An *sat_lvalue* equal to a SKETCH *a-type* object, according to the following table—

SOB_ATTRIBUTE	*an-attribute*	SOB_LONG	*a-long*
SOB_BIGNUM	*a-bignum*	SOB_LVECTOR	*a-lisp-vector*
SOB_BINARY	*a-binary-function*	SOB_NONLISP	*a-non-lisp-value*
SOB_CHAR	*a-char*	SOB_PORT	*a-port*
SOB_DOUBLE	*a-double*	SOB_SHORT	*a-short*
SOB_FIXNUM	*a-fixnum*	SOB_STRING	*a-string*
SOB_FLOAT	*a-float*	SOB_SYMBOL	*a-symbol*
SOB_FLONUM	*a-flonum*	SOB_TYPE	*a-type*
SOB_HUNK	*a-hunk*	SOB_UBIT	*a-ubit*
SOB_INT	*an-int*	SOB_UCHAR	*a-uchar*
SOB_IVECTOR	*an-immediate-vector*	SOB_ULONG	*a-ulong*
SOB_LARRAY	*a-lisp-array*	SOB_UNSIGNED	*an-unsigned*
SOB_LBIT	*an-lbit*	SOB_USHORT	*a-ushort*
SOB_LIST	*a-list*	SOB_VALUE	*a-value*

sob_case (ty_type) [C Function]

RETURNS: An integer code that discriminates between different numeric types and is suitable for use in a case statement. The codes returned have names such as *SOB_UBCASE* as per the following table—

Ty_type Value	Code Returned	Numeric Type
SOB_UBIT	**SOB_UBCASE**	unsigned 1 bit integer
SOB_CHAR	**SOB_CCASE**	signed 8 bit integer
SOB_UCHAR	**SOB_UCCASE**	unsigned 8 bit integer
SOB_SHORT	**SOB_SCASE**	signed 16 bit integer
SOB_USHORT	**SOB_USCASE**	unsigned 16 bit integer
SOB_LONG *SOB_INT*	**SOB_LCASE**	signed 32 bit integer
SOB_ULONG *SOB_UNSIGNED*	**SOB_ULCASE**	unsigned 32 bit integer
SOB_FLOAT	**SOB_FCASE**	signed 32 bit floating point number
SOB_DOUBLE	**SOB_DCASE**	unsigned 64 bit floating point number

If ty_type is not listed in the above table, 0 is returned.

sob_ltype (g_value) [C Macro]

RETURNS: The *sob_type* for the SKETCH type associated with the LISP type of g_value. This LISP type is the same as returned by ***has-lisp-type***. Thus if g_value were ***an-attribute***, ***sob_ltype*** would return something like *SOB_HUNK*: see ***has-lisp-type***!

sob_missing (x_type_case) [C Function]

RETURNS: The missing value appropriate to the data type ty_type with—

x_type_case = sob_case (ty_type).

This value is returned as a *double*. If ty_type has no missing value, some value is returned which is never taken by ty_type values: this is invariably SAT_DMISSING.

sob_nobject (t_name) [C Function]

RETURNS: The *sat_lvalue* which is the object whose *has-name* attribute value is the symbol *sat_nsymbol* (t_name).

BUG: Behavior is undefined if there is no such object but there is a bound symbol with the name t_name. An error is detected only if the symbol t_name is unbound.

sob_tsize (ty_type) [C Function]

RETURNS: The size in bits of a datum of type ty_type, or 0 if ty_type is not a valid type or has no specified size. This size in bits is the same as the *has-size* attribute of ty_type in LISP.

sob_type [C Type]
ty_ [Argument Prefix]

VALUE: A lisp value which is a pointer to a SKETCH *a-type* object.

sob_vcreate (ty_type) [C Function]

RETURNS: A newly created object of type ty_type. The object is the same as would be created by—

(*create-object* (*list* ty_type) *nil*),

except that element default values which are not constants, but which require computation to produce, are ignored, and their elements take the values they would have if no defaults were ever given for them.

Ty_type must have been defined by *declare-vector-type* or *define-vector-type*.

Note for purposes of *lint* that the value returned is of type *sat_lvalue*.

sob_vinit (ob_object ty_type) [C Function]

RETURNS: Ob_object after initializing it.

Note that for purposes of *lint* both ob_object and the value returned is of type sat_lvalue.

SIDE EFFECT: Sets all of the vector part of ob_object just as they would have been set had the object been created by—

(*create-object* (*list* ty_type) *nil*).

Ty_type must have been defined by *declare-vector-type* or *define-vector-type*. The hunk part of the object is not touched, and in fact the vector size and property list elements of the vector do not have to exist.

USE: To initialize vector objects created in the stack. E.g.—

```
function (...) ... {
        sag_talloc (transform, 1);
        ...
        sob_vinit (transform, SAG_TRANSFORM);
        ... }
```

allocates a SAG_TRANSFORM object in the stack and initializes it.

"stub" [SKETCH Term]

USE: A stub is an object most of whose attributes are yet undefined, but which has enough defined attributes to provide some kind of unique referent (e.g. name) for the object. Stubs are considered to be part of indexing, and are handled by the *make-object* operation. The general rules concerning stubs are as follows:

(1) If a stub is created while making an object, and an object with the same referent already exists, the stub is discarded, and the pre-existing object is returned as the result of making the object. An error check is made to be sure the pre-existing object and the newly created stub have the same type.

(2) If a non-stub is created while making an object, and a stub with the same referent already exists, the attributes of the pre-existing stub are filled in with the attribute values from the newly created object (by *move-object*), the newly created object is discarded, and the pre-existing object (the former stub) is returned as the result of making the object. An error check is made to be sure the pre-existing stub and the newly created object have the same type.

(3) If a non-stub is created while making an object, and a non-stub with the same referent already exists, the two objects with the same referents are checked for equality by *compare-object*. Inequality is an error. The newly created object is then discarded, and the pre-existing object is returned as the result of making the object.

(symbol-init-function 'g_value 'atd_descriptor 'at_attribute 'ty_type) [LISP Function]

(symbol-init-macro 'g_value atd_descriptor at_attribute ty_type) [LISP Macro]

USE ONLY WHEN: Defining non-standard SKETCH types and attributes.

SIDE EFFECT: Checks that g_value is a symbol, and calls *error* if not with a message that at_attribute must be initialized to a symbol for an object of type ty_type.

USE: Usable as the *has-init-function* or *has-init-macro* value for *an-attribute-function-table*.

(**symeval** 's_symbol) [LISP Special Function]

WARNING: When compiled, FRANZ *symeval* does not check for unbound variables and automatically bind them. Use *object-symeval* instead in compiled code that is to automatically bind unbound variables.

top-level-print [LISP Global Variable]

SIDE EFFECT: This variable, which is defined and used by the top level in the FRANZ EXTENSIONS package, is set by the OBJECTS PACKAGE to print only the name of any expression value with a non-*nil* *has-name* attribute, unless that name is *eq* to the expression that was evaluated (as stored in the global variable + by the top level).

(**s_type** 'at_attribute 'g_value ...) [LISP Macro]
(**s_type** 'ob_object) [LISP Macro]
(**s_type** 'ob_object 'at_attribute 'g_value ...) [LISP Macro]

WHERE: S_type is the name of a SKETCH type, ty_type.

RETURNS: The form with no ob_object returns a SKETCH object of type ty_type, at_attribute value g_value, and other attribute values as given by .... Unspecified attributes will be given default values determined by s_type and at_attribute. This form is equivalent to—

(*make-object* (*list* s_type at_attribute g_value ...) *nil*)

It is more efficient if each at_attribute is specified by its name, s_attribute.

The form with a single argument, ob_object, macro expands to ob_object. This form is used to tell other macros that ob_object is necessarily an object of type ty_type. For example,

(*has-name* (*an-attribute* x))

may be compilable to more efficient code than—

(*has-name* x).

However, this one argument form does not usually check to see that the type of ob_object in fact is ty_type, so the programmer must avoid mistakes.

The form with the ob_object argument and at_attribute/g_value argument pairs makes a new object. Attributes not specified by the at_attribute/g_value pairs are taken from the corresponding attributes of ob_object, rather than being given default values. The type of ob_object must be ty_type. This form is equivalent to—

(*make-object* (*list* s_type at_attribute g_value ...) (s_type ob_object))

It is more efficient if each at_attribute is specified by its name, s_attribute, and ob_object is an expression of the form—

(s_type ...)

which promises a non-*nil* value of the correct type.

NOTE: All symbols beginning with *a-* or *an-* should name SKETCH types, and all SKETCH types should have names beginning with these prefixes.

NOTE: It is permissible to override this definition by setting the function definition of s_type. The default macro definition of s_type will not replace an existing definition.

(uneval-object 'g_object ['g_index-switch ['g_backquote-switch]]) [SKETCH Operation Macro]

uneval-object [SKETCH Operation]

WHERE: We have written g_object instead of ob_object simply to emphasize that any LISP value can be considered to be a SKETCH object for the purposes of *uneval-object*.

RETURNS: A **LISP value** which when *eval*'ed will evaluate to g_object. More importantly, when *pretty-print*'ed, re-*read*, and then *eval*'ed this value will evaluate to g_object. This is the only general means that a SKETCH object may be transmitted from one program through a file to another program.

The result of *uneval-object* may contain calls to the fictitious macros *backquote* and *comma*, which will *pretty-print* as ` and , respectively. The argument to *backquote* may have the form of a dotted list, as in—

(*backquote* (... . (*comma* ...)))

Backquote is defined as a macro, and its presence also signals *pretty-print* to process the list specially.

Comma cannot occur outside a *backquote*'d argument.

A SKETCH object which is indexed by having a *has-name* attribute will be represented by an expression of the form—

(s_type *has-name* 's_name)

which evaluates to a stub for the object. If g_index-switch is absent or *nil*, an exception will be made for the g_object itself, which will be represented as a type and attribute list even if it has a *has-name* attribute. However no such exception will be made for the attribute values of g_object.

By using these rules, it is possible to output in any order a set of named objects which cross reference each other, and get the cross referencing right when the objects are input into another program load.

The behavior of the last two paragraphs is the behavior of the default *uneval-object* functions defined by *declare-hunk-type* and *declare-vector-type*. This default behavior can be overridden by defining special *uneval-object* functions for a particular type.

Sometimes the results of *uneval-object* are to be included as part of an argument to *backquote*. In this case, the result of an *uneval-object* will not to be evaluated unless it is a call to the *comma* pseudo-function. This situation is indicated by a present, non-*nil* g_backquote-switch. Otherwise, *uneval-object* is to operate normally, assuming that the result will be evaluated to obtain g_object. For example,

(*uneval-object* 'x *nil t*)

will return just x, whereas—

(*uneval-object* 'x *nil nil*)

will return 'x.

EFFICIENCY: This macro compiles more efficient code if g_object is an expression of the form—

(s_type ...)

whose type s_type is specified at compile time.

Also, if g_backquote-switch is not given or has a known value (*nil* or *t*) at compile time, and if g_object does not have the form—

(s_type ...),

uneval-object compiles in-line code to check whether g_object is a number, string, or symbol, and returns g_object or 'g_object as its value in that case.

UNEVAL-SWITCH: If an attribute, at_attribute, of an object of type ty_type has a non-*nil* value of

(*get-uneval-switch* ty_type at_attribute),

then in any call to *uneval-object*, this value will control the *uneval-object*'ing of the attribute value. If the switch is *no*, the attribute will not be included as part of the unevaled object. If the switch is *yes* or *nil*, the attribute will be included. If the switch is another symbol, that symbol will be taken as the name of a function to be called in place of *uneval-object* to uneval the attribute value. See HAS-UNEVAL-SWITCH under *an-attribute-descriptor*.

(**unpre-evaluate-object** 'ob_object) [LISP Function]

USE ONLY WHEN: Referencing pre-evaluated macro arguments in calls to *error* returned by the macro.

RETURNS: An expression which crudely attempts to undo possible pre-evaluation of macro arguments. Returns—

(*uneval-object* ob_object)

after stripping any *quote* function therefrom.

CHAPTER 6

CATALOGS

1. CATALOG FILES AND FILE CATALOGS. A catalog file is a file that stores a sequence of LISP and SKETCH objects in ASCII text. Each object is represented by a LISP expression which may be read (by the LISP *read* function) and evaluated (by the LISP *eval* function) to produce the object. Virtually any LISP or SKETCH object can be represented in this manner.

An object can be written into a catalog by first unevaluating it (using the SKETCH *uneval-object* macro), and then printing the result (using the SKETCH *pretty-print* macro, or LISP *print* function).

A file catalog is *a-catalog* object with a *has-file* attribute that is a symbol naming a catalog file. E.g.—

(*setq* the-catalog (*a-catalog has-file* 'my-file.ca))

where my-file.ca is the name of the catalog file. Note that the file name extension *.ca* is preferred for catalog files (but not required).

You can read this catalog by—

(*setq* the-object (*read-catalog* the-catalog))

which returns the next object in the catalog. The first object returned from a new catalog is the first object in the catalog. If you want to reset the catalog to the beginning, you can execute—

(*close-catalog* the-catalog)

which does not destroy the catalog object, but does release operating system resources used by that catalog, and causes the next *read-catalog* to begin back at the beginning of the catalog. There is no explicit open-catalog operation: it is implied by the first read (or write) of a catalog.

Executing—

(*setq* the-object (*read-catalog* the-catalog))

when the catalog is positioned at its end will return the symbol *end-of-catalog* as the value of the-object.

The-object may be written at the end of the catalog by—

(*write-catalog* the-catalog the-object)

Note that *write-catalog* always appends to the end of the catalog; it never causes information to be lost from the catalog. When you are done writing objects into the catalog, you should use—

(*close-catalog* the-catalog)

to be sure everything you wrote is properly transferred to disk.

After you start reading from a catalog, you should not write to the catalog until you have closed it. Similarly, after you start writing, you should not read until you have closed the catalog.

To start writing at the beginning of a catalog you first truncate the catalog. For catalog files, this is done with—

(*setq* the-catalog (*new-catalog* 'my-file.ca)),

which truncates my-file.ca and sets the-catalog to—

(*a-catalog has-file* 'my-file.ca)

It is also possible to read from a random location in a catalog file. To find the location of the last object read from the catalog by *read-catalog*, or written into the catalog by *write-catalog*, use—

(*setq* the-location (*get-catalog-location* the-catalog)),

after which the last object can be read at any later time by—

(*setq* the-object (*read-catalog* the-catalog the-location)),

regardless of where the catalog is positioned when this last statement is executed.

If you look at the value of the-location, by the way, you will find it to be a list of several numbers. Its complexity is due to the fact that objects in a catalog are sometimes packed (automatically) by referring to a previous object in the catalog, so that to position to an arbitrary object in a catalog requires positioning to the first previous unpacked object, and then reading forward to the desired object, unpacking as you read.

When opening catalog files, the directories searched are those in the list of directory names (represented by symbols) returned by—

(*status catalog-search-path*)

which is typically set by placing the statement

(*sstatus catalog-search-path* (|.| s_directory ...))

in the *sketch.rc* file (which is loaded whenever *sketch* is started).

2. INDEX CATALOGS. An index catalog permits objects in a second catalog to be referenced by meaningful names, called keys. The second catalog is called the indexed catalog. The keys are defined by an index function that returns the key of an object when called with the object as an argument. The keys become the locations returned by *get-catalog-location* and used by *read-catalog* when these functions are applied to the index catalog.

Suppose we have a catalog file named mine.ca containing objects some of which have a non-*nil* has-id attribute that we wish to use as a key. Then the following creates the appropriate index catalog—

(*a-catalog is-index-of* (*a-catalog has-file* 'mine.ca)
has-index-function '(*lambda* (x y) (has-id x)))

Here the indexed catalog is (*a-catalog has-file* 'mine.ca), the value of the *is-index-of* attribute of the index catalog. Also, we have introduced a *lambda* index function, instead of just using has-id directly, because has-id is actually a macro (like all attribute names), and because the index function must take a second argument.

The second argument, y, is the number (1, 2, 3, ...) of the object in the indexed catalog. So to use the object's number as its key, just use the index function—

'(*lambda* (x y) y).

There is a predefined function named ***catalog-number*** that equals this last *lambda*, so the symbol ***catalog-number*** may be used as the index function when you want object keys to equal the number of the object in the indexed catalog. In this case—

(***read-catalog*** ca_index-catalog 4)

would read the 4'th object in the indexed catalog.

By the way: one must not replace '(*lambda* ...) by #'(*lambda* ...), for those of you who know that this trick will compile the *lambda* function, because the index function should be something we can save in a file, as we shall see in a moment.

If the index function returns *nil* for an object, that object has no key. After reading that object ***get-catalog-location*** will return *nil*, which cannot be passed as a location to ***read-catalog***.

The default index function, or what you get when you specify *nil* as an index function, returns a key only for objects that are pairs of the form—

(*catalog-key* g_key).

For such an object g_key is returned as the key. Putting such objects at selected points in a catalog file enables one to position to these points. Note that what actually appears in the catalog file is the quoted list—

'(*catalog-key* g_key).

At the end of every catalog the symbol *end-of-catalog* appears as if it were an object in the catalog. This symbol always has itself as its key, regardless of how the index function is defined. That is,

(***read-catalog*** ca_index-catalog '*end-of-catalog*)

will always position both index and indexed catalogs at their ends and return the symbol *end-of-catalog*.

Note that operations on an index catalog are equivalent to operations on its indexed catalog, except that object location values are different.

If you use one of the index catalogs defined above that index mine.ca, then after closing the index catalog you will find a new file, mine.ci. The index function and the index itself are written into this file when the index catalog is closed. The index is roughly a list of triples each consisting of a key, the number of an object in the indexed catalog, and the location of the object in the indexed catalog. The index may not be complete. Later, if another index catalog is defined that indexes mine.ca with the same index function, the mine.ci file will be used to read the index, and save the time of having to read the entire mine.ca file to rebuild the index. Also, no index function need be specified for the index catalog if mine.ci exists; it will be read from mine.ci.

You can provide the name of a file to serve as mine.ci for any index catalog. It is a symbol which is the value of the index catalog's *has-index-file* attribute. Note that the index stored in one of these files may be incomplete, as the index is built incrementally as it is needed, and not completed until the end of the indexed catalog is read. See HAS-INDEX-FILE under ***a-catalog*** in the GLOSSARY.

When index files are being used. keys must be objects that will equal themselves when printed and re-read. Integers, symbols, character strings, and lists of these will work. Floating point numbers that originated when character strings with 5 or fewer digits were read into the computer may also work.

If you use an index catalog to write a catalog file like mine.ca, the index file, mine.ci, will be made when the index catalog is closed. Once mine.ci is completed, the index function is actually never needed again. In particular, it may be a symbol that has no function definition in environments in which mine.ca is read with an index catalog.

3. INCLUDED CATALOGS.

4. FILTER CATALOGS.

5. RANDOM PORTS.

6. TAPE VOLUMES.

7. HITLIST.

(1) Finish tutorial documentation.

(2) Implement tape volumes.

(3) Possibly improve packing algorithm.

8. GLOSSARY.

(**a-catalog** [*has-file* 's_file-name] [SKETCH Object]
[*has-filter* '(u_function ,ca_input-catalog)]
[*is-index-of* ca_indexed-file]
[*has-index-file* 's_index-file]
[*has-index-function* 'u_index-function])

(**has-file** 'ca_catalog) [SKETCH Attribute Macro]
(**has-filter** 'ca_catalog) [SKETCH Attribute Macro]
(**is-index-of** 'ca_catalog) [SKETCH Attribute Macro]
(**has-index-file** 'ca_catalog) [SKETCH Attribute Macro]
(**has-index-function** 'ca_catalog) [SKETCH Attribute Macro]

catalog-key [LISP Symbol]
(**catalog-number** ob_x x_number) [LISP Function]

VALUE: A catalog object that may be used to read or write LISP objects. There are several different kinds of catalogs, distinguishable by their attributes.

HAS-FILE: A file catalog has a file name in the *has-file* attribute. The file contains a sequence of LISP expressions which can be *read* and then *eval*uated to produce LISP values. These LISP expressions may be packed: each expression may be represented in a special notation that describes only its differences from the previous expression in the file.

A file catalog can be written as well as read. The values written are converted using *uneval-object* into an expression that will *eval*uate into the value being written. The values written are packed, but after packing 50 values a value is intentionally left unpacked to speed repositioning when reading the catalog.

HAS-FILTER: The value of this attribute consists of a two element list, or pair, of the form-

(u_function ca_input-catalog)

The function is a function of one variable which is applied to each value read from ca_input-catalog to produce an output value for the current catalog. If the function returns the symbol '*please-ignore*, the corresponding ca_input-catalog value is ignored.

A filter catalog cannot be written.

IS-INDEX-OF:

HAS-INDEX-FUNCTION:

HAS-INDEX-FILE: The value of the *is-index-of* attribute is a catalog, ca_indexed-catalog, called the indexed catalog. The current catalog is called the index catalog. Operations on the index catalog are equivalent to operations on the indexed catalog, except for operations involving locations. The locations of an index catalog are keys determined by u_index-function. This function is called by-

(*funcall* u_index-function ob_object x_number)

where ob_object is the x_number'th object in ca_indexed-catalog. The function (which must not have side effects) returns a key which is used to name the location of the object in the index catalog. The function may also return *nil* to indicate that the object does not have a well defined location in the index catalog.

Thus if u_index-function equals—

(*lambda* (x y) y),

the x_number'th object in the indexed catalog will have x_number as its key. The function *catalog-number* is defined to be equal to this particular function, and is more mnemonic.

The keys must be lisp objects that equal themselves when printed and re-read. E.g. integers, symbols, character strings, and lists of such. The keys must be unique: a non-unique key used as a location for the index catalog will locate any of the several objects in the indexed catalog that have that key. The symbol *end-of-catalog* is automatically the key of the end of the *end-of-catalog* symbol returned at the end of ca_indexed-catalog (regardless of the definition of u_index-function), and must not be a key of any object in ca_indexed-catalog.

If the *has-index-function* attribute is *nil*, the key for any object of the form—

(*catalog-key* g_x),

will be g_x, while no other objects will have a key.

The index catalog keeps an index table that translates keys into locations in the indexed catalog. This table may be incomplete if the entire indexed catalog has not been read and translated into keys. If a request is made to locate to an object with a key not yet in the index table, the remainder of the indexed catalog is read until the object with the key is found, or until the end-of-catalog is reached, in which case an error is signaled.

If s_index-file, the *has-index-file* attribute, is non-*nil* and names a readable file, then when the index is first needed, the *has-index-function* value and the index itself will be read from s_index-file. The *has-index-function* value, which must be a symbol or a printable *lambda* list, will be read from the beginning of s_index-file. The index will then be read.

There are two cases when the contents of s_index-file are ignored. First, if the *has-index-function* has a non-*nil* value before the file is read, and this value is not *equal* to the index function read from the file, then it is assumed that the contents of the file are not valid for the current application. Second, if the indexed catalog has a *has-file* attribute whose file has been modified more recently than s_index-file, it is assumed that the index stored in the file is not valid, though the index function read will still be used to replace a *nil* *has-index-function* value.

If the *has-index-file* attribute is non-*nil* and s_index-file is writable or creatable when the index catalog is closed, the *has-index-function* attribute and the index itself are written into that file, unless they are both identical to what was read from the file previously. In particular, if the index catalog was being used to write the indexed catalog, the complete index will be written. However, if the index file was merely being used to read the indexed file, and if *end-of-catalog* had not been reached, then only a partial index will be written.

In searching for the directory containing s_index-file, the same procedure is used as when searching for a catalog file.

If the *has-index-file* attribute is *nil*, and the indexed catalog is a file catalog whose file name ends in *.ca*, then a file name made by replacing the *.ca* by *.ci* will be used as if it were the value of the *has-index-file*, unless such a file already exists and contains a *has-index-function* value that disagrees with a non-*nil* initial value of that attribute for the index catalog.

If no file is available for use as the index file, the index is lost when the index catalog is closed. If the index catalog is not closed, any index constructed in MOS memory that would have been written out were the catalog closed will be lost. However, when SKETCH exits, all open index catalogs will be automatically closed.

WARNING: When using a *.ci* file with a *.ca* file that includes other catalog files, the index will not be automatically invalidated when the included catalog files are changed. To invalidate the index the user should *touch*(1) the including *.ca* file.

NOTE: The *has-index-file* and *.ci* files can be used by several SKETCH processes at once, and are (hopefully) protected against the various abnormal states that may arise during such use.

(a-tape-volume [SKETCH Object]
has-tape-format '(x_record_length ...)
[*has-name* 's_name]
[*has-drive* 'x_drive]
[*has-file-number* 'x_file-number]
[*has-record-number* 'x_record-number]
[*is-modified* 's_modified-switch])

WARNING: Tape volumes and operations thereupon are not yet implemented.

USE: *A-tape-volume* is an object describing the format of a magnetic tape. The contents of the tape are not described in detail: only the format.

This object is stored in the file named s_name (accessible using the *data-search-path* directory list. The optional attributes (including *has-name*) are supplied by SKETCH software when the tape volume is mounted (see *mount-tape*) and are not part of the object stored in the s_name file.

HAS-TAPE-FORMAT: This list describes the tape format. It is a list with one element for each file on the tape. That element is the record size in bytes of records in the file.

For input files, the record size may be an overestimate: the maximum possible size in bytes of any record in the file. For output files all records will be exactly the given size.

Files on the tape are separated by file marks. When a tape is written, two consecutive file marks are written after the last file.

HAS-NAME: S_name serves as a file name for a file containing nothing but the *a-tape-volume* object. The file named is found by searching the (*status catalog-search-path*) list of directories.

The file named by s_name is called the volume object file for the tape volume. It is read when the tape is mounted, and may be written when the tape is dismounted if its *has-tape-files* attribute has been modified.

S_name is also used as the name of the volume inside the LISP environment.

It is strongly recommended that s_name end with the extension *.tv* standing for "tape volume".

HAS-DRIVE: A symbol or number naming the tape drive. The following are standard—

0—9 Magnetic tape drive 0 through 9.

HAS-FILE-NUMBER: The number (1, 2, ...) of the file in which the tape is currently positioned.

HAS-RECORD-NUMBER: The number (1, 2, ...) of the record just before which the tape is currently positioned.

IS-MODIFIED: Non-*nil* if the *has-tape-format* list has been modified since the tape volume was mounted.

"catalog file" [SKETCH Term]
.ca [UNIX File Extension]

FILE FORMAT: A catalog file is a file created by *new-catalog*, *write-catalog*, and *close-catalog*. It holds a sequence of LISP values, including SKETCH objects such as *an-array* objects. Each LISP value is represented by a LISP expression which can be read and then evaluated to produce the value.

The preferred extension for catalog files is *.ca*.

(**catalog-pack** 'g_next-expression 'g_last-expression) [LISP Function]

USE ONLY WHEN: Maintaining catalog package.

RETURNS: The packed version of the unpacked g_next-expression in the context where the value returned is to be written into a packed file of LISP expressions, and the unpacked version of the previous expression written is g_last-expression.

NOTE: If g_next-expression cannot be packed, it is returned. Packing will have occurred only if the value returned is not *eq* g_next-expression.

(*status* **catalog-search-path**) [LISP Function]
(*sstatus* **catalog-search-path** (s_directory ...)) [LISP Function]
catalog-key [LISP Symbol]

VALUE: The list (s_directory ...) is a list of the names of directories which are searched for catalog files to be input. The first directory named in the list is the place where new catalog files are created (unless the name given the new file contains a directory name).

(catalog-unpack 'g_next-expression 'g_last-expression) [LISP Function]

USE ONLY WHEN: Maintaining catalog package.

RETURNS: The unpacked version of the packed g_next-expression in the context where g_next-expression has been read from a packed file of LISP expressions, and the unpacked version of the previous expression read is g_last-expression.

NOTE: If g_next-expression was already unpacked, it is returned. Unpacking will have occurred only if the value returned is not *eq* g_next-expression.

.ci [UNIX File Extension]

FILE FORMAT: A catalog index file associated with a *.ca* file. See *has-index-file* under *a-catalog*.

(close-catalog 'ca_catalog) [LISP Function]

SIDE EFFECT: Closes the catalog, flushing all information stored in ports associated with the catalog.

(copy-catalog 'g_input 'g_output ['(g_key ...)]) [LISP Function]
(append-catalog 'g_input 'g_output ['(g_key ...)]) [LISP Function]

WHERE: G_input is either *a-catalog*, or a symbol naming a file from which the catalog—

(*a-catalog has-file* g_input)

is made; and g_output is similar.

RETURNS: The number of items copied.

SIDE EFFECT: Items are read from the input catalog and written to the output catalog until and *end-of-catalog* is read. Both catalogs are then closed.

If g_output is a symbol, *copy-file* uses *new-catalog* to create the output catalog and truncate the file, whereas *append-catalog* simply creates the output catalog without truncating the file, and thereby appends to the file.

If '(g_key ...) is given, the catalogs must be in indexed catalog (if it is not, they are replaced by index catalogs whose *is-index-of* attributes are the original catalogs). The input is copied by first copying the object at location g_key and all objects following it that have no key; then doing the same for the object whose location is the next key in the '(g_key ...) list, and so forth to the end of the list. If an object at location g_key is not of the form—

(*catalog-key* ...)

then the object—

(*catalog-key* g_key)

is output just before it (but this last object is not included in the returned count of objects copied).

(*status* **data-search-path**) [LISP Function]
(*sstatus* **data-search-path** (s_directory ...)) [LISP Function]

VALUE: The list (s_directory ...) is a list of the names of directories which are searched for data files to be input. The first directory named in the list is the place where new data files are created (unless the name given the new file contains a directory name).

(**dismount-tape** ['x_drive]) [LISP Function]
(**dismount-tape** ['s_volume-name]) [LISP Function]

SIDE EFFECT: Dismounts the indicated *a-tape-volume* object: i.e. undoes *mount-tape*. Closes the drive. Writes the *a-tape-volume* object back to its file if it has been modified.

(**get-catalog-keys** 'ca_index-catalog) [LISP Function]

RETURNS: The list of all keys defined for an index catalog.

SIDE EFFECT: Reads to the end of the catalog.

(**get-catalog-location** 'ca_catalog) [LISP Function]

RETURNS: The location of the last value read from or written to ca_catalog. This location is some LISP value that can be understood by *read-catalog*.

(**get-random-port** 'p_port ['g_location 's_direction ['(s_directory-name ...)]]) [LISP Function]

USE ONLY WHEN: Copying between memory and random locations in the file system or on magnetic tape.

WHERE: G_location specifies a byte location within a file. It has one of the following forms:

s_file-name
(s_file-name [x_offset])
(s_file-name *end-of-file*)
(s_volume-name x_file x_record [x_offset])
(s_volume-name *end-of-volume*)
(s_volume-name x_file *end-of-file*)

S_file-name is the name of a file that is searched for in the directories (s_directory ...). The directory list defaults to (*status data-search-path*). If an output file does not already exist in one of these directories, it is created in the first directory in the list (unless the output file name contains a directory name: see the write mode of *search-path*).

S_volume-name is the name of a tape, x_file the number of a file on that tape (1, 2, 3, ...), and x_record the number of a record in that file (1, 2, 3, ...).

X_offset is the number of bytes in the file or tape record before the first byte to be read or written. For a file, *end-of-file* denotes the x_offset value to position to the end of the file (which is, in fact, the current length of the file).

For a tape volume *end-of-volume* denotes the x_file and x_record values necessary to start a new file at the end of the volume, while *end-of-file* denotes the x_record value necessary to add a record to the end of the file designated by x_file. In these cases the port must be for writing, and not for reading.

's_direction is either '*read* or '*write*.

RETURNS: A port positioned to the location g_location, or *nil* if it is not possible to produce such a port, or if g_location itself is missing (see SIDE EFFECT for this last case).

The port is suitable for reading if s_direction is '*read*, or for writing if s_direction is '*write*.

SIDE EFFECT: The random i/o package keeps a cache of ports which it repositions and passes to users. P_port is a user port being returned to this cache. P_port may be *nil* if no port is being returned.

The value of *get-random-port* is a port taken from the cache and positioned to g_location. If g_location is missing no port will be taken from the cache and the value returned will be *nil*. A call with missing g_location may be used to return a previously acquired port.

NOTE: The user may reposition a returned port to a different offset, or find its current offset. Other operations, aside from reading and writing, should not be performed on the port.

NOTE: The offset's in locations are identical to the port offsets that can be changed and inspected by *fseek*. This means that each tape record is treated like a complete file all by itself, with offset 0 corresponding to the beginning of the record. The port for a tape location will suffer an end-of-file at the end of the record.

NOTE: The user may not posses multiple ports referencing the same tape volume.

BUG: If you unlink a file for which you have recently had a random port, and then get a random port with the same file name, the port you get may be for the old file, which is now nameless. This is because the port associated with the file name is not closed and reopened. However, you can truncate a file successfully without unlinking it, because the ports are drained (but not closed) when they have no users.

(get-random-port-location 'p_port) [LISP Function]

USE ONLY WHEN: Copying between memory and random locations in the file system or on magnetic tape.

RETURNS: The current location of p_port, assuming the latter was gotten by a call to *get-random-port*. This location is one of the forms where x_offset is an explicit number.

(has-function 'ca_catalog) [SKETCH Attribute]

USE ONLY WHEN: Defining new types of catalog.

VALUE: The *has-function* attribute is automatically set from the other attributes when it is gotten. Thus most users need not worry about it.

The function calls

(*read-catalog* ca_catalog [g_location])
(*write-catalog* ca_catalog g_value)
(*get-catalog-location* ca_catalog)
(*close-catalog* ca_catalog)

normally translate into

(funcall (*has-function* ca_catalog) '*read* ca_catalog [g_location])
(funcall (*has-function* ca_catalog) '*write* ca_catalog g_value)
(funcall (*has-function* ca_catalog) '*locate* ca_catalog)
(funcall (*has-function* ca_catalog) '*close* ca_catalog)

The call

(funcall (*has-function* ca_catalog) '*read* ca_catalog)

must return '*end-of-catalog* at an end of file.

Some possible values for the *has-function* attribute are '*file-catalog*, which uses the *has-file* attribute, '*filter-catalog* which uses the *has-filter* attribute, and '*index-catalog* which uses the *is-index-of* attribute.

When

(funcall (*has-function* ca_catalog) '*read* ca_catalog)

returns a list of the form

(*please-include* ca_included-catalog)

the *read-catalog* function alters ca_catalog so that future requests to read it will return values from ca_include-catalog until the end of the latter.

(has-include 'ca_catalog) [SKETCH Attribute]

USE ONLY WHEN: Maintaining catalog package.

VALUE: When another catalog is being included in this catalog, a two element list of the form-

(ca_included-catalog g_location-of-included-catalog)

The location is the location within this catalog of the value of the form-

(*please-include* ca_included-catalog)

which caused the inclusion to start. If there is no inclusion currently in progress, the *has-include* attribute value is *nil.*

(lookat-tape ['x_drive]) [LISP Function]
(lookat-tape ['s_volume-name]) [LISP Function]

RETURNS: The *a-tape-volume* object for the given drive number or volume name, if one is mounted. When this prints its *has-drive*, *has-file-number*, and *has-record-number* attributes tell the drive and current position.

(make-catalog-index 'g_catalog ['s_index-file]) [LISP Function]

WHERE: G_catalog is either *a-catalog* or a symbol naming a file from which the catalog—

(*a-catalog has-file* g_input)

is made.

If s_index-file is not given, g_catalog must be a symbol ending in *.ca*, or a *has-file* catalog with a *has-file* attribute which is a symbol ending in *.ca*, and this symbol with *.ca* replaced by *.ci* will be used as the s_index-file value. The *has-index-function* is taken if available from the previous version of s_index-file, or is *nil.*

RETURNS: The number of items read from g_catalog.

SIDE EFFECT: Builds an index file named s_index-file suitable for use as the *has-index-file* attribute of an index file whose *is-index-of* attribute is g_catalog. S_index-file is initially truncated.

(**mount-tape** 's_volume-name ['x_drive]) [LISP Function]

RETURNS: The *a-tape-volume* object for the tape volume mounted.

SIDE EFFECT: Mounts the *a-tape-volume* object indicated. Reads this object from the file named s_volume-name. Assigns a *has-name* attribute to the object (which is not assigned in the file's version of the object). Assigns the drive to the object and opens the drive.

(**new-catalog** 's_file-name) [LISP Function]

RETURNS: A new, empty, catalog with *has-file* attribute value s_file-name. If any previous catalog existed with that name, it is emptied (its file is truncated).

(**new-data-file** 's_file-name) [LISP Function]

RETURNS: The name of the file found by

(*search-path* (status data-search-path) s_file-name 'a)

SIDE EFFECT: The file whose name is returned is truncated to 0 length if it exists, or created as a file of 0 length if it does not exist.

(**read-catalog** 'ca_catalog ['g_location]) [LISP Function]

please-ignore [LISP Symbol]
(**please-include** ca_catalog) [LISP List]

RETURNS: Returns the next value read from ca_catalog. If g_location is given, it must be a location returned from the call

(*get-catalog-location* ca_catalog)

and the value returned will be the value at the given g_location.

At the end of ca_catalog, the symbol *end-of-catalog* will be returned.

NOTE: If the symbol *please-ignore* is to be returned from *read-catalog*, that function skips that value and continues to the next value in the catalog.

NOTE: If a value of the form

(*please-include* ca_included-catalog)

is to be returned from *read-catalog*, that function instead returns the first value from ca_included-catalog (using *read-catalog* recursively to read that catalog), and in subsequent calls keeps returning values from ca_included-catalog until the symbol *end-of-catalog* is to be returned. Then the *read-catalog* function returns instead the next value read from ca_catalog.

.tv [UNIX File Extension]

FILE FORMAT: These files contain *a-tape-volume* object describing a single tape volume. There is only one such object per file.

(**write-catalog** 'ca_catalog 'g_value) [LISP Function]

SIDE EFFECT: Writes g_value at the end of ca_catalog. Notice that no part of ca_catalog can be overwritten (but see *new-catalog*, which truncates catalog files). The actual writing operation is essentially done by executing-

(*pretty-print* (*uneval-object* g_value) ...)

CHAPTER 7

ARRAYS

1. ROADMAP Since the tutorial has not yet been written, the following will have to do. Start with the *an-array* glossary entry, and then read the sar_demo.ou demonstration output listing, looking up the glossary entries as you go. C programmers should them read the *sar_array* glossary entry, followed by all the subsequent entries for names beginning with sar_ or SAR_.

2. HITLIST

(1) Write tutorial documentation.

(2) Possibly change *prepare-array* to use *array-copy-exponent* whenever making a new array, even if it is the same element type as the old array. This means revising many calls to *prepare-array* to have an explicit exponent.

(3) Define compare-object so constant arrays may have has-name attributes. In general, compare-object currently makes no sense for arrays.

(4) Document sar_slice, sar_duplicate, sar_aduplicate.

(5) Change sar_print output format to produce a LISP expression.

(6) Fix the bug in *mirror-array* involving dimension parity.

3. GLOSSARY.

(altered-duplicate-of-array 'ar_array 'ty_element-type ['x_exponent ['x_offset ['x_size]]]) [LISP Function]

USE ONLY WHEN: Creating from an array of structures an array of substructures, or numbers. Also occasionally used to change the apparent element type of an array.

RETURNS: A new array which is like that returned by *slice-of-array* but has a change of element type to ty_element-type and exponent to x_exponent. The parent and slice of the new array are equal, and are equal to the current slice of ar_array if no x_offset or x_size are given. Here equality means that the addresses of the elements with the same subscripts are the same, but the the element types and exponents need not be.

If x_offset is given but x_size is not, the addresses of the new array elements are offset from the addresses of the corresponding ar_array elements by x_offset times the size of the new array elements (determined by ty_element-type). The elements of the new array must lie within the corresponding elements of the old array.

If both x_offset and x_size are given, a new X dimension of size x_size is created for the new array, such that the new array elements with given Y, Z, T, U, V subscript values form a contiguous block of memory within an element of ar_array whose X, Y, Z, T, and U subscripts respectively equal the Y, Z, T, U, and V subscripts for the new array. In other words, the dimensions of the old array are pushed down in forming the new array: X becomes Y, Y becomes Z, Z becomes T, T becomes U, U becomes V, and V is discarded. The new array elements with 0 X subscript have addresses offset from the old array elements that contain them by x_offset times the size of the new array elements (determined by ty_element-type). SIDE EFFECT The elements of ar_array are first allocated, if this has not already been done.

DEFAULTS: X_exponent defaults to 0.

BUG: The size in bits of the new array elements must be an exact divisor of the size of the old array elements. This restriction is unfortunate, and is a consequence of storing the *has-increments* attributes in units of one element size. No fix is likely.

NOTE: The new array elements share memory with the corresponding elements of ar_array, so that changes to these ar_array elements will change the corresponding new array elements and vice versa.

NOTE: The *has-been-changed* and *is-readonly* attributes of the new array are set to the corresponding attributes of the old array. The *is-immovable* attribute is set to *nil*.

(**an-array** *has-sizes* '(x_xsize [x_ysize ...]) [SKETCH Type Macro]
[*has-element-type* 'ty_element-type]
[*has-exponent* 'x_exponent]
[*by-expression* 'g_expression]
[*by-value* 'g_value]
[*has-array-file* 'g_array-file]
[*has-offsets* '(n_xoffset [n_yoffset ...])]
[*has-scales* '(n_xscales [n_yscales ...])])

(**an-array** *has-parent-sizes* '(x_xparent-size [x_yparent-size ...]) [SKETCH Type Macro]
[*has-parent-increments* '(x_xparent-increment [x_yparent-increment ...])
[*has-parent-offsets* '(n_xparent-offsets [n_yparent-offsets ...])
[*has-parent-scales* '(n_xparent-scales [n_yparent-scales ...])
[*has-desired-sizes* '(x_xdesired-sizes [x_ydesired-sizes ...])
[*has-desired-origins* '(x_xdesired-origins [x_ydesired-origins ...])
[*has-steps* '(x_xsteps [x_ysteps ...])
[*has-element-type* 'ty_element-type]
[*has-exponent* 'x_exponent]
[*by-expression* 'g_expression]
[*by-value* 'g_value]
[*has-array-file* 'g_array-file]
[*is-readonly* 'g_readonly-switch]
[*is-immovable* 'g_immovable-switch])

(**an-array** 'ar_prototype [SKETCH Type Macro]
[*do-share-elements* g_share-elements]
...)

(allocate-array 'ar_array)	[LISP Macro]
an-array	[SKETCH Type Object]
ar_	[Argument Prefix]
car_	[Argument Prefix]
sar_	[Argument Prefix]
lar_	[Argument Prefix]
far_	[Argument Prefix]
dar_	[Argument Prefix]
ubar_	[Argument Prefix]
ucar_	[Argument Prefix]
usar_	[Argument Prefix]
ular_	[Argument Prefix]
(has-element-type 'ar_array)	[SKETCH Attribute Macro]
(has-exponent 'ar_array)	[SKETCH Attribute Macro]
(has-sizes 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-parent-sizes 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-desired-sizes 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-origins 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-desired-origins 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-steps 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-increments 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-parent-increments 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-parent-offsets 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-offsets 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-parent-scales 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(has-scales 'ar_array ['at/x_length])	[SKETCH Attribute Macro]
(by-expression 'ar_array)	[SKETCH Attribute Macro]
(by-value 'ar_array)	[SKETCH Attribute Macro]
(has-array-file 'ar_array)	[SKETCH Attribute Macro]
(has-array-format 'ar_array)	[SKETCH Attribute Macro]
(has-been-changed 'ar_array)	[SKETCH Attribute Macro]
(is-readonly 'ar_array)	[SKETCH Attribute Macro]
(is-immovable 'ar_array)	[SKETCH Attribute Macro]
default-array-element-type	[LISP Global Variable]
default-array-long-exponent	[LISP Global Variable]
default-array-short-exponent	[LISP Global Variable]

OVERVIEW: A SKETCH array object is an array of elements which are numbers or other SKETCH objects: any SKETCH object type with a *has-size* attribute specifying the length of objects of that type in bits can be the value of the array *has-element-type*.

A numeric element type can be either fixed point or floating point. For fixed point element types (*a-ubit, a-char, a-uchar, a-short, a-ushort, a-long,* or *a-ulong*), the array has a *has-exponent* attribute such that actual value of an element equals the stored value times—

[2] *has-exponent* attribute

A SKETCH array object actually defines two arrays: a parent array, and a slice of that parent. The slice is a part of the parent defined by giving origins for the slice in the parent, sizes for the slice, and multipliers called steps for converting slice subscripts into parent subscripts. These slice parameters can be changed, allowing the slice to move around inside the parent. The parent has some parameters of its own: sizes, increments, offsets, and scales, which cannot be changed.

SKETCH arrays have exactly 6 dimensions, named X, Y, Z, T, U, and V, in that order. Later dimensions may have their size set to 1 if they are not needed.

SKETCH arrays have two modes: allocated and unallocated. In allocated mode memory is allocated for the elements of the array, while in unallocated mode no memory is allocated for the elements. Arrays are initially unallocated, and become allocated only when their elements are referenced. The *allocate-array* macro checks whether an array is allocated, and allocates it elements if not. The call—

(*allocate-array* ar_array)

returns ar_array as its value, while ensuring that the elements of ar_array are allocated.

When the elements of an array are allocated, their initial values may be provided by the *has-array-file*, *by-expression*, or *by-value* attributes explained below.

Most computation on SKETCH arrays is done by C code. The call to *allocate-array* just given is typically used as an argument expression to a LISP call of a C function, in order to ensure that the array passed to the C function is allocated. E.g.—

(*ccheck* (_my_C_function (*allocate-array* my-array) ...))

Array elements can be stored in MOS memory, or in files, or in both places. Arrays which are stored in files have a *has-array-file* attribute, and an associated *has-been-changed* attribute. These keep track of where the array is stored in the file system, and whether the in-MOS version still matches the in-file version.

THE PARENT: The parent array is described by its sizes, increments, offsets, and scales.

The parent array subscript for a given dimension may range from 0 to the size of the dimension minus 1. Each time the subscript is incremented by 1, the address of the element referenced is incremented by the size of the element times the increment of the dimension. Thus if the dimension increment were 4, each unit increment in the subscript would increment the address by 4 times the element size.

Parent subscripts may be mapped onto a real number scale which is used for display purposes. Each dimension may have a scale and an offset. Subscript I is mapped onto the real number—

$$I * \text{scale} + \text{offset} + 0.5 * \text{scale},$$

where the last term is present because subscript I is thought of as denoting the interval—

$$[I - 0.5, I + 0.5)$$

and offset corresponds to the lower bound of the real number interval which the subscript 0 maps onto.

THE SLICE: The slice is determined by the parent and by the following parameters: sizes, origins, and steps. The sizes are the dimension sizes of the slice. The origins are the parent array subscripts onto which the slice 0 subscripts map. The steps are the parent array subscript changes that correspond to a the slice subscript changes of +1. Thus for each dimension—

$$\text{parent-subscript} = \text{origin} + \text{step} * \text{slice-subscript}$$
$$0 \leq \text{slice-subscript} < \text{slice-size}.$$

It is possible to try to specify a slice that does not fit inside the parent array. If this happens, the actual slice sizes and origins are modified until the slice does fit inside the parent, or if this is impossible, the slice sizes are all set to 0. Thus the slice has two sets of sizes: desired sizes and actual sizes; and two sets of origins: desired origins and actual origins. The actual sizes and origins are computed from the desired sizes and origins, plus the steps and parent parameters, by a process called clipping.

CLIPPING: Clipping reduces the actual sizes and changes the actual origins of the slice so that it will fit inside the parent array.

One form of clipping changes the actual slice origins until they are legal parent array subscripts. This is done by repetitively adding the the dimension's slice step to the actual slice origin until the actual origin is legal, and reducing the actual slice size by 1 for each step added. If this process will not work (because adding the step would move the origin away from 0 instead of toward 0), the actual size will be made 0.

The other form of clipping merely reduces the actual slice sizes until the largest slice subscripts map onto legal parent subscripts.

If any actual size is reduced to 0 by clipping, then all the actual sizes are forced to 0.

ARGUMENT PREFIXES: Prefixes such as *ubar_* and *sar_* specify that the argument is of *an-array* type and has a particular element type, according to the following table—

Argument Prefix		Element Type
ar_	—	unspecified
ubar_	*a-ubit*	1-bit unsigned integer.
car_	*a-char*	8-bit signed integer.
ucar_	*a-uchar*	8-bit unsigned integer.
sar_	*a-short*	16-bit signed integer.
usar_	*a-ushort*	16-bit unsigned integer.
lar_	*a-long*	32-bit signed integer.
ular_	*a-ulong*	32-bit unsigned integer.
far_	*a-float*	32-bit floating point number.
dar_	*a-double*	64-bit floating point number.

HAS-ELEMENT-TYPE: The type of the elements of an array. *A-type* object. Possible values include *a-long*, *a-short*, and *a-char* for signed 32, 16, and 8 bit integers; *a-ulong*, *a-ushort*, *a-uchar*, and *a-ubit* for unsigned 32, 16, 8, and 1 bit integers; and *a-double* and *a-float* for 64 and 32 bit floating point numbers.

Other values are possible: array elements may be C structures.

The default element type for array creation is stored in the global variable **default-array-element-type**, which itself defaults to *a-long*.

Any element type value must have a *has-size* attribute specifying the length of the array element in bits if the array elements are allocated. For example, (*has-size a-long*) equals 32. The *has-size* attribute of the element type does not have to be known for arrays whose elements are never allocated.

HAS-EXPONENT: A fixnum: block floating point elements of ar_array should be considered as integers which are multiplied by $2^{\text{x\_exponent}}$ to get the true numeric value of the element, where x_exponent is the *has-exponent* attribute of ar_array.

If ar_array's element type is *a-long*, the *has-exponent* attribute defaults to the value of the global variable **default-array-long-exponent**, which itself defaults to -16. If ar_array's element type is *a-short*, the *has-exponent* attribute defaults to the value of the global variable **default-array-short-exponent**, which itself defaults to -8. Otherwise the *has-exponent* attribute defaults to 0.

The *has-exponent* attribute is not used if the element type is *not a-long, a-ulong, a-short, a-ushort, a-char, a-uchar*, or *a-ubit*.

AT/X_LENGTH: Many array attributes are lists of numbers, one for each dimension. Examples are *has-sizes*, *has-increments*, and *has-origins*. When one of these attributes is gotten, a list of 6 numbers is normally returned, since that is the normal number of dimensions of an array.

If the at/x_length parameter is given as a positive integer while getting one of these attributes, then it specifies the length of the list returned. The numbers added to the end of the list, if this number is larger than 6, depend upon the particular attribute: e.g., *has-sizes* adds 1's, *has-increments* adds the minimum number of elements required to hold the array, and *has-origins* adds 0's.

If the at/x_length parameter is given the value *do-shorten*, the list gotten is shortened by omitting from its end any numbers equal to the number that would be added to the end of the list if it were to be lengthened. E.g., 1's are omitted from the end of *has-sizes*, and 0's from the end of *has-origins*.

For some attributes, like *has-offsets*, *nil* may be an element of the attribute value list, just as if it were a number. For these attributes *nil* is the element added to lengthen a list, or the element removed from the end to shorten the list.

HAS-SIZES:

HAS-PARENT-SIZES:

HAS-DESIRED-SIZES: The list of the integer dimension sizes of the array. There are three such lists: the parent sizes, the desired slice sizes, and the actual slice sizes. If the slice is the same as the parent, all three lists are the same, and may be denoted by the *has-sizes* attribute.

The parent sizes may be set when the array is initialized, but not *setf*. The desired sizes may be *setf* at any time, to change the definition of the slice. The actual sizes are computed according to the rules of CLIPPING above.

Allowable subscripts for a given dimension range from 0 through the actual dimension size minus 1.

The *has-sizes* attribute returns the actual sizes when it is read. However, it sets the desired sizes when it is *setf*. It also serves as the default for both parent and desired sizes when an array is initialized.

The integer sizes must be non-negative. If any are 0, the parent or slice is empty. Size integer lists may be abbreviated by omitting 1's at the end, as long as the list does not become empty.

When the *has-sizes*, *has-parent-sizes*, or *has-desired-sizes* attributes are read, an extra at/x_length argument may be given to specify the length of the list of sizes returned. This list will then be either truncated, filled out by adding 1's, or shortened by omitting 1's from the end.

When the *has-sizes* or *has-desired-sizes* element is *setf*, *nil* may be used as a new dimension size in order to indicate that the corresponding dimension size is not to be changed. Also, sizes omitted from the end are taken to be *nil* in this sense, and are not assumed to be 1. For example,

(*setf* (*has-sizes* x '(*nil* 8)))

will change only the 2'nd (Y) dimension's desired size.

When *an-array* is printed or unevaluated, the *has-parent-sizes* attribute label is replaced by the *has-sizes* attribute label in the output. If both desired and parent sizes are the same, only one sizes attribute is output, with the *has-sizes* label.

HAS-ORIGINS:

HAS-DESIRED-ORIGINS: A list of the origins of the current ar_array slice within its parent array. The origins are the parent subscripts to which the slice 0 subscripts correspond. Because of clipping, there are two sets of origins: the desired origins, and the actual origins. The desired origins may be *setf*, but the actual origins are computed according to the rules of CLIPPING above.

An origin list may be of any length with 0's omitted from the end.

The *has-origins* attribute returns the actual origins when it is read, but sets the desired origins when it is *setf* or used as an initial value.

When the *has-origins* or *has-desired-origins* attributes are read, an extra at/x_length argument may be given to specify the length of the list of origins returned. This list will then be either truncated, filled out by adding 0's, or shortened by omitting 0's from the end.

A value being *setf* to the desired origins may have a *nil* origin for a particular dimension, in which case the value of that dimension's desired origin will not be changed. Origins omitted from the end are taken to be *nil* in this sense, and are not assumed to be 0.

When *an-array* is printed or unevaluated, the *has-desired-origins* attribute label is replaced by the *has-origins* attribute label in the output.

HAS-STEPS: A list of the integer steps of the current slice within the parent array. For each dimension, the step is the increment in the parent array subscript that corresponds to a unit increment in the slice subscript. A step may be negative or zero. The steps may be *setf*.

The list may be of any length with 1's omitted from the end.

When the *has-steps* attribute is read, an extra at/x_length argument may be given to specify the length of the list of steps returned. This list will then be either truncated, filled out by adding 1's, or shortened by omitting 1's from the end.

When the steps are *setf*, the new value may have a *nil* step for a particular dimension, in which case the value of that dimension's step will not be changed. Steps omitted from the end are taken to be *nil* in this sense, and are not assumed to be 1.

HAS-INCREMENTS:

HAS-PARENT-INCREMENTS: A list of the integer increments for the array. An array is mapped onto a one dimensional vector by multiplying each subscript by its corresponding increment, summing these products, adding a vector origin, and using the result as a vector index. The vector origin specifies the array element which has all zero subscripts, and the index is measured in units one array element long.

Increments are therefore measured in units that are one array element long. Thus for elements of type *a-ubit*, increments are measured in units of 1 bit, whereas for elements of type *a-long*, increments are measured in units of 32 bits.

There are two sets of increments. The parent increments are set when the array is initialized, and cannot be *setf*. The slice increments always equal the parent increments times the steps, for each dimension. They cannot be *setf*, but change when the steps change.

A list of increments may be of any length. A value omitted from the end of a list of parent increments must equal the size N, in units of one array element, of the smallest vector that will include all elements with the subscripts of the previous dimensions ranging over their entire range, according to the sizes and increments of these previous dimensions. Here the dimensions are those of the parent array, and not the slice.

The *has-increments* attribute returns the slice increments when it is read, but sets the parent increments when it is used as an initial value. Neither kind of increments can be *setf*.

When the *has-increments* or *has-parent-increments* attributes are read, an extra at/x_length argument may be given to specify the length of the list of increments returned. If N is as above, then the list returned will be either truncated, filled out by adding N's, or shortened by omitting N's from the end.

When *an-array* is printed or unevaluated, the *has-parent-increments* attribute label is replaced by the *has-increments* attribute label in the output. Slice increments are never printed and never appear in an unevaluated array.

HAS-OFFSETS:

HAS-PARENT-OFFSETS: A list of *nil*'s or *flonum*'s. Each *flonum* is the value that -0.5 is mapped onto by the ruler associated with the corresponding dimension. A *nil* indicates there is no ruler for the dimension. Values omitted from the end of the offsets list must equal *nil*.

There are two such lists: the offsets of the parent array, and those of the slice. The latter may be computed from the parent offsets and scales, and the slice origins, by the formula—

slice-offset = parent-offset + parent-scale * actual-slice-origin.

A slice offset is *nil* if the corresponding parent offset or scale is *nil*.

The *has-offsets* attribute returns the slice offsets when it is read, but sets the parent offsets when it is used as an initial value or *setf*. The parent offsets can be *setf* only when their previous value is *nil*. Slice offsets cannot be initialized or *setf*.

When the *has-offsets* or *has-parent-offsets* attributes are read, an extra at/x_length argument may be given to specify the length of the list of offsets returned. This list will then be either truncated, filled out by adding *nil*'s, or shortened by omitting *nil*'s from the end.

When *an-array* is printed or unevaluated, the *has-parent-offsets* attribute label is replaced by the *has-offsets* attribute label in the output.

HAS-SCALES:

HAS-PARENT-SCALES: A list of *nil*'s or *flonum*'s. Each *flonum* is the scale of the ruler associated with the corresponding dimension. The scale is the increment in ruler range for 1 unit increment in ruler domain. A *nil* indicates there is no ruler for the dimension. Values omitted from the end of the offsets list must equal *nil*.

There are two such lists: the scales of the parent array, and those of the slice. The latter is the product of the parent scales and the slice steps.

The *has-scales* attribute returns the slice scales when it is read, but sets the parent scales when it is used as an initial value or *setf*. The parent scales can be *setf* only if their previous value is *nil*. The slice scales cannot be *setf*.

When the *has-scales* or *has-parent-scales* attributes are read, an extra at/x_length argument may be given to specify the length of the list of scales returned. This list will then be either truncated, filled out by adding *nil*'s, or shortened by omitting *nil*'s from the end.

When *an-array* is printed or unevaluated, the *has-parent-scales* attribute label is replaced by the *has-scales* attribute label in the output.

PROTOTYPES:

DO-SHARE-ELEMENTS: Evaluating an expression such as—

(*an-array* ar_prototype *has-element-type a-short* ...)

will make a copy of ar_prototype, replacing any ar_prototype attributes by those given explicitly in the expression. Ar_prototype is called the prototype.

If the elements of ar_prototype have been allocated, then the elements of the result will be allocated, and will be copies of the elements of ar_prototype. In this case the parent sizes of the result must equal those of ar_prototype. Because array elements must be copied, only numeric type elements can be handled. The result's *has-exponent* will be computed by *array-copy-exponent* if the result does not have the same element type as ar_prototype and the *has-exponent* attribute is not explicitly given a numeric value in the expression above. *Array-copy-exponent* will also be used if the *has-exponent* attribute is explicitly given the value *nil* in the expression above, even if the result has the same element type as ar_prototype.

If g_share-elements is present and non-*nil*, the elements of ar_prototype will be allocated if that has not already been done, and the result will share elements with ar_prototype. In this case no attempt must be made to change the parent size, parent increments, element type, or exponent of the prototype while making the result.

BY-EXPRESSION: If the *by-expression* attribute value, g_expression, is non-*nil*, it will be used to compute an initial value for each element of the parent array when the array's elements are allocated. The variables *X*, *Y*, *Z*, *T*, *U* and *V* may be used in g_expression as the parent element subscripts. No other variables should be used in g_expression (variables may be substituted into the expression when the expression is computed by using ' and ,).

The *by-expression* attribute of an array is set to *nil* right after it is used to initialize the array elements.

Currently the element type of the array must be numeric for *by-expression* to work.

Because the interpreter is used, this way of initializing an array is very slow if the array is large.

BY-VALUE: If the *by-value* attribute value, g_value, is non-*nil*, it will be used to compute an initial value for each element of the parent array, when the array's elements are allocated. G_value is a list of sublists of sublists ... of element values that are used to initialize the elements. The innermost lists correspond to the first (X) dimension: e.g.—

(*an-array has-sizes* '(2 3)
by-value '((00 01) (10 11) (20 21)))

and—

(*an-array has-sizes* '(2 3)
by-expression '(*plus X* (*product* 10 *Y*)))

give the same initial elements.

If elements or sublists are omitted from the end of a list, *nil* values will be assumed.

The *by-value* attribute of an array is set to *nil* right after it is used to initialize the array elements.

Currently the element type of the array must be numeric for *by-value* to work.

Because the interpreter is used, this way of initializing an array is very slow if the array is large.

HAS-ARRAY-FILE:

HAS_ARRAY_FORMAT:

HAS-BEEN-CHANGED: The *has-array-file* attribute value g_array-file can be a symbol—

s_file-name

equal to the name of the file in which the parent array is stored, or a list of the form—

(s_file-name x_offset)

specifying the name of a file in which the parent array is stored beginning at the given offset, or a list of the form—

(s_volume-name x_file-number x_record-number x_offset)

specifying the name of a magnetic tape volume containing the parent array, the file number of the file on the tape that contains the array, the record number of the record in the file that contains the array, and the offset of the parent array in that record.

The form—

s_file-name

is equivalent to—

(s_file-name 0).

The offset specified is the offset of the first byte of the parent array, which need not be the first byte of the (0 0 0 0 0 0) element if the parent array has negative increments.

The *has-been-changed* attribute keeps track of whether the MOS memory and file system versions of an array are equal. It is *t* if array elements are allocated and any parent element has been changed since the array was last written to or read from its *has-array-file* attribute. It is *nil* otherwise.

The *has-been-changed* attribute can be *setf* to *t*, but cannot be *setf* to *nil* or initialized. Any non-*nil* value is equivalent to *t*. Setting it to *t* will allocate the array elements if that has not already been done.

The mechanism for keeping track of changes is imprecise, in that whenever two arrays share elements, setting the *has-been-changed* attribute of one to *t* will set the same attribute of the other to *t*. And if this second array shares an element with a third array, the third array will also have its *has-been-changed* attribute to *t*, even if it does not share any elements with the first array. The errors in this bookkeeping will always set *has-been-changed* to *t* when it should be left *nil*. It will never be set *nil* when it should be left *t*.

The *has-array-format* attribute is the value of the **computer-format** global variable at the time the array was written on disk (it is *nil* if *has-array-file* is *nil*. This specifies the binary format used to store numbers on disk. Sometimes arrays written on one computer may be read on another computer with different number formats, provided an appropriate conversion routine is provided (there is currently no mechanism for doing this implicitly).

IS-READONLY: The *is-readonly* attribute is *t* if it is not permissible to write elements of ar_array; *nil* if it is permissible.

Can be *setf*. When *setf*, any non-*nil* value is equivalent to *t*.

Overlapping arrays may still be writable even if this array is not, in which case this array's elements might still get changed even if it is readonly.

IS-IMMOVABLE: The *is-immovable* attribute is *t* if it is not permissible to change the ar_array attributes which control the position of the current ar_array slice, namely the *has-desired-sizes*, *has-desired-origins*, and *has-steps* attributes; *nil* if these can be changed.

Can be *setf*. When *setf*, any non-*nil* value is equivalent to *t*.

GARBAGE COLLECTION: Array elements (but not the array proper) may be garbage collected if the array is not stored in a local or global variable, as a function argument or return value, or as an element of a list stored in one of these placed (but not as an element of a sublist of a list stored in one of these places).

An array whose elements have been garbage collected in this way has its *is-collected* attribute set to *t*. Normal array functions will signal an error if passed such an array. .

Thus it is impossible to store arrays in complex data bases, unless these arrays are bolted down by storing them in a global variable, or as an element of a list stored in a global variable.

The purpose of this form of element garbage collection is to allow the mark and sweep algorithm for finding unused arrays to run much faster than the similar algorithm that finds unused objects. This permits array allocation to use memory at a faster rate than normal object allocation, without the garbage collector becoming a major drain of CPU time.

(**an-array-summary** *has-count* 'x_count [SKETCH Object]
has-missing-count 'x_missing-count
has-mean 'f_mean
has-standard-deviation 'f_standard-deviation
has-maximum 'f_maximum
has-minimum 'f_minimum
has-sum 'f_sum
has-sum-squares 'f_sum-squares)

an-array-summary [SKETCH Type]
asum_ [Argument Prefix]
(**has-count** 'asum_summary) [SKETCH Attribute Macro]
(**has-missing-count** 'asum_summary) [SKETCH Attribute Macro]
(**has-mean** 'asum_summary) [SKETCH Attribute Macro]
(**has-standard-deviation** 'asum_summary) [SKETCH Attribute Macro]
(**has-maximum** 'asum_summary) [SKETCH Attribute Macro]
(**has-minimum** 'asum_summary) [SKETCH Attribute Macro]
(**has-sum** 'asum_summary) [SKETCH Attribute Macro]
(**has-sum-squares** 'asum_summary) [SKETCH Attribute Macro]

VALUE: An object that summarizes the elements of an array. In the various summary statistics missing values are not counted, except of course in *has-missing-count* which is the count of missing values. *Has-count* is the count of non-missing values, only. Some of the statistics may be missing (*nil*) if there are not enough non-missing values.

.ar [UNIX File Extension]

FILE FORMAT: A file containing array elements: e.g. a file such as *cache.ar* written when *uneval-object* is called by *write-catalog* and referenced in the *has-array-file* attributes of *an-array* objects.

array-block-region-sizes [LISP Global Variable]

USER ONLY WHEN: Having problems with memory overflow.

USE: **array-block-region-sizes** is a list of the sizes in bytes of the contiguous regions that will be allocated to hold array element blocks.

A new region is allocated when an array is to be allocated that will not fit into existing regions. Elements of the **array-block-region-sizes** list are removed until one is removed that is larger than the size of the array. All the removed elements are added to determine the size of the new region.

If the array is too large for this to work, no elements of the list are removed, and a region just large enough for the array is allocated. This desparation strategy is likely to cause later problems.

Setting **array-block-region-sizes** to a list whose only element is the total amount of storage available for arrays is the best way to avoid running out of space because, it avoids array storage fragmentation. However, such a setting will lead to a substantial delay when the single large region is allocated.

array-blocks-history [LISP Global Variable]
array-blocks-history-length [LISP Global Variable]

VALUES: **array-blocks-history** is a history of the time spent in sweeping and compacting array blocks. Its length is limited to **array-blocks-history-length** entries, and the latter defaults to 50.

(array-copy-exponent 'ar_array 'ty_element-type) [LISP Function]

WHERE: Ty_element-type and the element type of ar_array are both numeric.

RETURNS: The *has-exponent* value most appropriate for a copy of ar_array which has element type ty_element-type.

If ty_element-type is *a-float*, *a-double*, or *a-ubit*, the exponent is 0.

If ty_element-type is *a-long* or *a-ulong*, the elements of the array are scaled to fit into the low order 24 bits of the 32 bit integers.

If ty_element-type is *a-short*, *a-ushort*, *a-char*, or *a-uchar*, the elements are scaled to use all available precision of ty_element-type without any clipping on copying.

If ar_array has all zero or missing value elements, the exponents are -16 for *a-long* and *a-ulong*, -8 for *a-short* and *a-short*, and 0 for *a-char* and *a-uchar*.

(bounds-of-array 'ar_array [n_factor]) [LISP Function]

WHERE: N_factor defaults to 0.00001.

RETURNS:

(n_lower-bound n_upper-bound),

where n_lower-bound is the minimum of all the elements of ar_array minus epsilon, and n-upper-bound is the maximum of all the elements of ar_array, plus epsilon. Epsilon is chosen to be—

factor * ((maximum of all array elements) - (minimum of all array elements))

in order to expand the interval slightly. If epsilon would be zero, n_factor is substituted for it. If epsilon cannot be computed because the array has no non-missing elements,

(-n_factor n_factor)

is returned.

(collect-array-blocks) [LISP Function]

USE ONLY WHEN: Playing with the array block garbage collector (which is normally automatic).

SIDE EFFECT: Calls first *sweep-array-blocks* and then *compact-array-blocks*.

RETURNS: The sum of the number of fixnum's in all the free blocks.

(compact-array-blocks) [LISP Function]
compact-array-blocks-count [LISP Global Variable]
compact-array-blocks-time [LISP Global Variable]
compact-array-blocks-bytes [LISP Global Variable]

USE ONLY WHEN: Playing with the array block garbage collector (which is normally automatic).

SIDE EFFECT: Compacts all the blocks in the array block allocation area. Does *not* mark the free blocks first: call *sweep-array-blocks* to do that.

compact-array-blocks-count is incremented every time *compact-array-blocks* is called, **compact-array-blocks-time** has the time taken by the call added to it, and **compact-array-blocks-bytes** has the number of bytes moved by the compactification added to it. All these variables are initialized to 0. The time is measured in the same units as *ptime*: see **ptime-counts-per-second**.

RETURNS: The sum of the number of fixnum's in all the free blocks.

(copy-array 'ar_output 'ar_input) [LISP Function]

WHERE: Ar_output and ar_input are similar.

RETURN: Ar_output. Copies ar_input elements to the corresponding ar_output elements. Both arrays must have numeric element types, but these and the exponents need not agree.

BUG: Currently cannot handle non-numeric element types.

(copy-of-array 'ar_input) [LISP Function]

RETURN: A new array which is a copy of the current ar_input slice and has the same element type, exponent, offsets, scales, and parities as ar_input. More explicitly, the parent sizes, offsets, and scales of the result equal the slice sizes, offsets, and scales of ar_input.

BUG: Currently cannot handle non-numeric element types.

(duplicate-of-array 'ar_array) [LISP Function]

RETURNS: A new array which shares elements with ar_array, and has the exact same standard array attributes, except for the *is-immovable*, *has-array-file*, *by-expression*, and *by-value* attributes, which are set to *nil* in the result.

The elements of ar_array are allocated if this has not already been done.

Attributes of ar_array which are not documented in this package are not copied to the result.

(format-object 'ar_array ...) [LISP Function]

EQUIVALENT TO: The usual *format-object*, except that ar_array's *has-parent-sizes*, *has-parent-increments*, *has-desired-sizes*, *has-desired-origins*, *has-steps*, *has-parent-offsets*, *has-parent-scales*, and *has-parities* values are printed with some editing. The ends of the list values of these attributes are truncated in the same manner as the *do-shorten* option truncates the lists when they are gotten. Also, *has-parent-sizes* is renamed *has-sizes*, *has-parent-increments* is renamed *has-increments*, *has-parent-offsets* is renamed *has-offsets*, and *has-parent-scales* is renamed *has-scales*.

(has-array-descriptor 'ar_array) [LISP Macro]

EQUIVALENT TO:

(*allocate-array* ar_array)

Has-array-descriptor is an anachronism and will be removed in the future.

(has-element 'ar_array 'x_subscript ...) [SKETCH Attribute]
(has-element 'ar_array '(x_subscript ...)) [SKETCH Attribute]

WHERE: The element type of ar_array must be numeric, or *an-1bit*, or a vector type declared by *declare-vector-type*.

These latter vector elements should not have any single subelement that is stored in both the vector and hunk part (e.g. *in-both* subelements).

VALUE: The element of the current ar_array slice corresponding to the given subscripts.

A numeric value may be *nil* if the array element type allows signed numbers.

Numeric values will be *fixnum* if they are exact integers and *flonum* otherwise.

For vector values, only *in-vector* subelements are remembered in the array, and all *in-hunk* subelements will be set to *nil* in the element value read.

Subscripts range from 0 to 1 less than the size of the corresponding dimension. 0's may be omitted from the end of the subscript list.

WHEN SETF: Numeric values will be automatically converted from *fixnum* to *flonum* and vice versa if necessary.

Non-*nil* *an-1bit* values will be converted to *t*.

Only the *in-vector* subelements of a vector value will be stored in the array: the *in-hunk* subelements will not be stored, and will return as *nil* when the array element is read.

BUG: If for a vector element there is a subelement in both the vector and hunk part, the value returned will have inconsistent values for the subelement in the two different places it is stored, unless the subelement value is *nil*.

(**has-parent-ruler** 'ar_array 'x_dimension) [SKETCH Attribute Macro]
(**has-ruler** 'ar_array 'x_dimension) [SKETCH Attribute Macro]

VALUE: The rulers associated with the x_dimension dimension of ar_array. These are—

'(.size (.offset nil) .scale).

where the size, offset, and scale are for either the parent dimension or the slice dimension. This ruler maps -0.5 onto the offset and has the given scale.

Has-parent-ruler always refers to the parent dimension, while *has-ruler* refers to the slice dimension when gotten, and the parent dimension when *setf*.

Returns *nil* when gotten unless both offset and scale are non-*nil*.

(**inspect-array** 'ar_array [LISP Function]
['(i_size ...) ['(i_origin ...) ['(i_step ...)]]])

SIDE EFFECT: Applies *slice-of-array* to the argument list and prints the resulting slice using *print-array*. Then echos the current slice desired origins, and waits for the user to type a command. Typing one of the 8 letters surrounding 's' on the standard terminal keyboard moves the slice in the corresponding direction: left, right, up, down, or diagonal. Here a lower case letter moves by half the size of the slice X or Y dimension, and an upper case letter moves by the full size of the dimension.

Typing a set of integers in parentheses sets the origins to the values indicated (use a *nil* value for no-change).

Typing a control-D or $ exits the inspect-array function.

Except for the control-D, multiple commands may be typed on one line, in which case all are executed first, and then the current slice is printed on the console terminal. Just typing a carriage return by itself reprints the current slice.

(**integerize-ruler** 'rul_ruler) [LISP Function]

WHERE: Rul_ruler is a possibly unnormalized ruler that can be normalized.

RETURNS: A ruler with the same mapping as rul_ruler, but with a possibly different domain which is represented by a single non-negative integer N. N is chosen to be the largest integer such that—

[N-1.5 N-0.5)

intersects the domain of rul_ruler, if N is non-negative. If N would be negative, N is chosen to be 0. The domain of the ruler returned is—

[-0.5 N-0.5).

The ruler returned is normalized except for its domain description, which consists of just the integer N, rather than a list of two bounds.

(log-bound-of-array 'ar_array ['x_sign]) [LISP Function]

RETURNS: The logarithm base 2 of the smallest power of 2 such that all non-missing elements of ar_array have an absolute value less than that power of 2. If x_sign is given and > 0, all negative elements will be treated as missing, while if x_sign is given and < 0, all positive elements will be treated as missing. Zero elements are always treated as missing. If all the elements of ar_array are missing, *nil* is returned.

(lookat-arrays ['s_name ...]) [LISP Function]

RETURNS: A list of pairs (s_name ar_value) for all symbols s_name whose value is an array.

If no arguments are given, the arguments are taken to be the list of all symbols in the symbol table.

(map-by-ruler 'n_number 'rul_ruler) [LISP Function]

WHERE: Rul_ruler should be normalized. No checking is performed for invalid rul_ruler's (for efficiency reasons).

RETURNS: The number obtained by applying the affine transformation defined by rul_ruler to n_number.

(mirror-array 'ar_output 'ar_input) [LISP Function]

WHERE: There is a list of integers '(i_width ...) corresponding to dimensions such that the size of ar_output for a dimension equals the size of ar_input for the dimension plus 2 times the i_width value for the dimension. Each i_width must be non-negative, but may be zero.

RETURNS: Ar_output after setting its elements.

SIDE EFFECT: Ar_input is copied into—

(*slice-of-array* ar_output
(*has-sizes* ar_input) '(i_width ...)).

The rest of ar_output is filled in by mirror reflections of the slice. E.g., just "above" and "below" the slice in any dimension is placed a copy of the slice which is identical to the slice except its subscript order is reversed for the given dimension.

NOTE: The element types of ar_output and ar_input may differ, as long as both are numeric.

BUG: Currently each i_width may *not* exceed the size of the corresponding ar_input dimension.

BUG: Really each dimension should be given a parity which determines whether the elements with subscript 0 or size−1 should be duplicated or not. I.e., should a duplicate of the top row be placed just above the top row, or should a duplicate of the second row down be placed just above the top row, leaving the top row unduplicated. Sometimes the answer is yes and sometimes no, depending on the array, and the answer, or 'dimension parity', can be computed automatically. Currently the top row is always duplicated.

(mirror-of-array ar_array '(i_width ...)) [LISP Function]

RETURNS: A newly created array each of whose dimensions is larger than the corresponding ar_array dimension by 2 times the corresponding i_width. The new array is set by calling the *mirror-array* function with it as the output and ar_array as input.

(move-array 'ar_array 'x_dimension 'x_change) [LISP Function]
(move-array 'ar_array '(x_change ...)) [LISP Function]

RETURNS: Ar_array after its current slice is modified.

SIDE EFFECT: The form with x_dimension (see *X-dimension*) changes the origin of the specified dimension by the amount x_change * step, where step is the slice step parameter for the dimension. The form with the list of changes applies these changes to the X, Y, Z, T, U, and V dimensions in order, with changes not given or being given as *nil* being taken as 0 (no change).

(normalize-ruler 'rul_ruler) [LISP Function]

RETURNS: Rul_ruler after it is normalized. An error occurs if rul_ruler cannot be normalized or does not have the format of a valid ruler.

(place-array 'ar_array 'x_dimension 'x_size ['x_origin ['x_step]]) [LISP Function]
(place-array 'ar_array '(x_size ...) ['(x_origin ...) ['(x_step ...)]]) [LISP Function]

WHERE: Any x_size's must be non-negative, but x_origin's and x_step's may equal any integer.

RETURNS: Ar_array after its current slice is modified.

SIDE EFFECT: Modifies the current slice of ar_array by setting the parameters of its dimensions. The three parameters, x_size, x_origin, and x_step, refer to the desired size of the slice dimension, the desired value of the parent array subscript corresponding to slice subscript 0, and the desired increment in the parent array subscript that corresponds to a unit increment of the slice subscript. They are the same values as integers listed in the array's *has-desired-sizes*, *has-desired-origins*, and *has-steps* attributes.

The form with x_dimension given (see *X-dimension*) sets the three parameters for the specified dimension.

The form with the lists of x_size's, x_origin's, and x_step's sets the parameters for all dimensions in the usual order X, Y, Z, T, U, V.

Omitted parameters, or parameters explicitly given the value *nil*, default to the current slice parameter values: i.e. to no change in the parameter.

(prepare-array 'ar_array '(x_expand ...) ['ty_element-type ['x_exponent']] [*must-copy*]) [LISP Function]

array-expander [LISP Global Variable]

RETURNS: Either ar_array, or a new array which is a copy of the current slice of ar_array, or an expanded version thereof. The array returned will have the given element type and exponent, if they are given and not *nil*.

If the '(x_expand ...) list is *nil*, or all x_expand values are 0, the array returned will have the same sizes as the current slice of ar_array. In this case, if ar_array has the correct element type and exponent, and if the *must-copy* switch is absent, ar_array itself will be returned. Otherwise a new array will be returned whose elements are copies of those of ar_array.

If x_expand is non-*nil*, a new array is created whose sizes are larger than those of ar_array by the values in the '(x_expand ...) list rounded up to the next even integers. Then the call—

(*funcall* **array-expander** ar_new-array ar_array)

is executed, and its result returned as the value of *prepare-array*. The usual value for **array-expander** is '*mirror-array*, which is in fact the default value of **array-expander**.

The above call to **array-expander** will be made if the element type or exponent of the array must be changed, even if the sizes are not being changed.

NOTE: If no exponent is specified when changing the type of numeric elements to a block floating point type, an exponent is computed automatically using *array-copy-exponent*. If the element type is not changed, *array-copy-exponent* will not be used.

BUG: Cannot not handle non-numeric array element types.

(print-array ar_array [p_port [x_length]]) [LISP Function]

WHERE: The element type of ar_array must be numeric or *an-1bit*.

DEFAULTS: P_port defaults to *poport* and x_length defaults to the value of **line-length** (which itself defaults to 80).

SIDE EFFECT: The array is printed to the p_port. A line length of x_length columns is assumed. The X dimension is the columns dimension, the Y dimension is the rows dimension.

If the length is 80 columns, either 5, 10 or 80 columns will print on the first line of a row, depending on the array element type. Excess columns for a row are printed on subsequent lines which are each indented by increasing amounts.

If the Z, T, U, or V dimension sizes are above 1, then multiple two dimensional arrays are printed, each labeled by (* * Z T U V).

Missing values are printed as *nil*.

BUG: The output format is not LISP readable and leaves something to be desired when a row spills onto more than one line. It will probably be changed in the future.

(read-array-elements 'ar_array 'g_array-file) [LISP Function]

RETURNS: Ar_array after its elements have been modified.

SIDE EFFECT: Reads from the file location specified by g_array-file into the parent array of ar_array. Sets the *has-array-file* attribute of ar_array to g_array-file. Sets the *has-been-changed* attribute to *nil*.

See HAS-ARRAY-FILE under *an-array* for the layout of g_array-file.

(reorganization-of-array 'ar_array '(x_xorigin ...) '(x_xsize ...) ['(x_xincrement ...)]) [LISP Function]

USE ONLY WHEN: A crazy array reorganization is required.

WHERE: Unspecified origins default to 0, unspecified sizes default to 1, and unspecified increments are computed in the same manner that they are computed when a new array is created by *an-array*.

RETURNS: A new array that shares elements with ar_array. The parent sizes of the new array are x_xsize .... and the parent increments are x_xincrement. The parent origin element of the new array is the element of ar_array designated by the subscripts x_xorigin ... .

The current slice of the new array equals the new array's parent.

WARNING: If one is not careful, some elements of the new array may not be elements of ar_array. However, they will be elements of some ancestor of ar_array (a parent of ar_array, or of an array from which ar_array was made by *slice-of-array*, or something like that).

It is an error if there is no ancestor in which the new array will fit.

(reset-array 'ar_array) [LISP Function]

RETURNS: Ar_array after it is modified.

SIDE EFFECT: Resets the array slice so that it equals the parent.

(restrict-array 'ar_array 'x_dimension ['x_size ['x_origin ['x_step]]]) [LISP Function]

RETURNS: Ar_array after its current slice is modified.

EQUIVALENT TO: (*place-array* ar_array D x_size x_origin x_step) for all D >= x_dimension.

(reverse-array 'ar_array 'x_dimension) [LISP Function]

RETURNS: Ar_array after its current slice is modified.

SIDE EFFECT: Reverses the order of subscripts in the dimension specified by x_dimension within the current ar_array slice by negating the dimension's step and changing the dimension's desired origin appropriately to—

old desired origin + (desired size - 1) * (old step).

(round-ruler 'rul_ruler 'n_factor ['l_pattern]) [LISP Function]

WHERE: N_factor > 0.

RETURNS: A ruler made by from rul_ruler by the following alterations. First, rul_ruler is normalized. Second, the ruler scale is rounded away from zero until it is an exact multiple of n_factor. Third, the ruler is matched against l_pattern, and any bound that matches a *nil* in l_pattern is changed to *nil*. There should be only one such bound, and changing it to *nil* will provide the freedom necessary to recompute it and make the ruler valid.

The resulting ruler is unnormalized, since it has a *nil* bound.

L_pattern defaults to ((*t t*) (*t nil*)), which allows the second range bound to change.

"ruler" [SKETCH Term]
"normalized ruler" [SKETCH Term]
rul_ [Argument Prefix]

NORMALIZED RULERS: A normalized ruler is basically a list in the format—

((f_domain-bound-1 f_domain-bound-2)
(f_range-bound-1 f_range-bound-2)
f_scale),

where all the numbers are floating point. This list specifies an affine map from the oriented real interval—

[f_domain-bound-1 f_domain-bound-2)

to the oriented real interval—

[f_range-bound-1 f_range-bound-2).

Note that these intervals may have their bounds switched: that is—

f_domain-bound-1 > f_domain-bound-2

or—

f_range-bound-1 > f_range-bound-2

are allowed. In any case, the map sends f_domain-bound-1 onto f_range-bound-1, and f_domain-bound-2 onto f_range-bound-2.

F_scale is the multiplier used in the ruler affine map. A ruler is invalid unless—

(f_range-bound-2 - f_range-bound-1) ==
    f_scale * (f_domain-bound-2 - f_domain-bound-1).

INTEGER PARAMETERS: If any of the five parameters in a ruler are integers, the ruler is considered to be unnormalized. It may be normalized in this case simply by replacing the integers by their floating point equivalents.

CONVERSION TO DISCRETE BINS: To convert the real line to discrete bins, we assign each real number to the nearest integer: we *round*. As a consequence, each integer can be thought of as the midpoint of an interval of length 1.0. If the integers from 0 through N-1 are to represent N bins covering the real interval [L U], the ruler that should be used to represent this fact is—

((-0.5 N-0.5) (L U)).

A real interval in a ruler can be replaced by a single integer N. This integer stands for the interval—

[-0.5 N-0.5).

Such a ruler is an unnormalized ruler, and can be normalized by replacing the integer by the interval it represents.

PARTIAL RULERS: If any one of the five parameters of a ruler is omitted, it can be computed from the others. Rulers are permitted with *nil* as one of the parameter values. These are unnormalized rulers that may be normalized by replacing the *nil* with a computed value.

An exception is made if f_scale is *nil* and f_domain-bound-1 == f_domain-bound-2, in which case the ruler cannot be **normalized**, and is considered invalid.

It is also possible to specify a domain or range by a *nil* list, which will be replaced by (-0.5 *nil*). This is similar to having an integer specification of the domain or range, and not knowing the value of the integer.

SCALE FACTORS: If a ruler has a list (n_factor [l_pattern]) in place of f_scale, then the ruler is unnormalized. To normalize the ruler, it is treated as a partial ruler with missing scale, the scale is computed, and the function call—

(*round-ruler* the-ruler n_factor [l_pattern])

is called to compute a ruler which is then normalized and returned.

sar_array	[C Type]
ar_	[Argument Prefix]
SAR_ARRAY	[C Global Constant]
ar_array->**sar_type**	[C Macro]
ar_array->**sar_etype**	[C Macro]
ar_array->**sar_esize**	[C Macro]
ar_array->**sar_exponent**	[C Macro]
ar_array->**sar_ubbase**	[C Macro]
ar_array->**sar_cbase**	[C Macro]
ar_array->**sar_ucbase**	[C Macro]
ar_array->**sar_sbase**	[C Macro]
ar_array->**sar_usbase**	[C Macro]
ar_array->**sar_lbase**	[C Macro]
ar_array->**sar_ulbase**	[C Macro]
ar_array->**sar_fbase**	[C Macro]
ar_array->**sar_dbase**	[C Macro]
ar_array->**sar_ubase**	[C Macro]
ar_array->**sar_ibase**	[C Macro]
ar_array->**sar_edimensions**	[C Macro]
ar_array->**sar_cdimensions**	[C Macro]
ar_array->**sar_dimensions** + n	[C Macro]
ar_array->**sar_xsize**	[C Macro]
ar_array->**sar_xincrement**	[C Macro]
ar_array->**sar_ysize**	[C Macro]
ar_array->**sar_yincrement**	[C Macro]
ar_array->**sar_zsize**	[C Macro]
ar_array->**sar_zincrement**	[C Macro]
ar_array->**sar_tsize**	[C Macro]
ar_array->**sar_tincrement**	[C Macro]
ar_array->**sar_usize**	[C Macro]
ar_array->**sar_uincrement**	[C Macro]
ar_array->**sar_vsize**	[C Macro]
ar_array->**sar_vincrement**	[C Macro]

USE: An *sar_array* value is a pointer at *an-array* object.

SAR_ARRAY equals the LISP value *an-array*, and is the value of the ***sar_type*** element of *an-array* object.

In the description that follows we assume the elements of the array are allocated, as this is nearly always the case with C code. See UNALLOCATED ARRAYS below.

ARGUMENT PREFIX: The *ar_* prefix denotes C arguments of *sar_array* type.

SAR_TYPE: Ar_array->*sar_type* equals *SAR_ARRAY*, and is an *sob_type* value. It may be used to verify that ar_array points at an array descriptor.

SAR_ETYPE: Ar_array->*sar_etype* is the *has-element-type* attribute of ar_array. It is an *sob_type* value.

SAR_ESIZE: Ar_array−>*sar_esize* is size in bits of an element of ar_array. It is an *int* value and equals *sob_tsize* (ar_array−>*sar_etype*), which is the same as the *has-size* attribute of the *has-element-type* of the array. Redundant, but useful for speed.

SAR_EXPONENT: Ar_array−>*sar_exponent* is the *has-exponent* attribute of ar_array. It is an *int* value used for a block floating point array. It should equal 0 for other kinds of arrays.

SAR_UBBASE:

SAR_CBASE:

SAR_UCBASE:

SAR_SBASE:

SAR_USBASE:

SAR_LBASE:

SAR_ULBASE:

SAR_FBASE:

SAR_DBASE:

SAR_UBASE:

SAR_IBASE: Ar_array−>*sar_..base* is the base address of the array elements: that is, the address of the element with subscripts (0 0 0 0 0 0). The data type of this address is given according to the following table—

sar_ubbase	*unsigned*
sar_cbase	*char* *
sar_ucbase	*uchar* *
sar_sbase	*short* *
sar_usbase	*ushort* *
sar_lbase	*long* *
sar_ulbase	*ulong* *
sar_ibase	*int* *
sar_ubase	*unsigned* *
sar_fbase	*float* *
sar_dbase	*double* *

The various different versions of this structure element are used for the different types of array element: e.g. *sar_lbase* is used if the array elements are *long*'s.

The *sar_ubbase* value is exceptional in that it is a bit address for arrays of unsigned bit elements (all other addresses are byte addresses). When right shifted by 3 it addresses a byte. Its low order three bits address a bit within the byte, with the high order bit being at address 0 and the low order bit at address 7. This addressing system is compatible with most raster input/output devices.

The *sar_cbase* value may be cast to any normal C pointer type in order to get a pointer at the elements of the array. The only exception is for arrays of unsigned bits, as noted above.

The convention is that bit addressing (*sar_ubbase*) is used if—

(ar_array−>*sar_esize* & 07) != 0

and byte addressing (*sar_cbase*, ...) is used otherwise. I.e., byte addressing is used if the element size is a multiple of one byte.

SAR_EDIMENSIONS: Ar_array−>*sar_edimensions* is the number of dimensions of ar_array with 0 size after clipping. If non-zero, it indicates that the array is empty. Note, it is not the same thing as the number of 0 dimension *sar_size* values, as all the later are zeroed if any of them are zero.

SAR_CDIMENSIONS: Ar_array−>*sar_cdimensions* is the number of dimensions of ar_array which have been clipped, in the sense that their desired size does not equal their actual size.

SAR_DIMENSIONS: Ar_array−>*sar_dimensions* + n is a pointer to the array dimension descriptor for the n'th dimension of the array. The standard values of n are *SAR_X*, *SAR_Y*, *SAR_Z*, *SAR_T*, *SAR_U*, and *SAR_V*.

SAR_XSIZE SAR_XINCREMENT:

SAR_YSIZE SAR_YINCREMENT:

SAR_ZSIZE SAR_ZINCREMENT:

SAR_TSIZE SAR_TINCREMENT:

SAR_USIZE SAR_UINCREMENT:

SAR_VSIZE SAR_VINCREMENT: Ar_array−>*sar_xsize* is equivalent to—

(ar_array−>*sar_dimensions* + *SAR_X*) −> *sar_size*

and ar_array−>*sar_xincrement* is equivalent to—

(ar_array−>*sar_dimensions* + *SAR_X*) −> *sar_increment*.

Similarly for the other dimensions.

UNALLOCATED ARRAYS: Arrays whose elements are not allocated appear to be very much like allocated arrays. However, their *sar_...base* value is set to point to a region of virtual memory that is unimplemented, in order to catch reference errors. Also, their *sar_esize* element may be 0 (to allow for cataloged arrays whose element types are unknown to the current program and which will never be allocated by the current program).

sar_dimension [C Type]
adim_ [Argument Prefix]
adim_dimension->**sar_size** [C Macro]
adim_dimension->**sar_increment** [C Macro]

USE: An *sar_dimension* value is a pointer at an array dimension description structure. There is one such structure inside *an-array* object for each of the array's *SAR_MDIMENSIONS* dimensions.

ARGUMENT PREFIX: The *adim_* prefix denotes C arguments of *sar_dimension* type.

SAR_SIZE: Adim_dimension->*sar_size* is the actual size of the array dimension (as gotten by the *has-sizes* attribute of *an-array* object). It is an *int* value.

The dimension's subscripts are allowed to range from 0 through *sar_size*-1. This size equals the desired size if the dimension is not clipped, but will be less than the desired size if the dimension is clipped.

If the array is empty, this value will be zero, even if this dimension is not empty. In other words, if any dimension's *sar_size* is zero, all the other dimensions' *sar_size*'s will be set to zero.

SAR_INCREMENT: Adim_dimension->*sar_increment* is number of array elements that must be skipped in memory for each +1 change in the dimension's subscript. Note that this number, which is an *int*, is in units of array elements, and not bytes or bits. May be positive, negative, or zero.

It is the same increment as gotten by the *has-increments* attribute of *an-array* object.

SAR_MDIMENSIONS [C Macro Constant]

VALUE: The maximum number of dimensions allowed in an array. Usual value is 6.

SAR_MSIZE [C Macro Constant]

VALUE: The maximum *sar_size* of an individual dimension. Not commonly used, and should not be used to unnecessarily restrict software. Sometimes, however, it is convenient to allocate one dimension's worth of work area in the stack, and this constant is for such purposes. The usual value is 8192.

sar_place (ar_array, x_dimension, x_size, x_origin, x_step) [C Function]

SIDE EFFECT: Sets the desired size, desired origin, and step of the given dimension (*SAR_X*, *SAR_Y*, ...) of ar_array. Errors are indicated by a call to *sfe_error* followed by a return. The error switch should be checked for by the caller.

sar_similar (ar_array1, ar_array2) [C Macro]
sar_xsimilar (ar_array1, ar_array2, x_exclude) [C Function]

RETURNS: *SAT_YES()* if ar_array1 and ar_array2 are both *sar_array* objects (they have the right *sar_type* element value) and these two arrays have identical dimension *sar_size*'s for non-excluded dimensions. Otherwise returns *SAT_NO*().

Sar_similar excludes no dimensions, while *sar_xsimilar* excludes those dimensions D (D = 0, 1, ...) for which the bit 1<<D is on in x_exclude.

sar_write (ar_array) [C Macro]

RETURNS: *SAT_YES ()* if ar_array is writable (*is-readonly* attribute is *nil*), and *SAT_NO ()* if ar_array is readonly.

SIDE EFFECT: Sets the *has-been-changed* attribute of ar_array to *t* if the array is writable. This is very important in that it permits the array disk cacheing to work correctly.

NOTE: This macro should be used on every array that is to be written by a C function. The result returned by this macro should be checked by *sfe_assert*.

SAR_X [C Macro Constant]
SAR_Y [C Macro Constant]
SAR_Z [C Macro Constant]
SAR_T [C Macro Constant]
SAR_U [C Macro Constant]
SAR_V [C Macro Constant]

VALUE: The values 0 (*SAR_X*) through 5 (*SAR_V*) denoting the first 6 dimensions of an array. Arrays with fewer than 6 dimensions are treated as 6 dimensional arrays whose later dimensions (closer to *SAR_V*) have size 1.

{**sar_xfor_elements** (ar_array, type, base) { [C Macro]
... }}
{**sar_xfor_2_elements** (ar_array1, type1, base1,
ar_array2, type2, base2) { [C Macro]
... }}
{**sar_xfor_3_elements** (ar_array1, type1, base1,
ar_array2, type2, base2,
ar_array3, type3, base3) { [C Macro]
... }}
{**sar_xfor_4_elements** (ar_array1, type1, base1,
ar_array2, type2, base2,
ar_array3, type3, base3,
ar_array4, type4, base4) { [C Macro]
... }}
{**sar_for_elements** (ar_array, type) { [C Macro]
... }}
{**sar_for_2_elements** (ar_array1, ar_array2, type) { [C Macro]
... }}
{**sar_for_3_elements** (ar_array1, ar_array2, ar_array3,
type) { [C Macro]
... }}
{**sar_for_4_elements** (ar_array1, ar_array2, ar_array3,
ar_array4, type) { [C Macro]
... }}
X [C Local Variable]
Y [C Local Variable]
Z [C Local Variable]
T [C Local Variable]
U [C Local Variable]
V [C Local Variable]
xp [C Local Variable]

SIDE EFFECT: *Sar_xfor_elements* creates a loop which executes the body ... once for each element of ar_array. This macro is equivalent to—

```
register type xp; register int X;
register type yp; register int Y;
register type zp; register int Z;
register type tp; register int T;
register type up; register int U;
register type vp; register int V;
if (ar_array->sar_edimensions == 0)
for (V = 0, vp = (type) (base);
        V < (ar_array)->sar_vsize;
        ++ V, vp += (ar_array)->sar_vincrement)
for (U = 0, up = vp;
        U < (ar_array)->sar_usize;
        ++ U, up += (ar_array)->sar_uincrement)
for (T = 0, tp = up;
        T < (ar_array)->sar_tsize;
        ++ T, tp += (ar_array)->sar_tincrement)
for (Z = 0, zp = tp;
        Z < (ar_array)->sar_zsize;
        ++ Z, zp += (ar_array)->sar_zincrement)
for (Y = 0, yp = zp;
        Y < (ar_array)->sar_ysize;
        ++ Y, yp += (ar_array)->sar_yincrement)
for (X = 0, xp = yp;
        X < (ar_array)->sar_xsize;
        ++ X, xp += (ar_array)->sar_xincrement)
```

Notice that the macro begins with declarations, and that more declarations may immediately precede the macro. The pointer *xp* tracks through the elements of ar_array, and may be use in the array body to access the current element. The type of this pointer is the type argument to the macro. The address of the (0 0 0 0 0 0) element is the base argument to the macro. The variables *X*, *Y*, *Z*, *T*, *U*, and *V* are the subscripts of the current element.

The macros *sar_xfor_2_elements*, *sar_xfor_3_elements* and *sar_xfor_4_elements* are similar except that there are a separate set of pointers *x1p*, *x2p*, *x3p* and *x4p* for the four arrays ar_array1, ar_array2, ar_array3 and ar_array4. Thus on any iteration of the inner loop corresponding elements of two, three, or four arrays are being pointed at. Each of these arrays has its own base element address and element pointer type. However, the subscripts are restricted by the sizes of ar_array1, which should be the smallest of the arrays. There are variables *y1p*, *y2p*, *y3p*, *y4p*, *z1p*, ..., *v4p* for the other dimensions.

Sar_for_elements (ar_array, type) is an abbreviation for—

> *sar_xfor_elements* (ar_array, type, (ar_array)->*sar_cbase*).

Sar_for_2_elements, *sar_for_3_elements* and *sar_for_4_elements* are similar abbreviations where all arrays have the same element types and *sar_cbase* may be used for the base address in each case.

The x in *sar_xfor_elements* stands for "extended". The forms without the x are more commonly used.

EXAMPLE:

```
/* Add input1 to input2 and store in output */
{ sar_for_3_elements (output, input1, input2, long *) {
        * x1p = * x2p + * x3p; }}
```

{**sar_xfor_matrices** (ar_array, type, base) { [C Macro]
... }}

{**sar_xfor_2_matrices** (ar_array1, type1, base1, [C Macro]
ar_array2, type2, base2) {
... }}

{**sar_xfor_3_matrices** (ar_array1, type1, base1, [C Macro]
ar_array2, type2, base2,
ar_array3, type3, base3) {
... }}

{**sar_xfor_4_matrices** (ar_array1, type1, base1, [C Macro]
ar_array2, type2, base2,
ar_array3, type3, base3,
ar_array4, type4, base4) {
... }}

{**sar_for_matrices** (ar_array, type) { [C Macro]
... }}

{**sar_for_2_matrices** (ar_array1, ar_array2, type) { [C Macro]
... }}

{**sar_for_3_matrices** (ar_array1, ar_array2, ar_array3, [C Macro]
type) {
... }}

{**sar_for_4_matrices** (ar_array1, ar_array2, ar_array3, [C Macro]
ar_array4, type) {
... }}

SIDE EFFECT: *Sar_xfor_matrices* creates a loop which executes the body ... once for each 2 dimensional matrix of ar_array. This macro is equivalent to—

```
type zp; int Z;
type tp; int T;
type up; int U;
type vp; int V;
if (ar_array->sar_edimensions == 0)
for (V = 0, vp = (type) (base);
        V < (ar_array)->sar_vsize;
        ++ V, vp += (ar_array)->sar_vincrement)
for (U = 0, up = vp;
        U < (ar_array)->sar_usize;
        ++ U, up += (ar_array)->sar_uincrement)
for (T = 0, tp = up;
        T < (ar_array)->sar_tsize;
        ++ T, tp += (ar_array)->sar_tincrement)
for (Z = 0, zp = tp;
        Z < (ar_array)->sar_zsize;
        ++ Z, zp += (ar_array)->sar_zincrement)
```

Notice that the macro begins with declarations, and that more declarations may immediately precede the macro. The pointer *zp* tracks through the matrices of ar_array, and may be use in the array body as the address of the (0 0) element of the current matrix. The type of this pointer is the type argument to the macro. The address of the (0 0 0 0 0 0) element of ar_array is the base argument to the macro. The variables *Z*, *T*, *U*, and *V* are the subscripts of the current matrix.

The macros *sar_xfor_2_matrices*, *sar_xfor_3_matrices*, and *sar_xfor_4_matrices* are similar except that there are a separate set of pointers *z1p*, *z2p*, *z3p* and *z4p* for the three arrays ar_array1, ar_array2, ar_array3 and ar_array4. Thus on any iteration of the inner loop corresponding matrices of two, three or four arrays are being pointed at. Each of these arrays has its own base element address and element pointer type. However, the subscripts are restricted by the sizes of ar_array1, which should be the smallest of the arrays. There are variables *t1p*, *t2p*, *t3p*, *t4p*, *u1p*, ..., *v4p* for the other dimensions.

Sar_for_matrices (ar_array, type) is an abbreviation for—

sar_xfor_matrices (ar_array, type, (ar_array)->*sar_cbase*).

Sar_for_2_matrices, *sar_for_3_matrices*, and *sar_for_4_matrices* are similar abbreviations where all arrays have the same element types and *sar_cbase* may be used for the base address in each case.

The x in *sar_xfor_matrices* stands for "extended". The forms without the x are more commonly used.

{**sar_xfor_matrix_elements** (ar_array, type, base) { [C Macro]
... }}
{**sar_xfor_2_matrix_elements** (ar_array1, type1, base1, [C Macro]
ar_array2, type2, base2) {
... }}
{**sar_xfor_3_matrix_elements** (ar_array1, type1, base1, [C Macro]
ar_array2, type2, base2,
ar_array3, type3, base3) {
... }}
{**sar_xfor_4_matrix_elements** (ar_array1, type1, base1, [C Macro]
ar_array2, type2, base2,
ar_array3, type3, base3,
ar_array4, type4, base4) {
... }}
{**sar_for_matrix_elements** (ar_array, type) { [C Macro]
... }}
{**sar_for_2_matrix_elements** (ar_array1, ar_array2, type) { [C Macro]
... }}
{**sar_for_3_matrix_elements** (ar_array1, ar_array2, ar_array3 [C Macro]
type) {
... }}
{**sar_for_4_matrix_elements** (ar_array1, ar_array2, ar_array3, [C Macro]
ar_array4, type) {
... }}

SIDE EFFECT: *Sar_xfor_matrix_elements* creates a loop which executes the body ... once for each element of a 2 dimensional matrix in ar_array. This macro is equivalent to—

```
register type xp; register int X;
register type yp; register int Y;
if (ar_array->sar_edimensions == 0)
for (Y = 0, yp = (type) (base);
            Y < (ar_array)->sar_ysize;
            ++ Y, yp += (ar_array)->sar_yincrement)
for (X = 0, xp = yp;
            X < (ar_array)->sar_xsize;
            ++ X, xp += (ar_array)->sar_xincrement)
```

Notice that the macro begins with declarations, and that more declarations may immediately precede the macro. The pointer *xp* tracks through the elements of a matrix of ar_array, and may be use in the array body to access the current element. The type of this pointer is the type argument to the macro. The address of the (0 0) matrix element is the base argument to the macro. The variables X and Y are the subscripts of the current element.

The macros *sar_xfor_2_matrix_elements*, *sar_xfor_3_matrix_elements* and *sar_xfor_4_matrix_elements* are similar except that there are a separate set of pointers *x1p*, *x2p*, *x3p* and *x4p* for the three arrays ar_array1, ar_array2, ar_array3 and ar_array4. Thus on any iteration of the inner loop corresponding elements of two, three or four matrices are being pointed at. Each of these matrices has its own base element address and element pointer type. However, the subscripts are restricted by the sizes of the X and Y dimensions of ar_array1, which should be the smallest of the matrices. There are variables *y1p*, *y2p*, *y3p*, *y4p* for the Y dimension.

Sar_for_matrix_elements (ar_array, type) is an abbreviation for—

sar_xfor_matrix_elements (ar_array, type, zp).

and is intended to be used inside *sar_for_matrices*. *Sar_for_2_matrix_elements*, *sar_for_3_matrix_elements* and *sar_for_4_matrix_elements* are similar abbreviations where all arrays have the same element types and *z1p*, *z2p*, *z3p* and *z4p* are used as the base addresses.

The x in *sar_xfor_matrix_elements* stands for "extended". The forms without the x are more commonly used.

(set-array-by-expression 'ar_array 'g_expression) [LISP Function]

WHERE: The element type of ar_array must be numeric.

RETURNS: Ar_array after its elements have been set.

SIDE EFFECT: Sets all the elements of ar_array using g_expression to compute a value for each element. G_expression is an expression to be evaluated for each element. In that expression, the variables *X*, *Y*, *Z*, *T*, *U*, and *V* evaluate to the subscripts of the element. There should be no other variables in the expression (variables may be substituted for in the expression when the expression is created by using ‘ and ,).

(set-array-by-value 'ar_array ' g_value) [LISP Function]

RETURNS: Ar_array after setting its elements.

SIDE EFFECT: Sets the elements of ar_array using values taken from g_value. G_value is a list of sublists of sublists ... of element values. The innermost lists correspond to the first (X) dimension: e.g., if x is an array with sizes (2 3) then—

(*set-array-by-value* x '((00 01) (10 11) (20 21)))

and—

(*set-array-by-expression*
    x '(*plus* *X* (*product* 10 *Y*)))

set the elements to the same values.

NOTE: If elements or sublists are omitted from the end of a list, *nil* values will be assumed.

(slice-of-array ar_array) [LISP Function]
(slice-of-array ar_array '(x_size ...) ['(x_origin ...) ['(x_step ...)]]) [LISP Function]
(slice-of-array ar_array x_dimension x_size [x_origin [x_step]]) [LISP Function]

RETURNS: If only the ar_array argument is given, a new array whose parent is identical to the current slice of ar_array, and whose current slice is identical to its parent. The elements of ar_array are first allocated, if this has not already been done.

If other arguments are given, (*slice-of-array* ar_array ...) is equivalent to (*place-array* (*slice-of-array* ar_array) ...).

NOTE: The new array shares the elements of ar_array, so that changes to these ar_array elements will change the corresponding new array elements and vice versa.

NOTE: The *has-been-changed* and *is-readonly* attributes of the new array are set to the corresponding attributes of the old array. The *is-immovable* attribute is set to *nil*.

(summary-of-array 'ar_array) [LISP Function]

WHERE: The elements of ar_array must be numeric.

RETURNS: *A-summary* structure summarizing the array. All the usual *an-array-summary* structure attributes are included.

(sweep-array-blocks) [LISP Function]
sweep-array-blocks-count [LISP Global Variable]
sweep-array-blocks-time [LISP Global Variable]
sweep-array-blocks-bytes [LISP Global Variable]

USE ONLY WHEN: Playing with the array block garbage collector (which is normally automatic).

SIDE EFFECT: Marks as free all unused blocks in the array block memory allocation area so they may be reused. Does *not* compact the block allocation area: call *compact-array-blocks* subsequently to do that.

sweep-array-blocks-count is incremented every time *sweep-array-blocks* is called, **sweep-array-blocks-time** has the time taken by the call added to it, and **sweep-array-blocks-bytes** has the number of bytes recovered by the sweep added to it. All these variables are initialized to 0. The time is measured in the same units as *ptime*: see **ptime-counts-per-second**.

RETURNS: The sum of the number of fixnum's in all the free blocks.

(transpose-array 'ar_array 'x_dimension-1 'x_dimension-2) [LISP Function]

RETURNS: Ar_array after it is modified.

SIDE EFFECT: Transposes (exchanges) the two specified dimensions within *both* the current ar_array slice and its parent. The actual data elements are not moved: rather all the dimension parameters in the array object are exchanged.

(uneval-object 'ar_array [*t*]) [LISP Function]
default-array-file [LISP Global Variable]
cache.ar [UNIX File Name]

EQUIVALENT TO: The usual *uneval-object*, except that if the array has a *t has-been-changed* attribute, then—

(*write-array-elements* ar_array **default-array-file**)

is called. Also ar_array's *has-parent-sizes*, *has-parent-increments*, *has-desired-sizes*, *has-desired-origins*, *has-steps*, *has-parent-offsets*, *has-parent-scales*, and *has-parities* values are output with some editing. The ends of the list values of these attributes are truncated in the same manner as the *do-shorten* option truncates the lists when they are gotten. Also, *has-parent-sizes* is renamed *has-sizes*, *has-parent-increments* is renamed *has-increments*, *has-parent-offsets* is renamed *has-offsets*, and *has-parent-scales* is renamed *has-scales*.

NOTE: **default-array-file** is a global variable whose default value is—

(*cache.ar end-of-file*).

(write-array-elements 'ar_array 'g_array-file) [LISP Function]

SIDE EFFECT: Writes the parent of ar_array to the file location specified by g_array-file. Sets the *has-array-file* attribute of ar_array to g_array-file, and the *has-array-format* attribute to the value of **computer-format**. Sets the *has-been-changed* attribute to *nil*.

See HAS-ARRAY-FILE under *an-array* for the format of g_array-file.

NOTE: G_array-file may have one of the forms—

(s_file-name *end-of-file*)
(s_volume-name *end-of-volume*)
(s_volume-name x_file-number *end-of-file*).

In these cases the last element of the g_array-file list will be changed to an appropriate number before it is stored in the *has-array-file* attribute.

(**write-catalog** 'ca_catalog 'ar_array) [LISP Function]

SEE: *Uneval-object* (in this chapter), which is called to produce the exact value to be written into the catalog.

X-dimension [LISP Global Constant]
Y-dimension [LISP Global Constant]
Z-dimension [LISP Global Constant]
T-dimension [LISP Global Constant]
U-dimension [LISP Global Constant]
V-dimension [LISP Global Constant]

VALUE: The integers 0 (*X-dimension*) through 5 (*V-dimension*) identifying the different dimensions of arrays.

NOTE: The X dimension is first when listing subscripts, dimension sizes, etc., and the V dimension is last.

NOTE: The standard storage organization for arrays is a contiguous list of elements with the X subscript varying fastest and the V subscript slowest. Other organizations may be obtained by explicitly specifying the *has-parent-increments* attribute.

CHAPTER 8

BASIC ARITHMETIC

1. GLOSSARY.

(**absolute-value-array-elements** 'lar_output ['lar_input]) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the absolute value of the corresponding element in lar_input. The two arrays may have different exponents.

(**accumulate-filter** 'lar_array 'x_dimension) [LISP Function]

RETURNS: Lar_array is returned after the lar_array elements are modified to hold the desired result.

SIDE EFFECT: Applies a filter that computes the sum of all values with equal or lower subscripts for the given dimension of the given lar_array. Thus the output for subscript j in the given dimension is the sum of the input for subscripts 0, 1, ..., j.

BUGS: Overflow is handled by doing modulo arithmetic.

(**add-arrays** 'lar_output 'lar_input-1 ['lar_input-2]) [LISP Function]

WHERE: The arrays are similar and have the same exponent, and lar_input-2 defaults to lar_output.

RETURNS: Lar_output after its elements have been modified.

SIDE EFFECT: Adds each element of lar_input-1 to the corresponding element of lar_input-2 and stores the result in the corresponding element of lar_output.

BUG: The addition is done using modulo arithmetic in event of overflow.

(add-to-array-elements 'lar_array 'n_addend) [LISP Function]

RETURNS: Lar_array after its elements have been modified.

SIDE EFFECT: Adds n_addend to all elements of lar_array.

BUG: The addition is done using modulo arithmetic in the event of overflow.

(arccos-array-elements 'lar_output ['lar_input]) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the arc cosine of the corresponding element in lar_input. The two arrays may have different exponents.

(arcsin-array-elements 'lar_output ['lar_input]) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the arc sine of the corresponding element in lar_input. The two arrays may have different exponents.

(contrast-of 'ar_input '(x_width ...)
['n_background ['n_center]]) [LISP Function]

WHERE: The x_width are non-negative integers and both n_background and n_center default to 1.0.

RETURNS: An output array, lar_output, whose elements are the "convolution" in the sense of the *convolve* function of ar_input and a kernel with a special form which is parametrized by (x_width ...), n_background, and n_center. The kernel is of size-

(2 * x_width + 1, ...)

and has for all its elements except the center element the value-

- n_background / ((2 * x_width + 1) * ...).

where the denominator is the area of the kernel. The center element has the value-

n_center - n_background / ((2 * x_width + 1) * ...).

The effect is to output into each lar_output element (X, ...) n_center times the ar_input element (X+x_width, ...) minus n_background times the average of the ar_input elements in a (2*x_width+1 ...) rectangular box centered on the ar_input element just indicated.

BUGS: Missing values are not handled. Overflow is handled by doing modulo arithmetic.

(convolution-of 'ar_input 'ar_kernel) [LISP Function]

RETURNS: A new 2 dimensional array lar_output with exponent the same as ar_input (after its elements are converted to *a-long*'s) whose elements are computed by passing it and the other parameters to the *convolution* function. *Prepare-array* is applied to ar_input.

(convolve 'lar_output 'lar_input 'ar_kernel) [LISP Function]

WHERE: The arrays are treated as 2 dimensional, and the sizes of the lar_output dimensions must be one more than sizes of the corresponding lar_input dimensions minus the sizes of the corresponding ar_kernel dimensions.

RETURNS: Lar_output after its elements are set.

SIDE EFFECT: Stores the "convolution" of lar_input and ar_kernel in lar_output. To be more precise, what is stored is the convolution of lar_input and the matrix whose (X, Y)'th element is the (-X, -Y)'th element of ar_kernel; or in other words, the (X, Y)'th element of lar_output is the *scalar-product* of the ar_kernel and a slice of lar_input with origins (X, Y).

(cos-array-elements 'lar_output ['lar_input']) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the cosine of the corresponding element in lar_input. The two arrays may have different exponents.

(del2g-kernel '(n_xwidth n_ywidth)
['(n_xoffset n_yoffset)']) [LISP Function]

WHERE: N_xoffset and n_yoffset are ≥ 0 and default to 0.5.

RETURNS: A block floating point array with exponent -24 representing the kernel computed as the minus of the Laplacian operator applied to the Gaussian function. The Laplacian operator is assumed to be scaled by n_xwidth and n_ywidth, so actually the second derivative with respect to x is multiplied by n_xwidth ** 2, and the second derivative with respect to y by n_ywidth ** 2. The sizes of the X and Y dimensions are

2 * xsize + (*ceiling* n_xoffset) + 1

and

2 * ysize + (*ceiling* n_yoffset) + 1

where xsize and ysize are choosen as indicated below. The value of the point with subscripts (X Y) is-

```
  (1 / (pi * n_xwidth * n_ywidth))
* (1 - (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2)
* (exp (1 - (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2))
```

where

n_X = X - xsize - n_xoffset

and

n_Y = Y - ysize - n_yoffset.

This function is positive inside the ellipse

(n_X / n_xwidth) ** 2 + (n_Y / n_ywidth) ** 2 $\leq$ 1,

zero on that ellipse, and negative outside the ellipse. The total integral of the function is 0. The function is normalized to have the integral +1 inside the ellipse and the integral -1 outside the ellipse, where for these purposes the function is assumed to be continuous and extend to infinity.

Xsize and ysize are chosen to be large enough so that the integral of the continuous function over all points (X Y) outside the kernel returned by *del2g-kernel* is less than the value of the global variable **kernel-cutoff**. In computing these sizes, n_xoffset and n_yoffset are assumed to be 0, as a worst case.

(derivative-filter 'lar_array 'x_dimension 'x_width) [LISP Function]

RETURNS: A slice of lar_array is returned after the lar_array elements are modified so that the slice holds the desired result. The size of the given dimension for the slice is x_width-1 less than the size of that dimension for lar_array, and the slice origin is 0 for that dimension. Other dimensions are not affected.

SIDE EFFECT: Applies a derivative filter of the given x_width for the given x_dimension of the given lar_array. The derivative filter forms the sum of the terms -

(6 / (x_width ** 3 - x_width)) * (- x_width + 1 + 2 * i) * x (i)

for i = 0, 1, ..., x_width - 1. The normalization constant is chosen so that if x(i) = i the result will be 1. The output for subscript j in the given dimension is computed by letting x(i) equal the input for subscript j+i.

NOTE: The lar_array elements that are not in the returned slice are modified in undefined ways.

BUGS: Missing values are not handled. Overflow is handled by doing modulo arithmetic.

(dither 'x_size) [LISP Function]
(cached-dither 'x_size)
default-dither-size

WHERE: X_size is a power of two.

RETURNS: *Dither* and *cached-dither* return the Dither Matrix of the given size: the matrices D^n of the paper Jarvis, J.F., Judice, C.N., and Ninke, W.H., *A Survey of Tehniques for the Display of Continuous Tone Pictures on Bilevel Displays*, Computer Graphics and Image Processing, 5, 13-40 (1976), where n is the matrix size, x_size. The matrix is square.

Cached-dither remembers all dither matrices it has computed, saving them. It returns previously computed matrices without recomputing them. Only those of size <= 64 are saved at the moment.

default-dither-size is a global variable set to a default value suitable for the size parameter. It itself defaults to 8.

(dxg-kernel '(n_xwidth n_ywidth) [LISP Function]
['(n_xoffset n_yoffset)])

WHERE: N_xoffset and n_yoffset are ≥ 0 and default to 0.5.

RETURNS: A block floating point array with exponent -24 representing the kernel computed as the minus of the x partial derivative applied to the Gaussian function. The sizes of the X and Y dimensions are

2 * xsize + (*ceiling* n_xoffset) + 1

and

2 * ysize + (*ceiling* n_yoffset) + 1

where xsize and ysize are choosen as indicated below. The value of the point with subscripts (X Y) is-

(2 / ((*sqrt* pi) * n_xwidth * n_ywidth))
* (n_X / n_xwidth)
* (*exp* (- (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2))

where

n_X = X - xsize - n_xoffset

and

n_Y = Y - ysize - n_yoffset.

This function is positive for X > 0 and negative for X < 0.

The total integral of the function is 0. The function is normalized to have the integral +1 on the halfplane X > 0 and the integral -1 on the halfplane X < 0, where for these purposes the function is assumed to be continuous and extend to infinity.

Xsize and ysize are chosen to be large enough so that the integral of the continuous function over all points (X Y) outside the kernel returned by *dxg-kernel* is less than the value of the global variable ***kernel-cutoff***. In computing these sizes, n_xoffset and n_yoffset are assumed to be 0, as a worst case.

(**expand-missing** 'lar_output 'ar_input 'ar_original [LISP Function]
['(x_xsize x_ysize) ['x_count]])

WHERE: Lar_output, ar_input, and ar_original all have the same exponent and the same dimension sizes except for the X and Y dimensions. Lar_output and ar_original have the same X and Y dimensions, while the X and Y dimensions of ar_input are respectively 2*x_xsize and 2*x_ysize larger than the X and Y dimensions of lar_output.

X_xsize, X_ysize, and x_count all default to 1.

RETURNS: The number of non-missing elements of ar_input replaced by missing values in lar_output.

SIDE EFFECT: Copies the elements of ar_input to lar_output, replacing some of the non-missing values by missing values. The purpose of this is to expand a sky region (region of all missing values in a laser radar image) which has been shrunk by *shrink-missing*. Ar_original is the original array before it was shrunk by *shrink-missing*.

An element is replaced by a missing value when it is copied if (1) the element is not missing in ar_input, (2) the element is missing in ar_original, and (3) there are x_count or more missing values in the box in ar_input of size

(2*x_xsize+1 2*x_ysize+1)

centered on the element, not counting the element itself.

(**expand-missing-of** 'ar_input ar_original '(x_xsize x_ysize) ['x_count ['x_repeat ...]]]) [LISP Function]

WHERE: Ar_input and ar_original should have the same dimension sizes. X_repeat defaults to infinity, and may also be given as *nil* to specify infinity.

The part of the argument list beginning with '(x_xsize x_ysize) may be repeated as long as any repetition begins with a non-empty list value.

RETURNS: A new array lar_output which is computed by passing ar_input and the other parameters through *expand-missing* x_repeat times. As an optimization, the process stops when no more replacement is possible, so x_repeat can be a very large number.

If no replacement is done in computing lar_output, ar_input is returned in place of lar_output.

Prepare-array is applied to ar_input.

If more than one set of size/count/repeat parameters is given, these parameters are removed from the parameter list as they are used, and the process is repeated with the last lar_output substituted for ar_input.

(**exponentiate-array-elements** 'lar_output ['lar_input]) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the exponential function of the corresponding element in lar_input. The two arrays may have different exponents.

(**gaussian-kernel** '(n_xwidth n_ywidth) ['(n_xoffset n_yoffset)]) [LISP Function]

WHERE: N_xoffset and n_yoffset are ≥ 0 and default to 0.5.

RETURNS: A block floating point array with exponent -24 representing the kernel computed by the Gaussian function. The sizes of the X and Y dimensions are

2 * xsize + (*ceiling* n_xoffset) + 1

and

2 * ysize + (*ceiling* n_yoffset) + 1

where xsize and ysize are choosen as indicated below. The value of the point with subscripts (X Y) is-

```
  (1 / (pi * n_xwidth * n_ywidth))
* (exp (- (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2))
```

where

n_X = X - xsize - n_xoffset

and

n_Y = Y - ysize - n_yoffset

Xsize and ysize are chosen to be large enough so that the integral of the continuous function over all points (X Y) outside the kernel returned by *gaussian-kernel* is less than the value of the global variable ***kernel-cutoff***. In computing these sizes, n_xoffset and n_yoffset are assumed to be 0, as a worst case.

The normalization constant is chosen so that the integral of the kernel would be 1 if it where a continuous function extending to infinity.

(interpolation-filter 'lar_array 'x_dimension 'n_factor 'n_offset) [LISP Function]

RETURNS: A slice of lar_array is returned after the lar_array elements are modified so that the slice holds the desired result. The size of the given dimension for the slice is as large as possible subject to the conditions that for all slice subscripts j

- epsilon ≤ n_offset + j * n_factor
≤ x_input-dimension-size - 1 + epsilon.

where epsilon = 2^{-10} is included to compensate for rounding errors. The slice origin is 0 for that dimension. Other dimensions are not affected.

SIDE EFFECT: Interpolates the input values for the given dimension so that the output has the given size. Linear interpolition is used. Output subscript 0 has the value associated with input subscript n_offset, and in general output subscript j has the value associated with input subscript

n_offset + j * n_factor

Non-integer input subscripts given by this formula are handled by linearly interpolating input elements with the next lower and higher integer subscripts. Input subscripts less than 0 by an amount less than or equal to epsilon are treated as 0, and similarly input subscripts larger than the maximum input subscript by an amount less than or equal to epsilon are treated as the maximum input subscript.

NOTE: The lar_array elements that are not in the returned slice are modified in undefined ways.

(interpolation-of 'ar_input '(x_size ...)) [LISP Function]

WHERE: The x_size are non-negative.

RETURNS: An output array, lar_output, whose elements are the linear interpolation, in the sense of the *interpolation-filter* function, of the elements of ar_input. The dimension sizes of lar_output are (x_size ...). The expansion factors for *interpolation-filter* are automatically chosen to be positive and to give the right lar_output sizes (and the offsets are chosen to be zero).

BUG: Overflow is handled by doing modulo arithmetic.

kernel-cutoff [LISP Global Variable]

VALUE: A *flonum*, default 0.01. The amount of a kernel that may be discarded in order to make a kernel of infinite extent fit into a small finite array. Measured as a fraction bounding the integral of the discarded part of the kernel divided by the integral of the kernel over the part of the space where the kernel has the same sign it does in the discarded part. The measurement is generally made by using integration of the continuous kernel function, not its descrete representation.

(local-maxima-of 'ar_input '(x_size ...)) [LISP Function]

WHERE: The x_size ... are non-negative.

RETURNS: An output array, lar_output, whose elements are the maxima of the elements of a rectangular box centered at the corresponding point of lar_input. The sizes of the box are (2*x_size+1, ...). The maximum is computed successively along each dimension of ar_input by calling the function maximum-filter for that dimension. The dimensions of lar_output are made identical to the dimensions of ar_input, by first expanding lar_input by appropriate amounts. This, and the conversion of element type to a-long, are accomplished by passing the input array to the function prepare-array. The input array is returned only when it has type a-long and the x_size ... arguments are all zero.

(local-minima-of 'ar_input '(x_size ...)) [LISP Function]

WHERE: The x_size ... are non-negative.

RETURNS: An output array, lar_output, whose elements are the minima of the elements of a rectangular box centered at the corresponding point of lar_input. The sizes of the box are (2*x_size+1, ...). The minimum is computed successively along each dimension of ar_input by calling the function minimum-filter for that dimension. The dimensions of lar_output are made identical to the dimensions of ar_input, by first expanding lar_input by appropriate amounts. This, and the conversion of element type to a-long, are accomplished by passing the input array to the function prepare-array. The input array itself is returned only when it has type a-long and the x_size ... arguments are all zero.

(**log-array-elements** 'lar_output ['lar_input]) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the logrithm of the corresponding element in lar_input. The two arrays may have different exponents.

(**mark-missing** 'lar_output 'ar_input
'(n_minimum n_maximum)
['(n_lower n_upper) ['(x_xsize x_ysize) ['x_count]]]) [LISP Function]

WHERE: Lar_output and lar_input have the same exponent and the same dimension sizes except for the X and Y dimensions. The X and Y dimensions of lar_input are respectively 2*x_xsize and 2*x_ysize larger than the X and Y dimensions of lar_output. N_minumum, n_maximum, x_xsize, x_ysize, x_count, n_lower, and n_upper may be given as *nil* if they are missing. X_xsize and X_ysize default to 1, and x_count defaults to 2. N_minimum and n_lower default to negative infinity, while n_maximum and n_upper default to positive infinity.

RETURNS: The number of missing values in lar_output.

SIDE EFFECT: Copies ar_input to lar_output replacing bad pixel values by the missing value *nil*. Values outside the range from minimum to maximum, inclusive, are bad.

If either n_lower or n_upper is given, then a value is replaced by *nil* unless the

(2*x_xsize+1 2*x_ysize+1)

box centered on the pixel contains at least x_count pixels (not counting the center pixel) in the range

(pixel-value+n_lower pixel-value+n_upper),

inclusive.

(**mark-missing-of** 'ar_input '(n_minimum n_maximum)
['(n_lower n_upper) ['(x_xsize x_ysize) ['x_count]]]) [LISP Function]

RETURNS: A new array lar_output which is computed by passing it and the other parameters to the *mark-missing* function. *Prepare-array* is applied to ar_input.

(**maximize-array-elements-with** 'lar_array 'n_number) [LISP Function]

RETURNS: Lar_array after its elements have been modified.

SIDE EFFECT: Each element of the array is replaced by the maximum of the element value and n_number. In other words, elements with values below n_number are replaced by n_number.

(**maximize-arrays** 'lar_output 'lar_input-1 ['lar_input-2]) [LISP Function]

WHERE: The arrays are similar and have the same exponent, and lar_input-2 defaults to lar_output.

RETURNS: Lar_output after its elements have been modified.

SIDE EFFECT: Takes the maximum of each element of lar_input-1 with the corresponding element of lar_input-2 and stores the result in the corresponding element of lar_output.

(**maximum-filter** 'lar_array 'x_dimension 'x_width) [LISP Function]

RETURNS: A slice of lar_array is returned after the lar_array elements are modified so that the slice holds the desired result. The size of the given dimension for the slice is x_width-1 less than the size of that dimension for lar_array, and the slice origin is 0 for that dimension. Other dimensions are not affected.

SIDE EFFECT: Applies a filter which forms the maximum of the last x_width input values for the given dimension of the given lar_array. Thus the output for subscript j in the given dimension is the maximum of the input for subscripts j, j+1, ..., j+x_width-1.

NOTE: The lar_array elements that are not in the returned slice are modified in undefined ways.

BUG: Missing values are not handled.

(**minimize-array-elements-with** 'lar_array 'n_number) [LISP Function]

RETURNS: Lar_array after its elements have been modified.

SIDE EFFECT: Each element of the array is replaced by the minimum of the element value and n_number. In other words, elements with values above n_number are replaced by n_number.

(**minimize-arrays** 'lar_output 'lar_input-1 ['lar_input-2]) [LISP Function]

WHERE: The arrays are similar and have the same exponent, and lar_input-2 defaults to lar_output.

RETURNS: Lar_output after its elements have been modified.

SIDE EFFECT: Takes the minimum of each element of lar_input-1 with the corresponding element of lar_input-2 and stores the result in the corresponding element of lar_output.

(**minimum-filter** 'lar_array 'x_dimension 'x_width) [LISP Function]

RETURNS: A slice of lar_array is returned after the lar_array elements are modified so that the slice holds the desired result. The size of the given dimension for the slice is x_width-1 less than the size of that dimension for lar_array, and the slice origin is 0 for that dimension. Other dimensions are not affected.

SIDE EFFECT: Applies a filter which forms the minimum of the last x_width input values for the given dimension of the given lar_array. Thus the output for subscript j in the given dimension is the minimum of the input for subscripts j, j+1, ..., j+x_width-1.

NOTE: The lar_array elements that are not in the returned slice are modified in undefined ways.

BUG: Missing values are not handled.

(**multiply-array-elements** 'lar_output 'lar_input-1 ['lar_input-2]) [LISP Function]

WHERE: All three arrays are similar and lar_input-2 defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the product of the corresponding elements in the two inputs. The three arrays may have different exponents.

BUG: The multiplication is done using modulo arithmetic in event of overflow.

(**multiply-array-elements-by** 'lar_array 'n_multiplier) [LISP Function]

RETURNS: Lar_array after its elements have been modified.

SIDE EFFECT: Multiplies each element of lar_array by n_multiplier.

BUG: The multiplication is done using modulo arithmetic in event of overflow.

(**overlay-missing** 'lar_output 'lar_input) [LISP Function]

WHERE: Lar_output and lar_input must have the same dimension sizes and exponent.

RETURNS: Lar_output after setting some of its elements.

SIDE EFFECT: Replaces every missing element value in lar_output by the corresponding element value in lar_input.

(**power-array-elements** 'lar_output ['lar_input] 'n_exponent) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the n_exponent power of the corresponding element in lar_input. The two arrays may have different exponents.

(scalar-product 'ar_input-1 'ar_input-2) [LISP Function]

WHERE: The two arrays are similar.

RETURNS: A number. The scalar product of the two arrays. More precisely, the sum of the products of corresponding elements in the arrays. *Nil* is returned if either array has any missing values.

(set-array-elements 'lar_array 'n_value) [LISP Function]
(set-array-elements 'lar_array *nil*) [LISP Function]

RETURNS: Lar_array after its elements have been set.

SIDE EFFECT: Sets all elements of lar_array to n_value or *nil*.

(set-missing-to 'lar_array 'n_value) [LISP Function]

RETURNS: Lar_array after setting some of its elements.

SIDE EFFECT: Replaces every missing value in lar_array by n_value.

(shrink-missing 'lar_output 'ar_input ['x_count]) [LISP Function]

WHERE: Lar_output and ar_input have the same exponent and the same dimension sizes except for the X and Y dimensions. The X and Y dimensions of ar_input are larger by 2 than the X and Y dimensions of lar_output. X_count defaults to 2 and must be $>= 2$.

RETURNS: The number of missing values left in lar_output.

SIDE EFFECT: Copies the elements of ar_input to lar_output, replacing some of the missing values by estimates. In general the output element equals the input element unless the input value is missing and has at least x_count non-missing 2-dimensional 8-neighbors.

Missing values are replaced by values obtained through inspection of the 2-dimensional 8-neighbors of the missing value.

Given a pair of non-missing neighbors, the missing value is replaced by their average. Generally there are many such pairs, from which one is chosen whose two values are closest together. If there are many closest pairs, one is choosen which has the lest bend in the line from one of the pair points to the missing value point to the other pair point.

(shrink-missing-of 'ar_input
'x_count ...) [LISP Function]

WHERE: More than one x_count value may be given.

RETURNS: A new array lar_output which is computed by passing ar_input and x_count through *shrink-missing*. *Prepare-array* is applied to ar_input.

If more than one x_count parameter is given, each x_count parameter is removed from the parameter list as it is used, and the process is repeated with the last lar_output substituted for ar_input. As an optimization, the process stops when there are no missing values left in lar_output.

(sin-array-elements 'lar_output ['lar_input]) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the sine of the corresponding element in lar_input. The two arrays may have different exponents.

(square-root-array-elements 'lar_output ['lar_input]) [LISP Function]

WHERE: Both arrays are similar and lar_input defaults to lar_output.

RETURNS: Lar_output after its elements have been set.

SIDE EFFECT: Sets each element in lar_output to the square root of the corresponding element in lar_input. The two arrays may have different exponents.

(subtract-arrays 'lar_output ['lar_input-1] 'lar_input-2) [LISP Function]

WHERE: The arrays are similar and have the same exponent, and lar_input-1 defaults to lar_output.

RETURNS: Lar_output after its elements have been modified.

SIDE EFFECT: Subtracts each element of lar_input-2 from the corresponding element of lar_input-1 and stores the result in the corresponding element of lar_output.

BUG: The subtraction is done using modulo arithmetic in event of overflow.

(sum-filter 'lar_array 'x_dimension 'x_width) **[LISP Function]**

RETURNS: A slice of lar_array is returned after the lar_array elements are modified so that the slice holds the desired result. The size of the given dimension for the slice is x_width-1 less than the size of that dimension for lar_array, and the slice origin is 0 for that dimension. Other dimensions are not affected.

SIDE EFFECT: Applies a filter that computes the mean of the last x_width input values for the given dimension of the given lar_array. Thus the output for subscript j in the given dimension is the mean of the input for subscripts j, j+1, ..., j+x_width-1.

NOTE: The lar_array elements that are not in the returned slice are modified in undefined ways.

BUGS: Missing values are not handled. Overflow is handled by doing modulo arithmetic.

CHAPTER 9

BIT GRAPHICS

1. GLOSSARY.

(a-bitgraph-parameter-set *has-line-width* 'x_line-width [LISP Macro]
has-1-width 'x_1-width *has-1-height* 'x_1-height
has-5-width 'x_5-width *has-5-height* 'x_5-height
has-10-width 'x_10-width *has-10-height* 'x_10-height)

default-bitgraph-parameter-set [LISP Global Variable]

USE: *A-bitgraph-parameter-set* provides parameters for drawing bitgraphs that are dependent upon the resolution of the bitgraph. E.g., such parameters as would be different for a 1800×1800 point screen and for a 512×480 screen.

default-bitgraph-parameter-set is used by bitgraph functions that need such parameters as the default bitgraph parameter set.

HAS-LINE-WIDTH: This is the recommended minimum line width in pixels for drawing lines.

HAS-1-WIDTH:

HAS-1-HEIGHT:

HAS-5-WIDTH:

HAS-5-HEIGHT:

HAS-10-WIDTH:

HAS-10-HEIGHT: These are the recommended width and height of ruler lines. x_10-height and x_10-width are for the lines that mark every 10 measurement units. X_5-width and x_5-height are for the lines between these that mark every 5 measurement units, and x_1-width and x_1-height are for the other lines that mark every 1 measurement unit. Height (but not width) may be 0 to eliminate a line.

(**a-character-set** *has-file* 's_file
has-font 's_font
has-sizes '(x_xsize 'x_ysize)) [SKETCH Type Macro]

a-character-set [SKETCH Type Object]
cset_ [Argument Prefix]

(**read-character-set** 'cset_character-set) [SKETCH Attribute Macro]

(**has-file** 'cset_character-set) [SKETCH Attribute Macro]
(**has-font** 'cset_character-set) [SKETCH Attribute Macro]
(**has-sizes** 'cset_character-set) [SKETCH Attribute Macro]
(**has-x-range** 'cset_character-set) [SKETCH Attribute Macro]
(**has-y-range** 'cset_character-set) [SKETCH Attribute Macro]
(**has-width-range** 'cset_character-set) [SKETCH Attribute Macro]
(**has-width-estimate** 'cset_character-set) [SKETCH Attribute Macro]
(**has-been-read** 'cset_character-set) [SKETCH Attribute Macro]
(**has-dispatch-array** 'cset_character-set) [SKETCH Attribute Macro]
(**has-bitgraph-array** 'cset_character-set) [SKETCH Attribute Macro]

USE: *A-character-set* defines the bitgraph masks of a character set with a particular style and size.

The character set is defined in a file stored in the format of the Berkeley UNIX font library: see vfont(5). This file is read into memory where its information is stored in two arrays: the dispatch array and the bitgraph array. Reading the file and creating the arrays is not done until the function *read-character-set* is applied to the character set. Until then the character set attributes not set when the character set is created are *nil*.

HAS-BEEN-READ:

READ-CHARACTER-SET: The *has-been-read* attribute is *nil* if the character set has not been read from the font file (*has-file*), or if the dispatch and bitgraph array elements (see HAS-DISPATCH-ARRAY below) are not allocated. This latter happens when the character set is written into a catalog and read back from the catalog without allocating the elements of these arrays.

Otherwise *has-been-read* is *t*.

The function call—

(*read-character-set* cset_character-set)

reads the character set and allocates the array elements, if these have not already been done, and returns the character set as its value in any case. This function call is usually used as the argument to C functions requiring the character set after it has been read.

HAS-SIZES: These are the width (x_xsize) and height (x_ysize) in pixels between characters. For a character set with variable width characters, the width may not be the maximum width of any character, but should be the maximum likely average width of the characters in any set of English words containing at least 5 characters (including spaces).

X_xsize defaults to the value of the *has-width-estimate* attribute, while x_ysize defaults to the ceiling of *has-y-range* maximum minus the *has-y-range* minimum times 1.2.

HAS-X-RANGE:

HAS-Y-RANGE:

HAS-WIDTH-RANGE: These are lists of the form '(x_minimum x_maximum) giving the smallest and largest X and Y coordinates of any pixel in any character of the character set; and giving the smallest and largest width of any character in the character set. The width of a character is the number of pixels the cursor is to be moved to the right when the character is drawn (and has nothing to do with how many pixels the character turns on).

HAS-WIDTH-ESTIMATE: This is the width of the widest capital letter other than M or W. It may be used to estimate the horizontal size of the character set.

HAS-FONT: A symbol naming the font. Some of the fonts available and their has-sizes

Font	Has-Sizes
fixed-roman	(9 14), (12 21), (16 27), (19 32), (25 42)
hershey-bold	(13 19), (15 21), (18 25), (20 28), (22 31), (24 34), (26 36), (31 42), (35 49), (40 54), (44 59), (48 66), (53 72), (61 83), (79 107)
hershey-italic	(13 19), (15 21), (18 25), (20 27), (22 31), (24 33), (26 36), (31 41), (35 48), (40 54), (44 59), (48 66), (53 71), (61 82), (79 105)
hershey-roman	(13 19), (15 21), (18 25), (20 28), (22 34), (24 34), (26 36), (31 42), (35 49), (40 54), (44 59), (48 66), (53 72), (61 83), (79 107)
nonic-bold	(17 27), (21 32), (24 37)
nonic-italic	(16 27), (21 32), (24 37)
nonic-roman	(16 27), (20 32), (23 37)
screen-bold	(8 15), (9 17)
screen-roman	(6 9), (7 12), (7 15), (8 17), (8 17)
script	(51 51)
serif-roman	(7 13), (7 14), (7 15), (8 17), (9 19)
shadow	(31 42)
times-bold	(13 16), (15 19), (17 24), (19 26), (22 29), (24 31), (26 34), (30 39), (35 45), (39 50), (43 55), (48 61), (52 68), (60 78), (78 100)
times-italic	(12 17), (14 19), (16 23), (18 25), (20 28), (22 32), (24 34), (28 39), (32 45), (36 50), (40 56), (44 61), (48 67), (56 77), (72 99)
times-roman	(14 19), (16 23), (18 26), (20 28), (23 32), (25 35), (27 37), (32 45), (36 50), (41 56), (45 62), (50 69), (54 74), (63 88), (81 112)

If *a-character-set* has non-*nil* font and *has-sizes* it is enterred in the font data base maintained for use by the *find-character-set* function. See the description of that function.

HAS-FILE: The UNIX font file (see vfont(5) for format). This file is searched for using the directory list provided by—

(*status font-search-path*).

(bar-graph 'bgar_output 'lar_input 'rul_ruler ['s_mode]) [LISP Function]

WHERE: The X dimension size of bgar_output must be the X dimension size of lar_input times some integer x_width > 0.

S_mode must be either *draw*, *erase*, *reverse*, or *invisible*: the default is *draw*.

SIDE EFFECT: Logically OR's (if s_mode is *draw*) a bar graph of lar_input into lar_output. The bars run vertically (in the Y dimension) upward (toward negative Y values) for some bar height chosen by the associated lar_input element value. The height of the bar is determined by mapping the element value by the inverse of the affine transformation defined by rul_ruler, and *round*'ing the result to the nearest integer. Bars with negative height are not drawn, and bars with height larger than the Y dimension size of bgar_output are clipped to that Y dimension size.

The width of each bar in the X direction is x_width.

Missing elements of lar_input do not produce a bar.

(bar-graph-of 'ar_input x_height [x_width]) [LISP Function]

WHERE: X_height > 0, x_width > 0. X_width defaults to 1.

Lar_input must be one dimensional.

RETURNS: A new bitgraph array bgar_output whose elements are set by passing it to bar-graph. Bgar_output has X dimension size equal to x_width times the X dimension size of ar_input. Bgar_output has Y dimension size equal to x_height.

If ar_input's X dimension has a ruler, the ruler for bgar_output's X dimension is set to the ruler of ar_input's X dimension with the scale multiplied by x_width.

If ar_input has any non-missing element, the ruler for bgar_output's Y dimension is set to—

(x_height (0 x_maximum+1))

where x_maximum is the maximum value of any ar_input element. This ruler is also passed to *bar-graph*.

(**bitgraph-box** 'bgar_output 'n_xminimum 'n_xmaximum 'n_yminimum 'n_ymaximum ['s_mode]) [LISP Function]

WHERE: The limits n_xminimum and n_xmaximum may be exchanged without effect, and similarly n_yminimum and n_ymaximum.

S_mode must be either *draw*, *erase*, *reverse*, or *invisible*: the default is *draw*.

RETURNS: *Bitgraph-box* returns bgar_output after setting some of its bits.

SIDE EFFECT: Logically OR's (if s_mode is *draw*) a rectanglular box with horizontal and vertical sides given by the lines—

X == (*ceiling* n_xminimum)
X == (*floor* n_xmaximum)
Y == (*ceiling* n_yminimum)
Y == (*floor* n_ymaximum)

into bgar_output.

NOTE: (*has-line-width *default-bitgraph-parameter-set**)

is a good minimum value for

n_xmaximum - n_xminimum + 1

or

n_ymaximum - n_yminimum + 1

when the purpose is to draw a line that bounds an image, graph, table, or other figure.

(**bitgraph-line** 'bgar_output 'n_x1 'n_y1 'n_x2 'n_y2 ['n_width ['s_mode]]) [LISP Function]

WHERE: N_width may be *nil* to mean the same thing as a missing n_width argument.

S_mode must be either *draw*, *erase*, *reverse*, or *invisible*: the default is *draw*.

RETURNS: Bgar_output after setting some of its bits.

SIDE EFFECT: Draws the line joining the point (x1, y1), and the point (x2, y2), by logically OR'ing (is s_mode is *draw*) a parallelogram containing that line into bgar_output.

The parallelogram starts at the point

$$(x1 - xw/2 - yw/2,\ y1 - yw/2 + xw/2)$$

and has sides

$$(x2 - x1 + xw,\ y2 - y1 + yw)$$

and

$$(yw,\ -xw)$$

where

$$length = sqrt((x2 - x1)**2 + (y2 - y1)**2)$$
$$xw = n\_width * (x2 - x1) / length$$
$$yw = n\_width * (y2 - y1) / length$$

If n_width is *nil*, then xw and yw are computed as above and then transformed by—

$$m = (1 + 2^{-15}) * max(|xw|, |yw|)$$
$$xw = m * xw$$
$$yw = m * yw$$

which gives the smallest effective width that will draw a visible line.

In order to ensure that the line is drawn exactly the same way, no matter how it is presented, and in spite of rounding errors, the end points are first exchanged unless $x2 > x1$ or $x2 == x1$ and $y2 > y1$.

Before the parallelogram is drawn, the line is clipped so that x1 and x2 lie in approximately the range

$$(-0.874 + (|xw| + |yw|)/2, xsize - 0.126 - (|xw| + |yw|)/2)$$

and y1 and y2 lie in approximately the range

$$(-0.874 + (|xw| + |yw|)/2, ysize - 0.126 - (|xw| + |yw|)/2)$$

The order of operations is thus—

exchange of end points if necessary
clipping
specification of parallelogram
OR'ing of parallelogram

(**bitgraph-lines** 'bgar_output [n_dot-size] [s_mode] [*has-origins* '(n_xorigin n_yorigin)] [*has-zooms* '(n_xzoom n_yzoom)] ['(n_x n_y) ...] [nil] ['ar_array ...]) [LISP Function]

WHERE: N_dot-size, n_xzoom, and n_yzoom default to 1, while n_xorigin and n_yorigin default to 0.

S_mode must be either *draw*, *erase*, *reverse*, or *invisible*: the default is *draw*.

RETURNS: Bgar_output after modifying it.

SIDE EFFECT: Logically OR's (if s_mode is *draw*) into bgar_output lines whose end points are given by arguments of the form '(n_x n_y) or ar_array. For ar_array arguments, each row of the array represents a point, with the first column (X subscript equal 0) being the point's X coordinate, and the second column (X subscript equal 1) being the point's Y coordinate. The array must have exactly two columns (array's X dimension size).

Point arguments may also be missing: these are represented by *nil* arguments, or by array rows whose elements are missing. The missing points break the sequence of all points into subsequences of non-missing points. Each such subsequence defines a broken line obtained by connecting the subsequence points in order. It is these broken lines that are logically OR'ed (if s_mode is *draw*) into bgar_output.

The lines are drawn by moving a circular dot whose size in pixels is n_dot-size.

(**bitgraph-parallelogram** 'bgar_output 'n_x 'n_y 'n_x1 'n_y1 'n_x2 'n_y2 ['s_mode]) [LISP Function]

WHERE: S_mode must be either *draw*, *erase*, *reverse*, or *invisible*: the default is *draw*.

RETURNS: Bgar_output after setting some of its bits.

SIDE EFFECT: Logically OR's (is s_mode is *draw*) a parallelogram into bgar_output. The sides are formed by the vectors (n_x1 n_y2) and (n_x2 n_y2) drawn from the origin (n_x n_y). All the coordinates may be given as fractions.

Only one boundary in each direction has the property that points exactly on it are turned on. The other boundary in the direction has the property that points exactly on it are turned off.

It may be wise to use fractional coordinates to be sure the right things are turned on: e.g.-

(*bitgraph-parallelogram* bgar_output -0.5 -0.5 0.5 -0.5 21 21)

will draw a 20 point line at a 45 degree angle from (0 0).

In order to ensure that the parallelogram is drawn exactly the same way, no matter how it is presented, and in spite of rounding errors, transformations are made on the parameters to put them in cannonical form (for instance, x y is changed to the leftmost point of the parallelogram in the case where bgar_output has yincrement == 1).

A parallelogram any part of which lies outside the X and Y coordinate ranges-

(-0.875, xsize - 0.125) (-0.875, ysize - 0.125)

will not be drawn.

(**bitgraph-ruler** 'bgar_output 'rul_ruler [LISP Function]
'x_xminimum 'x_xmaximum 'x_ybase
's_mode
do-reverse g_reverse-switch]
[*has-bitgraph-parameter-set* 'bgps_parameter-set])

WHERE: X_xminimum may be above x_xmaximum.

The rul_ruler scale must be non-zero.

Bgar_output must have either xincrement or yincrement equal to 1 or -1.

The default value of bgps_parameter-set is **default-bitgraph-parameter-set**. Bgps_parameter-set contributes the parameters x_1-height, x_1-width, x_5-height, x_5-width, x_10-height, x_10-width.

If g_reverse-switch is non-*nil* the signs of x_1-height, x_5-height, and x_10-height are effectively changed.

S_mode must be either *draw*, *erase*, *reverse*, or *invisible*: the default is *draw*.

RETURNS: *Ruler* returns bgar_output after setting some of its bits.

SIDE EFFECT: *Bitgraph-ruler* makes ruler marks in bgar_output with the marks having a base on the horizontal (X direction) line defined by x_xminimum, x_xmaximum, and x_ybase. The marks themselves run in the Y direction.

The X coordinate is scaled so that unscaled coordinate x_x is mapped first onto x_x−x_minimum and then by rul_ruler to a real number called the scaled X coordinate. Marks are placed at scaled X coordinates ..., -2, -1, 0, 1, 2, ... Marks at ..., -20, -10, 0, 10, 20, ... are of (Y coordinate) height x_10-height and (unscaled X coordinate) width x_10-width. Marks at ..., -15, -5, 5, 15, ... have height x_5-height and width x_5-width. The rest of the marks have height x_1-height and width x_1-width.

If possible the ruler scale is multiplied by 10 repeatedly (without changing the value x_minimum maps onto) until doing so further would cause the marks to overlap. If necessary the ruler scale is divided by 10 repeatedly until none of the marks overlap. In order not to overlap, marks are required to have x_1-width space between them.

Marks can be surpressed by setting the appropriate heights to 0. However the widths may not be 0, and the scale will still be choosen as if the marks were being made.

If the heights are positive the marks are made in the positive Y coordinate direction, which is down. If the heights are negative the marks will be made in the opposite direction, up. The variable width marks are centered on their nominal X position. A mark is not made if any part of it would be outside the limits of bgar_output.

(**bitgraph-text** 'ubar_output '(x_xorigin x_yorigin) [LISP Function]
['s_mode] ['s_orientation] ['s_adjust ...] 'cset_character-set
's t_string ...)

WHERE: S_mode is *draw* (the default), *erase*, *reverse*, or *invisible*.

S_orientation is *mirror*, *left-rotate*, *left-mirror*, *top-rotate*, *top-mirror*, *right-rotate*, or *right-mirror*.

S_adjust is *left*, *right*, *over*, or *under*.

Nil's in the optinal part of the argument list are ignored.

SIDE EFFECT: Text is drawn at the indicated origin.

Each t_string argument is taken to be one or more separate lines of text. Only printing characters, the single space character, the tab character, and the line feed character are permitted in the strings. Tab stops are set every 8 characters from the beginning of each string, or from the previous line feed character. Tabs are translated into space characters.

The width and height of the total text consisting of all the lines is computed. This is used to form an imaginary box around the text. The text lines are then adjusted in the box according to some of the s_adjust parameters. Lastly, the box is positioned in the ubar_output according to the origin position, s_adjust parameters, and s_orientation parameter, and the text is drawn.

Individual characters that will not fit completely inside ubar_output are not drawn. Characters that will fit completely inside are always drawn.

S_ADJUST: The s_adjust parameters control the positioning of lines within the text box, and the positioning of the box relative to the origin. The possible s_adjust values are—

left	The origin is placed just to the left of the box. The lines of text with the least amount of blank space at their left are left justified in the box. The lines with the next least amount of blank space at their left have their first non-blank character printed directly under the character in the same column of the first line above them that has already been justified, if any, or the first line below them if there is no such line above. And so forth, until all lines are justified.
right	Like *left* but to the right instead of the left.
	If neither *left* or *right* is given, each line has blank space at its beginning and ending removed, and is then centered in the box. The origin is placed at the center of the box in the horizontal dimension.
under	The origin is placed just under the box.
over	The origin is placed just over the box.
	If neither *under* or *over* is given, the origin is placed at the center of the box in the vertical dimension.

S_ORIENTATION: This is one of the values—

nil	*mirror*
left-rotate	*left-mirror*
top-rotate	*top-mirror*
right-rotate	*right-mirror*

The entire text is rotated as indicated around the origin position. *Nil* means to do no rotation; *top-rotate* means to rotate 180 degrees to make the bottommost part of the characters nearest the top of the display. The mirror forms do not cause the characters to be mirror-imaged, nor do they reverse the order of the characters in the text. But they do switch which side of the text the origin is on, left or right, when viewed after any rotation.

(find-character-set 's_font '(n_xsize n_ysize)) [LISP Function]

(clear-character-sets ['s_font]) [LISP Function]
find-character-set [LISP Global Variable]
character-set-fonts [LISP Global Variable]

WHERE: **find-character-set** has the default value '*sbg/sbg_*.

RETURNS: *Find-character-set* returns *a-character-set* with the given *has-font* attribute and the largest *has-sizes* attributes available that are less than or equal to the given sizes.

If there are no character sets with the given *has-font* attribute, the catalog file with the name—

(*concat *find-character-set** s_font '.ca)

is read. It will presumably have character sets with s_font as their *has-font*.

Clear-character-sets clears the font data base of all character sets with a given *has-font*, or all character sets if no argument is given. **character-set-fonts** is a list of all the fonts in the font data base.

(get-character-bitgraph 'cset_character-set 's/x_character ['s_orientation]) [LISP Function]

USE ONLY WHEN: Diagnosing character set appearance.

WHERE: S_orientation is *nil*, *mirror*, *left-rotate*, or *right-mirror*.

RETURNS: The bitgraph array of a character in cset_character-set. The character is the first character of s_character, or has the ASCII code x_character.

The array presents the character in the given orientation; *nil* for normal, *mirror* for mirror image, *left-rotate* for the rotated 90 degrees to the left, and *right-mirror* for the mirrored character rotated 90 degrees to the right. In all these cases the Y dimension increment equals 1, so the four orientations actually select four different arrays in memory.

(get-character-display 'cset_character-set 's/x_character) [LISP Function]

USE ONLY WHEN: Diagnosing character set appearence.

RETURNS: The *a-bitgraph-character* descriptor for a character in cset_character-set. The character is the first character of s_character, or has the ASCII code x_character.

sbg_bit [x_x] [C Macro]
sbg_tobit [x_x] [C Macro]
sbg_frombit [x_x] [C Macro]
sbg_shift [x_x] [C Macro]
sbg_endbit [C Macro]
sbg_endtobit [C Macro]
sbg_endfrombit [C Macro]
sbg_endshift [C Macro]

WHERE: $0 <= x\_x <= 31$.

REQUIRES: #include <sbg/sbg_bit.h>

RETURNS: The subscripted expressions return a *ulong* which, when viewed as a one dimensional 32 bit array with xincrement == 1, has bit x_x only on in the case of *sbg_bit*, has only bits 0, 1, ..., x_x on in the case of *sbg_tobit*, or has only bits x_x, ..., 31 on in the case of *sbg_frombit*.

In the case of *sbg_shift* bits x_x, ..., 7 are on within each of the 4 bytes within the array: this is useful for right-to-left computers for masking a word in which every byte is to be right shifted by x_x. The mask for a word in which each byte is to be left shifted by x_x should be

~ *sbg_shift* [8 − x_x]

for $1 <= x\_x <= 8$.

sbg_bit, *sbg_tobit*, *sbg_frombit*, and *sbg_shift* are contiguous vectors of *ulong*'s, which may be stepped along by a *ulong* pointer p using a * p ++ expression. The expressions *sbg_endbit*, *sbg_endtobit*, *sbg_endfrombit*, and *sbg_endshift* are pointers to *ulong*'s that point at the first location after the respective vectors, so expressions such as p < *sbg_endbit* may be used to terminate loops.

sbg_box (bgar_output, x_xmin, x_xmax, x_ymin, x_ymax, g_mode) [C Function]

WHERE: G_mode must be one of *SBG_DRAW*, *SBG_ERASE*, *SBG_REVERSE*, or *SBG_INVISIBLE*.

SIDE EFFECT: Draws a box in bgar_output with the given minimum and maximum X and Y coordinates.

NOTE: X_xmin and x_xmax may be reverse (x_xmin greater than x_xmax) without effecting results (the function sorts these arguments). Similarly for x_ymin and x_ymax. If the box is too big to fit in bgar_output, the box is clipped.

sbg_character [C Type]
bgchar_ [Argument Prefix]
bgchar_character->**sbg_coffset** [C Structure Element]
bgchar_character->**sbg_corigin** [x_dimension] [C Structure Element]
bgchar_character->**sbg_csize** [x_dimension] [C Structure Element]
bgchar_character->**sbg_cwidth** [C Structure Element]

a-bitgraph-character [SKETCH Type Object]
an-allocate-bitgraph-character [SKETCH Type Object]
SBG_ACHARACTER [C Global Variable]

USE: This is an element in the dispatch array of a character set. It gives information about one character. The *sbg_coffset* is the index of the first *long* of the character's *a-ubit* matrix within the character set's bitgraph array. This matrix has *sbg_csize[SAR_X]* columns (X size) and *sbg_csize[SAR_Y]* rows (Y size). Each row, however, is expanded to an integral number of *long*'s by adding 0 bits on the end, so the actual number of columns is—

$$((sbg\_csize[SAR\_X] + 31) / 32) * 32$$

The character's bitgraph matrix is to be inserted in output at some displacement from the cursor location. This displacement is is given by *sbg_corigin[SAR_X]* in the X direction and *sbg_corigin[SAR_Y]* in the Y direction.

After inserting the character, the cursor is to be moved *sbg_cwidth* columns to the right.

A-BITGRAPH-CHARACTER:

AN-ALLOCATE-BITGRAPH-CHARACTER:

SBG_ACHARACTER: *A-bitgraph-character* is the SKETCH type of a *sbg_character* object. It is formally a pointer to a structure. *An-allocate-bitgraph-character* is the SKETCH type used as an array *has-element-type* if the bitgraph character structures are to be the array elements. It refers to the structure proper, and not a pointer to it. *SBG_ACHARACTER* is the C global variable equal to *an-allocate-bitgraph-character*.

sbg_dot (ux_ubbase, x_xincrement, x_yincrement, x_xoffset, x_yoffset, x_xdelta, x_ydelta, x_size, s_mode) [C Function]

WHERE: An output *a-ubit* array is located at base bit address ux_ubbase and has increments x_xincrement and y_yincrement.

X_xoffset, x_yoffset, x_xdelta, x_ydelta, and x_size are all measured in units of 1/2 pixel. X_xdelta and x_ydelta must be in the range from -2 through +2. X_size must be in the range from 2 to 28 and have been previously initialized by *initialize-bitgraph-point-size*.

S_mode is SBG_DRAW, SBG_ERASE, SBG_REVERSE, or SBG_INVISIBLE.

Either x_xincrement must equal 1, or x_xincrement and x_yincrement must both be exactly divisible by 8.

SIDE EFFECT: A dot is put into the output array all along the straight line from the position—

(x_xoffset+epsilon x_yoffset+epsilon),

to the position

(x_xoffset+x_xdelta+epsilon x_yoffset+x_ydelta+epsilon),

where these offset and delta coordinates are in units of 1/2'th pixel: i.e., have 2 times the resolution of the X and Y coordinates in the output array. The diameter of the dot is x_size, also in 1/2'th pixel units.

Epsilon is choosen to be a very small number. e.g. 0.001. It prevents anomalies such as some vertical lines with size = 2 being twice as wide as others.

If s_mode is SBG_DRAW the dot pixels are set to 1; if SBG_ERASE, the pixels are set to 0; if SBG_REVERSE, the pixels are complemented; and if SBG_INVISIBLE, the pixels are left untouched.

WARNING: No check is made to see if the dot will go off the edge of the array. Memory may be damaged if it does.

No check is made to see if the arguments are in their proper ranges.

sbg_line (bgar_output, x_x1, x_y1, x_x2, x_y2, x_width, g_mode) [C Function]

WHERE: X_x1, x_y1, x_x2, x_y2, and x_width are all in units of 2**-16, and x_width may equal *SAT_MISSING* to represent a *nil* or missing value.

G_mode must be one of *SBG_DRAW*, *SBG_ERASE*, *SBG_REVERSE*, or *SBG_INVISIBLE*.

SIDE EFFECT: Performs the same action as the LISP function call—

(*bitgraph-line* x1 y1 x2 y2 width)

where the LISP arguments are the floating point equivalents of the C arguments.

sbg_or (ux_outp, x_oinc, *ulx_inp, x_xsize, x_ysize) [C Function]

USE ONLY WHEN: Maintaining the SKETCH bitgraph package. Others may use this function, but must beware that no error checking whatever is done by the function.

WHERE: An output *a-ubit* array is located at bit address ux_outp (as for sar_ubbase), and an input *a-ubit* array at address *ulx_inp. The xincrement of both arrays must be 1; the yincrement must be x_oinc for the output array and 32*x_xsize for the input array, the xsize must be 32*x_xsize for both arrays, and the ysize must be x_ysize for both arrays.

SIDE EFFECT: Logically OR's the input array into the output array. Is designed as a high speed function used as a primitive function in the bitgraph package for or'ing characters, pixel shades, contours, etc into *a-ubit* arrays.

BUG: The current code will not work on a right to left computer (in the tradition of VAX and INTEL, not IBM or MOTOROLA) unless *ulongs* do not have to be aligned (which they do not on a VAX).

sbg_pgram (bgar_output, x_x, x_y, x_x1, x_y1, x_x2, x_y2, g_mode) [C Function]

WHERE: X_x, x_y, x_x1, x_y1, x_x1, and x_y1 are all in units of 2**-16.

G_mode must be one of *SBG_DRAW*, *SBG_ERASE*, *SBG_REVERSE*, or *SBG_INVISIBLE*.

SIDE EFFECT: Same as the LISP function—

(*bitgraph-parallelogram* output x y x1 y1 x2 y2)

sbg_ruler (bgar_output, f_xfirst, f_xstep, x_xminimum, x_xmaximum, x_ybase, x_1width, x_1height, x_5width, x_5height, x_10width, x_10height, g_mode) [C Function]

WHERE: X_xminimum may be above x_xmaximum.

G_mode must be one of *SBG_DRAW*, *SBG_ERASE*, *SBG_REVERSE*, or *SBG_INVISIBLE*.

SIDE EFFECT: Performs same action on bgar_output as the LISP function call—

```
(bitgraph-ruler output '((0 ...) (f_xfirst ...) f_xstep) xminimum xmaximum ybase mode
               (a-bitgraph-parameter-set has-1-width 1-width
                                         has-1-height 1-height
                                         has-5-width 5-width
                                         has-5-height 5-height
                                         has-10-width 10-width
                                         has-10-height 10-height))
```

sbg_s_or (ux_outp, x_oinc, *usx_inp, x_xsize, x_ysize) [C Function]

USE ONLY WHEN: Maintaining the SKETCH bitgraph package. Others may use this function, but must beware that no error checking whatever is done by the function.

WHERE: An output *a-ubit* array is located at bit address ux_outp (as for sar_ubbase), and an input *a-ubit* array at address *usx_inp. The xincrement of both arrays must be 1; the yincrement must be x_oinc for the output array and 16 * x_xsize for the input array, the xsize must be 16*x_xsize for both arrays, and the ysize must be x_ysize for both arrays.

SIDE EFFECT: Logically OR's the input array into the output array. Is designed as a high speed function used as a primitive function in the bitgraph package for or'ing characters, pixel shades, contours, etc into arrays.

BUG: The current code will not work on a right to left computer (in the tradition of VAX and INTEL, not IBM or MOTOROLA) unless *ushorts* do not have to be aligned (which they do not on a VAX).

NOTE: This function is similar to *sbg_or*, the only difference being that *ushorts* (16 bits) are used everywhere instead of *ulongs* (32 bits).

CHAPTER 10

ANALYTIC GEOMETRY

1. GLOSSARY.

1d-zero-transform	[LISP Global Variable]
1d-to-2d-zero-transform	[LISP Global Variable]
1d-to-3d-zero-transform	[LISP Global Variable]
2d-to-1d-zero-transform	[LISP Global Variable]
2d-zero-transform	[LISP Global Variable]
2d-to-3d-zero-transform	[LISP Global Variable]
3d-to-1d-zero-transform	[LISP Global Variable]
3d-to-2d-zero-transform	[LISP Global Variable]
3d-zero-transform	[LISP Global Variable]
1d-unit-transform	[LISP Global Variable]
2d-unit-transform	[LISP Global Variable]
3d-unit-transform	[LISP Global Variable]

sag_1d_zero_transform	[C Global Variable]
sag_1d_to_2d_zero_transform	[C Global Variable]
sag_1d_to_3d_zero_transform	[C Global Variable]
sag_2d_to_1d_zero_transform	[C Global Variable]
sag_2d_zero_transform	[C Global Variable]
sag_2d_to_3d_zero_transform	[C Global Variable]
sag_3d_to_1d_zero_transform	[C Global Variable]
sag_3d_to_2d_zero_transform	[C Global Variable]
sag_3d_zero_transform	[C Global Variable]
sag_1d_unit_transform	[C Global Variable]
sag_2d_unit_transform	[C Global Variable]
sag_3d_unit_transform	[C Global Variable]

VALUE: Unit and zero transforms of the given dimensions.

3d-x-unit-vector [LISP Global Variable]
3d-y-unit-vector [LISP Global Variable]
3d-z-unit-vector [LISP Global Variable]
2d-x-unit-vector [LISP Global Variable]
2d-y-unit-vector [LISP Global Variable]
1d-x-unit-vector [LISP Global Variable]
3d-zero-vector [LISP Global Variable]
2d-zero-vector [LISP Global Variable]
1d-zero-vector [LISP Global Variable]
0d-vector [LISP Global Variable]

sag_3d_x_unit_vector [C Global Variable]
sag_3d_y_unit_vector [C Global Variable]
sag_3d_x_unit_vector [C Global Variable]
sag_2d_x_unit_vector [C Global Variable]
sag_2d_y_unit_vector [C Global Variable]
sag_1d_x_unit_vector [C Global Variable]
sag_3d_zero_vector [C Global Variable]
sag_2d_zero_vector [C Global Variable]
sag_1d_zero_vector [C Global Variable]
sag_0d_vector [C Global Variable]

VALUE: Unit and zero vectors of the given dimensions and in the X, Y, and Z directions.

(a-cluster [*has-point-array* 'ar_point-array] [LISP Macro]
[*has-point-list* '(pt_point-1 ...)]
[*is-chain* 'g_chain-switch]
[*is-maximal-polygon* 'g_maximal-polygon-switch])

a-cluster [SKETCH Type Object]
cl_ [Argument Prefix]

(**has-point-array** 'cl_cluster) [SKETCH Attribute Macro]
(**has-point-list** 'cl_cluster) [SKETCH Attribute Macro]
(**is-chain** 'cl_cluster) [SKETCH Attribute Macro]
(**is-closed** 'cl_cluster) [SKETCH Attribute Macro]
(**is-maximal-polygon** 'cl_cluster) [SKETCH Attribute Macro]
(**has-dimension** 'cl_cluster) [SKETCH Attribute Macro]
(**has-count** 'cl_cluster) [SKETCH Attribute Macro]

"chain" [SKETCH Term]
"closed chain" [SKETCH Term]
"edge of chain" [SKETCH Term]
"maximal polygon" [SKETCH Term]

sag_cluster [C Type]
SAG_CLUSTER [C Global Variable]
cl_cluster->**sag_ctype** [C Structure Element]
cl_cluster->**sag_cparray** [C Structure Element]

cl_cluster->**sag_cplist** [C Structure Element]
cl_cluster->**sag_ccount** [C Structure Element]
cl_cluster->**sag_cdimension** [C Structure Element]
cl_cluster->**sag_cchain** [C Structure Element]
cl_cluster->**sag_cclosed** [C Structure Element]
cl_cluster->**sag_cmpolygon** [C Structure Element]

USE: *A-cluster* is a set of points. All the points must have the same dimension: 1, 2, or 3.

The set of points can be specified by either giving the *has-point-list* or the *has-point-array* attributes. The former is a list of *a-vector*'s that represent points. The later is an array whose X dimension size equals the dimension of the points and whose Y dimension equals the number of points. The array X coordinate values 0, 1, and 2 correspond to the X, Y, and Z point coordinates. The array Y coordinate values index the different points.

Only one of the two attributes, *has-point-list* or *has-point-array*, may be specified when creating a cluster. The other will be computed if accessed.

A-cluster is a chain if the first point is to be thought of as connected to the second point, the second point is connected to the third point, etc. The chain is closed if the last point is identical to the first point. More precisely, *a-cluster is-closed* if and only if it is a chain with at least one point and the first point equals (in the sense of *object-compare*) the last point (all chains with just one point are closed).

The edges of a chain are the line segments between consecutive points.

A maximal polygon is a closed chain of points lying in a plane whose edges equal those of the convex hull in the plane of the chain's set of points. The *is-maximal-polygon* attribute is *t* if the chain was computed in a way that makes it a maximal polygon; but a chain may still accidentally be a maximal polygon, even if the value of this attribute is *nil*.

HAS-POINT-ARRAY: The element type of this array is *a-short*. When *a-cluster* object is created, arrays with other element types may be specified as the *has-point-array*, but they will be copied if necessary to convert the element type to *a-short*.

IS-MAXIMAL-POLYGON: This may be *setf* (if it is belatedly realized that the cluster is in fact a maximal polygon).

SAG_CPLIST:

SAG_CPARRAY: These are C *sat_lvalue* values. *Sag_cparray* must be cast to a C *sar_array* value before used.

SAG_CCOUNT:

SAG_CDIMENSION: These are C *int* values.

SAG_CCHAIN:

SAG_CCLOSED:

SAG_CMPOLYGON: These are *on-lbit* values which take the C values 0 and 1 for the LISP values *nil* and *t*.

SAG_CLUSTER:

SAG_CTYPE: A LISP *a-cluster* object is a C *sag_cluster* structure. *Sag_ctype* is first element of an *sag_cluster* structure in C. It equals SAG_CLUSTER, which in turn equals—

sob_nobject ("a-cluster").

(**a-line** *has-start* 'pt_start
has-length 'n_length
[*has-direction* 'vec_direction]) [LISP Macro]

(**a-line** *has-start* 'pt_start
has-end 'pt_end [*is-infinite* 'g_infinite-switch]) [LISP Macro]

(**a-line** *has-start* 'pt_start
has-segment 'vec_segment [*is-infinite* 'g_infinite-switch]) [LISP Macro]

a-line [SKETCH Type]
lin_ [Argument Prefix]

(**has-length** 'lin_line) [SKETCH Attribute Macro]
(**has-start** 'lin_line) [SKETCH Attribute Macro]
(**has-direction** 'lin_line) [SKETCH Attribute Macro]
(**has-end** 'lin_line) [SKETCH Attribute Macro]
(**is-infinite** 'lin_line) [LISP Macro]
(**has-segment** 'lin_line) [LISP Macro]

sag_line [C Type]
SAG_LINE [C Global Variable]
lin_line->**sag_ltype** [C Structure Element]
lin_line->**sag_llength** [C Structure Element]
lin_line->**sag_linfinite** [C Structure Element]
lin_line->**sag_lstart** [C Structure Element]
lin_line->**sag_lend** [C Structure Element]
lin_line->**sag_ldirection** [C Structure Element]

USE: *A-line* represents a finite or infinite oriented line.

A finite line has a *has-start* point and a *has-end* point. Its *has-length* is the distance from the start point to the end point. If the length is non-zero, its *has-direction* attribute is a unit vector directed from the start to the end. If the length is zero, the *has-direction* is *nil*.

An infinite line has a *has-direction* attribute which is a unit vector in the direction of the line, and a *has-start* attribute which is a point on the line, and which must be perpendicular to the direction. The *has-end* and *has-length* attribute are *nil*.

One can get the *has-segment* and *is-infinite* attributes of a line, but these are not actually stored in the line. The *has-segment* attribute is the end of the line minus the start of the line for a finite line, and *nil* for an infinite line. The *is-infinite* attribute is *t* for an infinite line and *nil* for a finite line.

Any finite line may also be made by giving its start and end, or by giving its start and a *has-segment* value from which the end can be computed. An infinite line may also be made by specifying a finite line as just mentioned and adding a *t is-infinite* attribute.

Any direction given when *a-line* is created does not have to be a unit vector: it will be converted into one. It will also be converted to *nil* if the length is given as zero. Similarly the start given for an infinite vector does not have to be perpendicular to the direction: it will be changed to be so.

SAG_LLENGTH:

SAG_LINFINITE: *Sag_llength is a C float which takes the value SAT_FMISSING if the line is infinite. Sag_linfinite is a macro that tests whether sag_llength is missing.*

SAG_LSTART:

SAG_LEND:

SAG_LDIRECTION: These are all of C type *sat_lvalue*, and must be cast to the C type *sag_vector* before they are used.

SAG_LINE:

SAG_LTYPE: A LISP *a-line* object is a C *sag_line* structure. *Sag_ltype* is first element of a *sag_line* structure in C. It equals *SAG_LINE*, which in turn equals—

sob_nobject ("a-line").

WARNING ABOUT COMPARE-OBJECT: *A-line* will not equal itself when *uneval*'ed and then re-*eval*'ed. Two lines computed in different ways may be unequal when compared with *compare-object*, even though they are supposed to be equal in theory.

(an-ellipsoid *has-transform* 'trans_ortho
has-xradii '(n_xradius [n_yradius [n_zradius]])
[*has-center* 'vec_center]) [LISP Macro]
(an-ellipsoid *has-radius* 'n_radius
has-center 'vec_center)

an-ellipsoid [SKETCH Type]
ell_ [Argument Prefix]

(has-transform 'ell_ellipsoid) [SKETCH Attribute Macro]
(has-radii 'ell_ellipsoid) [SKETCH Attribute Macro]
(has-radius 'ell_ellipsoid) [SKETCH Attribute Macro]
(has-center 'ell_ellipsoid) [SKETCH Attribute Macro]

(has-dimension 'ell_ellipsoid) [SKETCH Attribute Macro]

RETURNS: *An-ellipse* object, which represents a finite 1, 2, or 3 dimensional ellipsoid. For a 1 dimensional ellipsoid, n_yradius and n_zradius are *nil*. Such an ellipsoid is just a pair of points. For a 2 dimensional ellipsoid only n_zradius is *nil*.

Transform_ortho is an orthogonal point transformation which defines a transformed coordinate system. In the transformed coordinate system the ellipsoid has the equation—

$$\left(\frac{X}{n\_xradius}\right)^2 + \left(\frac{Y}{n\_yradius}\right)^2 + \left(\frac{Z}{n\_zradius}\right)^2 = 1.0$$

(in which a coordinate is omitted if its corresponding radius is *nil*).

Trans_transform is M dimensional if the space in which the ellipsoid lives is M dimensional, even if the ellipsoid has fewer dimensions.

If n_radius and vec_center are given, all the radii are equal, the dimension M of the containing space is the dimension of vec_center, the linear part of the transform is the unit transform of the space, and vec_center is the *has-displacement* part of the transform.

HAS-RADIUS: Equal to the radii, such as n_xradius, if all the non-*nil* radii are *equal* and the dimension of the space containing the ellipsoid equals the dimension of the ellipsoid. Equal to *nil* if the radii are unequal or the dimension of the ellipsoid is less than the dimension of the space.

HAS-CENTER: The center of the ellipse: $-\vec{d}\cdot T^{-1}$, where $\vec{d}$ is the displacement part of trans_transform and T is the linear part.

HAS-DIMENSION: The number of non-*nil* radii from among n_xradius, n_yradius, and n_zradius.

SAG_ELLIPSOID:

SAG_ETYPE: A LISP *an-ellipsoid* object is a C *sag_ellipsoid* structure. *Sag_etype* is first element of an *sag_ellipsoid* structure in C. It equals SAG_ELLIPSOID, which in turn equals—

sob_nobject ("an-ellipsoid").

WARNING: *An-ellipsoid* will not equal itself when *uneval*'ed and then re-*eval*'ed. Two ellipsoids computed in different ways may be unequal when compared with *compare-object*, even though they are provably equal.

(angle-between-lines lin_line-1 lin_line-2) [LISP Macro]

RETURNS: The *flonum* angle in radians between the direction vectors of the lines. The angle is in the range $[0,\pi]$. If one of the lines is a zero length finite line, the result is *nil*.

(angle-between-vectors 'vec_vector-1 'vec_vector-2) [LISP Macro]

WHERE: Both vectors must have the same dimension.

RETURNS: The *flonum* equal to the angle between the vectors in radians. *Nil* if one of the vectors is of zero length.

In order to get accuracy, two different methods of computation are used: one for the case where the vectors are nearly parallel, and one for the case where the vectors are nearly perpendicular.

(a-transform [*has-xx* 'n_xx] [*has-xy* 'n_xy] [SKETCH Type Macro]
[*has-xz* 'n_xz] [*has-xt* 'n_xt]
[*has-yx* 'n_yx] [*has-yy* 'n_yy]
[*has-yz* 'n_yz] [*has-yt* 'n_yt]
[*has-zx* 'n_zx] [*has-zy* 'n_zy]
[*has-zz* 'n_zz] [*has-zt* 'n_zt]
[*has-tx* 'n_tx] [*has-ty* 'n_ty]
[*has-tz* 'n_tz] [*has-tt* 'n_tt]
[*is-orthogonal* 'g_orthogonal]
[*has-input-dimensions* 'x_input-dimensions]
[*has-output-dimensions* 'x_output-dimensions])

(a-transform [*has-displacement* 'vec_displacement] [SKETCH Type Macro]
has-axis 'vec_axis *has-angle* 'n_angle)

a-transform [SKETCH Type Object]
trans_ [Argument Prefix]

(has-inverse 'trans_transform) [SKETCH Attribute Macro]
(has-determinant 'trans_transform) [SKETCH Attribute Macro]
(has-axis 'trans_transform) [SKETCH Attribute Macro]
(has-angle 'trans_transform) [SKETCH Attribute Macro]
(has-displacement 'trans_transform) [SKETCH Attribute Macro]
(is-linear 'trans_transform) [SKETCH Attribute Macro]
(is-affine 'trans_transform) [SKETCH Attribute Macro]
(is-orthogonal 'trans_transform) [SKETCH Attribute Macro]
(has-input-dimension 'trans_transform) [SKETCH Attribute Macro]
(has-output-dimension 'trans_transform) [SKETCH Attribute Macro]

"linear transform" [SKETCH Term]
"projective transform" [SKETCH Term]
"orthogonal transform" [SKETCH Term]
"affine transform" [SKETCH Term]

sag_transform [C Type]
SAG_TRANSFORM [C Global Variable]
trans_transform->**sag_ttype** [C Structure Element]
trans_transform->**sag_tlinear** [C Structure Element]
trans_transform->**sag_taffine** [C Structure Element]
trans_transform->**sag_torthogonal** [C Structure Element]
trans_transform->**sag_tidimension** [C Structure Element]
trans_transform->**sag_todimension** [C Structure Element]
trans_transform->**sag_txx** [C Structure Element]
trans_transform->**sag_txy** [C Structure Element]
trans_transform->**sag_txz** [C Structure Element]
trans_transform->**sag_txt** [C Structure Element]
trans_transform->**sag_tyx** [C Structure Element]
trans_transform->**sag_tyy** [C Structure Element]
trans_transform->**sag_tyz** [C Structure Element]
trans_transform->**sag_tyt** [C Structure Element]
trans_transform->**sag_tzx** [C Structure Element]
trans_transform->**sag_tzy** [C Structure Element]
trans_transform->**sag_tzz** [C Structure Element]
trans_transform->**sag_tzt** [C Structure Element]
trans_transform->**sag_ttx** [C Structure Element]
trans_transform->**sag_tty** [C Structure Element]
trans_transform->**sag_ttz** [C Structure Element]
trans_transform->**sag_ttt** [C Structure Element]

WHERE: Vec_axis, if given, must be a unit vector with *has-length equal* to 1.0.

USE: *A-transform* object which is a linear, affine, or projective transformation of points. The points are represented by *a-vector* objects.

The *has-xx*, *has-xy*, ..., *has-tt* attributes are called the coordinates of the transformation. The kj'th coordinate of the transform multiplies the k'th coordinate of the input point to produce a term in the j'th coordinate of the output point. The coordinates are converted to *flonum*'s. Missing coordinates are set to *nil*.

For one dimensional transforms, coordinates involving Y or Z are *nil*. For two dimensional transforms, coordinates involving Z are *nil*. For a transform from 3 dimensional space to 2 dimensional space, the XZ, YZ, and ZZ coordinates are *nil*, but the ZX and ZY coordinates are not. And so forth.

The T dimension is used for projective transforms. Points in N dimensions are extended to N+1 dimensions by adding a T coordinate equal to 1. Then the N+1 dimensional point is transformed, the coordinates of the result are divided by the T coordinate of the result, and the T coordinate of the result is removed.

A linear transformation is a non-projective transformation: one not involving the T coordinate. If a transformation has each coordinate involving T equal to *nil* or to either 0.0 for non TT coordinates or to 1.0 for the TT coordinate, then the transformation is deemed to be linear and all coordinates involving T are set to *nil*.

An affine transformation is a projective transformation which never changes the T coordinate of a point (before division). Such is equivalent to a linear transformation followed by adding a vector, called the displacement vector of the affine transformation (vec_displacement, actually).

If a non-linear transform has its XT, YT, and ZT coordinates equal to *nil* or 0.0, and its TT coordinate equal to *nil* or 1.0, the transformation is deemed to be affine, the TT coordinate is set to 1.0, and any of the XT, YT, and ZT coordinates for which X, Y, or Z is an input dimension are set to 0.0.

A transform is orthogonal if it preserves distances and orientations. The later means that no reflections are involved in the transformation. An orthogonal transform may be linear or affine; but if it is not linear, it must be affine.

G_orthogonal is non-*nil* if the transform was computed in a way that made it orthogonal. If g_orthogonal is *nil*, the transform may or may not be orthogonal by accident.

The *has-displacement* attribute gives an alternate representation of the TX, TY, and TZ coordinates of an affine transformation.

The *has-axis* and *has-angle* attributes give an alternate representation of the XX, XY, XZ, YX, YY, YZ, ZX, ZY, and ZZ coordinates of an orthogonal 3D transform. Vec_axis must be a unit vector with *has-length equal* 1.0, provided n_angle is not 0.0. If n_angle is 0.0, vec_axis will be set to *nil*, even if a non-*nil* value is provided. G_orthogonal is set to *t* if n_angle is not *nil*.

IS-LINEAR:

IS-AFFINE:

IS-ORTHOGONAL:

SAG_TLINEAR:

SAG_TAFFINE:

SAG_TORTHOGONAL: Attributes which specify the kind of transformation. See USE above. In LISP these are *nil* or *t*; in C they are 0 or 1.

HAS-XX ...:

SAG_TXX ...: The coordinates of the transformation. See USE above. The LISP versions are *flonum*'s (other LISP numbers will be converted to *flonum*'s when stored). The C versions are *float*'s.

HAS-INPUT-DIMENSIONS:

HAS-OUTPUT-DIMENSIONS:

SAG_TIDIMENSIONS:

SAG_TODIMENSIONS: The dimension of the points input to or output from the transformation. 0, 1, 2, or 3. The LISP versions are *fixnum*'s; the C versions are *int*'s.

Normally the dimensions are computed automatically. However, the input dimension cannot be computed automatically when the output dimension is 0. Also, when a transform is created from a prototype and the dimensions are changed, the new dimensions have to be given explicitly.

HAS-INVERSE: (*has-inverse* trans_transform) is a *a-transform* that is the inverse of trans_transform. This attribute of trans_transform is computed the first time it is needed. For non-affine orthogonal transforms it is just the transpose.

If the transform is not invertible, this attribute is *nil* when computed. However, the fact that it has been computed and found to be *nil* is remembered, so it will not be computed again.

If the transform is invertible,

(*has-inverse* (*has-inverse* trans_transform))

is identical to trans_transform.

The inverse of an affine transform is affine. The inverse of an orthogonal transform is orthogonal.

The inverse of a projective non-affine transform is only determined up to a multiplier. The multiplier is set so the TT coordinate of the inverse is 1.0, if this is reasonable [check].

HAS-DETERMINANT: (*has-determinant* trans_transform) is a *flonum* which is the determinant of the part of trans_transform that does not involve the T coordinate. It is undefined (*nil*) for projective non-affine transformations.

It is computed the first time it is needed. For orthogonal transforms it is *equal* 1.0.

HAS-AXIS: *A-vector* which is the axis of rotation of an orthogonal point transformation. It is computed the first time it is needed. It is *nil* for non-orthogonal transforms and for the unit orthogonal transform (the one with **0.0** rotation angle).

HAS-ANGLE: A *flonum* which is the angle of rotation in radians of an orthogonal point transformation, about its axis. It is computed the first time it is needed. It is 0.0 for the unit transform, and *nil* for non-orthogonal transforms.

The axis and angle of rotation are always chosen so that the angle of rotation is always $>= -\frac{\pi}{2}$ and $< \frac{\pi}{2}$.

HAS-DISPLACEMENT: The displacement part of an affine transformation. *Nil* for non-affine transformations.

SAG_TRANSFORM:

SAG_TTYPE: A LISP *a-transform* object is a C *sag_transform* structure. *Sag_ttype* is first element of a *sag_transform* structure in C. It equals SAG_TRANSFORM, which in turn equals—

sob_nobject ("a-transform").

COMPUTING TRANSFORMS IN C:

SAG_VADJUST: Transforms which are empty, that is have all attributes missing, can be created in C by code such as—

sag_talloc (tr, 1)

...

sob_vinit ((*sat_lvalue*) tr, *SAG_TRANSFORM*)

or

register sag_transform tr = (*sob_transform*) *sob_vcreate* (*SAG_TRANSFORM*);

Note that in the first case the transform is in the stack, and cannot not have any *sag_taxis*, *sag_tdisplacement*, or *sag_tinverse* elements. Serious problems can arise if one attempts to compute these elements for a stack transform.

Note that in the second case the transform is in the heap and must be protected from garbage collection before any more heap allocations are done.

After an empty transform has been created, it may be completely defined by setting its non-missing *sag_txx*, *sag_txy*, ... *sag_ttt* coordinates, its *sag_tidimension* and *sag_todimension* dimension sizes, and its *sag_torthogonal*, *sag_tlinear*, and *sag_taffine* flags. The dimensions and flags are best set by calling—

sag_tsdimensions (tr)

after setting the coordinates.

The *sag_orthogonal* flag must be set before calling *sag_tsdimensions*, if it applies to the xx, ..., zz coordinates. *Sag_tsdimensions* will clear this flag for projective transforms, but leaves it untouched for other kinds of transformations.

The tx, ty, and tz coordinates can be set by calling—

sag_tsdisplacement (tr, vec_displacement)

and the xx, xy, xz, yx, yy, yz, zx, zy, zz coordinates can be set by calling—

sag_tsangle (tr, f_angle, vec_axis).

This last function also sets *sag_torthogonal*. Vec_axis may be *sat_nil* if f_angle is zero.

There are a variety of other functions for setting empty transforms: see *sag_tcompose*, *sag_tsum*, *sag_tdifference*, *sag_tpscalar*, *sag_tdiagonal*, etc.

WARNING ABOUT COMPARE OBJECT: Because floating point numbers are not exact, outputting *a-transform* into a catalog and reading it back will not read back *a-transform* that is exactly the same as the original. *Compare-object* will generally consider the transforms to be different.

(**a-vector** [*has-x* 'n_x] [*has-y* 'n_y [*has-z* 'n_z]] [SKETCH Type Macro]

[*has-length* 'n_length])

(**has-x** 'vec_vector) [SKETCH Attribute Macro]
(**has-y** 'vec_vector) [SKETCH Attribute Macro]
(**has-z** 'vec_vector) [SKETCH Attribute Macro]
(**has-length** 'vec_vector) [SKETCH Attribute Macro]
(**has-dimension** 'vec_vector) [SKETCH Attribute Macro]

a-vector [SKETCH Type Object]
vec_ [Argument Prefix]
pt_ [Argument Prefix]

sag_vector [C Type]
SAG_VECTOR [C Global Variable]
vec_vector->**sag_vtype** [C Structure Element]
vec_vector->**sag_vx** [C Structure Element]
vec_vector->**sag_vy** [C Structure Element]
vec_vector->**sag_vz** [C Structure Element]
sag_vlength (vec_vector) [C Macro]
sag_vadjust (vec_vector, f_length) [C Function]

ARGUMENT PREFIXES: *Vec_* and *pt_* both denote *a-vector* objects. The former is used to emphasize that the object represents a displacement vector, and the later to emphasize that the object represents a point in space.

USE: *A-vector* object represents a relative motion in space. It may also be used to represent a point in space by specifying the displacement of the point from some origin.

The X, Y, and Z coordinates are stored as *flonum*'s. Missing y and z coordinates are set to *nil*. A 1-dimensional point or vector has *nil* y and z coordinates, while a 2-dimensional point or vector has a *nil* z coordinate.

If n_length is given, the vector coordinates are scaled so the vector is of the given length. A unit vector can be made by specifying n_length as being 1.0.

HAS-X, ...:

SAG_VX, ...: The coordinates of the point. See USE above. The C versions are *float*'s.

HAS-LENGTH:

SAG_VLENGTH: A *flonum* which is the length of vec_vector. It is an attribute of vec_vector which is computed the first time it is needed. For unit length vectors it is *equal* 1.0.

The *has-length* attribute of *a-vector* is not printed when the vector is printed unless it equals 1.0, signifying a unit vector. Similarly it is not returned by *uneval-object* unless it is 1.0. Lastly, it does not participate in *compare-object* comparisons of vectors.

The C form, *sag_vlength* (vec_vector), is a macro so it can check whether the length attribute has been computed yet, and call a function to compute it if not. This macro returns a *float*.

SAG_VECTOR:

SAG_VTYPE: A LISP *a-vector* object is a C *sag_vector* structure. *Sag_vtype* is first element of a *sag_vector* structure in C. It equals SAG_VECTOR, which in turn equals—

sob_nobject ("a-vector").

COMPUTING VECTORS IN C:

SAG_VADJUST: Vectors which are empty, that is have all attributes missing, can be created in C by code such as—

sag_valloc (v, 1)

...

sob_vinit ((*sat_lvalue*) v, *SAG_VECTOR*)

or

register sag_vector v = (*sob_vector*) *sob_vcreate* (*SAG_VECTOR*);

Note that in the second case the vector is in the heap and must be protected from garbage collection before any more heap allocations are done. Also note that the 'v' in 'sob_v' refers to *a-lisp-vector* and not *a-vector*.

After an empty vector has been created, it may be completely defined by setting its non-missing *sag_vx*, *sag_vy*, and *sag_vz* coordinates. The rule that *sag_vx* may not be missing if *sag_vy* is not missing, and neither *sag_vx* nor *sag_vy* may be missing if *sag_vz* is not missing, must be obeyed. No other attributes of the vector need be set.

The length of a vector may be set or changed after the vector coordinates have been set by calling—

sag_vadjust (vec_vector, f_length).

The *sag_vx*, *sag_vy*, and *sag_vz* coordinates of the vector are adjusted by this call to make the vector length equal the desired length, if necessary.

There are a variety of other functions for setting empty vectors: see *sag_tpoint*, *sag_tvector*, *sag_tcovector*, *sag_vsum*, *sag_vdifference*, *sag_vmove*, *sag_vpscalar*, *sag_vpbetween*, *sag_vunit*, etc.

WARNING ABOUT COMPARE_OBJECT: Because floating point numbers are not exact, outputting *a-vector* into a catalog and reading it back will not read back *a-vector* that is exactly the same as the original. *Compare-object* will generally consider the vectors to be different.

(**center-of-gravity-of-cluster** 'cl_cluster) [LISP Function]

RETURNS: A point equal to the center of gravity of the points in the cluster (with equal weighting of points). Returns *nil* if the cluster is empty.

(**compose-transforms** 'trans_transform-1 'trans_transform-2) [LISP Macro]

RETURNS: The composition of the two transforms. That is, the transform which moves a point to the same place that it would be moved by first applying trans_transform-1 and then trans_transform-2.

(**distance-between-lines** 'lin_line-1 'lin_line-2) [LISP Macro]

RETURNS: The *flonum* distance from lin_line-1 to the lin_line-2. If both lines are infinite, nearly coplanar, and nearly parallel, the distance between the lines is defined to be the distance between their *has-start* points.

(**distance-between-point-and-line** 'pt_point 'lin_line) [LISP Macro]

RETURNS: The *flonum* distance between the point and the line.

(**distance-between-points** 'pt_p1 'pt_p2) [LISP Macro]

RETURNS: The *flonum* distance between points (represented by vectors). Both points must have the same dimension.

(**linearize-transform** 'trans_transform) [LISP Macro]

RETURNS: A linear transform whose X, Y, and Z coordinates equal those of trans_transform.

(**lines-are-parallel** lin_line_1, lin_line_2) [LISP Macro]

RETURNS: The symbol *t* if neither line is zero length and the directions of the lines are parallel according to *vectors-are-parallel*. Otherwise returns *nil*.

(**move-vector** 'vec_vector 'n_x ['n_y ['n_z]]) [LISP Macro]

RETURNS: *A-vector* equal to vec_vector with n_x added to its x coordinate, n_y to its y coordinate, and n_z to its z coordinate. There may be fewer n_x, n_y, and n_z arguments than the dimension of vec_vector, with the omitted arguments being treated as zero.

(**point-between-points** 'pt_point-1 'pt_point-2 'n_scalar) [LISP Macro]

WHERE: Both points must have the same dimension.

RETURNS: *Point-between-points* returns *a-vector* equal to—

n_scalar * pt_point-1 + (1 - n_scalar) * pt_point-2.

(**product-of-scalar-and-transform** 'n_scalar 'trans_transform) [LISP Macro]

RETURNS: *A-transform* equal to the scalar product of n_scalar and trans_transform.

(**product-of-scalar-and-vector** 'n_scalar 'vec_vector) [LISP Macro]

RETURNS: *A-vector* equal to the scalar product of n_scalar and vec_vector.

sag_langle (lin_line_1 lin_line_2) [C Function]
sag_lpdistance (vec_point lin_line) [C Function]
sag_ldistance (lin_line_1 lin_line_2) [C Function]

RETURNS: A floating point number equal to—

sag_langle The angle in radians between the directions of the lin_line_1 and lin_line2. Or *SAT_DMISSING* if either line is of zero length.

sag_lpdistance The distance from vec_point to lin_line.

sag_ldistance The distance between the lines. If the lines are both infinite and are parallel according to *sag_lparallel*, then this is the distance between their *has-start* points.

sag_lparallel (lin_line_1, lin_line_2) [C Function]

RETURNS: 1 if neither line is zero length and the directions of the lines are parallel according to *sag_vparallel*. Otherwise returns 0.

sag_tpoint (pt_result, pt_point, trans_transform) [C Function]
sag_tvector (vec_result, vec_vector, trans_transform) [C Function]
sag_tcovector (vec_result, trans_transform, vec_covector) [C Function]

SIDE EFFECT: The empty vector vec_result (or pt_result) is set to the point, vector, or covector transformed by trans_transform.

sag_tsproduct (trans_transform_1 trans_transform_2) [C Function]

RETURNS: A floating point number equal to the scalar product of trans_transform_1 and trans_transform_2. This is the sum of the products of corresponding elements, as in the scalar product of two vectors. If both transforms are linear, the tt components are *not* included in the sum.

sag_tsum (trans_transform_1, trans_transform_2, trans_transform_3) [C Function]
sag_tdifference (trans_transform_1, trans_transform_2, trans_transform_3) [C Function]
sag_tpscalar (trans_transform_1, f_scalar, trans_transform_2) [C Function]
sag_tdiagonal (trans_transform_1, x_dimension, f_scalar) [C Function]

SIDE EFFECT: The empty transform trans_transform_1 is set as follows—

sag_tsum The sum of trans_transform_2 and trans_transform_3.

sag_tdifference The difference trans_transform_2 minus trans_transform_3.

sag_tpscalar The product of f_scalar and trans_transform_2.

sag_tdiagonal The x_dimensional transform with the value f_scalar for all diagonal elements and zeros everywhere else.

The output transform can be one of the input transforms for any of these functions. If there is an error in any of these functions, the output transform may be set to an inconsistent state.

sag_vdimension (vec_vector) [C Function]

RETURNS: The dimension of vec_vector as an integer (0, 1, 2, or 3).

sag_vparallel (vec_vector1, vec_vector2) [C Function]

RETURNS: 1 if vec_vector1 and vec_vector2 both have non-zero length and they are parallel in the sense that the sin of the angle between them has absolute value less than 0.001. Otherwise returns 0.

The number 0.001 is choosen because if the sin of the angle between the vectors is X, the error of the computed unit vector that is perpendicular to the two vectors may have a norm as large as 10**-6/X. The 10**-6 arrises because the coordinates of vectors are stored in single precision floating point.

Thus if we quantize both the domain of angles and the domain of unit vectors, we get—

(error in angle domain) * (error in unit vector domain) == 10**-6.

So 0.001 splits the difference between the two domains.

sag_vsproduct (vec_vector_1 vec_vector_2) [C Function]
sag_vangle (vec_vector_1 vec_vector_2) [C Function]
sag_vdistance (pt_point_1 pt_point_2) [C Function]

RETURNS: A floating point number equal to—

sag_vsproduct The scalar product of vec_vector_1 and vec_vector_2.

sag_vangle The angle between vec_vector_1 and vec_vector_2, in radians, in the range $[0, \pi]$; or the value *SAT_DMISSING* if one of the vectors is of zero length.

In order to get accuracy, two different methods of computation are used: one for the case where the vectors are nearly parallel, and one for the case where the vectors are nearly perpendicular.

sag_vdistance The distance between pt_point_1 and pt_point_2 (length of their difference).

sag_vsum (vec_vector_1, vec_vector_2, vec_vector_3) [C Function]
sag_vdifference (vec_vector_1, vec_vector_2, vec_vector_3) [C Function]
sag_vpscalar (vec_vector_1, f_scalar, vec_vector_2) [C Function]
sag_vvproduct (vec_vector_1, vec_vector_2, vec_vector_3) [C Function]
sag_vpbetween (pt_point_1, pt_point_2, pt_point_3, f_scalar) [C Function]
sag_vunit (vec_vector, x_unit_dimension, x_total_dimension) [C Function]

SIDE EFFECT: The empty vector vec_vector_1 (or pt_point_1) is set as follows—

sag_vsum The sum of vec_vector_2 and vec_vector_3.

sag_vdifference The difference vec_vector_2 minus vec_vector_3.

sag_vpscalar The product of f_scalar and vec_vector_2.

sag_vvproduct The vector product of vec_vector_2 and vec_vector_3.

sag_vpbetween The sum of f_scalar times pt_point_2 and (1.0 − f_scalar) times pt_point_3.

sag_vunit The unit vector with dimension x_total_dimension in the in the x_unit_dimension direction (0 for X, 1 for Y, 2 for Z).

The output vector (or point) can be one of the input vectors for any of these functions (even *sag_vvector*). If there is an error in any of these functions, the output vector may be set to an inconsistent state.

(scalar-product-of-transforms 'trans_transform-1 'trans_transform-2) [LISP Macro]

WHERE: Both transforms must have the same dimensions.

RETURNS: A *flonum* equal to the scalar product of trans_transform-1 and trans_transform-2. The scalar product is the sum of the products of components, as for a vector. If both transforms are linear, the tt components are *not* included in the sum.

(scalar-product-of-vectors 'vec_vector-1 'vec_vector-2) [LISP Macro]

WHERE: Both vectors must have the same dimension.

RETURNS: A *flonum* equal to the scalar product of vec_vector-1 and vec_vector-2.

(sum-of-transforms 'trans_t1 'trans_t2) [LISP Macro]
(difference-of-transforms 'trans_t1 'trans_t2) [LISP Macro]

WHERE: Both transforms must have the same dimensions.

RETURNS: *A-transform* equal to the sum of trans_t1 and trans_t2, or the sum of trans_t1 and minus trans_t2.

(sum-of-vectors 'vec_v1 'vec_v2) [LISP Macro]
(difference-of-vectors 'vec_v1 'vec_v2) [LISP Macro]

WHERE: Both vectors must have the same dimension.

RETURNS: *A-vector* equal to the sum of vec_v1 and vec_v2, or the sum of vec_v1 and minus vec_v2.

(transform-line 'lin_line 'trans_transform) [LISP Function]

RETURNS: *A-line* equal to lin_line transformed into the coordinate system obtained by transforming points by trans_transform (see *transform-point*).

This is just a matter of transforming the start, end, and direction of lin_line by trans_transform.

(transform-point 'pt_point 'trans_transform) [LISP Macro]
(transform-vector 'vec_vector 'trans_transform) [LISP Macro]
(transform-covector 'trans_transform 'vec_vector) [LISP Macro]

RETURNS: *A-vector* equal to the pt_point or vec_vector transformed by trans_transform.

Transform-vector ignores the T coordinate, using only the XX, XY, YZ, YX, YY, YZ, ZX, ZY, and ZZ coordinates of trans_transform. This has the effect of transforming a displacement vector, rather than a point. It will not work if trans_transform is projective but not affine.

Transform-covector is like *transform-vector* but transforms the vector by the transpose of the transformation. In general—

(scalar-product-of-vectors (transform-vector v T) w)
=
(scalar-product-of-vectors v (transform-covector T w)).

NOTE: The argument order is determined by thinking of vectors as row vectors, and covectors as column vectors. The ik'th element of a transform corresponds to row i and column k.

(**transpose-transform** 'trans_transform) [LISP Macro]

RETURNS: The transpose of the linear transform trans_transform.

(**vector-product-of-vectors** 'vec_vector-1 'vec_vector-2) [LISP Macro]

WHERE: The vector arguments must be 3D.

RETURNS: A-*vector* equal to the vector product of vec_vector-1 and vec_vector-2.

(**vectors-are-parallel** 'vec_vector-1 'vec_vector-2) [LISP Macro]

WHERE: Both vectors must have the same dimension.

RETURNS: The symbol *t* if vec_vector1 and vec_vector2 both have non-zero length and they are parallel in the sense that the sin of the angle between them has absolute value less than 0.001. Otherwise returns 0.

The number 0.001 is chosen because if the sin of the angle between the vectors is X, the error of the computed unit vector that is perpendicular to the two vectors may have a norm as large as 10**-6/X. The 10**-6 arises because the coordinates of vectors are stored in single precision floating point.

Thus if we quantize both the domain of angles and the domain of unit vectors, we get—

(error in angle domain) * (error in unit vector domain) == 10**-6.

So 0.001 splits the difference between the two domains.

(**zero-transform** 'x_dimension) [LISP Macro]
(**unit-transform** 'x_dimension) [LISP Macro]

RETURNS: A zero or unit transform whose dimension is x_dimension.

(**zero-vector** 'x_total-dimension) [LISP Macro]
(**unit-vector** 'x_unit-dimension 'x_total-dimension) [LISP Macro]

RETURNS: A vector whose dimension is x_total-dimension. The vector components are all zero for *zero-vector*, and are all zero except for the x_unit-dimension'th component, which is 1.0, for *unit-vector*, where x_unit-dimension is 0, 1, or 2 to denote the *X-dimension*, *Y-dimension*, or *Z-dimension*.

CHAPTER 11

DISPLAY

1. USING DISPLAYS. A display is an array of pixels. For example, the SUN low resolution frame buffer is an array of 640X475 pixels: 640 pixels wide and 475 pixels high.

A display has an intensity array with stores for each pixel a code for a color and an intensity. Usually this is an 8-bit code with 256 possible values. One value encodes black. 127 values encode 127 intensity levels of white, from dark gray to bright white. 16 levels encode 16 intensity levels magneta (purple), from dark magenta to bright magneta. There are also 16 intensity levels for each of 7 other colors: red, brown, yellow, cyan, green, turquoise, and blue.

Just to start out, the commands—

```
-> (clear-display)
-> (flush-display)
```

will clear the display. Most display functions just act on an in-computer-memory copy of the display. *Flush-display* is required to move this out to the display proper.

You can write an image into the current display with the *display-image* function. For example—

```
-> (display-image (an-array has-sizes '(64 64)
                            by-expression '(+ X (* 64 Y))))
-> (flush-display)
```

will display the array as a black-and-white image. As an alternative—

```
-> (display-image (an-array has-sizes '(8 16)
                            by-expression '(+ X (* 8 Y)))
                  '(100 200) has-zooms '(20 10)
                  has-bounds '(-0.5 pseudocolor 127.5))
-> (flush-display)
```

will display an image so that its upper left point has coordinates (100 200) (100 pixels horizontally from the left and 200 vertical down from the top), will make each pixel 20 times as larger horizontally and 10 times as large vertically (the *has-zooms*), and will display the image using a pseudocolor scale instead of a gray scale. In the pseudocolor scale there are only 16 intensities, but each is modulated by 8 colors, in order to permit small differences of intensity to show up. According to the *has-bounds* argument, the array element value range from -0.5 through 127.5 is divided into 128 equal intervals each mapped onto a different intensity code by the *pseudocolor* scale. See *display-image* in the GLOSSARY for details.

Instead of flushing the display to the display device, you can write the display into a catalog, read it back, and display it by commands such as—

—> (*setq* c (*a-catalog has-file* 'foo))
—> (*write-display* c)
—> (*close-display* c)

—> (*playback-display* (*read-catalog* c))

See *write-display* and *playback-display* in the GLOSSARY for more details.

A display can also have bitgraph planes. Each bitgraph plane has one bit per pixel. The bitgraph planes are each associated with an intensity array code value that specifies a color and an intensity. When a bitgraph plane pixel bit has the value '1', the plane's code value replaces, or overlays, the code value specified for the pixel by the display's intensity array. But if the bit has the value '0', there is no overlay, and the intensity array's code value is used.

If there are several bitgraph planes, they have a priority ordering, with the higher priority planes overlaying the lower priority ones. It is standard for a display to have the following nine bitgraph planes—

blue turquoise cyan green yellow brown red magenta white.

Each of these bitgraph planes is displayed as the brightest intensity of the color which names the plane. The priority of these planes is from lowest to highest in the above list: *white* overlays everything, while *blue* is overlaid by all other planes.

Text can be displayed by commands such as the following—

—> (*clear-display*)

—> (*display-patom* '|Hello There| '(20 300) '*right-rotate* '*turquoise*)

—> (*display-text* (20 '(300 50) 2.0 '*left* '*cyan*)
    (*patom* '|This is a list-|)
    (*terpri*)
    (*pretty-print* '((Feb 1 1987) (file 1) (image 1))))
—> (*flush-display*)

Display-patom displays the text 'Hello There' centered on the coordinates (20 300). This text is rotated 90 degrees to the right, and displayed in the *turquoise* plane. *Display-text* above displays the text written by the *patom*, *terpri*, and *pretty-print* statements within the body of *display-text*. The *pretty-print* assumes a line length of 20. The text is located in the *cyan* plane at the coordinates (300 50): it is vertically centered on these coordinates, but horizontally it is left adjusted so these coordinates appear just to the left of the text. The character size is 2.0 times the normal size. See *display-text* in the GLOSSARY for more details.

Lines can be displayed by commands such as—

```
—> (display-lines 'yellow '(300 0) '(500 0) '(500 100) '(300 150) '(300 0))

—> (setq m (quotient pi 180.0))
—> (setq circle (an-array has-sizes '(2 361)
                          by-expression '(if (equal X 0)
                                             (sin (product m Y))
                                             (cos (product m Y)))))

—> (display-lines (new-window '(95 295) has-zooms '(50 50)
                          has-sizes '(2.2 2.2) has-cursor '(0.6 0.6))
        'green 3.0 circle)
—> (flush-display)
```

The first *display-lines* command draws 4 straight lines between the successive points, forming a box in the *yellow* plane. The second *display-lines* command draws straight lines between successive points defined by a 2×361 array. The array has been defined so that these points will outline a unit circle. To position and size this circle, a window on the display has been created by the *new-window* function. The window has apparent size 2.2×2.2, but the zooms are both 50, so the window is 110×110 display pixels in size. The window has a cursor which is placed at the center of the window, because the circle will be centered on the cursor (the circle coordinates range from -1 through +1). The reason the center of the window is at coordinates (0.6 0.6) instead of (1.1 1.1), is that the origin (0 0) is in the center of the upper left pixel of the window, so that the upper left corner of the window has coordinates (-0.5 -0.5). The window must be made a little larger than the circle to ensure that no lines of the circle are partly omitted, as they would be if any part of them extended beyond the window.

The second *display-lines* command draws lines that are three display pixels wide, as indicated by the 3.0. The first *display-lines* command draws lines that are the default of 1.0 display pixels wide. The widths of lines and sizes of text do not depend on the zooms, unlike the sizes of pixels drawn with *display-image*.

The circle can be by using *a-transform* instead of a window. The commands—

```
—> (display-lines (a-transform has-xx 30 has-xy 0
                               has-yx 0 has-yy 30
                               has-displacement (a-vector has-x 150 has-y 350))
                  'brown 2.0 circle)
—> (flush-display)
```

draw a concentric circle 3/5'ths as large as the previous circle (radius 30 instead of 50) with a line width of 2.0 pixels instead of 3.0 pixels and the color *brown* instead of *green*.

See *display-lines* in the GLOSSARY for more details.

2. WINDOWS.

3. MAKING DISPLAYS.

4. GLOSSARY.

(a-display [*has-sizes* '(x_xsize x_ysize)]
[*has-map* 's_map-name]
[*has-device* '(s_device-type ...)]
[*has-film* 's_film]
[*has-parent* 'dwin_window]
[*has-bitgraph-planes* '(s_plane-type-name-1 ...)]
[*has-intensity-array* 'ucar/usar/s_intensity-array]
[*has-bitgraph-array* 'ubar/s_bitgraph-array]
[*has-bitgraph-programs* 'h/s_bitgraph-programs]) — [SKETCH Type Macro]

dis_ — [Argument Prefix]
a-display — [SKETCH Type Object]

"Bitgraph Plane Name" — [SKETCH Term]
bpn_ — [Argument Prefix]

(**has-sizes** 'dis_display) — [SKETCH Attribute Macro]
(**has-map** 'dis_display) — [SKETCH Attribute Macro]
(**has-device** 'dis_display) — [SKETCH Attribute Macro]
(**has-film** 'dis_display) — [SKETCH Attribute Macro]
(**has-parent** 'dis_display) — [SKETCH Attribute Macro]
(**has-intensity-array** 'dis_display) — [SKETCH Attribute Macro]
(**has-bitgraph-planes** 'dis_display) — [SKETCH Attribute Macro]
(**has-bitgraph-array** 'dis_display) — [SKETCH Attribute Macro]
(**has-bitgraph-programs** 'dis_display) — [SKETCH Attribute Macro]
(**has-range** 'dis_display) — [SKETCH Attribute Macro]
(**has-primary-colors** 'dis_display) — [SKETCH Attribute Macro]
(**has-colors** 'dis_display) — [SKETCH Attribute Macro]
(**has-scales** 'dis_display) — [SKETCH Attribute Macro]
(**has-plane-types** 'dis_display) — [SKETCH Attribute Macro]

USE: *A-display* is a representation of a black-and-white or color image. The representation has three parts. First, there is an intensity array, which stores a gray or color scale intensity image. Second, there is an ordered set of bitgraph programs, each of which is a list of vectors, text, and fill areas that can be used to draw a binary image which is called a bitgraph plane. Third, there is a bitgraph array, which directly stores all the bitgraph planes as binary images.

Any of the three parts may be omitted.

A-display may be just a means of storing information, or it may be an actual output device display. The latter has a non-*nil has-device* attribute that specifies the nature of the output device. An output display can also store information in any way a non-output display can, except that an output display cannot store information its hardware cannot output.

Writing information into a display stores the information inside the computer.

For an output display, the copy of this information stored in the computer is usually not the same as the copy stored in the output device. To move the latest copy from the computer to the output device, an additional operation, *flush-display*, must be performed on the output display. This operation ensures that the output device holds the same information as the computer.

A-display may be copied into part of a larger display by the *merge-display* function. If the display has a non-*nil has-parent* attribute, then an operation such as *flush-display* that outputs the display will merge it into the parent and flush the parent.

The X dimension of a display is horizontal with X increasing from left to right. The Y dimension is vertical with Y increasing from top to bottom. The coordinates of display pixels and the sizes of display dimensions are integers (unlike display windows where they are floating point).

A display has X and Y dimension sizes given by x_xsize and x_ysize. Both intensity and bitgraph arrays have these same X and Y dimension sizes, and the information stored in array elements with particular X and Y coordinates determines what is displayed in the pixel at corresponding horizontal and vertical coordinates on a display screen.

The *has-map* attribute of a display is used to look up *a-display-map* (any one of several with the same primary name will do), and the *has-range*, *has-primary-colors*, *has-colors*, *has-scales*, and *has-plane-types* attributes of the display are inherited from this display map. See *a-display-map* for the meanings of these attributes.

If the *has-device* attribute is given, it implies that only certain *has-sizes* and *has-map* values are legal. In fact, neither of these attributes need be given in this case, for both will receive default values implied by the *has-device* value. Similarly the *has-parent* attribute can imply these and other display attribute values.

If the *has-primary-colors* attribute is not *nil*, its length is the Z dimension size of the intensity array, and its elments are the colors cooresponding to the different Z coordinates (0, 1, 2, ... in that order). But if the *has-primary-colors* attribute is *nil*, the intensity array has Z dimension size 1, and the array stores codes that are mapped by some display map onto different colors. In this case the map from codes to colors is determined from the software view by *has-map*, and from the hardware view by *has-map*, *has-device*, and, when a camera is being used, by *has-film*.

The *has-bitgraph-planes* attribute specifies a bitgraph plane type name for each bitgraph plane. These type names are looked up in the *has-plane-types* attribute to give *a-plane-type* object for each bitgraph plane. This specifies the color, line type (*solid*, *long-dashed*, etc.), character font (*display*, *times-roman*, *times-bold*, etc.), area fill pattern, normal line width, and normal character size for the plane.

The bitgraph array stores one plane for each of its Z coordinates. Bitgraph planes are named by the Z coordinates they would have were there a bitgraph array. The elements of the *has-bitgraph-planes* list are for the planes with Z coordinates 0, 1, 2, ..., in that order. The number of elements in this list equals the Z dimension size of the bitgraph array.

Bitgraph planes with higher Z coordinates overlay those with lower Z coordinates.

BITGRAPH PLANE NAMES:

BPN_: The name of a bitgraph plane in *a-display* can be its Z coordinate, which is an integer. Or the name can be a pair, (type-name M), if the plane is the M'th plane in the *has-bitgraph-planes* list that has the given type name. Lastly, the name can be a type name by itself. to name the first plane with the given type name in the *has-bitgraph-planes* list (i.e. M = 1 implied).

MERGING DISPLAYS: A display used to store information is often merged into a display used to output information. This is easy and obvious when both displays have the same resolution, *has-map*, and and *has-bitgraph-planes* attributes, but can become difficult or impossible in other cases.

Intensity arrays can be scaled to output devices of different resolution, but usually the results will look nice only when the scale factors are integers or the resolution of the output display is very much higher than that of the input display.

Bitgraph arrays simply cannot be scaled to different resolutions.

Bitgraph programs are easily scaled to any resolution. If bitgraph planes are to be scaled later, they should be stored in bitgraph program form, even if they are also stored (redundantly) in bitgraph array form.

The *has-scales* attribute of a display is used when images is stored in the intensity array, and cannot meaningfully be changed when the intensity array is merged later into another display.

Similarly most of the *has-plane-types* attribute information is used when images is stored in a bitgraph array, and cannot meaningfully be changed when the bitgraph array is merged later into another display. The one exception is the plane color, which can be changed (see *a-plane-type*).

In fact, the colors of planes can be changed when they are merged later into another display, as long as the planes are stored in bitgraph array or bitgraph program form.

Any attribute inherited from the *has-map* display map can be changed when a bitgraph program is merged later into another display. This includes all the *has-plane-types* information. Thus bitgraph programs are very flexible.

A-display objects can be stored on disk and played back: i.e., read and merged into an output display. In general, storing information solely in an intensity array is very device dependent, but provides the fastest playback. Storing as much information as possible in bitgraph programs maximizes device independence, but slows playback. Bitgraph arrays are an intermediate step between bitgraph programs and intensity arrays, and their use provides some flexibility (colors can be changed and planes selected or omitted, but resolution cannot be changed) with some playback speed loss.

HAS-SIZES: The horizontal, or X coordinate, size in pixels, and the vertical, or Y coordinate. size in pixels.

The *has-sizes* attribute can be implied by the *has-device* or *has-parent* attribute. or can be directly specified, but must not end up being *nil*.

HAS-MAP: This attribute provides the primary part of the name of *a-display-map*, which in turn provides the *has-range*, *has-primary-colors*, *has-colors*, *has-scales*, and *has-plane-types* attributes.

The *has-map* attribute can be implied by the *has-device* or *has-parent* attribute, or can be directly specified, but must not end up being *nil*.

The most common values for this attribute are *standard* for displays with intensity arrays, and *standard-bitgraph* for arrays with no intensity arrays.

Displays with the *standard has-map* value store an 8-bit code for each pixel in an intensity array that has a Z dimension size of 1 and a *has-range* attribute equal to 256. The 8-bit codes are mapped to particular colors and intensities by display maps with this primary name. The *has-primary-colors* attribute in this case is *nil*.

Displays with the *standard-bitgraph has-map* value have no intensity array, but only bitgraph planes.

The *has-map* attribute value, s_map-name, is merely part of the name of the display map that will actually be used. It is, however, the part of the name important to the programmer. The other parts of the map name are used to select slightly different maps depending on the monitor, camera film, frame buffer, and display processor being used, but all the slightly different maps with the same s_map-name are intended to give essentually the same visual result.

See *a-display-map*.

HAS-RANGE:

HAS-PRIMARY-COLORS:

HAS-COLORS:

HAS-SCALES:

HAS-PLANE-TYPES: These attributes are inherited from *a-display-map* named by the *has-map* attribute. See HAS-MAP above and *a-display-map*.

HAS-BITGRAPH-PLANES: This attribute is a list whose elements are the type names of the display's bitgraph planes. The type names are looked up in the *has-plane-types* attribute to find *a-plane-type* objects that specify the color, line type, character font, area fill pattern, normal character size, and normal line width associated with each plane.

The elements of the *has-bitgraph-planes* list correspond to the planes with Z coordinates 0, 1, 2, ..., in that order. The length of the list determines the number of planes. There may be at most 32.

The *has-bitgraph-planes* attribute defaults to the bitgraph planes of the *has-parent* attribute if this latter attribute is non-*nil*.

HAS-INTENSITY-ARRAY: The intensity array is an array of *a-uchar*'s if the *has-range* attribute is not greater than 256, or an array of *a-ushort*'s otherwise. Its X and Y dimension sizes are x_xsize and x_ysize. Its Z dimension size is 1 if the *has-primary-colors* attribute is *nil*, and is otherwise the length of the *has-primary-colors* list.

This attribute may be given the explicit values *nil* or *t* when the display is created, to indicate whether the intensity array is to be absent or present. If no explicit value is given, an intensity array is created for the display if the *has-ranges* attribute is not *nil*, unless the display has a *has-parent* with no intensity array or a *has-device* that does not support intensity arrays.

HAS-BITGRAPH-ARRAY: The bitgraph array is an array of *a-ubit*'s. Its X and Y dimensions are x_xsize and x_ysize. Its Z dimension is the number of bitgraph planes, which equals the length of the *has-bitgraph-planes* list.

This attribute may be given the explicit values *nil* or *t* when the display is created, to indicate whether the bitgraph array is to be absent or present. If no explicit value is given, a bitgraph array is created for the display if the *has-bitgraph-planes*, *has-colors*, and *has-plane-types* attributes are not *nil*, unless the display has a *has-parent* with no bitgraph array or a *has-device* that does not support bitgraph arrays.

HAS-BITGRAPH-PROGRAMS: This is a hunk whose length is the number of bitgraph planes and whose elements are the bitgraph programs for the planes, in Z coordinate order.

This attribute may be given the explicit values *nil* or *t* when the display is created, to indicate whether the bitgraph programs are to be absent or present. If no explicit value is given, a bitgraph programs are created for the display if the *has-bitgraph-planes* attribute is not *nil*, unless the display has a *has-parent* with no bitgraph programs or a *has-device* that does not support bitgraph programs.

HAS-PARENT: If non-*nil* this is a window into which the display is merged by—

(*display-merge* dis_display dwin_parent)
(*flush-display* dwin_parent)

every time the display is flushed. *Expose-display* similarly merges the display into its parent and calls *expose-display* on the parent. Lastly *close-display* also calls itself on the parent.

A display also gets default values for its *has-sizes*, *has-map*, and *has-bitgraph-planes* attributes from its *has-parent*. The in addition the display will not acquire *has-intensity-array*, *has-bitgraph-array*, or *has-bitgraph-programs* by default if its *has-parent* does not have them.

HAS-DEVICE: This is a list that describes the hardware that implements the display, if there is such (many displays are merely used for storage). The following are the possible values—

(network <host> <port> <device> <processor> <monitor> <camera>)

A display accessible through the network via the display deamon system. The <host>, <port>, and <device> are as described in the appendix on DISPLAY DEAMONS. The <host> and <port> identify the server; the <device> identifies a particular display device on the <host> (the server services many displays on the same host).

The <processor>, <monitor>, and <camera> are as described under *a-display-map* S_PROCESSOR, S_MONITOR, and S_CAMERA. They are used to help select the display device's pseudocolor map and camera parameters (*has-map-array* and *has-camera-parameters* attributes of *a-display-map*).

<camera> may be omitted if there is none.

HAS-FILM: This is as described under *a-display-map*, S_FILM. It is used to select the pseudocolor map and camera parameters (*has-map-array* and *has-camera-parameters* attributes of *a-display-map*) during an *expose-display* operation. This *has-film* attribute may be *setf* at any time in order to allow changing films.

(**a-display-map** *has-ids* [SKETCH Type Macro]
'((s_map-name [s_monitor [s_film] [s_processor]]) ...)
[*has-primary-colors* '(s_primary-color-1 ...)]
[*has-range* 'x_range]
[*has-map-array* 'ucar_map-array]
[*has-camera-parameters* 'g_camera-parameters]
[*has-colors* '((s_color-1 (x_color-11 ...) ...)]
[*has-scales* '((s_scale-1 ucar/usar_scale-1) ...))]
[*has-plane-types* '((s_plane-type-1 plt_plane-type-1) ...))])

dmap_ [Argument Prefix]
a-display-map [SKETCH Type Object]

(**has-ids** 'dmap_map) [SKETCH Attribute Macro]
(**has-primary-colors** 'dmap_map) [SKETCH Attribute Macro]
(**has-range** 'dmap_map) [SKETCH Attribute Macro]
(**has-map-array** 'dmap_map) [SKETCH Attribute Macro]
(**has-colors** 'dmap_map) [SKETCH Attribute Macro]
(**has-scales** 'dmap_map) [SKETCH Attribute Macro]
(**has-plane-types** 'dmap_map) [SKETCH Attribute Macro]

USE: *A-display-map* provides various maps that relate to particular display hardware.

The *has-map-array* attribute is used with color displays whose intensity array records only one integer per pixel. It maps these integers onto color intensity N-tuples used by a color display monitor.

The *has-camera-parameters* attribute is used with displays that have camera hardware. Camera parameter settings determine how colors will look.

The *has-colors* attribute is an association list that maps color names onto intensity array values. For example, a color named *red* would map onto the intensity array value that gives the brightest red color.

The *has-scales* attribute is an association list that maps names of scales onto scales. A scale maps a set of equal sized intervals onto intensity array values. For example, a scale named *red* might map 16 intervals onto 16 different intensity levels of the color red.

The *has-plane-types* attribute is an association list that maps names of bitgraph plane types into *a-plane-type* objects that parameterize bitgraph planes. E.g. The name *red-italic* might map onto *a-plane-type* object with *has-color red* and *has-font display-italic*.

The *has-range* and *has-primary-colors* attributes of a display map parameterize the intensity array of any display using the map. The *has-range* attribute specifies the range of values that the elements of the array may take. The *has-primary-colors* attributes specifies the Z dimension size of the array, and the color associated with each Z coordinate. It is *nil* when the Z dimension size is 1 and the Z coordinate is not associated with colors, as is the case when the map has a non-*nil has-map-array* attribute.

Display maps have ID's that name them. The ID's principal part is 's_map-name. ID's have secondary parts, s_monitor, s_film, and s_processor, which name the monitor, the film for camera monitors, and the display processor used by the display hardware. Different display maps with the same s_map-name can be used to provide different *has-map-array*'s and *has-camera-parameters* for different monitors, films, and processors. The idea is that all display maps with the same s_map-name should look the same to the person viewing the display, and the differences in monitors, films, and processors should be compensated for by changing *has-map-array* and *has-camera-parameters*.

When a **display** map is made it is filed in a data base from whence it can be retrived via the *find-display-map* function. It is filed in this data base under each of its different ID's, and for each ID replaces any previous display map of that ID in the data base. The function *remake-display-maps* is useful for revising this data base.

Display map attributes are used only in certain contexts, and need not be present or correct in contexts where they are not used. *Has-range*, *has-primary-colors*, *has-colors*, and *has-scales* are used only for displays which have intensity arrays. *Has-plane-types* is used only for displays which have bitgraph programs. *Has-map-array* is used only with displays that have display hardware; and *has-camera-parameters* only with displays that have camera hardware.

All display maps with the same s_map-name should have the same *has-range*, *has-primary-colors*, *has-colors*, *has-scales*, and *has-plane-types* attributes if the maps are for displays with intensity and bitgraph arrays.

HAS-RANGE: The elements of the display's intensity array must be in the range from 0 through x_range—1.

X_range must not be greater than 65536. If x_range is not greater than 256, the intensity array elements must be of type *a-uchar*. Otherwise they must be of type *a-ushort*.

The *has-range* attribute is *nil* if display's with this map have no intensity array.

HAS-PRIMARY-COLORS: The primary colors,

(s_primary-color-1 ...)

name the Z subscripts of a display's intensity array. A typical value is—

(red green blue)

when the Z dimension size is 3.

If *has-primary-colors* is *nil*, the intensity array (if it exists) has only one integer element per pixel, its Z dimension size equals 1, and the *has-map-array* attribute maps the intensity integers onto color intensities (see below). The ***has-primary-colors*** attribute must be *nil* if the ***has-map-array*** attribute is non-*nil*.

HAS-MAP-ARRAY: For a display associated with a non-*nil* ***has-map-array***, ucar_map-array, each pixel in the intensity array is represented by a single integer that may be used as a Y coordinate to access a row of ucar_map-array. The elements of this row represent the intensities of colors in a manner understood by the hardware named by the ***has-ids*** attribute. Ucar_map-array has x_range rows.

Typically x_range is 256 and ucar_map-array has 3 columns with intensities for the colors red, green, and blue, in that order.

Ucar_array may be *nil* when *a-display-map* is being used for the sake of its *has-colors*, *has-scales*, *has-plane-types*, and *has-range* attributes, rather than for its ability to specify a map array for particular display hardware. It must be *nil* when the *has-primary-colors* attribute is non-*nil*, or when the *has-range* attribute is *nil*.

HAS-CAMERA-PARAMETERS: The *has-camera-parameters* attribute is a lisp object that sets the camera parameters for the camera named by s_camera (see HAS-IDS). Its form depends upon the type of camera. For the Matrix camera, it is a symbol which when viewed as a character string equals the response given by the camera to the pair of commands—

M
P

A typical value is for a color picture is—

|MCSA#1
P#1 COLOR EX800 R(R500G750B650)

C(R095G084B108) B(R378G378B378)
|

while a typical value for a black and white picture is—

|MWNA#1
P#1 NEG EX120 C077 B310
|

HAS-IDS: The map may be denoted by one of many ID's. These give a generic map name, s_map-name, the name of a monitor, s_monitor, and the name of a frame buffer processor, s_processor. An example ID is—

(*standard mitsubishi-c-3910 sun-475*).

Here *standard* names the map from the programmers point of view, *mitsubishi-c-3910* names the monitor (T.V. display), and *sun-475* names the display processor which stores the image in a digital frame buffer and converts it to analogue monitor signals.

Another example ID is—

(*standard matrix polaroid-559 sun-475*).

This ID is used with camera hardware. The camera monitor is a *matrix* which includes one of several camera mounts, and the film type is *polaroid-559*. The *has-map-array* and *has-camera-parameters* attributes need to be matched to the camera monitor and film type for best results.

An ID may be made less specific by omitting components at its end. Display maps are stored in a data base when they are made, and may be retrieved by the *find-display-map* function using the ID's to identify the map.

All display maps with the same s_map-name are intended to give the same visual result. The s_monitor, s_film, and s_processor components of the name are intended to make minor modifications to the map in order to give the same visual result on different monitors and with different processors. All maps with the same s_map-name should have *equal has-range*, *has-primary-colors*, *has-colors*, *has-scales*, and *has-plane-types* attribute values if the maps are for displays that have intensity or bitgraph arrays. However, the display package makes no check of this rule.

S_MAP-NAME: The most common value for s_map-name is *standard*. This map has an x_range of 256 and is as follows—

Intensity Array Element Value	Color	Hardware Intensities
0	black	0
1-127	white	32, 34, 35, ..., 255
128-143	magenta	85, 96, 108, ..., 255
144-159	red	85, 96, 108, ..., 255
160-175	brown	64, 77, 89, ..., 255
176-191	yellow	64, 77, 89, ..., 255
192-207	green	64, 77, 89, ..., 255
208-223	cyan	64, 77, 89, ..., 255
224-239	turquoise	64, 77, 89, ..., 255
240-255	blue	85, 96, 108, ..., 255

If the actual hardware intensities for each color are on a scale from 0 through 255, they will be chosen so that the most intense color component will have an actual hardware intensity equal to the hardware intensity given in the above table.

S_MONITOR: The following are some possible s_monitor values—

mitsubishi-c-3910	Mitsubishi C-3910 monitor.
mitsubishi-c-3419	Mitsubishi C-3419 monitor.
mitsubishi-c-3919	Mitsubishi C-3919 monitor.

S_CAMERA: The following are some possible s_camera values—

matrix	Matrix Camera.

S_FILM: The following are some possible s_film values—

polaroid-559	Polaroid 559 film.
polaroid-809	Polaroid 809 film.
polaroid-891	Polaroid 891 film.
polaroid-552	Polaroid 552 film.
kodak-VPS-III	Kodak VPS III film.

S_PROCESSOR: Some possible values for s_processor are—

comtal-512	COMTAL Vision-One/20 display processor organized as a 512×512×8-bit frame buffer.
sun-475	SUN 640×475×8-bit frame buffer.

HAS-COLORS: This attribute is a list of the form—

((s_color-1 (x_color-11 x_color-12 ...)) (s_color-2 (x_color-22 ...)) ...)

which maps symbols naming colors to lists of integers that that represent intensity array elements. The integers in one list represent the different Z components of one intensity array pixel value. The number of integers in the list must equal the Z dimension size of the intensity array. If this size is 1, the list of one integer may be replaced by a single integer, as in—

((s_color-1 x_color-1) (s_color-2 x_color-2) ...).

By way of example, any *standard* display map (see S_MAP-NAME above) has the following value for its *has-colors* attribute—

((black 0) (white 127) (magenta 143) (red 159) (brown 175) (yellow 191)
(green 207) (cyan 223) (turquoise 239) (blue 255))

HAS-SCALES: A scale maps N equal sized intervals of the real line onto N different intensity array pixel values. The scale does not itself specify the locations of the intervals: it is merely a vector of N pixel values. This vector is actually an array (e.g. ucar/usar_scale-1) whose Y dimension indexes the intervals. Thus the Y dimension size is the number of intervals. The X dimension of this scale array corresponds to the Z dimension of the display's intensity array: each scale array row gives the Z components for one intensity pixel.

The *has-scales* attribute maps scale names such as s_scale-1 onto scale arrays such as ucar/usar_scale-1.

If s_map-name is *standard*, the following scale names are defined by *has-scales*—

Scale Name	Number of Intervals	Value
black	1	Pure black.
gray	127	Dark gray to bright white.
pseduocolor	128	16 intensity levels, near black to white, modulated by 8 colors in the order magenta, red, brown, yellow, green, cyan, turquoise, blue.
positive	64	16 intensity levels, near black to white, modulated by 4 colors in the order green, cyan, turquoise, blue.
negative	64	16 intensity levels, white to near black, modulated by 4 colors in the order magenta, red, brown, yellow.
colors	9	9 colors on the order magenta, red, brown, yellow, green, cyan, turquoise, blue, white. Each color is represented by its brightest intensity level.
magenta	16	16 intensity levels, from dark magenta to bright magenta.
red	16	16 intensity levels, from dark red to bright red.
brown	16	16 intensity levels, from dark brown to bright brown.
yellow	16	16 intensity levels, from dark yellow to bright yellow.
green	16	16 intensity levels, from dark green to bright green.
cyan	16	16 intensity levels, from dark cyan to bright cyan.
turquoise	16	16 intensity levels, from dark turquoise to bright turquoise.
blue	16	16 intensity levels, from dark blue to bright blue.

HAS-PLANE-TYPES: *A-plane-type* specifies parameters used in drawing a bitgraph plane: color, line width in pixels, line type (*solid, long-dashed, dot-dashed*), character size, character font (*display, times-roman, times-italic, times-bold*), character size in pixels, and fill pattern (*nil* is the default which means solid fill). The *has-plane-types* attribute maps plane type names such as s_plane-type-1 onto plane types such as plt_plane-type-1.

The *standard-bitgraph* display map defines the plane names white, magenta, red, brown, yellow, green, cyan, turquoise, and blue. If color is available these all have the *display* font, a line width of 1.0, and the line type *solid*. If color is not available these plane type names are defined as follows—

Plane Type Name	Font	Line Width	Line Type
white	*roman*	1.0	*solid*
red	*roman*	1.0	*long-dashed*
green	*roman*	1.0	*dotted*
turquoise	*roman*	1.0	*dot-dashed*
yellow	*roman*	1.0	*short-dashed*
magenta	*bold*	2.0	*long-dashed*
cyan	*bold*	2.0	*dotted*
blue	*bold*	2.0	*dot-dashed*
brown	*bold*	2.0	*short-dashed*

Standard display maps provide the same plane type names as *standard-bitgraph* maps. *Standard* display maps work only with displays that acutally have color available. In all cases the line type is solid.

The normal character size for both *standard* and *standard-bitgraph* plane type names is (6 12): that is, 6 pixels wide and 12 tall. The display font provides different fonts for the character sizes (relative to 6X12) 1.0, 1.2, 1.5, 2.0. 2.7, 3.2, and 4.2. The width of the characters is the controlling dimension, so for example the 1.5 character set is 1.5*6 = 9 pixels wide.

(a-display-window ***has-parent*** 'dis/dwin_parent [SKETCH Type Macro]
[***has-sizes*** '(n_xsize n_ysize)]
[***has-upper-left*** 'dwpt_upper-left]
[***has-lower-right*** 'dwpt_lower-right]
[***has-origins*** 'dwpt_origins]
[***has-zooms*** '(n_xzoom n_yzoom)]
[***has-orientation*** 's_orientation]
[***has-transform*** 'trans_transform]
[***has-cursor*** 'dwpt_cursor]
[***has-plane*** 'bpn_line-plane]
[***has-line-plane*** 'bpn_line-plane]
[***has-area-plane*** 'bpn_area-plane]
[***has-text-plane*** 'bpn_text-plane])

dwin_	[Argument Prefix]
a-display-window	[SKETCH Type Object]
"display window point"	[SKETCH Term]
dwpt_	[Argument Prefix]
(**has-parent** 'dwin_display)	[SKETCH Attribute Macro]
(**has-sizes** 'dwin_display)	[SKETCH Attribute Macro]
(**has-upper-left** 'dwin_display)	[SKETCH Attribute Macro]
(**has-lower-right** 'dwin_display)	[SKETCH Attribute Macro]
(**has-origins** 'dwin_display)	[SKETCH Attribute Macro]
(**has-zooms** 'dwin_display)	[SKETCH Attribute Macro]
(**has-orientation** 'dwin_display)	[SKETCH Attribute Macro]
(**has-transform** 'dwin_display)	[SKETCH Attribute Macro]
(**has-cursor** 'dwin_display)	[SKETCH Attribute Macro]
(**has-plane** 'dwin_display)	[SKETCH Attribute Macro]
(**has-line-plane** 'dwin_display)	[SKETCH Attribute Macro]
(**has-area-plane** 'dwin_display)	[SKETCH Attribute Macro]
(**has-text-plane** 'dwin_display)	[SKETCH Attribute Macro]
current-display-window	[LISP Global Variable]
(**get-current-display-window**)	[LISP Macro]
playback-display-window	[LISP Global Variable]
(**get-playback-display-window**)	[LISP Macro]

USE: *A-display-window* specifies the context parameters used by display commands. Included are parameters for translating the command coordinates into display coordinates, for clipping lines, images, and text that fall outside a rectangular display region, for determining the current cursor position, and for selecting the current bit plane with which characters and lines are drawn.

CURRENT-DISPLAY-WINDOW:

GET-CURRENT-DISPLAY-WINDOW: The **current-display-window** variable holds *a-display-window* value which is the display window upon which most display commands operate by default.

This variable can also hold a function or *lambda* expression to be *funcall*'ed to return *a-display-window* value. It is often defined to be a *lambda* expression that sets **current-display-window** to *a-display-window* value and returns that value.

A call to *get-current-display-window* will return *a-display-window* derived from **current-display-window**, or will signal an error if no window is obtainable in this way.

PLAYBACK-DISPLAY-WINDOW:

GET-PLAYBACK-DISPLAY-WINDOW: **Playback-display-window** and *get-playback-display-window* are analogous to **current-display-window** and *get-current-display-window*, except that the window returned is used only by a few functions: in fact, at the moment, only by the *playback-display* function. However, the **current-display-window** is usually defined in terms of the **playback-display-window**.

HAS-PARENT: *A-display-window* is associated with *a-display*, the *has-parent* attribute of the window. Drawing in the window actually draws in its parent display.

If a window is created with another window as a parent, the *has-parent* attribute is changed to equal the parent display of the parent window, and the *has-sizes*, *has-zooms*, *has-orientation*, *has-origins*, *has-upper-left*, and *has-lower-right* parameters are adjusted accordingly.

HAS-SIZES: The window appears to be a rectangular display with sizes—

(n_xsize n_ysize).

This is mapped onto a box in the window's parent (see *has-orientation*, *has-zooms*, etc. below).

The coordinates of a window are like those of a display. The upper left pixel of the window has a center with zero coordinates. The X coordinates of this pixel lie in the range from -0.5 to +0.5. Similarly for the Y coordinates. If n_xsize and n_ysize are integers, the lower right pixel of the window has a center with coordinates (n_xsize−1 n_ysize−1), X coordinates in the range from n_xsize−1.5 to n_xsize−0.5, and Y coordinates in the range from n_ysize−1.5 to n_ysize−0.5.

Any line written into the window that would be partly outside the window is clipped: the parts outside are not drawn. Similarly any image written into the window is clipped: the parts outside are not drawn. However, any character written into the window is omitted, and not written at all, if any part of it would lie outside the window.

HAS-ORIENTATION:

HAS-ZOOMS:

HAS-UPPER-LEFT:

HAS-LOWER-RIGHT:

HAS-ORIGINS: *A-display-window* maps onto a box in its parent display. To specify the box one typically gives its upper left corner and, indirectly, its sizes. The sizes of the box are the products of the sizes of the window and the zooms of the window, and it is these latter two quantities that are given directly.

The orientation of the window in the box is a symbol that tells how to rotate the window to fit it into the box. *Left-rotate*, *right-rotate*, and *top-rotate* mean to rotate the window 90 or 180 degrees so that the window's bottom becomes the left, right, or top of the box. *Nil* (the default) means not to rotate the window at all.

The orientation may also be *left-mirror*, *right-mirror*, or *top-mirror* to specify that the left and right of the window are interchanged before it is rotated; or the orientation may be just plain *mirror* to interchange left and right without rotating.

The zooms are the ratios of the sides of the window to the sides of the box they are mapped onto. Thus n_xzoom is the length of the side of the box that the bottom (or top) of the window is mapped onto, divided by n_xsize. Similarly for n_yzoom, the left (or right) of the box, and n_ysize.

The *has-upper-left* and *has-lower-right* attributes give the upper left and lower right corners of the box onto which the window is mapped. The *has-origins* attribute gives the point in the box that the upper left pixel of the window is mapped onto. If the orientation is *nil*, then—

$$\text{n\_xupper-left} - 0.5 = \text{n\_xorigin} + (-0.5) * \text{n\_xzoom}$$

since the left edge of the upper left pixel of the window has x coordinate —0.5, while the left edge of the box has x coordinate n_xupper-left—0.5. A similar relation holds for the y coordinate. The relations for the lower right corner of the box, and for other orientations, can be derived similarly, remembering that the right of the window has x coordinate n_xsize—0.5. Thus, for example, if the orientation is *rotate-left*, then—

$$\text{n\_xupper-left} - 0.5 = \text{n\_xorigin} - (\text{n\_ysize} - 0.5) * \text{n\_yzoom}$$

since the rotation puts the origin near the upper right corner of the box and also exchanges the X and Y axises.

Notice that the numbers n_xupper-left, n_yupper-left, n_xlower-right, n_ylower-right, n_xorigin, n_yorigin, n_xsize, n_ysize, n_xzoom, and n_yzoom may all be non-integer floating point numbers. In this respect *a-display-window* is not like *a-display*, for the sizes of the latter must be integers.

If the *has-parent* given to create a window is itself *a-display-window*, the corners, origins, zooms, and orientation are all specified relative to the parent window, but are converted to be relative to the parent display of that window when the *has-parent* attribute is converted to be that parent display (see HAS-PARENT above).

If *has-origins* is given, neither *has-upper-left* nor *has-lower-right* may be given. It is illegal to give all four of the attributes *has-upper-left*, *has-lower-right*, *has-sizes*, and *has-zooms*.

DEFAULT FOR HAS-SIZES:

DEFAULT FOR HAS-ZOOMS:

DEFAULT FOR HAS-UPPER-LEFT: If a window is created with a prototype, as in—

(*a-display-window* (*get-current-display-window*)
has-origins '(10 10) *has-sizes* '(128 128))

the *has-origins*, *has-upper-left*, *has-lower-right*, *has-sizes*, and *has-zooms* attributes are *not* inherited from the prototype, unless none of these attributes are given, and the *has-parent* and *has-orientation* attributes are also not given. (Giving a *nil* value is equivalent to not giving a value for any of these attributes except *has-orientation*.)

Except in this case, *has-upper-left* defaults to '(0.0 0.0) if neither it nor *has-origin* nor *has-lower-right* is given; *has-zooms* defaults to '(1.0 1.0) unless it can be computed because *has-sizes*, *has-upper-left*, and *has-lower-right* have all been given; and lastly, *has-sizes* defaults, if it cannot be computed, to the largest value which will allow the window to fit inside its *has-parent*.

WINDOW POINTS:

HAS-TRANSFORM:

DWPT_: A window point is a point in the coordinate system of *a-display-window*. The window coordinate system places the point (0 0) at the center of the upper left pixel of the window, and increases the X and Y coordinates by 1 as one moves 1 pixel right or down. The X axis is horizontal and the Y axis vertical.

A window point can be specified directly as a pair of numbers (either *fixnum*'s or *flonum*'s).

A window point can also be specified as a point in the sense of the ANALYTIC GEOMETRY package (actually *a-vector* value), in which case it is transformed into the window coordinate system by the value of the window's *has-transform* attribute, which is *a-transform* value. A *nil has-transform* value is considered to be equivalent to the unit transform of 2 dimensional space. See the ANALYTIC GEOMETRY package for more details.

A *dwpt_* argument is a window point: either a list of two numbers or *a-vector* value. The later is converted into a list of two numbers. If a window point is stored in *a-display-window*, it is always stored in the form of a list of two numbers, and will be read in that form.

HAS-CURSOR: The cursor is a window point. It acts as an offset to the position where objects (e.g. lines and images) are drawn in the window.

The *has-cursor* attribute can be *setf*, and so can the its individual coordinates. It may be *setf* to a window point (see WINDOW POINT above).

The cursor defaults to (0 0).

HAS-LINE-PLANE:

HAS-TEXT-PLANE:

HAS-AREA-PLANE:

HAS-PLANE: Lines are drawn by default in the bitgraph plane specified by x_line-plane, which is the Z coordinate of the plane. This attribute may be *setf* anytime. It may be *setf* to the name of a plane (either a symbol or a symbol/integer pair: see BITGRAPH PLANE NAMES under *a-display*), in which case the name is converted to the plane Z coordinate before it is stored.

Characters are drawn similarly in the bitgraph plane specified by x_text-plane; and areas are filled in the bitgraph plane specified by x_area-plane.

The ***has-plane*** attribute, when *setf*, sets all three of the other plane attributes to the same value. *Has-plane* may be read without error only if all three of the other plane attributes have the same value, which is, of course, the value read.

The *has-line-plane*, *has-area-plane*, and *has-text-plane* attributes default to the value 0 if the window's parent display has a non-*nil has-bitgraph-planes* attribute; or to the value *nil* otherwise.

HAS-INTENSITY-ARRAY:

HAS-BITGRAPH-PLANES:

HAS-BITGRAPH-ARRAY:

HAS-BITGRAPH-PROGRAMS:

HAS-DEVICE:

HAS-FILM:

HAS-RANGE:

HAS-PRIMARY-COLORS:

HAS-COLORS:

HAS-SCALES:

HAS-MAP:

HAS-PLANE-TYPES: These are all attributes of the *has-parent* of *a-display-window*, but we frequently abuse language and speak as if they were the attributes of the window. They are not, and programs cannot treat them as such.

(a-plane-type [*has-color* 's_color] [SKETCH Type Macro]
[*has-fill-pattern* 'ubar_fill-pattern]
[*has-line-type* 's_line-type]
[*has-line-width* 'n_line-width]
[*has-character-font* 's_character-font]
[*has-character-sizes* '(n_xcharacter-size n_ycharacter-size)])

plt_ [Argument Prefix]
a-plane-type [SKETCH Type Object]

(has-color 'plt_display) [SKETCH Attribute Macro]
(has-fill-pattern 'plt_display) [SKETCH Attribute Macro]
(has-line-type 'plt_display) [SKETCH Attribute Macro]
(has-line-width 'plt_display) [SKETCH Attribute Macro]
(has-character-font 'plt_display) [SKETCH Attribute Macro]
(has-character-sizes 'plt_display) [SKETCH Attribute Macro]

USE: *A-plane-type* parameterizes a bitgraph plane. It specifies color, fill pattern (a bitgraph array) for filling areas, line type (*solid*, *long-dashed*, etc.), normal line width, character font (*display*, *times-roman*, etc.), and normal character sizes.

HAS-COLOR: A symbol naming the color of the plane's pixels: e.g. *red*. When the plane is used, the color must appear in the display's ***has-colors*** list (which is inherited from the display's map).

A *nil* value is equivalent to *white*.

HAS-FILL-PATTERN: A bitgraph array which is used to fill areas. This two dimensional array of *a-ubit*'s is replicated starting in the lefthand corner of a display to make a mask that covers the entire display. When an area in the display needs to be filled, the pixels in the area corresponding to one bits in this mask are turned on.

A *nil has-fill-pattern* attribute value is equivalent to an array with all its bits on, giving a solid area fill.

HAS-LINE-TYPE: This is a symbol chosen from the list—

solid
dotted
long-dashed
short-dashed
dot-dashed

It describes the type of line drawn by the ***display-lines*** function.

A *nil* value is equivalent to *solid*.

HAS-LINE-WIDTH: Specifies the normal value for a line's width in pixels, for lines drawn by the *display-lines* function. Fractional widths are possible: the nearest available width is used. The actual line width used to draw a line is the product of this number and the line width specified by the *display-lines* function that draws the line.

A *nil* value is equivalent to 1.0.

HAS-CHARACTER-FONT: The style of the character set used by *display-print* and related functions. See the *has-font* attribute of *a-character-set* in the BITGRAPH ARRAYS package.

A *nil* value is equivalent to *display*.

HAS-CHARACTER-SIZES: N_xcharacter-size and n_ycharacter-size are the width and height in pixels of a box into which each character is to fit. The character set choosen for use by a *display-print* or similar function is the largest whose characters will fit in the box, in a sense we will now describe.

N_ycharacter-size is actually the distance between successive lines of text. and must be somewhat greater than the actual height of the characters. N_xcharacter-size is an upper bound on the probable average horizontal spacing between characters, for capitalized English words. For variable width fonts, words with lower case letters generally have characters somewhat closer together than n_xcharacter-size would indicate, while strange words like HMMMMM might have characters farther apart than x_character-size would indicate. See HAS-SIZES under *a-character-set* for more details. The character set is actually found using the *find-character-set* function.

The actual character sizes used to select a character set are the product of these sizes and the character size number specified by the *display-print* or similar function that draws the characters.

"bitgraph programs" [SKETCH Term]

(w (n_xsize n_ysize) (n_xorigin n_yorigin) [Bitgraph Program Window]
(n_xzoom n_yzoom) s_orientation <statement> ...)

(t [n_character-size] [Bitgraph Program Statement]
[s_horizontal-adjust] [s_vertical-adjust]
[s_orientation] (n_xorigin n_yorigin) s/t_string ...)

(l [n_width] (n_xorigin n_yorigin) [Bitgraph Program Statement]
[(n_x n_y)] ... [*nil*]... [sar_array] ...)

(f (n_xorigin n_yorigin)) [Bitgraph Program Statement]

VALUE: A bitgraph program is a list of bitgraph program windows. Each bitgraph program window consists of a list with some window parameters followed by bitgraph program statements. The statements are of three types: text statements output text, line statements output lines, and fill statements fill areas (respectively the *t*, *l*, and *f* statements above).

The window parameters define the sizes, zooms, origins, and orientation of the window: see *a-display-window*.

See *display-text* for text statement parameters; *display-lines* for line statement parameters; and *fill-display-area* for fill statement parameters.

(check-bitgraph-program-syntax 'l_program) [LISP Function]

SIDE EFFECT: Checks the bitgraph program for syntax errors, and calls *error* if any are found.

(clear-display ['s_background] ['dwin_window]) [LISP Function]
(clear-intensity ['s_background] ['dwin_window]) [LISP Function]
(clear-bitgraph ['bpn_plane] ['dwin_window]) [LISP Function]

WHERE: S_background must be a symbol naming a *has-color* color of dwin_window (default is *black*).

Bpn_plane may be *nil* or omitted to specify the set of all bitgraph planes.

Dwin_window defaults to the value of the value of (*get-current-display-window*).

Nil arguments are ignored (so any argument may be *nil*).

SIDE EFFECT: Clears the intensity and bitgraph plane parts of a display window. *Clear-display* clears everything; *clear-intensity* just clears the intensity array; while *clear-bitgraph* just clears one or all bitgraph planes.

It is an error to try to clear a bitgraph plane if dwin_window does not cover its parent display entirely and the parent display has bitgraph programs.

The intensity array is set to a specified value, s_background, when it is cleared.

(close-display [dwin_display]) [LISP Function]

WHERE: Dwin_window defaults to the value of (*get-current-display-window*).

SIDE EFFECT: Releases the resources of the parent display of dwin_window. These resources, things like the display monitor hardware and communications channels to that hardware, are allocated the first time the display is flushed. After closing the display, the resources will be reallocated the next time the display is flushed.

(compose-display-orientations 's_first 's_second) [LISP Function]

RETURNS: Returns the display orientation that is the composition of the display orientation s_first followed by the display orientation s_second. Display orientations are the possible values of the *has-orientation* attribute of *a-display-window*.

(display-bitgraph 'ubar_array 's/l_name ['dwin_window] ['dwpt_upper-left] [*has-zooms* '(n_xzoom n_yzoom)] [*has-sizes* '(n_xsize n_ysize)] [*has-orientation* 's_orientation]) [LISP Function]

(display-bitgraph 'ubar_array '(s/l_name-1 ...) ['dwin_window] ['dwpt_upper-left] [*has-zooms* '(n_xzoom n_yzoom)] [*has-sizes* '(n_xsize n_ysize)] [*has-orientation* 's_orientation]) [LISP Function]

USE ONLY WHEN: Displays have only bitgraph arrays and do not have bitgraph programs.

The *display-bitgraph* function is an anachronism that should be replaced where possible by calls to *display-text*, *display-lines*, and *fill-display-area*, because the latter are output resolution independent, while *display-bitgraph* depends upon the application knowing the resolution of the output display.

WHERE: The dwin_window, dwpt_upper-left, n_xzoom, n_yzoom, n_xsize, n_ysize and s_orientation arguments are passed to the *new-window* function to get a window referred to as dwin_window below.

The zooms of dwin_window relative to its parent display must be 1.0. The pixels of dwin_window must be exactly aligned with the pixels of its parent display.

If dwin_window has a bitgraph array, each s/l_name is a bitgraph plane name of a plane in dwin_window into which a plane in ubar_array maps; or s/l_name may be *nil* to indicate a plane in ubar_array is not to be merged into dwin_window. The planes of ubar_array correspond to the Z coordinates of ubar_array, and these taken in order (0, 1, 2, ...) correspond to the elements of the list of s/l_names's (if there is only one name it may be given directly, instead of as in a list).

If dwin_window has no bitgraph array, but only an intensity array, each s/l_name is just a symbol naming a color in the has-colors list of dwin_window; or s/l_name is *nil* to indicate a plane in ubar_array is not to be merged into dwin_window.

SIDE EFFECT: Copies the information in ubar_array into the bitgraph array of dwin_window, if dwin_window has one, or the intensity array of dwin_window otherwise.

The merging is done just as it would be by *merge-display* if ubar_array were an attribute of *a-display*. If dwin_window has a bitgraph array, ubar_array is merged into that. if dwin_window has an intensity array, but no bitgraph array. ubar_array is merged into the intensity array. It is not an error for dwin_window to have bitgraph programs, but they will be uneffected by *display-bitgraph*.

(display-image 'ar_array ['dwin_window] [LISP Function]
['dwpt_upper-left] [*has-zooms* '(n_xzoom n_yzoom)]
[*has-sizes* '(n_xsize n_ysize)]
[*has-orientation* 's_orientation]
[*has-bounds* '(n_black n_white)]
[*has-bounds* '(n_bound-1 [—] s_scale-1 ... n_bound-N)]
[*has-missing* 's_missing-color]
[*has-low* 's_low-color]
[*has-high* 's_high-color]
[*has-contrasts* '(x_xcontrast x_ycontrast)]
[*do-pseudocolor* 'g_pseudocolor-switch])

WHERE: The dwin_window, dwpt_upper-left. n_xzoom, n_yzoom, n_xsize, n_ysize and s_orientation arguments are passed to the *new-window* function to get a window referred to as dwin_window below.

X_xcontrast and x_ycontrast. if given, must be > 0.

S_missing-color defaults to *red*.

There are two forms for the *has-bounds* argument. The simple form (n_black n_white) specifies the range of image values which will map onto the display's *gray* scale (or *pseudocolor* scale with the *do-pseudocolor* option).

If the *has-bounds* argument is *nil*, n_black and n_white default to the minimum and maximum value in ar_image.

The more complex form of the *has-bounds* argument gives a list of bounds and between each the name s_scale of a scale that is used to map the image values between the bounds. Each s_scale names a vector that is associated with two bounds, a lower bound and an upper bound. An image value that is greater than or equal to the lower bound, and less than the upper bound, is converted to a vector subscript by the formula

(*floor* ((image value — lower bound) * vector size)
/ (upper bound — lower bound))

This subscript indexes a vector element which yields a value for an intensity array pixel.

The scales are defined by the *has-scales* attribute of the window's parent, which is inherited from the parent's display map. See HAS-SCALES under *a-display-map* for the scales available when the *standard* display map is in use.

In the case of the complex *has-bounds* list, the bounds must be in ascending order. However, any bound may be *nil*. If the first bound is *nil*, it will be replaced by the minimum of all the image elements and the first non-*nil* bound. Similarly, if the last bound is *nil*, it will be replaced by the maximum of all the image elements and the last non-*nil* bound. If any other bounds are *nil*, they will be replaced in a fashion that will make all the scales between each pair of non-*nil* bounds appear to be concatenated.

Nil bounds may be completely omitted from the list. Thus the bounds list-

-0.5 black red 99.5

effectively makes a new 17 interval scale by concatenating the 1 interval scale *black* with the 16 interval scale *red*.

If a scale is immediately preceded by a – in the *has-bounds* list, the order of the values in the scale is reversed: e.g. a black to white scale becomes a white to black scale.

SIDE EFFECT: Displays the image in dwin_window. If the image is too small, it is placed in the upper left corner of the window, and if too large, only the upper left corner of the image is displayed.

If the window has zooms that are not equal to 1.0, each pixel in the window's parent display is given the value of the image pixel determined by mapping the display pixel's coordinates back to image pixel coordinates and rounding to the nearest image pixel.

If the *has-contrasts* argument is given, the image will have the average of a rectangle of size

(2 * x_xcontrast + 1 2 * x_ycontrast + 1)

subtracted from each element by the *contrast-of* function.

The *has-missing* argument specifies the color in which missing values will be displayed (default *red*). *Has-low* specifies the color in which values below the lowest *has-bound* bound will be displayed. If *has-low* is *nil* (the default), values lower than the lowest bound will be treated as equal to the lowest bound. Similarly, *has-high* specifies the color in which values equal to or above the highest *has-bound* bound will be displayed. If *has-high* is *nil* (the default), these values are treated as being just below the highest bound.

(display-lines [dwin_window] [n_width] [LISP Function]
[bpn_plane] [trans_transform]
[dwpt_point] ... [*nil*] [[–] ar_array] ...)

WHERE: Dwin_window defaults to the value of (*get-current-display-window*), n_width defaults to 1.0, bpn_plane defaults to the *has-line-plane* attribute of dwin_window, and trans_transform defaults to the *has-transform* attribute of dwin_window.

Nil arguments before the first dwpt_point, –, or ar_array are ignored.

SIDE EFFECT: Draws in bpn_plane a sequence of straight lines connecting points in the display window. Each point is either an argument (e.g. dwpt_point), or is a row in an array argument (e.g. ar_array) whose first, second, and third columns are the X, Y, and Z coordinates of the points, respectively (there may be no Z coordinate, or even no Y coordinate). The points defined by an array are transformed by trans_transform *unless* the array is proceeded by a -, in which case they are not transformed. Any dwpt_point points that are *a-vector* objects are also transformed by trans_transform.

A *nil* argument separates a preceeding sequence of lines from a following sequence. Thus many different sequences can be displayed by one call to *display-lines*. An ar_array point with a missing coordinate separates line sequences like the *nil* argument.

Line sequences are displayed by dragging a circular dot of size determined by n_width in straight lines between line sequence points. If a line sequence consists of a single point, at single dot is drawn at that point.

All line end points are offset from the window origin by the window cursor position.

The boundaries of the window clip the lines. The line end points need not lie inside the window.

(display-text ('x_line-length ['(n_xorigin n_yorigin)] [LISP Macro]
['bpn_plane] ['s_adjust ...] ['n_size] ['s_orientation] ['dwin_window])
g_statement ...)

(display-print 'g_value ['(n_xorigin n_yorigin) [LISP Function]
['bpn_plane] ['s_adjust ...] ['n_size] ['s_orientation] ['dwin_window])

(display-patom 'g_value ['(n_xorigin n_yorigin)] [LISP Function]
['bpn_plane] ['s_adjust ...] ['n_size] ['s_orientation] ['dwin_window])

(display-pretty-print 'x_line-length 'g_value [LISP Function]
['(n_xorigin n_yorigin)] ['bpn_plane] ['s_adjust ...] ['n_size]
['s_orientation] ['dwin_window])

(make-display-text-string 'x_string-size-in-bytes) [LISP Function]

WHERE: Dwin_window defaults to the value of (*get-current-display-window*), n_xorigin and n_yorigin default to 0.0, n_size defaults to 1.0, and s_adjust and s_orientation default to *nil*. Bpn_plane defaults to the *has-text-plane* attribute of dwin_window.

Nil arguments are ignored.

SIDE EFFECT: Text is drawn at the origin location in bpn_plane. The origin actually used is that given by (n_xorigin n_yorigin) displaced by the window cursor.

The text is first written into a string buffer. The result may be 1 or more lines (the last line need not end with a line feed). *Display-print* writes as *print*; *display-patom* as *patom*; and *display-pretty-print* as *pretty-print*. During the execution of *display-pretty-print* the LISP global variable ***line-length***, the line length used by *pretty-print*, is bound to x_line-length.

Display-text is a macro which (1) binds a port that writes into a string to the normal output port, *poport*, (2) binds x_line-length to *line-length* for use by *pretty-print*, (3) evaluates and remembers s_adjust, n_size, s_orientation, and dwin_window, (4) executes the body: g_statement ..., and (5) writes the the string into dwin_window using the remembered s_adjust, n_size, and s_orientation.

After the text is written into the string buffer, the font and size of the characters are determined. The normal character size for the bit plane being used is multiplied by n_size to determine character size. Then the width and height of the text in pixel positions are computed. These is used to form an imaginary box around the text. The text lines are then adjusted in the box according to some of the s_adjust parameters (as described below). Lastly, the box is positioned in the display window according to the origin position, s_adjust, and s_orientation parameters, and the text is drawn.

S_ADJUST: The s_adjust parameters control the positioning of lines within the text box, and the positioning of the box relative to the origin. The possible s_adjust values are—

left	The origin is placed just to the left of the box. The lines of text with the least amount of blank space at their left are left justified in the box. The lines with the next least amount of blank space at their left have their first non-blank character printed directly under the character in the same column of the first line above them that has already been justified, if any, or the first already justified line below them if there is no such line above. And so forth. until all lines are justified.
right	Like *left* but to the right instead of the left.
—	If neither *left* or *right* is given, each line has blank space at its beginning and ending removed, and is then centerred in the box. The origin is placed at the center of the box in the horizontal dimension.
under	The origin is placed just under the box.
over	The origin is placed just over the box.
—	If neither *under* or *over* is given, the origin is placed at the center of the box in the vertical dimension.

N_SIZE: This parameter is a number which multiplies the normal character size for the bitgraph plane in which the text is placed. Both horizontal and vertical sizes are multiplied by n_size. The character set actually used is the largest available whose sizes do not exceed those requested. (Acutally, the horizontal size is made as large as possible first, and then the vertical size.)

S_ORIENTATION: This is one of the values—

nil	*mirror*
left-rotate	*left-mirror*
top-rotate	*top-mirror*
right-rotate	*right-mirror*

The entire text is rotated as indicated around the origin position. *Nil* means to do no rotation; *top-rotate* means to rotate 180 degrees to make the bottommost part of the characters near the top of the display.

The mirror forms do not cause the characters to be mirror-imaged, nor do they reverse the order of the characters in the text. But they do switch which side of the text the origin is on, left or right, when viewed after any rotation. An attempt is also made to keep the amount of blank space between the text and the origin the same as it would have been if the text had not been mirrored.

The orientation of the text specified by s_orientation is composed with any orientation specified for the window in which the text is displayed to get the orientation actually used for display of the text.

MAKE-DISPLAY-TEXT-STRING: The string used by *display-print* etc. is initialized when the system is loaded and has a size of 16384 bytes. A call to *make-display-text-string* will create a new string of a different size for the use of *display-print* etc. This is useful only if the old string is too small.

(expose-display ['x_count] ['dis/dwin_display]) [LISP Function]

WHERE: Dis/dwin_display defaults to the value of (*get-current-display-window*).

X_count defaults to 1.

Nil arguments are ignored (so any argument may be *nil*).

SIDE EFFECT: Flushes dis/dwin_display as per *flush-display*, and then takes x_count identical pictures of the current state of the display. The *has-device* attribute value determines the procedure for doing this. A timer value is used in the flush which allows enough time for the camera to take the pictures, and *expose-display* returns before the camera is done.

The pseudocolor map in the display hardware may be changed temporarily while the picture is being taken, causing the picture to look different temporarily on any television monitor. This is done to compensate for differences between the television monitor and the camera plus film, so that after the film is developed, it will look the same as the television monitor normally looks.

The *has-camera-parameters* attribute of the display map selected by the display's *has-map* and *has-film* attributes will be used to set the camera parameters. See HAS-CAMERA-PARAMETERS under *a-display-map*.

(find-display-map '(s_map-name [s_monitor ...])) [LISP Function]
(find-display-map 's_map-name) [LISP Function]
(find-display-maps '(s_map-name [s_monitor ...])) [LISP Function]
(find-display-maps 's_map-name) [LISP Function]

RETURNS: *Find-display-map* returns the display map which has an ID that most precisely matches the ID given as the argument. A match is more precise if more components are given: e.g., '(standard matrix polaroid-891 sun-475) matches '(standard) but matches '(standard matrix polaroid-891) more precisely. *Find-display-map* returns *nil* if there is no matching map.

Find-display-maps returns a list of all the display maps that whose ID matches the ID given as the argument.

(flush-display ['f_delay] ['dis/dwin_display] [*t*]) [LISP Function]

WHERE: Dis/dwin_display defaults to the value of (*get-current-display-window*). If dis/dwin_display is a display window, it is replaced by its parent display.

Nil arguments are ignored (so any argument may be *nil*).

SIDE EFFECT: If dis_display has no *has-device* or *has-parent*, this function does nothing.

If dis_display has a *has-device*, *flush-display* delays until all commands sent to the display have actually been executed by the display hardware, and then returns. If f_delay is given, a timer is set which will ensure that the display hardware will not accept any subsequent display command until f_delay seconds after the last of these commands is finished.

If dis_display has a *has-parent*, then dis_display is merged into this parent by the *merge-display* function, and then *flush-display* is executed on the parent.

A record is kept of which parts of the display have been changed since the last flush, so as to avoid redundant work. The *t* argument causes this flush to believe the entire display has been changed since the last flush. It is useful if the screen is destroyed accidentally and needs to be refreshed.

(make-display-map-array ['n_gamma] [LISP Function]
'(n_red n_green n_blue x_size [n_first [n_last]])
...)

RETURNS: An array suitable for use as *a-display-map has-map-array* value for a display that takes 8-bit red, blue, and green intensities (in that order). The array is made by filling in x_size rows from each list argument of the form

(n_red n_green n_blue x_size [n_first [n_last]]).

The first three elements of this list define the relative sizes of the red, green, and blue components of some color.

The assumption is that the intensities on the screen are the gamma power of the numbers in the pseudocolor map, and the total intensity of a color (R G B) is

R ** gamma + G ** gamma + B ** gamma.

The numbers in the map are to give a linear scale of increasing total intensities.

N_first ** gamma is the desired intensity for the first element filled in. N_last ** gamma is the desired intensity for the last element filled in. If n_last is omitted or given as *nil*,

(red ** gamma + green ** gamma + blue ** gamma) ** (1/gamma).

is used as n_last. If n_first is omitted or given as *nil*, ((n_last ** n_gamma) /x_size) ** (1/gamma)

will be used as n_first.

The X size of the resulting map array is 3. The Y size is the sum of all the x_size's.

N_gamma defaults to 1.0. Values for gamma can be inserted anywhere in the argument list and will affect subsequent arguments only.

(merge-display 'dis_display ['dwin_window] ['dwpt_upper-left] [*has-zooms* '(n_xzoom n_yzoom)] [*has-sizes* '(n_xsize n_ysize)] [*has-orientation* 's_orientation] [*has-plane-map* '((bpn_source-1 bpn_target-1) ...)] [*has-color-map* '((s_source-1 s_target-1) ...)] [*has-map-map* '((s_source-1 s_target-1) ...)]) [LISP Function]

display-plane-map [LISP Global Variable]
display-color-map [LISP Global Variable]
display-map-map [LISP Global Variable]

WHERE: The dwin_window, dwpt_upper-left, n_xzoom, n_yzoom, n_xsize, n_ysize and s_orientation arguments are passed to the *new-window* function to get a window referred to as dwin_window below.

The *has-plane-map* argument maps bitgraph plane names of dis_display onto bitgraph planes of dwin_window. The entries consist of a dis_display plane name followed by a dwin_window plane name. In addition, an entry of the form—

(s_source s_target)

implies entries of the form—

((s_source x_N) (s_target x_N))

for each integer x_N, except for those x_N for which *has-plane-map* already has an entry with bpn_source equal to (s_source x_N).

The *has-color-map* argument maps the names of colors associated with dis_display to colors associated with dwin_window. It is only used when dis_display has a bitgraph array or bitgraph programs, while dwin_window has only an intensity array, and neither bitgraph array or programs. In this case the bitgraph information of dis_display is merged directly into the intensity array of dwin_window and a mapping of colors is possible.

The *has-map-map* argument maps display *has-map* names. This is nessary to pass error checks if dwin_window's parent has a different *has-map* attribute than dis_display, and both have intensity arrays.

The *has-plane-map* argument defaults to the value of the **display-plane-map** global variable, which itself defaults to *nil*. The *has-color-map* argument defaults to the value of the **display-color-map** global variable, which itself defaults to *nil*. The *has-map-map* argument defaults to the value of the **display-map-map** global variable, which itself defaults to *nil*.

SIDE EFFECT: Copies the information in dis_display into dwin_window. The upper left corner of dis_display is copied to the upper left corner of dwin_window. If dis_display is too small, the right or bottom of dwin_window is left unchanged, and if dis_display is too large, its right or bottom is not copied. The intensity array of dis_display is copied into the intensity array of dwin_window, if both intensity arrays exist. The pixels of dis_display are expanded by the zooms of dwin_window when copied.

If dis_display has bitgraph programs, and dwin_window has either bitgraph programs or a bitgraph array or both, each bitgraph plane M of dis_display is logically OR'ed into the bitgraph plane N of dwin_window with the same bitgraph plane name.

If instead dwin_window only has an intensity array, and no bitgraph programs or bitgraph array, the bitgraph planes of dis_display overlay the intensity information in dwin_window. This overlay is done after any intensity information in dis_display is copied into dwin_window. Information from planes with higher numbers in dis_display overlays information from planes with lower numbers. The bitgraph array of dis_display is used if it exists and dwin_window has zooms equal to 1. Otherwise dis_display's bitgraph program is used, and the dis_display plane types are recompiled for dwin_window.

If dis_display and dwin_window both have bitgraph arrays but no bitgraph programs, and if the zooms of dwin_window equal 1.0, the bitgraph planes of dis_display will be logically OR'ed into those of dwin_window as indicated above. Similarly if instead dwin_window has an intensity array but no bitgraph programs or bitgraph array.

It is not an error if only one of dis_display and dwin_window have intensity arrays, or if only one has bitgraph planes. It is an error if dis_display has a bitgraph array and no bitgraph programs, and either dwin_window has bitgraph programs, or dwin_window has zooms not equal to 1 or an upper left corner with non-integer coordinates.

(move-display-cursor-by 'n_xdisplacement 'n_ydisplacement ['dwin_window]) [LISP Function]

(move-display-cursor-by '(n_xdisplacement n_ydisplacement) ['dwin_window]) [LISP Function]

(move-display-cursor-by 'vec_vector ['dwin_window]) [LISP Function]

(move-display-cursor-to 'n_xposition 'n_yposition ['dwin_window]) [LISP Function]

(move-display-cursor-to '(n_xposition n_yposition) ['dwin_window]) [LISP Function]

(move-display-cursor-to 'pt_point ['dwin_window]) [LISP Function]

WHERE: Dwin_window defaults to the value of (*get-current-display-window*).

SIDE EFFECT: Resets the coordinates of the cursor point for the window. *Move-display-cursor-by* adds displacements to the current coordinates, while *move-display-cursor-to* replaces the coordinates. A *nil* displacement or position may be used to indicate the associated cursor coordinate is not to be modified.

(new-plane 'bpn_plane) [LISP Function]

EQUIVALENT TO: (*setf* (*has-plane* (*get-current-display-window*)) bpn_plane).

(new-window ['dwin_window] ['dwpt_upper-left] ...) [LISP Function]

WHERE: Dwin_window defaults to the value of (*get-current-display-window*) and dwpt_upper_left defaults to '(0.0 0.0). The dwin_window and dwpt_upper-left arguments may be given in any order, and may be mixed with *nil* arguments which are ignored.

EQUIVALENT TO:

```
(a-display-window dwin_window
                  has-parent dwin_window
                  has-upper-left dwpt_upper-left
                  ...)
```

except that dwpt_upper-left is offset by the dwin_window *has-cursor* position.

Note that because dwin_window is both the prototype and the parent, the new window will inherit *has-transform* and other attributes from it, while also having its upper left corner, sizes, zooms, and orientation defined in terms of it.

(playback-display 'dis_display ['dwin_window]) [LISP Function]
(playback-display ['dwin_window]) [LISP Function]
(playback-display 'ca_catalog ['dwin_window]) [LISP Function]
(playback-display 's_file-name ['dwin_window]) [LISP Function]

WHERE: Dwin_window defaults to the value of (*get-playback-display-window*).

SIDE EFFECT: This function is used to play back displays stored in a catalog.

If dis_display is given the function is equivalent to—

(*merge-display* dis_display dwin_window)
(*flush-display dwin_window)*

If no argument is given the function is equivalent to—

(*flush-display dwin_window t*)

If ca_catalog is given the function positions to the beginning of the catalog and enters interactive mode wherein it types the prompt 'playback> ' and reads commands. A carriage return reads the next catalog entry and displays it if it is a display, or pretty-prints it otherwise. A lisp expression followed by a carriage return reads the catalog entry whose location equals the lisp expression and displays or pretty-prints it. However, in this case if the entry read has the form—

(*catalog-key* location-used-to-read-entry),

it is replaced by the catalog entry following it before being displayed or pretty-printed. A '.' character lists all the allowable locations if ca_catalog is an index catalog. A '?' character lists help information. And a control-D or the character '$' causes the function to exit.

Giving s_file-name is just like giving the catalog—

(*a-catalog is-index-of* (*a-catalog has-file* 's_file-name)).

(remake-display-maps '(s_map-name [s_monitor ...]) 'at_attribute 'g_value ...) [LISP Function]

(remake-display-maps 's_map-name 'at_attribute 'g_value ...) [LISP Function]

EQUIVALENT TO:

(*dolist* (d (*find-display-maps* '(s_map-name [s_monitor] ...)))
(*a-display-map* d at_attribute g_value ...))

This is useful for resetting attributes like *has-plane-types* and *has-scales* for all display maps matching a given ID.

(write-display 'ca_catalog] ['dis/dwin_display]) [LISP Function]

WHERE: Dis/dwin_display defaults to the value of (*get-current-display-window*). If dis/dwin_display is a display window, it is replaced by its parent display.

Nil arguments are ignored (so any argument may be *nil*).

SIDE EFFECT: If dis_display has no *has-parent*, this function does the same thing as—

(*write-catalog* ca_catalog dis_display).

If dis_display has a *has-parent*, then dis_display is merged into this parent by the *merge-display* function, and then *write-display* is executed on the parent.

A display that has been written into a catalog can be read back from the catalog and redisplayed with the *merge-display* function.

CHAPTER 12

HISTOGRAMS

1. GLOSSARY.

(auto-clip 'lar_array 'x_area 'n_range ['(n_minimum n_maximum) ['n_extension]]) [LISP Function]

RETURNS: Clipping bounds for lar_array in the form of a range specified by a list with two elements:

(n_lower-bound n_upper-bound).

X_area represents a number of points in lar_array, hence an area if lar_array is two dimensional, a volume if three dimensional, etc. Lar_array is histogrammed with a resolution of n_range/10. Then a tight range is chosen as small as possible so that no histogram interval of size n_range (10 histogram points) wholy outside the tight range has as many as x_area points. This implies that if lar_array is thought of as an image, any object with values that are all within n_range of each other and whose values are also wholy outside the tight range must be smaller than x_area points.

During the computation of the histogram points with missing values and points with values outside the range (n_minimum n_maximum) are ignored. Thus the tight range will be inside the range (n_minimum n_maximum).

Then a loose range is computed so that the difference between each loose bound and its associated tight bound is = n_extension times the size of the loose range. The default value of n_extension is 0.10.

Lastly the intersection of the loose range and the range (n_minimum n_maximum) is returned.

(**histogram** 'lar_output 'ar_input-1 'rul_ruler-1 ... 'ar_input-N 'rul_ruler-N) [LISP Function]

WHERE: Where, if there are N ar_input/rul_ruler pairs, lar_output is treated as N dimensional. N must be from 1 through 4. Ar_input-1, ar_input-2, ..., ar_input-N must all be similar. The exponent of lar_output must be 0 or negative.

RETURNS: Lar_output after modifying its elements.

SIDE EFFECT: Adds the N dimensional histogram of ar_input-1, ..., ar_input-N to lar_output.

Each non-missing valued element from an input array is assigned a subscript obtained by applying the inverse of the affine transformation defined by the ruler associated with the input array, and *round*ing the result to the nearest integer.

Each corresponding set of elements in the input arrays is mapped onto N subscripts, and these select an element of lar_output. Ar_input-1 provides the *X-dimension* subscripts; ar_input-2 the *Y-dimension* subscripts; ar_input-3 the *Z-dimension* subscripts; and ar_input-4 the *T-dimension* subscripts. The selected element of lar_output is incremented by 1. If any element in a corresponding set of input elements is missing, or any subscript defined by these elements is out of range, the corresponding set is ignored.

NOTE: The elements in lar_output are *not* initialized to zero.

BUG: Lar_output elements may not have missing values.

(**histogram-of** 'ar_input-1 'rul_ruler-1 ... 'ar_input-N 'rul_ruler-N) [LISP Function]

WHERE: N must be from 1 through 4. Ar_input-1, ar_input-2, ..., ar_input-N must all be similar.

If the range specification (second element) of rul_ruler-M is *nil*, the pair computed by

(*bounds-of-array* ar_array-M)

will be substituted for it.

For example, the ruler (nil nil 2) will produce a histogram with bins of size 2 (the scale), with a lower bound on the first bin (first range bound) slightly smaller than the lower bound of all the elements in the associated array, and with as many bins as needed to ensure that all array elements are assigned to some bin.

As another example, the ruler (128 nil (2)) will produce a histogram with 128 bins (histogram dimension size), a lower bound for the first bin (first range bound) equal to the lower bound of all the elements of the associated array, and a bin size (scale) which is the smallest multiple of 2 that will fit all the array elements into the 128 bins.

RETURNS: A new N dimensional array lar_output with exponent 0 whose elements are computed by passing it and the other parameters to the *histogram* function. Before being passed the rulers have *integerize-ruler* applied to them, and the resulting domain size is used to determine the corresponding dimension size of lar_output.

The *has-ruler* attributes for the lar_output dimensions are set to the rulers used.

CHAPTER 13

EDGES

1. MAKING LINKED EDGE CHAINS

With the functions defined in this package, it is possible to detect and extract edges in a 2-D array, where the edges are defined as the zero-crossings of the convolution with a laplacian-of-gaussians operator or with some other contrast operator.

First, the input array must be convolved with the appropriate kernel; for example, if a1 is the input array, its contrast array dga1 is obtained by

(setq dga1 (convolution-of a1 (del2g-kernel '(2.0 2.0))))

where the widths of the Gaussian kernel, here (2.0 2.0), must be adapted to the desired scale of details. Second, the zero crossings of the contrast array and the slopes at these zero-crossings are estimated and stored in the arrays ea1, sa1, respectively by

(desetq (sa1 ea1) (zero-edges-of dga1 '(2 2)))

At this point, a list lc of chains of linked edge points can be obtained as the value returned by

(setq lc (linked-edges-of ea1 sa1))

Each chain in lc has the attributes *has-point-list* set to the list of points in the edge and *has-strength* to the average strength over the edge, as defined in the strength array sa1.

In addition to the above procedure, it is also possible to generate edge chains with information related to the ranges on each side on the edge. For this purpose, the initial array a1 must first be convolved with a smoothing operator corresponding to the contrast operator used to generate dga1. For example, the smoothed array ga1 is obtained with

(setq ga1 (convolution-of a1 (gaussian-kernel '(2.0 2.0) '(0.0 0.0))))

The list of extended edge chains xlc is then obtained with

(setq xlc (x-linked-edges-of ea1 sa1 ga1)).

In addition to *has-point-list* and *has-strength*, the chains in xlc have their *has-minimum-ranges* attribute set to the list of ranges on the lower side of the edge, corresponding to each point in *has-point-list*; the *has-min-median* is set to the median of the values in the above list. Finally, the *has-maximum-ranges* and *has-max-median* attributes are set to corresponding values.

2. ARRAY SIZES, OFFSETS, ETC.

The sizes of the various arrays described above are briefly discussed in this section. Let the sizes of the input array a1 be (N M), and the kernel sizes be (K K). Note first that (K K) is not equal to (2 2) in the above example, as (2 2) determines the side of the main lobe of the laplacian-of-gaussian, not the size of the kernel array; with the default *kernel-cutoff*, K=12 in this example. The array dga1 obtained with the function *convolution-of* has sizes (N+1 M+1); the result array sizes are larger than the input by one element whenever the kernel sizes are even. Furthermore, individual elements of dga1 must be considered as offset by (0.5 0.5) from the elements of a1. In order to obtain dga1, the function *convolution-of* internally expanded the array a1 to sizes (N+K M+K).

The arrays sa1 and ea1 have sizes (N M) and (N M 2) respectively. The function *zero-edges-of* always expects its input array to be offset by half-pixels in each dimension, and outputs arrays with sizes one less than the input sizes, and with no offsets. In order to obtain the final results, the input array to *zero-edges-of*, dga1, which has sizes (N+1 M+1), is internally expanded to sizes (N+2W-1 M+2W-1), where W denotes the value of each width argument to the function *zero-edges-of*. Reasonable values for the width arguments are integers close to the widths of the kernel used to generate dga1.

In the case of the extended edge chains, the minimum and maximum ranges are estimated as the minimum and maximum on a 5x5 box in ga1 around each edge point. The array ga1 must not be offset by half-pixels, so that the convolution kernel used for ga1 is obtained by explicitly forcing the optional offsets to (0 0). Although the initial array a1 itself could be supplied as the third argument to the *x-linked-edges-of* function, using the smoothed array ga1 is more compatible with the detection of edges as zero crossings of dga1 which is equivalent to the laplacian of ga1.

3. GLOSSARY.

(zero-edges 'ar_edges 'lar_input ['x_resolution]) [LISP Function]

WHERE: The element type of ar_edges is *a-char*.

There are integers x_xwidth $\geq$ 1 and x_ywidth $\geq$1 such that the X and Y dimension sizes of ar_edges are 2*x_xwidth-1 and 2*x_ywidth-1 less than the corresponding sizes of lar_input. The Z dimension size of ar_edge is exactly 2.

X_resolution defaults to the value of the ****default-edge-resolution**** variable, which itself defaults to 8.

RETURNS: Ar_edges after its elements are set.

SIDE EFFECT: Zero crossing edges are found in lar_input and stored in ar_edges. Ar_input should be the convolution of an image with a difference of gaussians or other contrast operator. Its zero contours are the edges to be found.

Each 2×2 box in lar_input is checked to see if it contains an edge (0 contour). If it contains 1, that is output into ar_edges according to the scheme below. If it contains 0 or more than 1, ar_edges is marked to indicate the absence of an edge for the box. The rational for the more than 1 case is that the edges involved would have nearly zero strength (gradient in lar_input at the edge location) and should not therefore be treated as edges.

The 2×2 lar_input boxes are mapped onto R×R point boxes where R is the resolution. If R were 8, the box would have 32 boundary points numbered as follows:

0	1	2	3	4	5	6	7	8
31								9
30								10
29								11
28								12
27								13
26								14
25								15
24	23	22	21	20	19	18	17	16

The ar_edges elements with coordinates (X Y 0) and (X Y 1) receive a pair of box boundary numbers from the 2X2 ar_input box whose upper left corner subscripts are

(X+x_xwidth-1 Y+x_ywidth-1).

This pair of points define a straight line which is the edge within the box. The pair is chosen so that going from the (X Y 0) boundary point to the (X Y 1) boundary point places the more intense part of the image on the righthand side.

If there is no edge in the ar_input box, a pair of SAT_CMISSING values is output to the two ar_edges elements.

To avoid redundancy edges equal to the right or bottom edge of the box are surpressed (they will be the left or top edge of an adjacent box. Edges are also surpressed if their two boundary points are equal (i.e. 0 length edges, which would only occur at the box corners).

(zero-edges-of 'ar_input '(x_xwidth x_ywidth) [LISP Function]
['x_resolution])

WHERE: X_xwidth and x_ywidth should be the widths of the difference of gaussians or similar kernel convolved with the original image to produce ar_input.

RETURNS: A list (lar_output lar_edges) of two arrays as produced by *zero-edge-strength* and *zero-edges*. *Prepare-array* is applied to ar_input.

WARNING: The two arrays returned will not be protected against garbage collection unless they are immediately stored in variables by *desetq* or the equivalent.

(**zero-edge-strength-of** 'lar_output 'lar_input 'lar_work 'ar_edges) [LISP Function]

WHERE: The X and Y dimension sizes of lar_output, lar_work, and lar_input are the same, and there are integers x_xwidth ≥ 1 and x_ywidth ≥ 1 such that these sizes are larger than the X and Y sizes of ar_edges by 2*x_xwidth-1 and 2*x_ywidth-1. Ar_edges has been computed from ar_input by using the *zero-edges* function. Bfa_work is a temporary work area.

X_xwidth and x_ywidth should be the widths of the difference of gaussians or similar kernel convolved with the original image to produce ar_input.

RETURNS: A slice of lar_output which has the same X and Y dimension sizes as ar_edges.

SIDE EFFECT: For each element with subscripts (X Y) in lar_output, the maximum and minimum are found of the 2*x_xwidth $\times$ 2*x_ywidth box in ar_input with upper left corner subscripts (X Y). The difference, the maximum minus the minimum, is output to the lar_output element if the corresponding element of ar_edge has an edge. This is a measure of the strength of that edge. If there is no edge, 0 is stored in the lar_output element. The corresponding ar_edge elements have subscripts (X Y ...).

The theory is that the strength is roughly the size of the gradient across the edge, and this can be measured roughly by the maximum minus the minimum on a box.

CHAPTER 14

LINEAR FIT

1. GLOSSARY.

(box-linear-fit 'lar_output 'lar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE:

lar_output X dimension size =
(lar_input X dimension size - x_xsize + x_xstep) / x_xstep

and similarly for the Y dimension. The Z dimension size of lar_output must be 4.

The exponents of lar_output and lar_input must be identical.

RETURNS: Lar_output after setting its elements.

SIDE EFFECT.: For each (X Y) coordinates in lar_output, a linear fit is done of all the points in the (x_xsize x_ysize) box with upper lefthand corner (X*x_xstep Y*x_ystep) in lar_input. The following linear fit parameters are recorded in the elements (X Y Z) of lar_output:

Z = 0 Constant.

Z = 1 X Derivative.

Z = 2 Y Derivative.

Z = 3 Standard Deviation.

The equation of fit is

value at (X1 Y1) = Constant +
(X Derivative) * (X1 - X * x_xstep - (x_xsize - 1) / 2)
(Y Derivative) * (Y1 - Y * x_ystep - (x_ysize - 1) / 2)

where
(X*x_xstep+(x_xsize-1)/2 Y*x_ystep+(x_ysize-1)/2)

is the center of the box.

The constant, derivatives, and standard deviation are recorded as missing if there are fewer than 3 non-missing values in the box, or if the non-missing points lie in a straight line. If there are 4 non-missing values not on a straight line, just the standard deviation is recorded as missing.

(box-linear-fit-of 'ar_input '(x_xsize x_ysize)
'(x_xstep x_ystep)) [LISP Function]

RETURNS: An array lar_output whose elements are set by calling *box-linear-fit* with lar_output, ar_input, (x_xsize x_ysize), and (x_xstep x_ystep) as arguments. *Prepare-array* is run on ar_input to change its numeric type and expand it by (x_xsize-1 x_ysize-1) if appropriate.

CHAPTER 15

TEXTURE

1. GLOSSARY.

(box-horizontal-total-variation 'lar_output 'lar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE:

lar_output X dimension size =
(lar_input X dimension size - x_xsize + x_xstep) / x_xstep

and similarly for the Y dimension.

The exponents of lar_output and lar_input must be identical.

x_xsize and x_ysize must be greater than 1.

RETURNS: Lar_output after setting its elements.

SIDE EFFECT.: For each (X Y) coordinates in lar_output, the normalized horizontal total variation is computed for all the points in the (x_xsize x_ysize) box with upper lefthand corner (X*x_xstep Y*x_ystep) in lar_input. The computed quantity is the sum of absolute values of differences between pairs of horizontal neighbors in the box, divided by the total number of differences involved.

BUG: Cannot handle missing values.

(box-horizontal-total-variation-of 'ar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE: x_xsize and x_ysize must be greater than 1.

RETURNS: An array lar_output whose elements are set by calling *box-horizontal-total-variation* with lar_output, ar_input, (x_xsize x_ysize), and (x_xstep x_ystep) as arguments. *Prepare-array* is run on ar_input to change its numeric type and expand it by (x_xsize-1 x_ysize-1) if appropriate.

(box-minimum-total-variation 'lar_output 'lar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE:

lar_output X dimension size =
(lar_input X dimension size - x_xsize + x_xstep) / x_xstep

and similarly for the Y dimension.

The exponents of lar_output and lar_input must be identical.

x_xsize and x_ysize must be greater than 1.

RETURNS: Lar_output after setting its elements.

SIDE EFFECT: For each (X Y) coordinates in lar_output, the normalized minimum total variation is computed for all the points in the (x_xsize x_ysize) box with upper lefthand corner (X*x_xstep Y*x_ystep) in lar_input. The computed quantity is the minimum of the normalized horizontal total variation and vertical total variation in the box, where the normalized horizontal total variation is the sum of absolute values of differences between pairs of horizontal neighbors in the box, divided by the total number of differences involved, and the other variation is similarly defined for vertical neighbors.

BUG: Cannot handle missing values.

(box-minimum-total-variation-of 'ar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE: x_xsize and x_ysize must be greater than 1.

RETURNS: An array lar_output whose elements are set by calling *box-minimum-total-variation* with lar_output, ar_input, (x_xsize x_ysize), and (x_xstep x_ystep) as arguments. *Prepare-array* is run on ar_input to change its numeric type and expand it by (x_xsize-1 x_ysize-1) if appropriate.

(box-standard-deviation 'lar_output 'lar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE:

lar_output X dimension size =
(lar_input X dimension size - x_xsize + x_xstep) / x_xstep

and similarly for the Y dimension. The Z dimension size of lar_output must be 2.

The exponents of lar_output and lar_input must be identical.

RETURNS: Lar_output after setting its elements.

SIDE EFFECT: For each (X Y) coordinates in lar_output, the mean and standard deviation are computed for all the points in the (x_xsize x_ysize) box with upper lefthand corner (X*x_xstep Y*x_ystep) in lar_input. These parameters are recorded in the elements (X Y Z) of lar_output:

Z = 0   Mean.

Z = 1   Standard Deviation.

The mean and standard deviation are recorded as missing if there are no non-missing values in the box. If there is only 1 non-missing value, just the standard deviation is recorded as missing.

(**box-standard-deviation-of** 'ar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

RETURNS: An array lar_output whose elements are set by calling ***box-standard-deviation*** with lar_output, ar_input, (x_xsize x_ysize), and (x_xstep x_ystep) as arguments. *Prepare-array* is run on ar_input to change its numeric type and expand it by (x_xsize-1 x_ysize-1) if appropriate.

(**box-vertical-total-variation** 'lar_output 'lar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE:

lar_output X dimension size =
(lar_input X dimension size - x_xsize + x_xstep) / x_xstep

and similarly for the Y dimension.

The exponents of lar_output and lar_input must be identical.

x_xsize and x_ysize must be greater than 1.

RETURNS: Lar_output after setting its elements.

SIDE EFFECT.: For each (X Y) coordinates in lar_output, the normalized vertical total variation is computed for all the points in the (x_xsize x_ysize) box with upper lefthand corner (X*x_xstep Y*x_ystep) in lar_input. The computed quantity is the sum of absolute values of differences between pairs of vertical neighbors in the box, divided by the total number of differences involved.

BUG: Cannot handle missing values.

(**box-vertical-total-variation-of** 'ar_input '(x_xsize x_ysize) '(x_xstep x_ystep)) [LISP Function]

WHERE: x_xsize and x_ysize must be greater than 1.

RETURNS: An array lar_output whose elements are set by calling *box-vertical-total-variation* with lar_output, ar_input, (x_xsize x_ysize), and (x_xstep x_ystep) as arguments. *Prepare-array* is run on ar_input to change its numeric type and expand it by (x_xsize-1 x_ysize-1) if appropriate.

APPENDIX A

SKETCH INDEX

APPENDIX B

CONFIGURATION

1. DESCRIPTION The files in this package are those likely to be changed when moving from one hardware/software operating system to another. For example, all names of directories outside SKETCH are in the files of this package.

When porting SKETCH to a new system, the source files of this package should always be read and modified as necessary. Some of these files are-

sco_load.l	Loaded into SKETCH evaluators, but not compilers.
sco_compile.l	Loaded into SKETCH compilers, but not evaluators.
sco_common.l	Loaded into both SKETCH evaluators and compilers.
sco_global.h	Same as FRANZ h/global.h, with additions to keep lint happy.
sco_defs1.mk	Included in every SKETCH makefile. See MAKING FILES Appendix.

The names defined in these files are parts of other packages, and are therefore documented elsewhere.

Some of the contents of these files may be overridden by *top-level-rc-riles*, which are usually files with the name *sketch.rc* or *sketchcom.rc* in the current directory, parent directories of the current directory, or users home directory. See *top-level-rc-files*.

APPENDIX C

MAKING FILES

1. MAKING TARGET FILES FROM SOURCE FILES. The SKETCH *make* package facilitates the making of files from other files, for example the making of foo.o from foo.c. All that is necessary to enable SKETCH *make* is to—

(1) Create a directory for the package (set of related functions) you are writing. This is called the package directory. Usually several related package directories are subdirectories of a common directory called a global directory. The global directory has to be built properly, but this will usually have already been done: see GLOBAL DIRECTORIES below.

(2) Put an executable (x permission set) file named *make* in this directory which has the form—

```
#!/bin/csh -f
if (-r csh.rc) then
        source csh.rc
else if (-r ../csh.rc) then
        source ../csh.rc
else
        source ../../csh.rc
        endif
exec psearch smk/smk_make.sh $argv:q
```

A typical value for ../csh.rc in the global directory is—

```
set path=(. /u1/walton/sketch4/users /sketch/sketch4b/ll/sun3.5 \
        /sketch/sketch4b/ll/sun3.5/tps /usr/local/bin /usr/ucb /usr/bin /bin)
rehash
setenv CPP_PATH "-I/u1/walton/sketch4/users -I/sketch/sketch4b/ll/sun3.5"
setenv COMPUTER_TYPE sun3
```

This combination of *make* and *csh.rc* establishes a path of global directories that may be searched for names of the form—

<package_name>/<filename>.

Psearch (which is part of the TPS system: see TEAM PROGRAMMER SYSTEM below) performs such a search for the *smk/smk_make.sh* program, which executes the *make* command for a SKETCH package directory.

(3) Put into the package directory a file named *makefile.mk* which might, for example, be the following—

```
CFILES=foo.c
LFILES=bar.l
OFILES=foo.o bar.o
```

This file defines *make* macros named CFILES and OFILES. Macro definitions for *make* consist of the macro name, followed by an equal sign, followed by the character string to which the macro is being defined, followed by the end of line (continuation across lines is allowed by using the backslash \ before the line-ends that are to be ignored).

CFILES is a list of all the *.c* source files in the package; LFILES a list of all *.l* source files; and OFILES a list of all the *.o* target files. It is the job of *make* to make target files, which do not initially exist, from source files, which are typically text files edited by the programmer.

(4) You may now make the target files from the source files by UNIX commands such as—

make foo.o
make bar.o
make foo.o bar.o
make all
make

The first two commands make only one of the files, the one designated. The third command makes both files. The last two commands make all the target files listed in *makefile.mk*. This is because making the target *all* is defined to be the same as making all the files listed in the *ALL_FILES* macro, which by default is defined by—

ALL_FILES=$(*LHFILES*) $(*OFILES*) $(*FILES*) \
$(*OTHER_TARGET_FILES*)

Also, *make* by itself, with no target mentioned, is defined to be equivalent to making the target *all*.

Note that *make* macros are invoked by the form—

$(<macro-name>)

The macros LHFILES, FILES, and OTHER_TARGET_FILES, not having been defined in our *makefile.mk*, are defined to be null strings.

The *make* command makes target files by executing UNIX commands. It prints these commands out just before executing them. Thus the following is typical—

```
% make foo.o
rm -f foo.o
cc -O -I/u1/walton/sketch4/users -I/sketch/sketch4b/ll/sun3.5 -c foo.c
chmod a-w foo.o
% make bar.o
rm -f \#bar.\#
sketchcom -q bar.l -o \#bar.\#
chmod a-w \#bar.\#
mv -f \#bar.\# bar.o
```

If you give *make* the *-n* option, as in "*make -n all*", it will print the UNIX commands it would execute to make the target files, but will not execute these commands.

The *make* command also pays attention to some subtleties—

(1) It will not make foo.o if that file exists and has a creation date later than foo.c. Instead it will either print—

'foo.o' is up to date.

if you explicitly asked for foo.o as a target, or do nothing if foo.o was an implicit target (as when *all* is the explicitly asked for target).

Similarly if bar.o exists and has a creation date later than bar.l.

(2) SKETCH *make* will usually change the files it makes to be read-only. This indicates to the *backup*(1tps) program whose use is described under INSTALLING PUBLIC VERSIONS below that the files are not changeable (but may be deleted and replaced), and therefore it is safe to link to these files rather than waste disk space by copying them.

SKETCH *make* will use "*rm -f*" and "*mv -f*" to remove previous copies of target files in order to avoid protection problems when these previous copies are read-only.

(3) SKETCH *make* will take pains to avoid leaving erroneous target files around when there is an error in making a target file. This is why the target file is often made under a pseudonym such as #bar.# above, and *mv*'ed to its final name only after it has been correctly made. (The backslashes \ in front of the #'s in the printed *make* output disappear when the command is read by the UNIX shell: they are necessary because # without \ is a comment character to the UNIX *make*(1) program.)

(4) Simple UNIX commands such as *cc* may not work for SKETCH, and may have to be replaced. For example, in one system a compiler table was too small to handle the large C language *for* loops used in SKETCH, so an alternative version of the compiler was constructed and *cc* was replaced by—

cc -B/sketch/sketch4b/berkeley/vax4.3/pcc/ -t0

2. PACKAGE DEFINING MACROS. To define a package you define macros in *makefile.mk*. Table 1 describes the macros most commonly used for this purpose.

TABLE 1: PART 1

PACKAGE DEFINING MACROS

Macro Name "Default Value"	**Definition**
CHAPTER ""	The chapter number, or appendix letter, of the package documentation chapter or appendix.
PREFIX ""	A several letter prefix that appears a the beginning of most file names and C language global names in the package. Some examples: 'smk', 'sob', and 'sar'. The prefix is separated from the rest of a name by an underline. Some example file names: 'sar_lisp', 'sob_type.l', and 'smk_defs1.mk'.
PACKAGE_DIRECTORY "$*(PREFIX)*"	This is the name of the package directory within its containing global directory. Normally this equals the package prefix, but it does not have to.
TITLE ""	The title of the package's documenting chapter or appendix.

TABLE 1: PART 2

PACKAGE DEFINING MACROS

Macro Name "Default Value"	**Definition**
LISP "$(*SKETCH*)"	The LISP evaluator environment into which *.o* files of this package are to be loaded to produce an evaluator environment for users of this package. Defaults to $(*SKETCH*), which in turn defaults to the SKETCH evaluator program.
DEMO_LISP "$(*LISP*) *−l* $(*PREFIX*)*_load*"	The LISP evaluator environment which is used to make an *.ou* file from a *.l* file by running the *demo* function. Defaults to $(*LISP*) with the evaluator files of this package loaded in by a *−l* switch.
LISZT "$(*SKETCHCOM*)"	The LISP compiler environment into which *.o* files of this package are to be loaded to produce a compiler environment for users of this package. Defaults to $(*SKETCHCOM*), which in turn defaults to the SKETCH compiler program.
DEMO_LISZT "$(*LISZT*) *−l* $(*PREFIX*)*_compile*"	The LISZT compiler environment which is used to make a *.ou* file from a *.cl* file by running the *demo* function. Defaults to $(*LISZT*) with the complier files of this package loaded in by a *−l* switch.
CFILES ""	All *.c* source files in the package. These are the C language files.

TABLE 1: PART 3 PACKAGE DEFINING MACROS	
Macro Name "Default Value"	**Definition**
COMMON_LFILES ""	All *.l* source files in the package that are to be loaded into both the LISP evaluator environment and the LISP compiler environment. See *COMPILE_LFILES*, *LOAD_LFILES*, and *OTHER_LFILES*.
COMPILE_LFILES ""	All *.l* source files in the package that are to be loaded into the LISP compiler environment, but not the LISP evaluator environment. See *COMMON_LFILES*, *LOAD_LFILES*, and *OTHER_LFILES*.
CSFILES ""	All *.cs* source files in the package. These files are written in a combination of assembly language and C macro language.
DEMO_CLFILES ""	All *.cl* source files in the package that are demonstration programs to be run by the compiler $*(DEMO_LISZT)* (and not the evaluator).
DEMO_LFILES ""	All *.l* source files in the package that are demonstration programs to be run by the evaluator $*(DEMO_LISP)*.
DEMO_OUFILES ""	All *.ou* target files in the package that are the output of demonstration programs. These files are made by running *.l* files through the evaluator $*(DEMO_LISP)* or *.cl* files through the the compiler $*(DEMO_LISZT)* and saving the standard output.
DOFILES ""	All *.do* source files in the package. These are documentation files that are processed by *eqn*(1), *pic*(1), *tbl*(1), and *troff*(1) to produce miscellaneous documentation.

TABLE 1: PART 4

PACKAGE DEFINING MACROS

Macro Name "Default Value"	Definition
FFILES ""	All *.f* source files in the package. These are the FORTRAN language files.
FILES ""	All target files with no extension in the package. $*(PREFIX)_lisp* and $*(PREFIX)_liszt* should be listed here if they are required by users of the package. If any other files are listed, explicit instructions for making them must be included in *makefile.mk*.
HFILES ""	All *.h* source files in the package. These are C language files that are *#include*'d in other *.c* files, and do not themselves have any corresponding *.o* file.
INSTALL_ RCFILES ""	All *.rc* source files that are to be installed in the $*(INSTALL_DIRECTORY)* (defined as a UNIX environment variable in *csh.rc*) by the *install* command.
LCFILES ""	All *.lc* source files in the package. These are C language files written with special conventions that make them directly callable by LISP code. These files are also compiled in a special manner.
LHFILES ""	All *.lh* target files in the package. These are C language files that are created from *.l* files that use the *declare-hunk-type* or *declare-vector-type* functions from the SKETCH Objects Package. These files are then *#include*'d into *.c* files.
LIBRARIES ""	A list of the libraries to be searched by *ld*(1) after other files in this package are loaded. Used when making *.ex* files, but not when loading *.o* files into LISP environments.
LNFILES ""	All symbolic link files defined in this package.

TABLE 1: PART 5 PACKAGE DEFINING MACROS	
Macro Name "Default Value"	**Definition**
LOAD_LFILES ""	All *.l* source files in the package that are to be loaded into the LISP evaluator environment, but not the LISP compiler environment. See *COMPILE_LFILES*, *COMMON_LFILES*, and *OTHER_LFILES*.
MAFILES ""	All *.ma* source files in the package. These are documentation files that are processed by *eqn*(1), *pic*(1), *tbl*, and *troff*(1) to produce the package chapter or appendix.
MKFILES "*makefile.mk*"	All *.mk* source files in the package. These are input to *make* commands. In most packages there is only one such file: *makefile.mk*.
OFILES ""	All *.o* target files in the package. These may be loaded into LISP environments, or combined by *ld* to produce executable programs.

TABLE 1: PART 6

PACKAGE DEFINING MACROS

Macro Name "Default Value"	Definition
OTHER_CLFILES ""	All *.cl* source files in the package that are *not* listed in *$(DEMO_CLFILES)*.
OTHER_DEMO_TARGET_FILES ""	All target files *not* made by *all* that are to be made before making *$(DEMO_OUFILES)*. For example, foo.o might be listed if it is to be made from foo.l before a demonstration program is run.
OTHER_INSTALL_SOURCE_FILES ""	All source files that are to be installed in the *$(INSTALL_DIRECTORY)* by the *install* command, and which are not already listed in *$(HFILES)*, *$(SHFILES)*, *$(DEMO_LFILES)*, *$(DEMO_CLFILES)*, or *$(INSTALL_RCFILES)*, and which are not one of the two files *make* or *makefile.mk*.
OTHER_INSTALL_TARGET_FILES ""	All target files that are to be installed in the *$(INSTALL_DIRECTORY)* by the *install* command, and which are not already listed in *$(LHFILES)*, and which are not one of the two files *$(PREFIX)_chap.in* or *COUNT*.

TABLE 1: PART 7 PACKAGE DEFINING MACROS	
Macro Name "Default Value"	**Definition**
OTHER_LFILES ""	All *.l* source files in the package that are *not* to be loaded into either the LISP evaluator environment or the LISP compiler environment. See *COMPILE_LFILES*, *COMMON_LFILES*, and *LOAD_LFILES*.
OTHER_RCFILES ""	All *.rc* source files not listed in *$(INSTALL_RCFILES)*.
OTHER_SOURCE_FILES "*make*"	Source files not listed elsewhere. Usually this consists of shell files with no extension, including the *make* file which is in every SKETCH package directory.
OTHER_TARGET_FILES ""	Target files not listed elsewhere.
SHFILES ""	All *.sh* source files in the package. These are written in either the *sh*(1) or *csh*(1) language.

3. PROGRAM CODE FILE EXTENSIONS. The extension of a file must tell the language or format of the file and its role in the scheme of making files from other files. For this reason SKETCH *make* uses a large number of distinctive file name extensions.

Figure 1 indicates the file extensions and *make* paths involved in making *.o* files that are loaded into *lisp* and its derivatives (*liszt*, *sketch*, and *sketchcom*).

Cc(1), *liszt*(1), *f77*(1), and *as*(1) are standard UNIX programs, and *.c*, *.l*, *.f*, *.s*, and *.o* are standard UNIX file extensions. */lib/cpp* is the C language macro pre-processor, which simply substitutes macros in the input text, but does no other part of C language compilation. Thus *.cs* files may use C macros in assembly language code. The strange combination of programs used to process *.lc* files have almost the same effect as *cc*, but make certain substitutions in the assembly language code (using *fixmask* and *sed*(1)) before it is passed through the C language optimizer */lib/c2*. These substitutions are required to write FRANZ LISP *lambda* functions directly in C on the VAX (currently these substitutions are not done on the SUN).

Figure 2 indicates the file extensions and *make* paths involved in making *.ex* files that are directly executable. This figure is almost the same as Figure 1. One difference is that *cc*(1) is used to make the *.ex* files from the *.o* files. In this use, *cc* merely calls the UNIX loader, *ld*(1), adding a program startup file to be beginning of the list of files loaded or libraries searched, and adding the standard C library to the end of this list. The other difference is the *.l* and *.lc* are made into *.ex* files not by compilation, but rather by loading them into *sketch* or *sketchcom* and doing a *dumplisp*. The *.l* and *.cl* files are supposed to

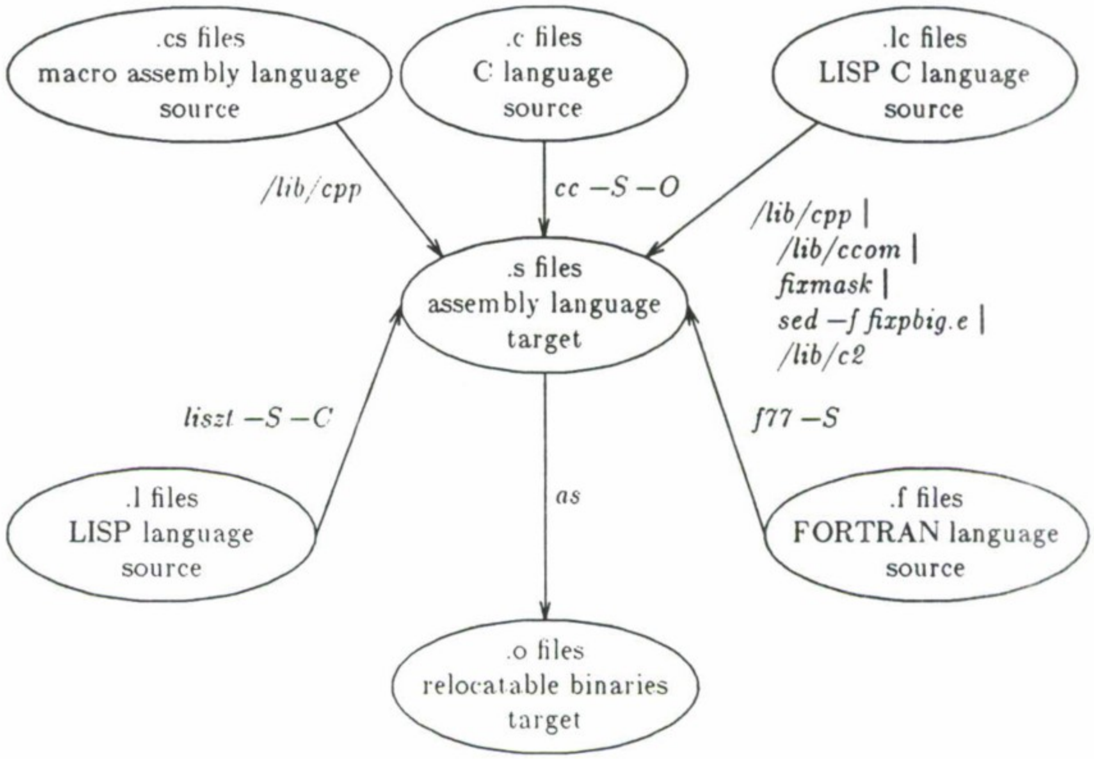

FIGURE 1: MAKING .o FILES

load other pre-compiled files and set global variables before the *dumplisp*.

Note that the autorun facility of *lisp*(1) (the −r option to *liszt*) is not supported, but similar effects can be obtained by writing shell files and using the *-I* and *-E* flags provided by the FRANZ EXTENSIONS package: see *top-level-switches* in that package.

Lint output is produced in the form of *.nt* files just like compilation produces *.s* files. See Figure 3. *Lint.sh* is a shell file that runs the standard UNIX *lint* program but removes certain meaningless warning messages from the output, so that the goal of linting with no messages is reasonable.

C compilations depend upon *.h* files as well as *.c* files. A *.lh* file is similar to a *.h* file, but is made from a *.l* file by the process of Figure 4. See *declare-hunk-type* and *declare-vector-type* in the SKETCH OBJECTS Package for an explanation of what is output into **C-definition-code-port**.

If a *.c* file is changed after its corresponding *.o* file is *made*, then the *.o* file will be remade the next time it or *all* is *made*. However the same is not true if a *.h* or *.lh* file on which the *.o* file also depends is changed. When *.h* or *.lh* files are changed, any *.o*, *.s*, or *.nt* files that depend upon them must be removed by hand. One could avoid this if one wanted to by adding a line such as—

<file1>.o <file1>.s <file1>.nt: <file2>.h <file3>.lh

to *makefile.mk*, thus explicitly giving the dependency involved. However, if a change is made to <file2>.h which will not effect <file1>.o, this line would force the unnecessary

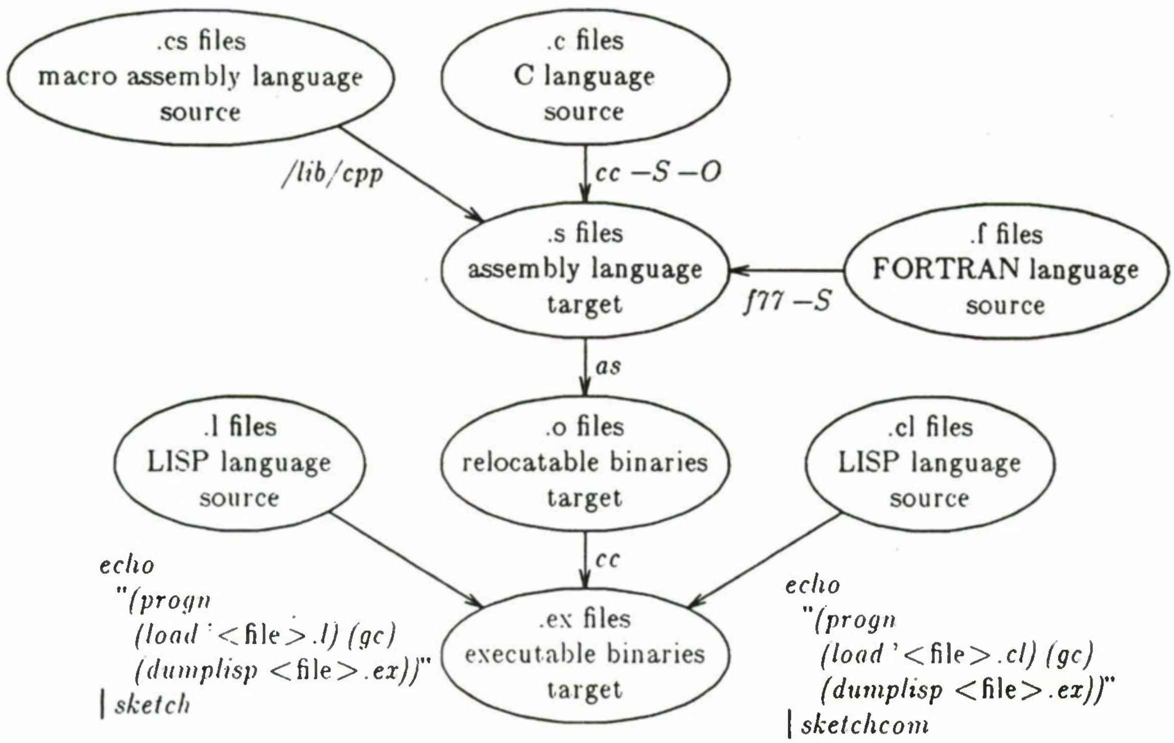

FIGURE 2: MAKING .ex FILES

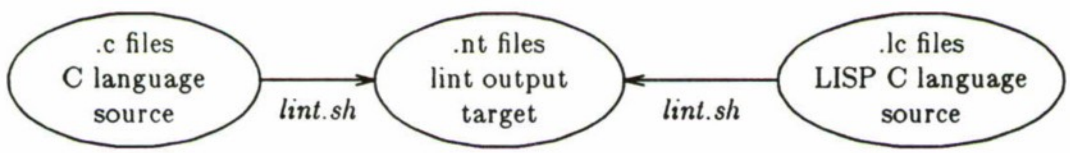

FIGURE 3: MAKING .nt FILES

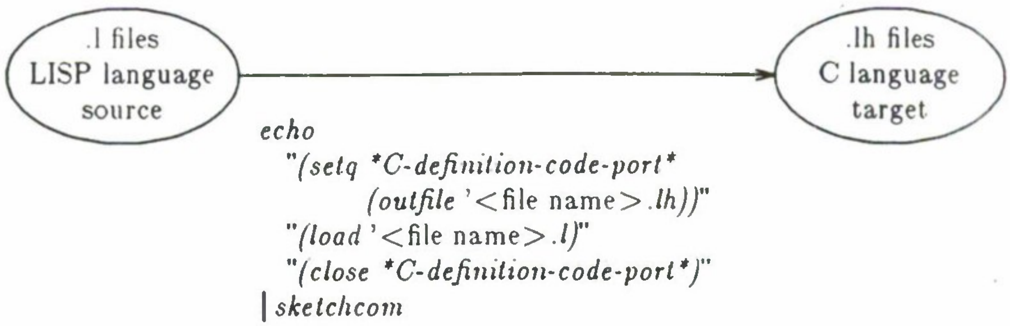

FIGURE 4: MAKING .lh FILES

recompilation of <file1>.o. Since one change to a *.h* file often affects only a few of the *.o* files that depend on the *.h* file, it is usually more efficient to remove *.o* files by hand when a *.h* file affects them.

Lastly, Figure 5 indicates how a *.ou* file is made from an *.l* or *.cl* file. The *.ou* file is the printed output that would result if the *.l* or *.cl* file were typed into *sketch*.

4. SPECIAL TARGETS USED FOR MAINTAINING CODE. Table 2 lists

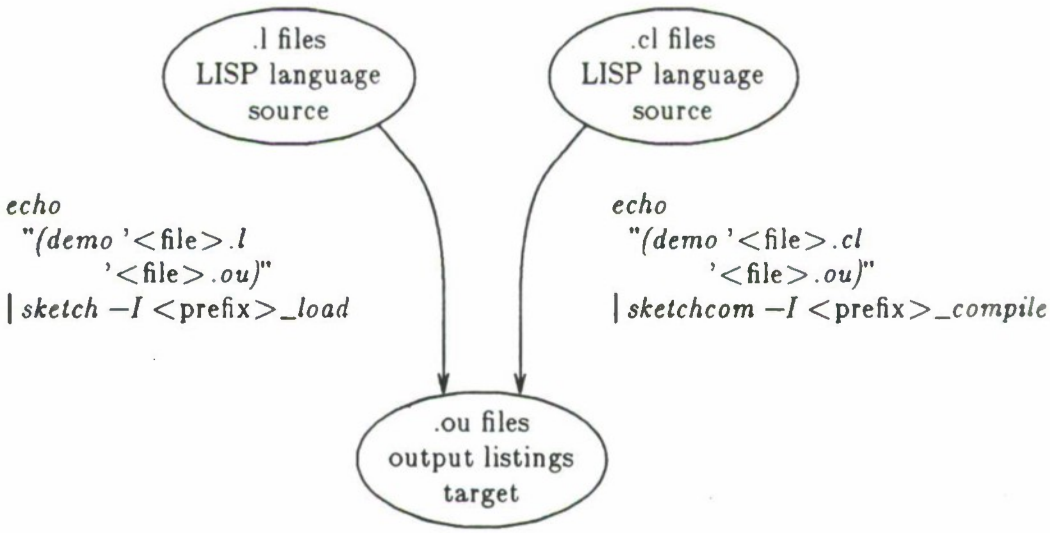

FIGURE 5: MAKING .ou FILES

TABLE 2: PART 1 SPECIAL TARGETS USED FOR MAINTAINING CODE	
all	Makes all code target files, or more explicitly, makes $(*ALL_FILES*) which defaults to "$(*LHFILES*) $(*OFILES*) $(*FILES*) $(*OTHER_TARGET_FILES*)".
all.lhfiles	Makes all *.lh* files, or more explicitly, $(*LHFILES*).
clean	Removes all code target files, and also all documentation targets and other miscellaneous non-source files. More explicitly, removes $(*ALL_FILES*) *WC COUNT* **.ex *.lh *.o *.s *.tr *.nt *.ns *.vs *.in *.sp *.he *.ou* and #*.
ex.clean **lh.clean** **o.clean** **s.clean** **nt.clean** **ou.clean**	Removes only *.ex* files, or only *.lh* files, etc.
compile	Makes $(*PREFIX*)*_compile.o*, which contains all of this package that is to be added to the $(*LISZT*) compiler environment.
count **COUNT**	Makes the *COUNT* file, which gives a line count breakdown of all $(*SOURCE_FILES*), for both code and documentation.
demo	Makes all $(*DEMO_OUFILES*) *.ou* files. First makes the same files made by *all*, and also all the $(*OTHER_DEMO_TARGET_FILES*).
lint	Makes a *.nt* file for each *.c* and *.lc* file (more explicitly, for each file in $(*CFILES*) and $(*LCFILES*)). Makes all $(*LHFILES*) files first.
$(PREFIX)_lisp	Makes the $(*PREFIX*)*_lisp* file, which is the $(*LISP*) evaluator environment with this package added. First makes $(*OFILES*), then loads $(*PREFIX*)*_load.o* into $(*LISP*), and lastly *dumplisp*'s the result into $(*PREFIX*)*_lisp*.

TABLE 2: PART 2 SPECIAL TARGETS USED FOR MAINTAINING CODE	
#lisp	Makes the *#lisp* file by exactly the same procedure as the *$(PREFIX)_lisp* file is made.
list	Outputs all the names of the files in $(*LIST_FILES*), which by default equals $(*SOURCE_FILES*). Each name is on a separate line.
$(PREFIX)_liszt	Makes the *$(PREFIX)_liszt* file, which is the $(*LISZT*) compiler environment with this package added. First makes $(*OFILES*), then loads *$(PREFIX)_load.o* into $(*LISZT*), and lastly *dumplisp*'s the result into *$(PREFIX)_liszt*.
#liszt	Makes the *#liszt* file by exactly the same procedure as the *$(PREFIX)_liszt* file is made.
load	Makes *$(PREFIX)_load.o*, which contains all of this package that is to be added to the $(*LISP*) evaluator environment.
print	Prints all the code source files. Specifically prints *$(LFILES)* *$(CLFILES)* *$(HFILES)* *$(CFILES)* *$(LCFILES)* *$(CSFILES)* *$(FFILES)* *$(MKFILES)* *$(SHFILES)* *$(RCFILES)* and *$(OTHER_PRINT_FILES)*, the last of which defaults to $(*OTHER_SOURCE_FILES*). Also makes and prints the *WC* file, which lists all the files printed and their line, word, and character counts. Files are printed in alphabetical order of their name, except that the WC file is printed first.
print_with_count	Just like *print*, but makes and prints the *COUNT* file instead of the *WC* file. The *COUNT* file is more meaningful, but takes longer to make than *WC*.
release_source	Releases all source files using *release*(1tps).
wc **WC**	Makes the *WC* file, which gives the line count, word count, and byte count of the files printed by *print*.

some special targets used for maintaining code.

5. DOCUMENTATION FILE EXTENSIONS. Figure 6 indicates the file extensions and *make* paths involved in making documentation targets. Each package typically has one *.ma* manual file which is successively made into *.tr*, *.vs*, and *.vo* files to print the package manual. In the first step the glossary, extracted from all the package *$(GLOSSARY_FILES)* (which includes all source files except *.ma* and *.do* files), is appended to the end of the *.ma* file to make the *.tr* file: see *sma_manual.sh* in the

MANUALS Appendix. Also, the chapter title, as defined by the *make* *$(CHAPTER)* and *$(TITLE)* macros, is prepended to the *.ma* file in this first step.

The *.vo* files actually do not exist: making them merely causes the *.vs* files to be printed. Such non-existent target files are called pseudofiles.

Instead of making *.vs* and *.vo* files, one can make *.ns* and *.no* files. The difference is that the former use *ditroff*(1) for phototypesetter like printers, and the later use *nroff*(1) for typewriter like printers.

Also one can make *.he* and *.ho* files instead using *nroff*(1). The *.he* files are like *.ns* *nroff*(1) output files, but contain only the glossary, and have specially formatted section headers that can be extracted by computer programs. They are designed for use by the on-line help facility (which is not yet implemented). The *.ho* pseudofile is used to print the *.he* file, but the only reason for doing this is to check that the file formating is OK (in particular, are some lines too long).

Lastly, one can also make *.sp* files made by running the *.tr* file derived from the *.ma* file through the *spell*(1) program to produce a list of potentially misspelled words.

Besides *.ma* files packages may have miscellaneous documents represented by *.do* files. These can be printed by the same mechanisms as *.ma* files, with the difference that no chapter title or glossary is added, and no *.he* files may be made. Figure 7 depicts this.

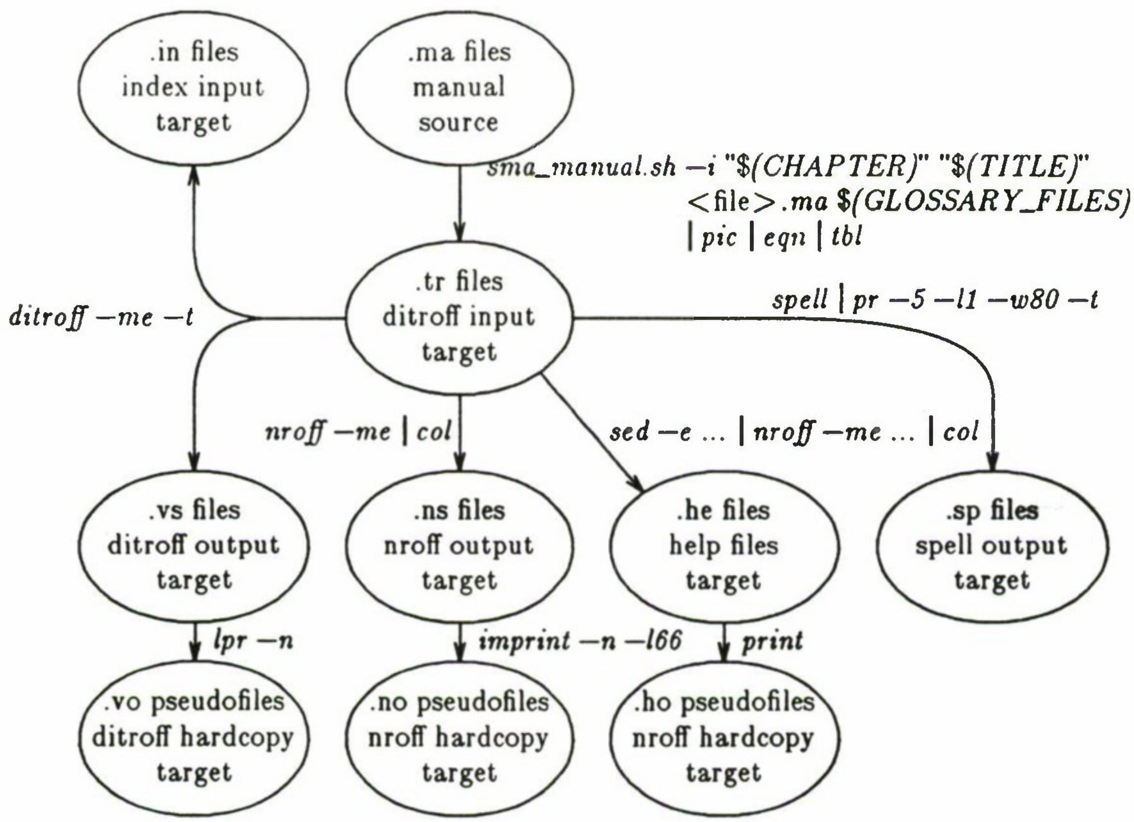

FIGURE 6: FILES MADE FROM .ma FILES

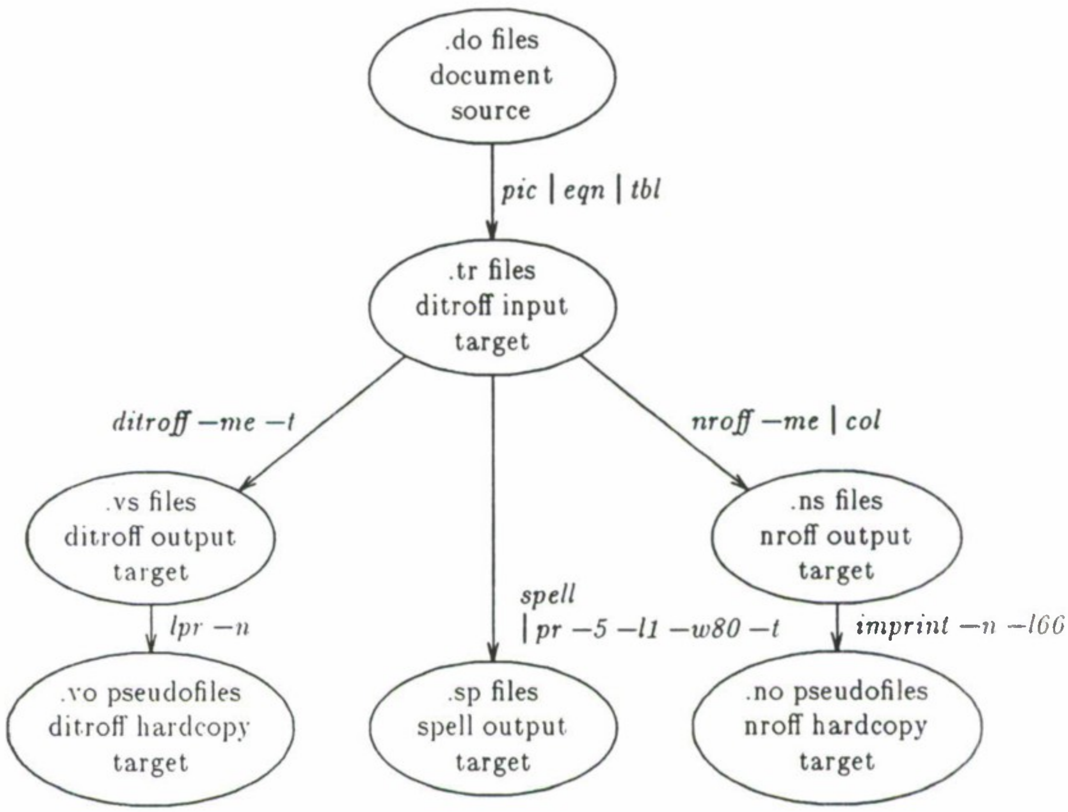

FIGURE 7: FILES MADE FROM .do FILES

6. SPECIAL TARGETS FOR MAINTAINING DOCUMENTATION. Table 3

TABLE 3 SPECIAL TARGETS USED FOR MAINTAINING DOCUMENTAION	
chap	Makes $(*PREFIX*)_chap.vo which prints the package documentation chapter or appendix.
chap.vs	Makes the $(*PREFIX*)_chap.vs file which is the *troff*(1) output file format of the package documentation chapter or appendix. Making this file takes a lot of computer time, but printing is takes little, so often this file is made in background and then printed later.
help	Makes the $(*PREFIX*)_chap.he file which is the package documentation chapter or appendix glossary in a format suitable for use by the on-line *help* command.

lists some special targets used for maintaining documention.

7. THE TEAM PROGRAMMER SYSTEM. The Team Programmer System (TPS) is a set of program development utility programs that are distributed with SKETCH and used by SKETCH. Here we briefly describe the TPS programs used by SKETCH in an essentual way.

TPS has commands to search the directories in the PATH environment variable for a file. The *fsearch* program is used to find the complete name of a file (or each of a list of files) by searching the list of directories given by PATH for the file. E.g., if

PATH=.:/sketch/sketch4b/ll/sun3.5

then

fsearch -rx smk/smk_make.sh sma/sma_index.sh

might return the line

```
/sketch/sketch4b/ll/sun3.5/smk/smk_make.sh \
        /sketch/sketch4b/ll/sun3.5/sma/sma_index.sh
```

The options −rwx may be used to require files to have read, write, or execute privileges to be included in the search.

The *psearch* command does an *fsearch* like action on its first argument, and then calls that first argument as a UNIX program, passing the rest of the *psearch* arguments to that program. Thus one might execute

psearch smk/smk_make.sh chap

Now if there were no slash (/) in the name of the command, the UNIX shell would do the same thing. Unfortunately, the shell does not search the PATH directories if there is a slash *anywhere* in the command name (not just at the beginning). So *psearch* is necessary when there is a slash in the middle of the command name.

TPS also has commands to install files in public locations. The key program is the *backup* program, used as in

backup –D/sketch/sketch4b/ll/sun3.5/sar sar_defs.h sar_load.o

This program takes files (sar_defs.h and sar_load.o) in the current directory and makes copies in another directory (/sketch/sketch4b/ll/sun3.5/sar) which holds the publically accessible versions of these files. The program also makes backups in the public directory of any previous public versions of these files (see the TPS documentation for details). Lastly, the *backup* program is intelligent in two ways. First, if a public version of a file already exists and equals the current directory version of the file, then the program does nothing, neither making a copy of the files or making backups. Second, if a copy is to be made of a read-only file, and the current and public directory are on the same file system, no copy is actually made, but instead the public version of the file is linked to the current directory version. This saves disk space, and is the reason that the SKETCH *make* facility makes all target files read-only.

A companion to the *backup* program is the *release* program, used as in

release –D/sketch/sketch4b/ll/sun3.5/sar sar_defs.h sar_load.o

This program merely destroys all backups of the indicated files that are in the public directory (not backups in the current directory). It also makes the public directory versions of the files read-only (if they are not already such).

The *backup* and *release* programs can be used without the *–D* option to backup and release files in the current directory. The backups are made in the current directory. However, linking of read-only files is never done, and instead *backup* always make a new copy of the file being backed up that is owned and writable by the person running the *backup* program, so that that person can edit the file.

More details about TPS programs are contained in the TPS documentation.

8. INSTALLING PUBLIC VERSIONS.

9. GLOBAL DIRECTORIES. A global directory is a directory that contains package subdirectories. For example, a global directory might contain subdirectories named smk and sma for the MAKING FILES and WRITING MANUALS packages, respectively.

A global directory should contain three files: *csh.rc*, *sketch.rc*, and *sketchcom.rc*. These files establish environment for the *make*, *sketch*, and *sketchcom* programs, respectively.

These *.rc* files establish a sequence of global directories recorded in various directory search paths. If a file is not found in the current global directory, then the next global directory in the path is searched, and so forth. For example, *csh.rc* might be

```
set path=(. /u1/walton/sketch4/users /sketch/sketch4b/ll/sun3.5 \
        /sketch/sketch4b/ll/sun3.5/tps /usr/local/bin /usr/ucb /usr/bin /bin)
rehash
setenv CPP_PATH "-I/u1/walton/sketch4/users -I/sketch/sketch4b/ll/sun3.5"
```

if the global directory were /u1/walton/sketch4/users. If a file is not in this directory, then /sketch/sketch4b/ll/sun3.5 is searched.

Suppose the /sketch/sketch4b/ll/sun3.5 directory contains an sed package subdirectory. If a copy of this subdirectory were made in /u1/walton/sketch4/users and modified, then all jobs run within the /u1/walton/sketch4/users directory and its subdirectories would use the modified sed package, while all jobs within /sketch/sketch4b/ll/sun3.5 and its subdirectories would use the original sed package.

Thus it is possible to build a tree of different versions of SKETCH.

The *PATH* UNIX environment variable (which is set from the *path csh* variable) not only lists the global directory sequence, but also lists other places to obtain program files, such as '.' for the current directory and /sketch/sketch4b/ll/sun3.5/tps for the TPS programs (see THE TEAM PROGRAMMER SYSTEM above).

A typical *sketch.rc* file is

```
(setq lisp-library-directory
      (tilde-expand '~/sketch/sketch4b/ll/sun3.5/lisp/lisplib))
(sstatus data-search-path (|.| /msmi1/data4 /sketch/sketch4b/ll/sun3.5))
(sstatus catalog-search-path (|.| /msmi1/exper4 /sketch/sketch4b/ll/sun3.5))
(sstatus cache-search-path (|.| /msmi1/data4 /sketch/sketch4b/ll/sun3.5))
(sstatus font-search-path (|.| /sketch/sketch4b/ll/sun3.5/vfonts
                               /sketch/sketch4b/ll/sun3.5/fonts))
(sstatus load-search-path (|.| /msmi1/exper4 /u1/walton/sketch4/users
                               /sketch/sketch4b/ll/sun3.5
                               /sketch/sketch4b/ll/sun3.5/lisp/lisplib))
(load 'display.rc)
```

Here /msmi1/exper4 is a global directory in which experiment jobs, in the form of interpreted LISP programs, are placed, and /msmi1/data4 is a global directory holding binary data files: e.g. arrays. Files in all these global directories must be referenced by giving a package subdirectory name, which serves to identify the directory that contains the file and makes the order in which global directories are listed in *sketch.rc* unimportant (unless a package in one global directory has the same subdirectory name as a package in another). Thus a data file might be referred to as ex1/jul81/t..1.ar which would be inside the jul81 dataset in the ex1 package subdirectory of one of the various global directories (/msmi1/data4 probably).

The display.rc file mentioned at the end of *sketch.rc* initializes the display system: see the DISPLAY chapter.

Sketchcom.rc is usually symbolically linked (by *ln -s*) to *sketch.rc*, so that the compiler sees the same initializing file as the evaluator.

A global directory has a *makefile.mk* file which can contain most of the same macros as a package directory *makefile.mk*: see Table 1. Macros that are not used in a global directory are *CHAPTER*, *PREFIX*, *PACKAGE_DIRECTORY*, and *TITLE*. Instead or these, the macros of Table 5 may be defined in a global directory.

10. TARGETS IN GLOBAL DIRECTORIES. Making a target in a global directory is like making a target in a package directory, except that targets that make files whose name includes $(*PREFIX*) or $(*PACKAGE_DIRECTORY*) cannot generally be made in the global directory. However, the following targets work in the global directory—

TABLE 5

GLOBAL DIRECTORY DEFINING MACROS

Macro Name "Default Value"	Definition
INDEX_APPENDIX ""	The appendix letter of the index of all the packages in $(*PACKAGES*) which is built by the *global_index.<xx>* targets.
INDEX_TITLE ""	The title of the index of all packages in $(*PACKAGES*) which is built by the *global_index.<xx>* targets.
LINK_DIRECTORY "*<illegal_value>*"	The global directory into which $(*LINK_FILES*) are linked by the *link make* target. This can be defined for package directories too, but generally is not.
PACKAGES ""	The list of package subdirectories that are to be processed by the <xxx>*.packages* targets of this global directory. These subdirectories need not be in the global directory itself as long as they are in the path of global directories.

clean_install *backup_install* *release_install*	With $(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*) replaced by $(*INSTALL_DIRECTORY*).

Making a target named <target>*.packages* in the global directory makes <target> in each package subdirectory listed in $(*PACKAGES*), for each package <target> whose name does not include a package $(*PREFIX*).

The following <target>*.packages* targets pass the indicated *make* macro values to the package *make* commands when they execute—

backup_install.packages	INSTALL_DIRECTORY=$(*INSTALL_DIRECTORY*)
release_install.packages	INSTALL_DIRECTORY=$(*INSTALL_DIRECTORY*)
clean_install.packages	INSTALL_DIRECTORY=$(*INSTALL_DIRECTORY*)
list.packages	LIST_FILES=$(*PACKAGE_LIST_FILES*)
link.packages	LINK_FILES=$(*PACKAGE_LINK_FILES*) LINK_DIRECTORY= $(*LINK_DIRECTORY*)/<package-directory>

Actually, the directory names are converted to non-relative form before they are passed. The <package-directory> refers to the element of the $(*PACKAGES*) macro value.

Sometimes it is desirable for *demo.packages* to pass the $(*DEMO_LISP*) and $(*DEMO_LISZT*) values to the package *make*, so there is a variant named *global.demo.packages* that does this by passing-

DEMO_LISP=$(*DEMO_LISP*)
DEMO_LISZT=$(*DEMO_LISZT*)

When defining $(*PACKAGE_LIST_FILES*) and $(*PACKAGE_LINK_FILES*) one must double the dollar signs. For example—

make 'PACKAGE_LIST_FILES=$$(HFILES)' list.packages

TABLE 6

SPECIAL TARGETS

USED IN GLOBAL DIRECTORIES

global_count **GLOBAL_COUNT**	Makes the file *GLOBAL_COUNT* by combining the *COUNT* file and all the <package>/*COUNT* files for every <package> in $*(PACKAGES)*. You must make *count.packages* first: it is not done automatically.
global.demo.packages	Like *global.packages*, but passes the global directory definition of $*(DEMO_LISP)* and $*(DEMO_LISZT)* to the packages to produce the demo *.ou* files. This is used in public global directories where the binary files required to use the package defintions of $*(DEMO_LISP)* and $*(DEMO_LISZT)* are not available.
global_index.tr **global_index.vs** **global_index.vo**	Makes the file *global_index.tr* by combining all the index files <package>/<prefix>_chap.in for every <package> in $*(PACKAGES)*. Other documentation target files are made from *global_index.tr* in the usual ways. You must make *index.packages* first: it is not done automatically.
manual	Prints the entire manual by making the following in order: *title.vo, index.packages, global_index.vo, chap.packages.*

Table 6 gives some other special targets that can be used in global directories only.

11. DEFINING UNIX PROGRAMS. The UNIX programs invoked by *make* and the flags passed to these programs are mostly defined by macros. Thus they can be replaced and modified by redefining their macros in the *makefile.mk*.

For example, every time a LISP evaluator environment is needed, SKETCH *make* uses $(*LISP*), which is usually defined by the default definitions—

LISP=$(SKETCH)
SKETCH=sketch

If we put—

LISP=lisp

into *makefile.mk*, we would get the *lisp*(1) program instead, whenever a LISP evaluator environment was needed by *make*.

The LISP evaluator environment, $(*LISP*), and the LISP compiler environment, $(*LISZT*), are the programs most commonly redefined in *makefile.mk*. The default definitions of various programs are in the *sco_defs1.mk* file of the *sco* (configuration) package, and may need to be changed when SKETCH is ported to a new computer.

Program flags are defined by macros separate from the program. Usually a program like $(*LISZT*) will have a related set of flags named by appending "*_FLAGS*" to the program macro name. Thus $(*LISZT_FLAGS*), which defaults to "*-q*" to suppress verbose output from the LISP compiler.

Macro definitions may be overridden by providing new definitions as arguments to the *make* UNIX command. For example,

make 'LISZT_FLAGS=' ...

will *make* target files with *liszt* flags defined to be the null string, thus causing verbose *liszt* compiler output.

Not all program flags are controlled by this mechanism. For example the *-S* flag to *cc* that makes a *.s* file from a *.c* file is not: it is controlled by the kind of file being made.

Table 7 lists the program and flag macros most commonly used while making target program code files. *−D* flags for the C macro preprocessor are put into $(*CPP_FLAGS*), while libraries to be searched by the loader are put into $(*LIBRARIES*).

There are a very large number of UNIX program macros not in Table 7: see the glossary for the program you want to modify.

12. HITLIST.

(1) Finish tutorial documenation.

(2) Check to be sure make index makes global index before local indexes: there may be a bug here? it may also ignore existing index.vs?

(3) Be sure all global directory targets are documented.

(4) Be sure count and index can be made from global install directory.

(5) Make global #lisp and #liszt (via environments, whatever that means?).

TABLE 7: PART 1 PROGRAM AND FLAG MACROS		
Program Macro "default"	**Flags Macro** "default"	**Use**
$(AS) *"as"*	$(AS_FLAGS) ""	Assemble *.s* files to make *.o* files.
$(CC) *"cc"*	$(CC_FLAGS) *"–O"* $(CPP_FLAGS) "" $(LD_FLAGS) "" $(LIBRARIES) ""	C Compiler. The flags are separated into 3 groups: compiler proper (cc), macro preprocessor (cpp), and loader (ld). *$(LIBRARIES)* is like *$(LD_FLAGS)*, but placed after the list of file names passed to the loader, instead of before that list.
$(LINT) *"psearch smk/smk_lint.sh"*	$(LINT_FLAGS) "" $(CPP_FLAGS) "" $(LINT_LIBRARIES) *"$(LIBRARIES)"*	Lint. The flags are separated into 2 groups: lint proper (lint) and the macro preprocessor (cpp). *$(LINT_FLAGS)* and *$(LINT_LIBRARIES)* are both passed to lint proper, but the first goes at the beginning of the argument list and the second goes at the end. *Smk/lint.sh* is a special version of *lint* that gets rid of certain meaningless warning messages.

TABLE 7: PART 2 PROGRAM AND FLAG MACROS		
Program Macro "default"	**Flags Macro** "default"	**Use**
$(LISP) "*$(SKETCH)*"	$(LISP_FLAGS) ""	Lisp evaluator environment.
$(DEMO_LISP) "*$(LISP) −I $(PREFIX)_load*"		Lisp evaluator environment used to make *.ou* files from *.l* files via the *demo* function.
$(LISZT) "*$(SKETCHCOM)*"	$(LISZT_FLAGS) "*−q*"	Lisp compiler environment.
$(DEMO_LISZT) "*$(LISZT) −I $(PREFIX)_compile*"		Liszt compiler environment used to make *.ou* files from *.cl* files via the *demo* function.
$(SKETCH) "*sketch*"		SKETCH evaluator environment.
$(SKETCHCOM) "*sketchcom*"		SKETCH compiler environment.

(6) Explain environment use. Make them use .o files. Explain declare (macros t). Explain need to def macros before use.

13. GLOSSARY.

all [MAKE Target]

WHEN MADE: Makes $(*ALL_FILES*).

$(ALL_FILES) [MAKE Macro]

VALUE: A list of all the package files that must be made from other files (does not include intermediate files that may not have to be made). Default value: "$(*LHFILES*) $(*OFILES*) $(*FILES*) $(*OTHER_TARGET_FILES*).

WARNING: These files are all removed by cleaning.

all.lhfiles [MAKE Target]

WHEN MADE: Makes $(*LHFILES*).

$(AS) [MAKE Macro]

VALUE: The name of the UNIX *as*(1) program. Usual default: "*as*".

$(AS_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *as*(1) program. Usual default: "".

$(BACKUP) [MAKE Macro]

VALUE: The name of the TPS *backup*(tps) program. Usual default: "*backup*".

$(BACKUP_FLAGS) [MAKE Macro]

VALUE: The flags for the TPS *backup*(tps) program. Usual default: "".

backup_install [MAKE Target]

VALUE: Installs $(*INSTALL_FILES*) in

$(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*)

by using *backup*(tps) with the *−D* and *−remove* options.

In the global directory, $(*INSTALL_DIRECTORY*) is used in place of

$(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*).

.c [UNIX File Extension]

FILE FORMAT: A C source file. Can be made into an *.ex*, *.o*, *.s*, or *.nt* file (all $(*LHFILES*) are all made first).

$(C2) [MAKE Macro]

VALUE: The name of the UNIX C compiler optimizer program. Usual default: "*/lib/c2*".

.ca [UNIX File Extension]

FILE FORMAT: SKETCH catalog files. See CATALOGS chapter.

$(CC) [MAKE Macro]

VALUE: The name of the UNIX *cc*(1) program. Usual default: "*cc*".

$(CC_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *cc*(1) program. These should not include flags for the macro preprocessor cpp, the loader *ld*(1), and *lint*(1), which are separate. Default value: "*-O*".

$(CCOM) [MAKE Macro]

VALUE: The name of the UNIX C compiler program. Usual default: "*/lib/ccom*".

$(CFILES) [MAKE Macro]

VALUE: A list of all the *.c* files in the package. These are assumed to be source files. Default value: "".

chap [MAKE Target]

WHEN MADE: Makes $(*PREFIX*)*_chap.vo*.

$(CHAPTER) [MAKE Macro]

VALUE: The number (1, 2, 3, etc.) of the package chapter or the letter (A, B, C, etc.) of the package appendix.

chap.vs [MAKE Target]

WHEN MADE: Makes $(*PREFIX*)*_chap.vs* and $(*PREFIX*)*_chap.in*.

.ci [UNIX File Extension]

FILE FORMAT: SKETCH catalog file index. See *has-index-file* under *a-catalog* in the CATALOGS chapter.

Can be made from a *.ca* file. The procedure to do this invokes *make-catalog-index* in $(*SKETCH*). In order for this to work, $(*PAKCAGE_DIRECTORY*) must be the name of the current directory relative to one of the

(*status catalog-search-path*)

directories.

.cl [UNIX File Extension]

FILE FORMAT: A LISZT source file. Can be loaded directly into LISZT by the *load* function, or made into a *.ex* or *.ou* file.

The only proper way to make a *.cl* file into a *.ex* file is directly, without intervening *.s* or *.o* files, in which case the *.cl* file is *load*'ed into $(*LISZT*) and the result *dumplisp*'ed to the *.ex* file.

clean [MAKE Target]
ex.clean [MAKE Target]
he.clean [MAKE Target]
lh.clean [MAKE Target]
nt.clean [MAKE Target]
ns.clean [MAKE Target]
o.clean [MAKE Target]
ou.clean [MAKE Target]
s.clean [MAKE Target]
sp.clean [MAKE Target]
tr.clean [MAKE Target]
vs.clean [MAKE Target]

WHEN MADE: *Clean* removes all non-source files. More explicitely, it removes $(*ALL_FILES*) *WC COUNT* **.ex* **.lh* **.o* **.s* **.tr* **.nt* **.ns* **.vs* **.in* **.sp* **.he* **.ou* and #*.

<xx>*.clean* removes all files with extension ".<xx>".

Clean and <xx>*.clean* are defined with a double colon :: and can therefore be added to by defining new entries such as—

clean::
rm -f ...

WARNING: If you execute—

make clean ... &>make.ou &

you will remove the *make.ou* file before you can look at it.

clean_install [MAKE Target]

VALUE: Removes the directory

$(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*)

with *rm −rf* and remakes it with *mkdir*. Gives the directory group write permission.

In the global directory, $(*INSTALL_DIRECTORY*) is used in place of

$(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*),

and *rm −f* is used in place of *rm −rf*, so that only ordinary files, and not subdirectories, are removed.

Clean_install is defined with a double colon :: and can therefore be added to by defining new entries such as—

```
clean_install::
        rm -rf ...
        mkdir ...
        chmod g+w ...
```

$(CLFILES) [MAKE Macro]

VALUE: A list of all the *.cl* files in the package. These are assumed to be source files. Default value:

"$(*DEMO_CLFILES*) $(*OTHER_CLFILES*)".

$(COL) [MAKE Macro]

VALUE: The name of the UNIX *col*(1) command used to remove reverse line feeds. Usual default: "*col*".

$(COLUMNS) [MAKE Macro]

VALUE: The name of the UNIX command to print a list of words in 5 columns on a terminal. Usual default: "*pr -5 -l1 -w80 -t*". Used, for example, to output spelling errors.

$(COMMON_LFILES) [MAKE Macro]

VALUE: A list of all the *.l* files in the package that are included in both $(*PREFIX*)*_load.l* and $(*PREFIX*)*_compile.l*. These are assumed to be source files. Default value: "".

compile [MAKE Target]

WHEN MADE: Makes $(*PREFIX*)*_compile.o*.

$(COMPILE_LFILES) [MAKE Macro]

VALUE: A list of all the *.l* files in the package that are included in $(*PREFIX*)*_compile.l* but not $(*PREFIX*)*_load.l*. These are assumed to be source files. Default value: "".

$(COMPUTER_TYPE) [MAKE Macro]
$(COMPUTER_TYPE) [UNIX Environment Variable]

VALUE: The type of the computer, either *sun3* or *vax*, on which SKETCH is running. Each compute type has its own set of SKETCH directories, but these may share sources in a common directory: see the ***link make*** target.

Must be set in *csh.rc* files. $(*COMPUTER_TYPE*) is a UNIX Environment variable turned by *make* into a *make* macro.

$(COUNT) [MAKE Macro]

VALUE: The name of the line counting program. Usual default:

psearch sma/sma_count.sh.

count [MAKE Target]
COUNT [MAKE Target]
COUNT [UNIX File Name]

WHEN MADE: $(*COUNT*)'s $(*SOURCE_FILES*) and puts the result into the file named *COUNT*.

$(COUNT_FLAGS) [MAKE Macro]

VALUE: The flags for the line counting program. Usual default: "".

$(CPP) [MAKE Macro]

VALUE: The name of the UNIX *cpp* C macro processor program. Usual default: "*/lib/cpp*".

$(CPP_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX C macro preprocessor *cpp*. Default value: "$(CPP_PATH)".

$(CPP_PATH) [MAKE Macro]
$(CPP_PATH) [UNIX Environment Variable]

VALUE: An argument for *cc* of the form "*–I*directory ...". Gives the directories to be searched for *.h* files.

Must be set in *csh.rc* files. $(*CPP_PATH*) is a UNIX Environment variable turned by *make* into a *make* macro.

.cs [UNIX File Extension]

FILE FORMAT: A macro assembly language source file. Can be made into a *.s*, *.o* or *.ex* file. Is made into a *.s* file by running through the C macro preprocessor, but not the rest of the C compiler.

$(CSFILES) [MAKE Macro]

VALUE: A list of all the *.cs* files in the package. These are assumed to be source files. Default value: "".

csh.rc [UNIX File]

VALUE: A file in the global directory that defines the directories used by *make*. It should have roughly the form—

set path=(<this-directory> <next-directory> ...
<tps-directory> /usr/local/bin /usr/ucb /usr/bin /bin)
rehash
setenv CPP_PATH "-I<this-directory> -I<next-directory> ..."
setenv INSTALL_DIRECTORY <this-directory>/pub
setenv COMPUTER_TYPE <computer_type>

where the chain of directories to be searched for SKETCH program and data files is <this-directory> <next-directory> ..., and the directory in which the public versions of these files are to be installed is usually named pub relative to this directory (but a relative name cannot be used in *csh.rc*). The <computer_type> is typically either sun3 or vax.

demo [MAKE Target]

VALUE: Makes $(DEMO_OUFILES).

$(DEMO_CLFILES) [MAKE Macro]

VALUE: A list of all the *.cl* files in the package that are demonstrations which can be loaded into $*(PREFIX)_liszt* by the *demo* function. These are assumed to be source files. Default value: "".

$(DEMO_LFILES) [MAKE Macro]

VALUE: A list of all the *.l* files in the package that are demonstrations which can be loaded into $*(PREFIX)_lisp* by the *demo* function. These are assumed to be source files. Default value: "".

$(DEMO_LISP) [MAKE Macro]

VALUE: The name of the LISP evaluator program used to make *.ou* files from *.l* files. Usual default:

"$(*LISP*) -I $(*PACKAGE_DIRECTORY*)/$(*PREFIX*)_*load*"

in package directories, and just "$(*LISP*)" in global directories.

$(DEMO_LISZT) [MAKE Macro]

VALUE: The name of the LISZT evaluator program used to make *.ou* files from *.cl* files. Usual default:

"$(*LISZT*) -I $(*PACKAGE_DIRECTORY*)/$(*PREFIX*)_*compile*"

in package directories, and just "$(*LISZT*)" in global directories.

$(DEMO_OUFILES) [MAKE Macro]

VALUE: A list of all the *.ou* files that can be made from demo *.l* files. These files are made by the "*make demo*". Default value: "".

$(DEMO_TARGET_FILES) [MAKE Macro]

VALUE: A list of all the files that should be made before making any file listed in $(*DEMO_OUFILES*). Default value:

"$(*ALL_FILES*) $(*OTHER_DEMO_TARGET_FILES*)"

However, *demo.packages* (but not *local.demo.packages*) explicitly sets this to equal—

"$(*OTHER_DEMO_TARGET_FILES*)"

$(DITROFF) [MAKE Macro]

VALUE: The name of the UNIX *troff*(1) program. Usual default: "*ditroff -me*". Use the value "*iroff -me -rv1*" with an IMAGEN printer.

$(DITROFF_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *troff*(1) program. Usual default: "".

.do [UNIX File Extension]

FILE FORMAT: A document source file. Can be made into a *.tr*, *.sp*, *.vs*, *.vo*, *.ns*, or *.no* file. Is made into a *.tr* file by passing through *pic*(1), *eqn*(1), and *tbl*(1).

$(DOFILES) [MAKE Macro]

VALUE: A list of all the *.do* files in the package. These are assumed to be source files. Default value: "".

$(EQN) [MAKE Macro]

VALUE: The name of the UNIX *eqn*(1) program. Usual default: "*eqn*".

WARNING: Do not use *eqn* constructs in files to be *nroff*ed.

$(EQN_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *eqn*(1) program. Usual default: "".

.ex [UNIX File Extension]

FILE FORMAT: An executable program file. Usually such files named xxx.*ex* are linked into file names xxx before they are used, but since *make* does not handle files with no extension automatically, it is not possible to drop the *.ex* extension completely. Rather, the xxx.*ex* file is made as an intermediate step, and the xxx file is made from it by including in *makefile.mk* the lines-

```
xxx:    xxx.ex
        rm -f xxx
        ln xxx.ex xxx
```

Can be made from *.l*, *.cl*, *.c*, *.cs*, *.f*, *.s*, or *.o* files.

.f [UNIX File Extension]

FILE FORMAT: A FORTRAN source file. Can be made into a *.ex*, *.o*, or *.s* file.

WARNING: FORTRAN source files are not yet implemented.

$(FFILES) [MAKE Macro]

VALUE: A list of all the *.f* files in the package. These are assumed to be source files. Default value: "".

$(FILES) [MAKE Macro]

VALUE: A list of all the package files that have no extension and that must be made from other files. These files are made by "*make all*". Default value: "".

WARNING: These files are all removed by cleaning.

global_count [MAKE Target]
GLOBAL_COUNT [MAKE Target]

WHEN MADE: Makes the file *GLOBAL_COUNT* by combining the *COUNT* file and all the <package>/*COUNT* files for every <package> in $(*PACKAGES*). This combination is done by the $(*COUNT*) program.

These *make* targets can only be made in a global directory.

global_index.tr [MAKE Target]
global_index.vs [MAKE Target]
global_index.vo [MAKE Target]
$(INDEX_APPENDIX) [MAKE Macro]
$(INDEX_TITLE) [MAKE Macro]

WHEN MADE: These make *global_index.tr*, *global_index.vs*, and *global_index.vo* where *global_index.tr* is made by running the $(*INDEX*) program against the <package_directory>/<prefix>_chap.in files for each package prefix listed in $(*PACKAGES*).

$(*INDEX_APPENDIX*) becomes the index appendix letter, while $(*INDEX_TITLE*) becomes the index appendix title. These default to "A" and "INDEX" respectively.

These *make* targets can only be made in a global directory.

$(GLOSSARY_FILES) [MAKE Macro]

VALUE: A list of all the source files in the package that may contain glossary entries. Default value: "$(*PRINT_FILES*)".

.h [UNIX File Extension]

FILE FORMAT: A C source file containing definitions included in various other C files by means of the C preprocessor *#include* statement.

.he [UNIX File Extension]

FILE FORMAT: Help file. Can be displayed on the screen. Can be made from a *.ma* or *.tr* file.

help [MAKE Target]

WHEN MADE: Makes $(*PREFIX*)_*chap.he*.

$(HE_PRINT) [MAKE Macro]

VALUE: The name of the UNIX program that prints *.he* files. Should handle underlining. Usual default: "*print*". Use the value "*imprint*" with an IMAGEN printer.

$(HE_PRINT_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX $(*HE_PRINT*) program. Usual default: "".

$(HFILES) [MAKE Macro]

VALUE: A list of all the *.h* files in the package. These are assumed to be source files. Default value: "".

.ho [UNIX File Extension]

FILE FORMAT: A fictitious file which when made causes the corresponding *.he* file to be printed. Can be made from a *.ma*, *.tr*, or *.he* file.

.in [UNIX File Extension]

FILE FORMAT: A $(*MANUAL*) index file. Made as a side effect of making a *.ma*, *.do*, or *.tr* file into a *.vs*, or *.vo* file.

Because of the way this file is made it currently contains not only index entries, but also all error messages from the troff job that made it. These are extracted so the user can see them by the make job that runs troff.

$(INDEX) [MAKE Macro]

VALUE: The name of the SKETCH index program. Usual default:

psearch sma/sma_index.sh.

index [MAKE Target]

WHEN MADE: Makes $(*PREFIX*)_*chap.in* (and maybe also $(*PREFIX*)*chap.vs*).

$(INDEX_FLAGS) [MAKE Macro]

VALUE: The flags for the SKETCH index program. Usual default: "".

$(INSTALL_DIRECTORY) [MAKE Macro]
$(INSTALL_DIRECTORY) [UNIX Environment Variable]

VALUE: The directory in which files are installed by the *backup_install* and *release_install make* targets.

Must be set in *csh.rc* files. $(*INSTALL_DIRECTORY*) is a UNIX Environment variable turned by *make* into a *make* macro.

$(INSTALL_FILES) [MAKE Macro]

VALUE: A list of all files that are to be installed by the *backup_install* and *release_install make* targets. Default value:

"$(INSTALL_SOURCE_FILES) $(INSTALL_TARGET_FILES)".

$(INSTALL_RCFILES) [MAKE Macro]

VALUE: A list of all *.rc* source files that are to be installed by the *backup_install* and *release_install make* targets. Default value: "" in a package directory and

"install_csh.rc install_sketch.rc"

in a global directory.

$(INSTALL_SOURCE_FILES) [MAKE Macro]

VALUE: A list of all source files that are to be installed by the *backup_install* and *release_install make* targets. Default value:

"make makefile.mk $(HFILES) $(SHFILES) $(DEMO_LFILES) $(DEMO_CLFILES) $(INSTALL_RCFILES) $(OTHER_INSTALL_SOURCE_FILES)".

$(INSTALL_TARGET_FILES) [MAKE Macro]

VALUE: A list of all target files that are to be installed by the *backup_install* and *release_install make* targets. Default value:

"$(LHFILES) $(PREFIX)_chap.in COUNT $(OTHER_INSTALL_TARGET_FILES)".

in a package directory, and

"$(LHFILES) COUNT $(OTHER_INSTALL_TARGET_FILES)".

in a global directory.

.l [UNIX File Extension]

FILE FORMAT: A LISP source file. Can be loaded directly into LISP by the LISP *load* function, or made into a *.ex*, *.o*, *.s*, *.lh*, or *.ou* file.

A *.l* file is made into a *.ex* file directly, without intervening *.s* or *.o* files, by *load*'ing the *.l* file into $(*LISP*) and *dumplisp*'ing the result to the *.ex* file.

.lc [UNIX File Extension]

FILE FORMAT: A C language source file containing *lambda*, *nlambda*, or *macro* functions. Compilation of this file into assembly language is done specially by running the compiler output through special filters so that the object file will fit into the LISP interpreter environment. Can be made into a *.o*, *.s*, or *.nt* file (all $(*LHFILES*) are all made first).

$(LCFILES) [MAKE Macro]

VALUE: A list of all the *.lc* files in the package. These are assumed to be source files. Default value: "".

$(LD_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *cc*(1) program when it is used to call *ld*(1) to produce an executable program. Usual default: "".

$(LFILES) [MAKE Macro]

VALUE: A list of all the *.l* files in the package. These are assumed to be source files. Default value:

"$(*COMMON_LFILES*) $(*COMPILE_LFILES*)
$(*LOAD_LFILES*) $(*DEMO_LFILES*)
$(*OTHER_LFILES*)".

.lh [UNIX File Extension]

FILE FORMAT: A C definitions file, like a *.h* file, which is made from a *.l* file via the **C-definition-code-port** facility (see the SKETCH objects package). Can be made from a *.l* file.

$(LHFILES) [MAKE Macro]

VALUE: A list of all the package *.lh* files that must be made from other files. All of these files must be made before any invocation of $(*CC*) or $(*LINT*). Unfortunately, this is not easily expressed, so removal of a *.lh* file and any files dependent upon it, and remaking the *.lh* file, must be done by hand whenever something is done that might change the *.lh* file.

However, making *all* or *lint* will make all the $(LHFILES) first.

Default value: "".

WARNING: If you change anything that would change a *.lh* file, you should remove by hand any file that would be affected and do a "*make all*". Or you may simply do a "*make clean all*".

WARNING: These files are all removed by cleaning.

$(LH_LISZT) [MAKE Macro]

VALUE: The name of the LISP compiler environment used to make *.lh* files from *.l* files. Usual default: "$(*LISZT*)".

$(LIBRARIES) [MAKE Macro]

VALUE: The library flags and file names for the UNIX *cc*(1) program when it is used to call *ld*(1) to produce an executable program. These flags and names are placed after the files being loaded, as opposed $(*LD_FLAGS*) which appear before the files being loaded in the *ld* argument list. Default value: "".

link [MAKE Target]

VALUE: Executes—

$(*LN*) $(*LN_FLAGS*) $(*LINK_DIRECTORY*)/file .

(notice the . at the end denoting the current directory) for every file in $(*LINK_FILES*) that is not readable in the current directory. This links the file in $(*LINK_DIRECTORY*) to the file in the current directory. Note that $(*LINK_DIRECTORY*) defaults to—

../../src/$(*PACKAGE_DIRECTORY*)

in a package directory, and to—

../src

in a global directory, while $(*LN_FLAGS*) defaults to '*—s*', so that the links are normally symbolic.

Normally you must first do some linking by hand to get SKETCH *make* to work; namely you must do—

ln -s <link_directory>/{make,makefile.mk} .

You must also be sure *../csh.rc* or *csh.rc* is defined.

Link.packages passes to each package *make* the definition—

LINK_DIRECTORY=../$(LINK_DIRECTORY)/<package_directory>

Link is defined with a double colon :: and can therefore be added to by defining new entries such as—

link::

    if test ! -r file; \
      then $(LN) $(LN_FLAGS) $(LINK_DIRECTORY)/file .; fi

$(LINK_DIRECTORY) [MAKE Macro]

VALUE: The directory into which $(*LINK_FILES*) are linked by the *link make* target. Default value: "../../src/$(PACKAGE_DIRECTORY)" in a package directory, and "../src" in a global directory.

Link.packages supplies the definition—

LINK_DIRECTORY=
../$(LINK_DIRECTORY)/$$(PACKAGE_DIRECTORY)

to each package *make*.

$(LINK_FILES) [MAKE Macro]

VALUE: A list of all the files in the package that are linked by the *link make* target. Default value: "$(*SOURCE_FILES*) $(*LNFILES*)".

$(LINT) [MAKE Macro]

VALUE: The name of the UNIX *lint*(1) program. Usual default:

psearch smk/smk_lint.sh.

This default lint program gets rid of warning messages of the following kinds:

... defined (...), but never used
returns value which is always ignored
returns value which is sometimes ignored

lint [MAKE Target]

WHEN MADE: Makes *.nt* files corresponding to all $(*CFILES*) and $(*LCFILES*). First makes all $(*LHFILES*).

$(LINT_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *lint*(1) program. These should not include flags for the macro preprocessor *cpp*, the loader *ld*(1), and *cc*(1), which are separate. Usual default: "".

$(LINT_LIBRARIES) [MAKE Macro]

VALUE: The library flags and file names for the UNIX *lint*(1) program. These flags and names are placed after the files being linted, as opposed $(*LINT_FLAGS*) which appear before the files being linted in the *lint* argument list. Default value: "$(*LIBRARIES*)".

$(LISP) [MAKE Macro]

VALUE: The name of the LISP evaluator program. Usual default: "$(*SKETCH*)".

$(PREFIX)_lisp [MAKE Target]

VALUE: A version of lisp made by executing

(*load* $(*PREFIX*)*_load*)

in $(*LISP*) after making $(*OFILES*).

#lisp [MAKE Target]

VALUE: Makes *#lisp* the same way as $(*PREFIX*)*_lisp* is made.

list [MAKE Target]

VALUE: Outputs the names of the files in $(*LIST_FILES*) to the standard output, one name per line.

List.packages prefixes each name in a package directory by '$(*PACKAGE_DIRECTORY*)/'.

$(LIST_FILES) [MAKE Macro]

VALUE: A list of all the files in the package that are listed by the ***list make*** target. Default value: "$(*SOURCE_FILES*)".

$(LISZT) [MAKE Macro]

VALUE: The name of the LISP compiler program. Usual default: "$(*SKETCHCOM*)".

$(PREFIX)_liszt [MAKE Target]

VALUE: A version of liszt made by executing

(*load* $(*PREFIX*)_*compile*)

in $(*LISZT*) after making $(*OFILES*).

#liszt [MAKE Target]

VALUE: Makes *#liszt* the same way as $(*PREFIX*)_*liszt* is made.

$(LISZT_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *liszt*(1) program. Usual default: "-*q*". Use the value "" to get verbose output.

$(LN) [MAKE Macro]

VALUE: The name of the UNIX *ln*(1) program. Usual default: "*ln*".

$(LNFILES) [MAKE Macro]

VALUE: A list of all the symbolic links in the package. Default value: "make" (*make* is usually linked to ../*package_make.sh*).

$(LN_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *ln*(1) program. Usual default: "−*s*".

load [MAKE Target]

WHEN MADE: Makes $(*PREFIX*)_*load.o*.

$(LOAD_LFILES) [MAKE Macro]

VALUE: A list of all the *.l* files in the package that are included in $(*PREFIX*)_*load.l* but not $(*PREFIX*)_*compile.l*. These are assumed to be source files. Default value: "".

.ma [UNIX File Extension]

FILE FORMAT: A *manual* source file. Can be made into a *.tr*, *.sp*, *.vs*, *.vo*, *.ns*, *.no*, or *.he* file. Is made into a *.tr* file by applying the $(*MANUAL*) program to add glossary entries from the $(SOURCE_FILES) and by passing the result through *pic*(1), *eqn*(1), and *tbl*(1).

$(MAFILES) [MAKE Macro]

VALUE: A list of all the *.ma* files in the package. These are assumed to be source files. Default value: "".

$(MAKE) [MAKE Macro]

VALUE: The name of the UNIX *make*(1) program, relative to the directory in which it will operate. Usual default: "*make*".

make [UNIX File]

VALUE: *Make* is a UNIX command file that replaces (modifies) the standard UNIX *make* (1) command. *Make* is usually symbolically linked to *../package_make.sh* in a package subdirectory, or to *global_make.sh* in a global directory.

../package_make.sh generally begins with

```
#!/bin/csh -f
if (-r csh.rc) then
        source csh.rc
else if (-r ../csh.rc) then
        source ../csh.rc
else
        source ../../csh.rc
        endif
```

and ends with

```
exec psearch smk/smk_make.sh $argv:q
```

It does not have to contain anything else.

global_make.sh is similar but begins with

```
#!/bin/csh -f
if (-r csh.rc) then
        source csh.rc
else
        source ../csh.rc
        endif
```

and ends with

```
exec psearch smk/smk_makeglobal.sh $argv:q
```

$(MAKE_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *make*(1) program. Usual default: "".

$(MANUAL) [MAKE Macro]

VALUE: The name of the SKETCH manual program. Usual default:

psearch sma/sma_manual.sh.

manual [MAKE Target]

WHEN MADE: Prints a complete manual by making the following in order: *title.vo*, *index.packages*, *global_index.vo*, and *chap.packages*.

This *make* target can only be made in a global directory.

$(MANUAL_FLAGS) [MAKE Macro]

VALUE: The flags for the SKETCH manual program. Usual default: "".

.mk [UNIX File Extension]

FILE FORMAT: A *make* source file.

$(MKFILES) [MAKE Macro]

VALUE: A list of all the *.mk* files in the package. These are assumed to be source files. Default value: "makefile.mk".

.no [UNIX File Extension]

FILE FORMAT: A fictitious file which when made causes the corresponding *.ns* file to be printed. Can be made from a *.ma*, *.do*, *.tr*, or *.ns* file.

$(NROFF) [MAKE Macro]

VALUE: The name of the UNIX *nroff*(1) program. Usual default: "*nroff -me*".

$(NROFF_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *nroff*(1) program. Usual default: "".

.ns [UNIX File Extension]

FILE FORMAT: A *lpr* source file (typically output by *nroff*(1)). Can be made from a *.ma*, *.do*, or *.tr* file. Can be made into a *.no* file.

.nt [UNIX File Extension]

FILE FORMAT: The output from *lint*'ing a *.c* file. Can be made from a *.c* or *.lc* file.

.o [UNIX File Extension]

FILE FORMAT: A binary object file containing a program, as produced by a compiler, such as those for LISP, C, and FORTRAN. LISP object files can be loaded into LISP by the *load* function, and others by the *cload* function. Can be made into a *.ex* file, or made from a *.l*, *.c*, *.lc*, *.cs*, *.f*, or *.s* file.

$(OFILES) [MAKE Macro]

VALUE: A list of all the package *.o* files that must be made from other files. These files are made by "*make all*". Default value: "".

WARNING: These files are all removed by cleaning.

$(OTHER_CLFILES) [MAKE Macro]

VALUE: A list of all the *.cl* files in the package that are not in $(*DEMO_CLFILES*). These are assumed to be source files. Default value: "".

$(OTHER_DEMO_TARGET_FILES) [MAKE Macro]

VALUE: A list of all the files other than those in $(*ALL_FILES*) that should be made before making any file listed in $(*DEMO_OUFILES*). E.g., the *.o* file for any *.c* file used exclusively by the demo. Default value: "".

$(OTHER_INSTALL_SOURCE_FILES) [MAKE Macro]

VALUE: A list of all source files that are to be installed by the *backup_install* and *release_install make* targets, but which are not listed in $(*HFILES*), $(*SHFILES*), $(*INSTALL_RCFILES*), $(*DEMO_LFILES*), or $(*DEMO_CLFILES*). The files *make* and *makefile.mk* should also be excluded from this list. Default value: "".

$(OTHER_INSTALL_TARGET_FILES) [MAKE Macro]

VALUE: A list of all target files that are to be installed by the *backup_install* and *release_install make* targets, but which are not listed in $(*LHFILES*). The index file $(*PREFIX*)*_chap.in* and the COUNT file should also be excluded from this list. Default value: "".

$(OTHER_LFILES) [MAKE Macro]

VALUE: A list of all the *.l* files in the package that are included in neither *$(PREFIX)_load.l* nor *$(PREFIX)_compile.l*, and are not in *$(DEMO_LFILES)*. These are assumed to be source files. Default value: "".

$(OTHER_PRINT_FILES) [MAKE Macro]

VALUE: A list of all the printable source files in the package that are not listed elsewhere, as in *$(LFILES)* or *$(CFILES)*. Default value: "*$(OTHER_SOURCE_FILES)*".

$(OTHER_RCFILES) [MAKE Macro]

VALUE: A list of all the *.rc* files in the package that are not listed in *$(INSTALL_RCFILES)*. These are assumed to be source files. Default value: "".

$(OTHER_SOURCE_FILES) [MAKE Macro]

VALUE: A list of all the source files in the package that are not listed elsewhere, as in *$(LFILES)* or *$(CFILES)*. Default value: "".

$(OTHER_TARGET_FILES) [MAKE Macro]

VALUE: A list of all the non-source files in the package that are not listed elsewhere, namely in *$(LHILES)*, *$(OFILES)*, or *$(FILES)*. These files are made by "*make all*". Default value: "".

WARNING: These files are all removed by cleaning.

.ou [UNIX File Extension]

FILE FORMAT: A copy of the standard output of some program. Can be made from a *.l* file by executing—

(*demo* '<x>*.l*'<x>*.ou*)

in the *$(DEMO_LISP)* program. Can be made similarly from a *.cl* program using *$(DEMO_LISZT)* in place of *$(LISP)*.

$(PACKAGE_DIRECTORY) [MAKE Macro]

VALUE: The package directory name; e.g. "sar" for the SKETCH array package. Default: *$(PREFIX)*.

Must be the same as *$(PREFIX)* in order for *.packages make* targets in a global directory to work, since *$(PACKAGES)* lists only one name per package, and presumes that this name is both the *$(PREFIX)* and *$(PACKAGE_DIRECTORY)* for the package.

$(PACKAGE_LINK_FILES) [MAKE Macro]

VALUE: *Link.packages* supplies the definition—

LINK_FILES=$(PACKAGE_LINK_FILES)

to each package *make*. Default value: "$$(*SOURCE_FILES*) $$(*LNFILES*)". Be sure to double the dollar signs in any definition you supply.

$(PACKAGE_LIST_FILES) [MAKE Macro]

VALUE: *List.packages* supplies the definition—

LIST_FILES=$(PACKAGE_LIST_FILES)

to each package *make*. Default value: "$$(*SOURCE_FILES*)". Be sure to double the dollar signs in any definition you supply.

$(PACKAGES) [MAKE Macro]
"global directory" [SKETCH Term]
"package directory" [SKETCH Term]
<make_target>**.packages** [MAKE Target Extension]
global.demo.packages [MAKE Target Extension]

VALUE: $(*PACKAGES*) is a list of package subdirectories that are to be processed by the <xxx>*.packages* targets of this global directory. These subdirectories need not be in the global directory itself as long as they are in the path of global directories (and can be found by *fsearch* (tps)).

Making a target named <target>*.packages* in the global directory makes the target <target> in each package subdirectory listed in $(*PACKAGES*), for each package target <target> whose name does not include a package prefix.

The following <target>*.packages* targets pass the indicated *make* macro values to the package *make* commands when they execute—

backup_install.packages	INSTALL_DIRECTORY= $(*INSTALL_DIRECTORY*)
clean_install.packages	INSTALL_DIRECTORY= $(*INSTALL_DIRECTORY*)
list.packages	LIST_FILES= $(*PACKAGE_LIST_FILES*)
link.packages	LINK_FILES= $(*PACKAGE_LINK_FILES*) LINK_DIRECTORY= ../$(*LINK_DIRECTORY*)/ $$(*PACKAGE_DIRECTORY*)

Actually, $(*INSTALL_DIRECTORY*) is converted to non-relative form before it is passed, but $(*LINK_DIRECTORY*) is not.

Sometimes it is desirable for *demo.packages* to pass the $(*DEMO_LISP*) and $(*DEMO_LISZT*) values to the package *make*, so there is a variant named *global.demo.packages* that does this by passing-

DEMO_LISP=$(*DEMO_LISP*)
DEMO_LISZT=$(*DEMO_LISZT*)

When defining $(*PACKAGE_LIST_FILES*) and $(*PACKAGE_LINK_FILES*) one must double the dollar signs. For example—

make '*PACKAGE_LIST_FILES*=$$(*HFILES*)' *list.packages*

Making a target *not* of the form <xxx>*.packages* in a global directory is like making the target in a package directory, except that targets that make files whose name includes $(*PREFIX*) or $(*PACKAGE_DIRECTORY*) cannot generally be made in the global directory. However, the following targets work in the global directory—

clean_install	With
backup_install	$(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*)
release_install	replaced by $(*INSTALL_DIRECTORY*).

WARNING: There is a strange bug in some versions of the UNIX *make*(1) that causes its -*n* option not to work for xx.packages targets. Use MAKE_FLAGS=-n instead.

$(PATH) [MAKE Macro]
$(PATH) [UNIX Environment Variable]

VALUE: A list of colon (:) separated directory names. These directories are searched in order for programs to run by *csh*, *sh*, and *psearch*. The same directories are searched in order by *fsearch* for data files.

Must be set in *csh.rc* files. $(*PATH*) is a UNIX Environment variable turned by *make* into a *make* macro.

$(PIC) [MAKE Macro]

VALUE: The name of the UNIX *pic*(1) program. Usual default: "*pic*".

WARNING: Do not use *pic* constructs in files to be *nroff*ed.

$(PIC_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *pic*(1) program. Usual default: "".

$(PREFIX) [MAKE Macro]

VALUE: The package name prefix. E.g. "sar" for the SKETCH array package, which prefixes file names such as "sar_array.l" and C code global names such as "sar_array".

$(PRINT) [MAKE Macro]

VALUE: The name of the UNIX *print*(1) program. Usual default: "*print*". Use the value "*imprint*" with an IMAGEN printer.

print [MAKE Target]

WHEN MADE: Makes the WC file and $(*PRINT*)'s *WC* and $(*PRINT_FILES*), the latter in alphabetical order.

$(PRINT_FILES) [MAKE Macro]

VALUE: A list of all the source files in the package which are to be printed by the *make print* target. Default value:

"$(*LFILES*) $(*CLFILES*) $(*HFILES*) $(*CFILES*)
$(*LCFILES*) $(*CSFILES*) $(*FFILES*)
$(*MKFILES*) $(*SHFILES*) $(RCFILES)
$(*OTHER_PRINT_FILES*)".

Because *OTHER_PRINT_FILES* is defined by default to be $(*OTHER_SOURCE_FILES*), *PRINT_FILES* is by default just the same as *SOURCE_FILES* with *.ma* and *.do* files omitted.

$(PRINT_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *print*(1) program. Usual default: "". Use the value "*-2 -O*" with an IMAGEN printer.

print_with_count [MAKE Target]

WHEN MADE: Makes the COUNT file and $(*PRINT*)'s *COUNT* and $(*PRINT_FILES*), the latter in alphabetical order.

$(RCFILES) [MAKE Macro]

VALUE: A list of all the *.rc* files in the package. These are assumed to be source files. Default value: "$(*INSTALL_RCFILES*) $(*OTHER_RCFILES*)".

$(RELEASE) [MAKE Macro]

VALUE: The name of the TPS *release*(tps) program. Usual default: "*release*".

$(RELEASE_FLAGS) [MAKE Macro]

VALUE: The flags for the TPS *release*(tps) program. Usual default: "".

release_install [MAKE Target]

VALUE: Releases $(*INSTALL_FILES*) in the directory

$(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*)

by using *release*(tps) with the *−D* and *−b* options.

In the global directory, $(*INSTALL_DIRECTORY*) is used in place of

$(*INSTALL_DIRECTORY*)/$(*PACKAGE_DIRECTORY*).

release_install_source [MAKE Target]

VALUE: Releases $(*INSTALL_SOURCE_FILES*) using *release*(tps).

release_source [MAKE Target]

VALUE: Releases $(*SOURCE_FILES*) using *release*(tps).

.s [UNIX File Extension]

FILE FORMAT: A non-macro, assembly language source file. Can be made into a *.o* or *.ex* file, or made from a *.l*, *.c*, *.lc*, *.f*, or *.cs* file. These files should not be used as source files, as they are cleaned out by *s.clean*. Use *.cs* files instead.

.sh [UNIX File Extension]

FILE FORMAT: A *sh*(1) or *csh*(1) source file.

$(SHFILES) [MAKE Macro]

VALUE: A list of all the *.sh* files in the package. These are assumed to be source files. Default value: "".

$(SKETCH) [MAKE Macro]

VALUE: The name of the SKETCH program. Usual default: *sketch*.

$(SKETCHCOM) [MAKE Macro]

VALUE: The name of the SKETCH compiler program. Usual default: *sketchcom*.

sketch.rc [UNIX File]

VALUE: A file in the global directory that defines the directories used by *sketch*. It should have roughly the form—

(sstatus cache-search-path (<this-directory> <next-directory> ...))
(sstatus catalog-search-path (<this-directory> <next-directory> ...))
(sstatus data-search-path (<this-directory> <next-directory> ...))
(sstatus font-search-path (<this-directory> <next-directory> ...))
(sstatus load-search-path (<this-directory> <next-directory> ...
<some-directory>/lisp/lisplib))
(setq *display-file-name* '/dev/<display-device>)
(setq *camera-file* '/dev/<camera-device>)

where the chain of directories to be searched for SKETCH program and data files is <this-directory> <next-directory> ..., and the FRANZ LISP library is int <some-directory>/lisp/lisplib. Relative directory names must not be used in *sketch.rc*.

smk_lint.sh... [UNIX Command]
$REAL_LINT [UNIX Environment Variable]

EQUIVALENT TO: *Lint*(1), but runs the output through *fgrep*(1) to remove all warning messages of the form—

...), but never used...
...returns value which is always ignored...
...returns value which is sometimes ignored...
...nonportable character comparison...
...File with unknown suffix...

NOTE: The UNIX *lint* command may be changed for this shell file by defining the environment variable *REAL_LINT* to equal a substitute.

$(SOURCE_FILES) [MAKE Macro]

VALUE: A list of all the source files in the package. Symbolic links listed in $(*LNFILES*) are not included. Default value:

"$(*LFILES*) $(*CLFILES*) $(*HFILES*) $(*CFILES*)
$(*LCFILES*) $(*CSFILES*) $(*FFILES*)
$(*MAFILES*) $(*DOFILES*) $(*MKFILES*)
$(*SHFILES*) $(CFILES) $(*OTHER_SOURCE_FILES*)".

.sp [UNIX File Extension]

FILE FORMAT: The output of the $(*SPELL*) program run against a *.tr* file and filtered through the $(*COLUMNS*) program. Can be made from a *.ma*, *.do*, or *.tr* file.

$(SPELL) [MAKE Macro]

VALUE: The name of the UNIX *spell*(1) program. Usual default: "*spell*".

spell [MAKE Target]

WHEN MADE: Makes $(*PREFIX*)*_chap.sp*.

$(SPELL_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *spell*(1) program. Usual default: "".

$(TBL) [MAKE Macro]

VALUE: The name of the UNIX *tbl*(1) program. Usual default: "*tbl*".

$(TBL_FLAGS) [MAKE Macro]

VALUE: The flags for the UNIX *tbl*(1) program. Usual default: "".

$(TITLE) [MAKE Macro]

VALUE: The title of the package chapter or appendix.

.tr [UNIX File Extension]

FILE FORMAT: A *troff* source file. Can be made into a *.vo*, *.vs*, *.sp*, *.ns*, *.no*, or *.he* file, or made from a *.ma* or *.do* file. *Pic*(1), *eqn*(1), and *tbl*(1) must be run to make a *.tr* file; only *troff* will be run on the *.tr* file itself.

.vo [UNIX File Extension]

FILE FORMAT: A fictitious file which when made causes the corresponding *.vs* file to be printed. Can be made from a *.ma*, *.do*, *.tr*, or *.vs* file.

.vs [UNIX File Extension]

FILE FORMAT: A *lpr -n* source file. Can be made into a *.vo* file, or made from a *.tr*, *.ma*, or *.do* file.

$(VS_PRINT) [MAKE Macro]

VALUE: The name of the UNIX program that prints *.vs* files. Usual default: "*lpr -n*". Use the value "*lpr -n -Pip*" with an IMAGEN printer.

$(VS_PRINT_FLAGS) [MAKE Macro]

VALUE: The flags for the $(*VS_PRINT*) program. Usual default: "".

wc [MAKE Target]
WC [MAKE Target]
WC [UNIX File Name]

WHEN MADE: Applies *wc*(1) to $(*SOURCE_FILES*) and puts the result into the file named *WC*. If there are any symbolic links in $(*LNFILES*), an *ls -l* listing of these is appended to *WC*.

APPENDIX D

WRITING MANUALS

1. CHAPTERS. The SKETCH manual is written using the *ME* macro package for the *troff*(1) text processing system. The manual is organized into chapters, one per package. Each chapter consists of ordinary manual sections, followed by a glossary. The glossary is embedded in the source code.

A SKETCH chapter is printed by the command

make chap

which invokes

sma_manual.sh -i "$(*CHAPTER*)" "$(*TITLE*)" $(*PREFIX*)*_chap.ma* $(*GLOSSARY_FILES*)

followed by \flpic(1), fleqn(1), *tbl*(1), and *troff -me*(1). The ordinary package manual sections are in the file $(*PREFIX*)*_chap.ma*, where $(*PREFIX*) is the package prefix. The glossary entries are in the files listed in $(*GLOSSARY_FILES*), which by default is just the list of all code source files (e.g. $(*LFILES*) and $(*CFILES*): see the appendix titled *MAKING FILES*). $(*CHAPTER*) is the chapter number, and $(*TITLE*) the chapter title.

The file $(*PREFIX*)_chap.ma is written as a sequence of sections each beginning with a *ME* section header–

.sh 1 SECTION TITLE.

The SECTION TITLE should be capitalized and terminated by a period. Spaces included in the SECTION TITLE must be preceded by a backslash \.

If $(*CHAPTER*) is a single capital letter, instead of a number, everything is the same except the result is called an appendix instead of a chapter.

2. GLOSSARY ENTRIES. Glossary entries are included in source files. The method of inclusion depends upon the language in which the source file is written. For example, a *.l* file prefixes each glossary entry line by ';<tab>', except for blank entry lines for which the <tab> may be omitted. A simple example is-

```
;	.En ( some-function " 'g_some-argument)" "[LISP Function]"
;	.Pa RETURNS
;	Some value computed from g_some-argument.
```

Another simple example in a *.c* file is

```
/*
.En "" some-function " (g_some-argument)" "[C Function]"
.Pa RETURNS
Some value computed from g_some-argument.
*/
```

The *.En* TROFF command line is required at the beginning of each glossary entry. Its 4 arguments are simply concatenated to form an output line: except that the second argument is surrounded by \fB and \fP to make it boldface, and the fourth argument is right adjusted in the output line instead of being put next to the other arguments.

The fourth argument is intended to describe the role of the name that is the second argument. The following is a list of standard fourth arguments—

[Argument Prefix]	[MAKE Macro]
[C Function]	[MAKE Target]
[C Macro]	[SKETCH Attribute Object]
[C Global Variable]	[SKETCH Attribute Macro]
[C Global Constant]	[SKETCH Object]
[C Structure Element]	[SKETCH Term]
[LISP Function]	[SKETCH Type Object]
[LISP Special Function]	[SKETCH Type Macro]
[LISP Macro]	[TROFF Command]
[LISP Global Variable]	[UNIX Command]
[LISP Global Constant]	[UNIX File Extension]
[LISP Property]	

The *.Cn* TROFF command will have the same effects as a *.En* command, except it does not start a new glossary entry, but continues the current one. It provides a way of specifying several different titles for one entry.

There are problems in the glossary sorting system (see below) which cause it to fail if either of the first two arguments to *.En* or *.Cn* contain spaces or tabs. In these two arguments use \0 instead of \(space). *Troff* treates \0 as a space the width of one digit.

The *.Xn* TROFF command can be used to add extra lines to a *.En* or *.Cn* command. The extra line is indented to the spot at which \kI last appeared in a previous line, as in

```
;        .En ( some-function " \kI'g_argument-1 'g_argument-2" "[LISP Function]"
;        .Xn "'g_argument-3 'g_argument-4)"
```

Other letters can be used in place of I: see *.Xn* in the glossary.

Glossary entries are included in various kinds of files according to the rules in Table 1. In order for glossary entries to be correctly extracted, each file must have an extension that specifies the language in which it is written.

The glossary is alphabetized by sorting each entry according to the second argument of the *.En* TROFF command that begins the entry. An index essentually consists of a glossary with everything outside *.En*, *.Cn*, and *.Xn* commands discarded. For index purposes, *.Cn* and *.En* are treated identically, and both start an new index entry. These index entries are sorted on the second argument to the *.En* or *.Cn* commands.

3. THE HELP FACILITY. The on-line manual, or help facility, is not yet completely implemented.

The first step in the making of an on-line manual is to *make* the

$(PREFIX)_chap.he

help file from the

TABLE 1

RULES FOR INCLUDING GLOSSARY ENTRIES IN FILES

.l file:	Preface all entry lines with a ';<tab>'. Omit the <tab> for blank entry lines. End entries with a line not beginning with ';<tab>'.
.sh file:	Preface all entries with a ': <<\DOCUMENTATION' line. End entries with a 'DOCUMENTATION' line for *sh* files, and a '\DOCUMENTATION' line for *csh* files.
.c, *.h*, *.cs*, or *.lc* file:	Preface all entries with a '/*' line. End entries with a '*/' line.
.ma or *.do* file:	Preface all entry lines with '.\"<tab>'. Omit the <tab> for blank entry lines. End entries with a line not beginning with '.\"<tab>'.
.mk file:	Preface all entry lines with a '#<tab>'. Omit the <tab> for blank entry lines. End entries with a line not beginning with '#<tab>'.

TABLE 1

$(PREFIX)_chap.ma

file by running the *.ma* file through *tbl*(1) to make a *.tr* file, and then through *nroff*(1) and *col*(1) to make a *.he* file. *Col* filters the output of *nroff* to remove all special motion control characters, except for backspace in sequences such as—

_(backspace)X

which are used to underline the characters such as X.

The *.he* file actually has only glossary entries, and not the rest of the manual chapter. There are also extra lines of the form—

################.En <name>

beginning the glossary entry with the given <name> (second argument to *.En*), and

################.Cn <extra-name>

for each <extra-name> defined by a glossary entry (second argument to *.Cn*).

Making—

$(PREFIX)_chap.ho

will have the effect of printing

$(PREFIX)_chap.he

in such a way that it can be proofread. This proofreading is necessary to check that problems are not caused by the fact that *nroff* cannot fit as many characters on a line as *troff*. Pay special attention to glossary entry headers.

In the future programs will be provided, hopefully based on the UNIX *refer* package, to index the *.he* files and rapidly retrieve glossary entries therefrom.

4. THE INDEX. If the *-i* option is used with *sma_manual.sh*, then when *troff* is run on the output of *sma_manual.sh*, an index file will be written into the standard error file. Several such index files many be combined into an appendix by the *sma_index.sh* UNIX command.

If the apparatus described int eh appendix titles *MAKING FILES* is used, index files are made automatically as a side effect of printing the manual chapter. The manual chapter (or appendix) has a principal source file named <xxx>*.ma*, and the associated index file is named <xxx>*.in*. A global index that includes all the package *.in* files can be printed by making the target *fast_index* or *slow_index* in the global directory which is a parent of the package directory. See the appendix titled *MAKING FILES* for more details.

5. HITLIST

(1) Finish the help facility.

6. GLOSSARY.

sma_count.sh [-c] file ... [UNIX Command]

EFFECT: Counts non-blank manual lines and code lines, and reports the results. The totals are reported for each file name directory, prefix, and extension, along with a grand total. A file name is of the form—

<directory>/<prefix>_<body>.<extension>

where only the last /, last _, and last . are recognized considered. The output is lines of the form—

<directory>/	<number code lines>	<number manual lines>
<prefix>_	<number code lines>	<number manual lines>
.<extension>	<number code lines>	<number manual lines>

followed by a separator line—

################################

followed by one line of the form—

<dir>/ <prefix>_ <body> .<ext> <# manual lines> <# code lines>

for each file. In either case a missing directory <dir> is denoted by '.', and a missing <prefix>, or extension <ext> by 'NONE'.

Glossary manual lines may be included in *.l*, *.sh*, *.c*, *.h*, *.lc*, *.cs*, *.ma*, *.do*, or *.mk* files with *.En* entries: see *sma_manual.sh* for the scheme used to include such lines. Non-glossary lines in these files are counted as code lines, except for *.ma* files where non-glossary lines are counted as manual lines. *.do* files are treated like *.ma* files, though they usually contain no glossary lines. Files with no extension are treated like *.sh* files.

Blank manual and code lines are not counted. Certain essentually blank lines, such as lines containing only '.' in troff input, only semi-colons in LISP input, or only '/*' or '*/' in C input, are also not counted. Blank glossary lines are those that are blank in this sense after any comment header (e.g. the ';<tab>' for *.l* files) has been stripped off.

If the -c option is present, the input files are outputs from previous *sma_count.sh* runs, and are combined to produce a composite input file. Any directory part to each input file name is added to the beginning of the directories listed in that file.

sma_index.sh appendix-letter 'appendix-title' [index-file ...] [UNIX Command]

EFFECT: A troff/nroff script is output for a index with given appendix-letter and title. The index files are *.in* files made by the *-i* option to *sma_manual.sh*.

The appendix letter and title obey the same rules as the chapter number and title do for sma_manual.sh. In particular, single capital letters should be used for the appendix letter: if numbers are used, a chapter will be created instead of an appendix.

NOTE: The nroff/troff script output presumes that the *me* macro package will be input to nroff or troff separately.

sma_manual.sh [-i] chapter-number 'chapter-title' chapter-file [glossary-file ...] [UNIX Command]

EFFECT: A troff/nroff script is output for a chapter with given number and title. The initial part of the chapter is defined by the chapter-file. The final section, entitled "GLOSSARY", is constructed from the glossary-files, if present. These can be *.l*, *.sh*, *.c*, *.h*, *.lc*, *.cs*, *.ma*, *.do*, or *.mk* files with *.En* entries. These *.En* entries must be included as comments according to the following scheme-

.l file:	Preface all entry lines with a ';<tab>'. Omit the <tab> for blank entry lines. End entries with a line not beginning with ';<tab>'.
.sh file: Files with no extension:	Preface all entries with a ': <<\DOCUMENTATION' line. End entries with a 'DOCUMENTATION' line for *sh* files and a '\DOCUMENTATION' line for *csh* files.
.c, *.h*, *.cs*, or *.lc* file:	Preface all entries with a '/*' line. End entries with a '*/' line.
.ma or *.do* file:	Preface all entry lines with '.\"<tab>'. Omit the <tab> for blank entry lines. End entries with a line not beginning with '.\"<tab>'.

.mk file: Preface all entry lines with a '#<tab>'.
Omit the <tab> for blank entry lines.
End entries with a line not beginning with '#<tab>'.

APPENDICES: If the chapter number is a single capital letter, an appendix will be created instead of a chapter.

-i OPTION: The *-i* option causes an index to be output to the error output stream during troff processing. The *sma_index.sh* UNIX command can process this index to produce final output.

NOTE: The nroff/troff script output presumes that the *me* macro package will be input to nroff or troff separately.

APPENDIX E

FRANZ FIXES

1. FRANZ FIXES. These are miscellaneous bug fixes to FRANZ LISP. They are individually documented in the glossary.

2. GLOSSARY.

(atom ...) [LISP Function]

EFFECTS VERSIONS: Opus 38.92 only.

FIXED BUG: Compiled *atom* incorrectly returned *nil* for *vector* and *vectori* values. Interpreted *atom*, however, correctly returned *t*.

(defconst ...) [LISP Macro]

FIXED BUG: FRANZ *defconst* does not declare constants to be special symbols if it is just loaded in.

(defprop ...) [LISP Macro]

FIXED BUG: FRANZ *defprop* does not compile function values for a property. The fixed version compiles *lambda*'s, *lexpr*'s, and *macro*'s.

liszt ... [UNIX Command]
(liszt-declare l_declaration ...) [LISP Nlambda Function]

EQUIVALENT TO: Standard FRANZ *liszt*, but modified so that it may be called with no arguments, will read and execute LISP expressions in its standard input, will remember any *special* declarations made when *defvar* or *defconst* are executed, and can be saved by *dumplisp* to make a new version of the compiler.

(*status feature complr*) is set in this compiler and may be used to conditionalize code for loading or execution inside the compiler.

New versions of the compiler made by *dumplisp* will not read in *.lisztrc* files.

LISZT-DECLARE: In general any call to *liszt-declare* will remember things in the *liszt* environment that would normally be entered by a *declare* statement. The arguments to *liszt-declare* have the same syntax as the arguments to *declare*. However, *localf* declarations are not remembered.

Note that the arguments of *liszt-declare* are not evaluated.

(macroexpand 'g_expression) [LISP Function]

FIXED BUG: FRANZ *macroexpand* did not handle *nlambda*'s. All *nlambda*'s in lisp and liszt are now handled.

(setf ...) [LISP Macro]

FIXED BUG: *Setf* cannot handle a single symbol returned by expansion of a macro first argument to *setf*. Such expansions should be handled the same as original input arguments, so *setf* should become *setq* in this case.

FIXED BUG: If *setf* cannot find a macro definition for an expression first argument, it gives up. It should try *apply* anyway if no function definition exists for the function symbol of the first argument, so that the function may become defined by the undefined function error handlers.

APPENDIX F

DISPLAY DEAMON

1. GLOSSARY.

"display daemon" [SKETCH Term]
"display protocol" [SKETCH Term]

USE: A SKETCH display daemon is a program that runs on a computer with display hardware, and makes that hardware available over networks.

The SKETCH display protocol is the language used to communicate with a display daemon on another computer (or even on your own computer).

STARTING DAEMONS: A daemon may be started by executing the following command in a directory containing the sdd package binaries—

framed [*—debug*] <port-number>
<device> <device-program> <device-argument> ...

The *-debug* option causes a trace of the commands received and the number of bytes sent and received. The trace is written to the UNIX standard error file.

The <port-number> is the port on which the daemon will listen. Port 1201 is typical. See ***a-display*** in the SKETCH Display Package for ways to set the <port-number> to which user of the daemon tries to connect.

The <device> is the name of the device being supported, also specified by ***a-display*** in the SKETCH Display Package. Typical values are *cgone0* and *fb* for a SUN3 computer.

The <device-program> designates which program will actually do the work of the deamon. This is a binary program in the sdd package directory. The most useful current value is *pixrectd-nocamerad,* which uses the SUN3 *pixrect* facility to access a SUN3 display device, and which does not support any camera. This program takes one <device-argument>, the file name of the hardware display device, typically either */dev/cgone0* or */dev/fb*.

The *framed* program can support multiple devices with different <device> identifiers. To do so multiple

<device> <device-program> <device-argument> ...

sequences are included in the *framed* argument list and separated by slash (/) arguments. E.g.

framed 1201 cgone0 pixrectd-nocamerad /dev/cgone0
/ fb pixrectd-nocamerad /dev/fb

However, the *-debug* option is not likely to work well if two deamons are serving two different devices at once.

CONNECTIONS: To talk to a display daemon you must open a connection to the daemon. First you need to know the host and port number of the daemon. The port number is given as an argument to the daemon when it is started. The following code is typical for making a connection—

```
#define ushort USHORT
#include <sys/types.h>
#undef ushort

#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

...

    char * host;
    int port

    ...

    int s, i;
    struct sockaddr_in sin;
    register struct hostent *h;

    ...

    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) ... notate error given by errno ...
    h = gethostbyname(host);
    if (h == NULL) ... notate error ...
    sin.sin_family = h->h_addrtype;
    bcopy(h->h_addr, (caddr_t)&sin.sin_addr, h->h_length);
    sin.sin_port = htons(port);

    i = connect(s, &sin, sizeof(sin));
    if (i < 0) ... notate error given by errno ...
```

At this point s is a socket descriptor of a connection into which data may be sent by—

```
  i = send (s, (caddr_t)data, sizeof data, 0) < 0);
  if (i < 0) ... notate error given by errno ...
```

and received by a subroutine such as—

```
        static int
receive (s, buffer, length)
        int s;
        register char * buffer;
        register int length;
{
        register int c;

        errno = 0;
        while (length > 0 && (c = recv (s, buffer, length, 0)) != 0) {
                if (c < 0) return (c);
                buffer += c;
                length -= c; }
        return (length == 0 ? 0 : -1); }
```

PROTOCOL: The protocol consists of messages. Each is a sequence of 32 bit integers (in network standard format: see the UNIX subroutines *ntol* and *lton* in *byteorder* (3N)), followed by zero or more byte strings. The integers tell the length of the byte strings (directly or indirectly).

There are two kinds of messages: requests and responses. The first integer of a request indicates the request type; currently one of—

open close clear map camera write flush nop.

The following requests have a response message whose first integer is the same as that of the corresponding request—

open close flush.

The nop, or no-operation message consists of a single 4-byte integer (the request type) that is ignored when sent either as a request or a response. One nop must be sent after every request for which a response is expected, before the response is read, to flush the request across to the daemon. Similarly the daemon sends a nop after every response before reading the next request.

Any request can also return an error response message. This will be read after the next request for which a response is expected.

REQUEST/RESPONSE CODES: The following are the current values of the request and response codes (the requests and responses are documented elsewhere)—

Name	Value	Meaning.
FR_OPEN	0xAAAAAAA1	Open display device.
FR_CLOSE	0xAAAAAAA2	Close display device.
FR_ERROR	0xAAAAAAA3	Error message response.
FR_CLEAR	0xAAAAAAA4	Clear all pixels to a single value.
FR_MAP	0xAAAAAAA5	Set color map.
FR_CAMERA	0xAAAAAAA6	Set camera parameters.
FR_WRITE	0xAAAAAAA7	Write block of pixels.
FR_FLUSH	0xAAAAAAA8	Flush memory to display proper.
FR_NOP	0xAAAAAAA9	No operation.

ERROR RECOVERY: If there is any error on a connection, the user should close the connection (via the UNIX *close* (2) routine, *not FR_CLOSE*) and reopen a new connection to the display device. The user should keep a copy of the device memory (map, camera parameters, and display pixels), so it can reinitialize these when the display device is reopened after an error.

FR_CAMERA [C Macro]
FR_CAMERA camera_string_size camera_string [SKETCH Display Daemon Request]

USE: Stores camera parameters in the camera.

For a Matrix camera, these parameters are represented by an ASCII character string, camera_string, of camera_string_size bytes, with a format to be determined later when this feature is implemented.

The camera parameters should be sent after *FR_OPEN*'ing the display device and before using the camera.

This request has no response if it is successful. If it is not successful an error message is returned (which is normally read later when the user is trying to read the response from an *FR_FLUSH*).

FR_CLEAR [C Macro]
FR_CLEAR xorigin yorigin xsize ysize pixel_value [SKETCH Display Daemon Request]

USE: Similar to *FR_WRITE* except that all pixels in the subimage are set to the same value. This value is passed as an integer in network standard format. If pixels are one byte long, then this integer is in the range from 0 through 255.

This request has no response if it is successful. If it is not successful an error message is returned (which is normally read later when the user is trying to read the response from an *FR_FLUSH*).

FR_ERROR [C Macro]
FR_ERROR message_size message_string [SKETCH Display Daemon Request]

USE: This is a response to any request during the execution of which the display daemon detects an error. The message_string is a character string of message_size bytes (without a NUL byte on the end). The message string should be formated in such a way that it can be read by the LISP *read* function and converted into a valid LISP object, which becomes an error message printable via *pretty-print*.

FR_FLUSH [C Macro]
FR_FLUSH delay_time exposure_count [SKETCH Display Daemon Request]
FR_FLUSH [SKETCH Display Daemon Response]

USE: The image and color maps stored in the display daemon are written to the display device (this may or may not have been previously done). After this is done, the state of the display device is not changed until first exposure_count pictures have been taken by the camera, and then delay_time milliseconds have elapsed. During this period new *FR_WRITE*, *FR_CLEAR*, and *FR_MAP* requests may be processed if they do not effect the display device state, but only change the memory of the display daemon.

The *FR_FLUSH* requests sends a response after the display device image and color map are written, but before the camera exposures have been taken or the delay has occurred. To get a response after the exposures and delay, send a second *FR_FLUSH* with zero exposure_count and zero delay_time, or send an *FR_CLOSE*. Neither of these two requests will respond until exposures and delays of previous requests have finished.

FR_MAP [C Macro]
FR_MAP xsize ysize map_type map_string [SKETCH Display Daemon Request]

USE: Stores a color map in the display device daemon (the color map may not be written to the display hardware until the next FR_FLUSH request, but on the other hand it may be written to the hardware sooner).

The meaning of xsize and ysize, and the format of map_string, can be display device specific. However, the following standard is recommended where applicable: xsize = 3, ysize = 256, with map_string consisting of 3*256 bytes, each an unsigned integer representing an intensity for one color. The first three bytes are the red, green, and blue intensities, in that order, for image pixels of value 0; the next three bytes are for pixels of value 1; etc.

The display device stores two color maps. If type == 0, the normal color map is set, whereas if type == 1, the camera map is set. The later is used temporarily only while the camera is taking a picture (see *FR_FLUSH*).

The normal map should be sent after *FR_OPEN*'ing the display device and before writing any image via *FR_WRITE* or *FR_CLEAR*.

The camera map should be sent after *FR_OPEN*'ing the display device and before using the camera.

This request has no response if it is successful. If it is not successful an error message is returned (which is normally read later when the user is trying to read the response from an *FR_FLUSH*).

FR_NOP [C Macro]
FR_NOP [SKETCH Display Daemon Request]
FR_NOP [SKETCH Display Daemon Response]

USE: If this is received as either a request or a response, it is ignored.

Some network implementations seem to be unable to deliver all the bytes of a message to the receiver until another message has been sent. For the sake of these, an *FR_NOP* should be sent after each request for which a response is expected. Similarly the daemon may send an *FR_NOP* after each response.

FR_OPEN [C Macro]
FR_OPEN user_id_size device_size [SKETCH Display Daemon Request]
processor_size monitor_size camera_size
user_id_string device_string
processor_string monitor_string camera_string
FR_OPEN [SKETCH Display Daemon Response]

FR_CLOSE [C Macro]
FR_CLOSE [SKETCH Display Daemon Request]
FR_CLOSE [SKETCH Display Daemon Response]

USE: The open display daemon request opens a specific hardware display, making it usable until it is closed. The request consists of a number of character strings, the size in bytes of each being given with the request integers. The user_id_string identifies the user so that other users can find him when they find that the display is busy. The device_string specifies which display device is to be used (one daemon may handle many displays for many different users on one computer). Device_string must match a device argument given to the display daemon when it is started (see *display daemon*).

The processor_string refers to the display processor type, the monitor_string to the display monitor type, and the camera_string to the display camera type. See *a-display* in the SKETCH Display Package. Currently the daemon does not use these.

The close request terminates use of a particular piece of display hardware, and releases all resources. It should be followed by closing the connection (and not by trying to open another display device).

If the daemon successfully processes the open or close request, it returns an open or close response, consisting of one 4-byte integer with the same type code as the request (*FR_OPEN* or *FR_CLOSE*). Otherwise the daemon returns an error message (*FR_ERROR*).

If there is any error on a connection, the user should close the connection (via the UNIX *close* (2) routine, *not FR_CLOSE*) and reopen a new connection to the display device. The user should keep a copy of the device memory (map, camera parameters, and display pixels), so it can reinitialize these when the display device is reopened after an error.

FR_WRITE .[C Macro]

FR_WRITE xorigin yorigin [SKETCH Display Daemon Request]
xsize ysize psize pixel_string

USE: Stores a rectangular subimage within the display device image. The coordinates of the upper left pixel of the subimage are xorigin and yorigin. The 0,0 pixel of the display device is its upper left corner, the x coordinate is horizontal, and the y coordinate is vertical (increasing from top to bottom). The size of the subimage in pixels is given by xsize and ysize. The number of bytes per pixel is given by psize (which is normally equal to 1). The pixels are stored in pixel_string. Each pixel is a contiguous string of bytes; and each horizontal row is a contiguous string of pixels. Therefore pixel_string is

xsize * ysize * psize

bytes long.

For one byte pixels, the pixels are interpreted as a number from 0 through 255 that indexes the color map (see *FR_MAP*). It is suggested that the formats assumed by the SUN pixrect package be used to guide pixel format standards in other cases.

This request has no response if it is successful. If it is not successful an error message is returned (which is normally read later when the user is trying to read the response from an *FR_FLUSH*).

SKETCH DEMONSTRATION PROGRAMS

VERSION 4B

April 1989

1.	FRANZ EXTENSIONS.	sfe_demo
2.	ATOMS.	sat_demo
3.	OBJECTS.	sob_cdemo, sob_hdemo, sob_vdemo
4.	CATALOGS.	sca_demo
5.	ARRAYS.	sar_demo, sar_gcdemo
6.	BASIC ARITHMETIC.	sba_demo
7.	BIT GRAPHICS.	sbg_demo
8.	ANALYTIC GEOMETRY.	sag_demo
9.	DISPLAY.	sdi_demo, sdi_cdemo
10.	HISTOGRAMS.	shi_demo
11.	EDGES.	sed_demo
12.	LINEAR FIT.	slf_demo
13.	TEXTURE.	stx_demo

```
-> (assert (=& 1 1)
           '(a very very very very very very very very very long error
              message that should NOT be printed))
nil
-> (assert (=& 1 2)
           '(a very very very very very very very very very long error
              message that SHOULD be printed))
ERROR    (a very very very very very very very very very long error message
           that SHOULD be printed)

BREAK -
<1> ^D
[Return to top level]
-> (ceiling 1.5)
2
-> (ceiling 2.0)
2
-> (ceiling 2)
2
-> (copy-list '(1) 5 'empty)
(1 empty empty empty empty)
-> (floor 2.5)
2
-> (floor 2.0)
2
-> (floor 2)
2
-> (ftime)
6.09708e+08
-> (has-setf-function 'car)
#<bcd @ 0x9AC02, lambda>
-> *is-compiler*
nil
-> (list-depth '(1 2 (3 4) (5 6 (7 8) 9)))
3
-> (check-list '(1 2 3))
3
-> (check-list '(1 2 . 3))
-1
-> (check-list '(1 2 3) 'fixp)
3
-> (check-list '(1 2 3) 'numberp)
3
-> (check-list '(1 2 three) 'numberp)
-1
-> pi
3.14159
-> sqrt-pi
1.77245
-> *ptime-counts-per-second*
60
-> (round 1.5)
2
-> (round 1.5)
1
-> (round 1.0)
```

```
1
-> (defcache f (2 equal) (x) (print 'COMPUTED) (terpri) (first x))
f
-> (f '(1 2 3 4))
COMPUTED
1
-> (f '(a b c d))
COMPUTED
a
-> *f-cache*
(((a b c d) a) ((1 2 3 4) 1))
-> (f '(1 2 3 4))
1
-> (f '(1 2 3 4))
1
-> (f '(1 2 3 4))
1
-> *f-cache*
(((1 2 3 4) 1) ((a b c d) a))
-> (f '(x y))
COMPUTED
x
-> *f-cache*
(((x y) x) ((1 2 3 4) 1))
-> *gc-count*
0
-> *gc-history*
nil
-> (xtime '(do i 0 (|1+| i) (>& i 100000) (cons nil nil)))
75.133
(compute-time - 75.15 seconds / gc-time - 11.8 seconds for 8 gcs)
-> *gc-count*
8
-> *gc-history*
(  (compute - 9.76667 seconds)
   (gc - 1.51667 seconds fixnum 144 flonum 100 symbol 391 list 428 + 0
       vectori 50 hunk0 50 hunk1 50 hunk2 50 hunk3 120)
   (compute - 9.5 seconds)
   (gc - 1.48333 seconds fixnum 144 + 0 flonum 100 symbol 391 list 428
       vectori 50 hunk0 50 hunk1 50 hunk2 50 hunk3 120)
   (compute - 9.4 seconds)
   (gc - 1.58333 seconds fixnum 144 + 0 flonum 100 symbol 391 list 428
       vectori 50 hunk0 50 hunk1 50 hunk2 50 hunk3 120)
   (compute - 9.41667 seconds)
   (gc - 1.46667 seconds fixnum 144 + 0 flonum 100 symbol 391 list 428
       vectori 50 hunk0 50 hunk1 50 hunk2 50 hunk3 120)
   (compute - 9.51667 seconds)
   (gc - 1.48333 seconds fixnum 144 + 0 flonum 100 symbol 391 list 428
       vectori 50 hunk0 50 hunk1 50 hunk2 50 hunk3 120)
   (compute - 9.45 seconds)
   (gc - 1.36667 seconds fixnum 144 + 0 flonum 100 symbol 391 list 428
       vectori 50 hunk0 50 hunk1 50 hunk2 50 hunk3 120)
   (compute - 9.46667 seconds)
   (gc - 1.5 seconds fixnum 144 + 0 flonum 100 symbol 391 list 428 vectori
       50 hunk0 50 hunk1 50 hunk2 50 hunk3 120))
->
```


Goodbye

```
-> (cload '(_sat_mtest _sat_ntest _sat_ftest) 'sat/sat_cdemo)
nil
-> (exec cat sat_cdemo.c)
#include <sat/sat_defs.h>

        int
sat_mtest (port) sat_lvalue port; {
        FILE * p = port->sat_lport;
        sfe_assert (sat_fmissing (SAT_FMISSING), "\
(sat_mtest - ! sat_fmissing (SAT_FMISSING))");
        sfe_assert (sat_dmissing (SAT_DMISSING), "\
(sat_mtest - ! sat_dmissing (SAT_DMISSING))");
        sfe_assert (! sat_fmissing (SAT_FMINIMUM), "\
(sat_mtest - sat_fmissing (SAT_FMINIMUM))");
        sfe_assert (! sat_fmissing (SAT_FMAXIMUM), "\
(sat_mtest - sat_fmissing (SAT_FMAXIMUM))");
        sfe_assert (! sat_dmissing (SAT_DMINIMUM), "\
(sat_mtest - sat_dmissing (SAT_DMINIMUM))");
        sfe_assert (! sat_dmissing (SAT_DMAXIMUM), "\
(sat_mtest - sat_dmissing (SAT_DMAXIMUM))");

        fprintf (p, "SAT_CMISSING = %d\n", SAT_CMISSING);
        fprintf (p, "SAT_CMAXIMUM = %d\n", SAT_CMAXIMUM);
        fprintf (p, "SAT_CMINIMUM = %d\n", SAT_CMINIMUM);
        fprintf (p, "SAT_UCMAXIMUM = %d\n", SAT_UCMAXIMUM);
        fprintf (p, "SAT_SMISSING = %d\n", SAT_SMISSING);
        fprintf (p, "SAT_SMAXIMUM = %d\n", SAT_SMAXIMUM);
        fprintf (p, "SAT_SMINIMUM = %d\n", SAT_SMINIMUM);
        fprintf (p, "SAT_USMAXIMUM = %d\n", SAT_USMAXIMUM);
        fprintf (p, "SAT_LMISSING = %d\n", SAT_LMISSING);
        fprintf (p, "SAT_LMAXIMUM = %d\n", SAT_LMAXIMUM);
        fprintf (p, "SAT_LMINIMUM = %d\n", SAT_LMINIMUM);
        fprintf (p, "SAT_ULMAXIMUM = %u\n", SAT_ULMAXIMUM);
        fprintf (p, "SAT_IMISSING = %d\n", SAT_IMISSING);
        fprintf (p, "SAT_IMAXIMUM = %d\n", SAT_IMAXIMUM);
        fprintf (p, "SAT_IMINIMUM = %d\n", SAT_IMINIMUM);
        fprintf (p, "SAT_UMAXIMUM = %u\n", SAT_UMAXIMUM);
        fprintf (p, "SAT_FMISSING = %15.10e\n", SAT_FMISSING);
        fprintf (p, "SAT_FMAXIMUM = %15.10e\n", SAT_FMAXIMUM);
        fprintf (p, "SAT_FMINIMUM = %15.10e\n", SAT_FMINIMUM);
        fprintf (p, "SAT_DMISSING = %25.20e\n", SAT_DMISSING);
        fprintf (p, "SAT_DMAXIMUM = %25.20e\n", SAT_DMAXIMUM);
        fprintf (p, "SAT_DMINIMUM = %25.20e\n", SAT_DMINIMUM);
        return (0); }

        int
sat_ntest (root, count)
        register sat_lvalue root;
        int count;
        {
        register int i = 0;
        while (i < count) {
                int i0 = i;
                register int j;
                register sat_lvalue tail;
                for (i = i0, j = 0, tail = root; j < 111; ++ j, ++ i) {
```



```
                        tail->sat_lrest = sat_nlist (sat_nil, sat_nil);
                        tail = tail->sat_lrest;
                        tail->sat_lfirst = sat_nfixnum (i); }
                for (i = i0, j = 0, tail = root; j < 111; ++ j, ++ i) {
                        tail = tail->sat_lrest;
                        sfe_assert (tail->sat_lfirst->sat_lint == i, "\
(sat_ntest - C allocation foulup - bad sat_lfirst)"); }
                sfe_assert (tail->sat_lrest == sat_nil, "\
(sat_ntest - C allocation foulup - bad sat_lrest)"); }
        return (0); }

        int
sat_ftest (string, count, port) char * string; int count; sat_lvalue port; {
        FILE * p = port->sat_lport;
        fprintf (p, "sat_sformat (...) = %s\n", sat_sformat (string));
        fprintf (p, "sat_snformat (...) = %s\n", sat_snformat (string, count));
        fprintf (p, "sat_tformat (...) = %s\n", sat_tformat (string));
        fprintf (p, "sat_tnformat (...) = %s\n", sat_tnformat (string, count));
        return (0); }
0
(compute-time - 1.8 seconds)
-> (_sat_mtest poport)
SAT_CMISSING = -128
SAT_CMAXIMUM = 127
SAT_CMINIMUM = -127
SAT_UCMAXIMUM = 255
SAT_SMISSING = -32768
SAT_SMAXIMUM = 32767
SAT_SMINIMUM = -32767
SAT_USMAXIMUM = 65535
SAT_LMISSING = -2147483648
SAT_LMAXIMUM = 2147483647
SAT_LMINIMUM = -2147483647
SAT_ULMAXIMUM = 4294967295
SAT_IMISSING = -2147483648
SAT_IMAXIMUM = 2147483647
SAT_IMINIMUM = -2147483647
SAT_UMAXIMUM = 4294967295
SAT_FMISSING = -3.4028234664e+38
SAT_FMAXIMUM = 3.4028232636e+38
SAT_FMINIMUM = -3.4028232636e+38
SAT_DMISSING = -1.79769313486231570000e+308
SAT_DMAXIMUM = 1.79769313486231550000e+308
SAT_DMINIMUM = -1.79769313486231550000e+308
0
-> (_sat_ftest "This is a test string" 100 poport)
sat_sformat (...) = |This is a test string|
sat_snformat (...) = |This is a test string|
sat_tformat (...) = "This is a test string"
sat_tnformat (...) = "This is a test string"
0
-> (_sat_ftest "This is a test string" 4 poport)
sat_sformat (...) = |This is a test string|
sat_snformat (...) = This
sat_tformat (...) = "This is a test string"
sat_tnformat (...) = "This"
```

```
0
-> (_sat_ftest "foobar" 4 poport)
sat_sformat (...) = foobar
sat_snformat (...) = foob
sat_tformat (...) = "foobar"
sat_tnformat (...) = "foob"
0
-> (_sat_ftest "#foobar" 4 poport)
sat_sformat (...) = |#foobar|
sat_snformat (...) = |#foo|
sat_tformat (...) = "#foobar"
sat_tnformat (...) = "#foo"
0
-> (_sat_ntest (cons nil nil) 200)
0
-> (_sat_ntest (cons nil nil) 1000000)
0
(compute-time - 58.9667 seconds / gc-time - 34.9333 seconds for 16 gcs)
-> (_SAT_MAD 1000000 1000000 0 2000000)
500000
-> (_SAT_MAD -1000000 1000000 4000000 2000000)
-499998
-> (_SAT_MAS 2 2 0 0 5)
128
-> (_SAT_MAS 2 2 128 0 5)
4224
-> (_SAT_MAS 0 0 0 4 -20)
16384
-> (_SAT_MAS 0 0 128 0 -2)
32
-> (setq x (_SAT_MAS 0 0 1 0 31))
-2147483648
-> (_SAT_MAS (expt 2 16) (expt 2 15) x 0 -2)
1073741824
->
Goodbye
```



```
-> (comment '(a demo for sob_common.l))
comment
-> (comment '(first make Bill and Jill unbound so this demo can be rerun
                    several times))
comment
-> (makunbound 'Bill)
Bill
-> (makunbound 'Jill)
Jill
-> (a-person has-name 'Bill has-weight 183 has-height 70 has-age 45)
Bill
-> Bill
(a-person has-name Bill has-weight 183 has-height 70 has-age 45)
-> (has-weight Bill)
183
-> (a-person has-name 'Lizzy has-age 6)
Lizzy
-> (a-person has-name 'Dizzy has-age 8)
Dizzy
-> (has-children Bill)
nil
-> (setf (has-children Bill) (list Lizzy Dizzy))
(Lizzy Dizzy)
-> (has-children Bill)
(Lizzy Dizzy)
-> Bill
(a-person has-name Bill has-weight 183 has-height 70 has-age 45
          has-children (Lizzy Dizzy))
-> (eq (first (has-children Bill)) Lizzy)
t
-> (print Lizzy)
Lizzynil
-> Lizzy
(a-person has-name Lizzy has-age 6)
-> (a-person Bill has-name 'Jill has-weight 153 has-age 43 has-husband Bill)
Jill
-> Jill
(a-person has-name Jill has-weight 153 has-age 43 has-husband Bill
          has-height 70 has-children (Lizzy Dizzy))
-> (get-attribute has-husband Jill)
Bill
-> (setf (get-attribute has-weight Bill) 195)
195
-> (push (a person has-name 'Fizzy has-age 1) (has-children Bill))
(Fizzy Lizzy Dizzy)
-> Bill
(a-person has-name Bill has-weight 195 has-height 70 has-age 45
          has-children (Fizzy Lizzy Dizzy))
-> (push (first (has-children Bill)) (has-children Jill))
(Fizzy Lizzy Dizzy)
-> Bill
(a-person has-name Bill has-weight 195 has-height 70 has-age 45
          has-children (Fizzy Lizzy Dizzy))
-> Jill
(a-person has-name Jill has-weight 153 has-age 43 has-husband Bill
          has-height 70 has-children (Fizzy Lizzy Dizzy))
```

```
-> (uneval-object Bill)
(a-person has-name 'Bill has-weight 195 has-height 70 has-age 45
          has-children
              `(  ,(a-person has-name 'Fizzy) ,(a-person has-name 'Lizzy)
                  ,(a-person has-name 'Dizzy)))
-> (uneval-object a-person)
(a-type has-name 'a-person has-allocation-count 5 has-parameters 2)
-> (has-type Bill)
a-person
-> a-person
(a-type has-name a-person has-allocation-count 5 has-parameters 2)
-> has-weight
(an-attribute has-name has-weight)
-> (a-person has-name 'George has-weight 108)
George
-> (eq George (a-person has-name 'George))
t
-> (eq George (a-person has-name 'George has-weight 108))
t
-> (comment '(the following should have an error complaining that the new
               and previous values of George are not equal))
comment
-> (a-person has-name 'George has-weight 110)
ERROR - (new value for George does not equal previous value)
WHILE EVALUATING - (make-name-function-2 (George a-person George))
WHILE EVALUATING - (make-object (list a-person has-name 'George has-weight
                                      110)
                                nil)

BREAK -
<1> ^D
[Return to top Level]
->
Goodbye
```



```
-> (exec rm -f sob_hdemo.lh)
0
-> (setq *C-definition-code-port* (outfile 'sob_hdemo.lh))
#<port sob_hdemo.lh>
-> (comment (allow this demo to be run many times by unbinding an-hfoo))
comment
-> (makunbound 'an-hfoo)
an-hfoo
-> (declare-hunk-type (an-hfoo sob_hfoo sob_h) is-read-init-write
                      (has-fiddle nil sob_hfiddle) (has-fee 9) has-fum
                      is-read-init has-fi)
an-hfoo
-> (setq x (an-hfoo has-fi 'my-fi has-fee 100))
(an-hfoo has-fee 100 has-fi my-fi)
-> (has-fiddle x)
nil
-> (setf (has-fiddle x) 100)
100
-> x
(an-hfoo has-fiddle 100 has-fee 100 has-fi my-fi)
-> (has-fi x)
my-fi
-> (macroexpand '(has-fiddle (an-hfoo x)))
(cxr 2 x)
-> (macroexpand '(setf (has-fiddle (an-hfoo x)) 100))
(rplacx 2 x 100)
-> (macroexpand '(setf (has-fi (an-hfoo x)) 'hi-there))
(never-set-function
   'hi-there
   (progn
      (declare (special *descriptor*an-hfoo*has-fi*1*))
      (cond
         (  (boundp '*descriptor*an-hfoo*has-fi*1*)
            *descriptor*an-hfoo*has-fi*1*)
         (t  (get-attribute-descriptor-function
               '*descriptor*an-hfoo*has-fi*1* an-hfoo has-fi 1))))
   has-fi x)
-> (declare-hunk-type an-hfoo is-read-init-write has-fiddle (has-fee 9)
                      has-fum is-read-init has-fi)
an-hfoo
-> (close *C-definition-code-port*)
t
-> (setq *C-definition-code-port* nil)
nil
-> (exec cat sob_hdemo.lh)
typedef struct sob_hstruct * sob_hfoo;
struct sob_hstruct [
        sat_lvalue sob_hplist;
        sob_type sob_htype;
        sat_ltype sob_hfiddle;
        sat_ltype sob_hPAD3;
        sat_ltype sob_hPAD4;
        sat_ltype sob_hPAD5;
        ];
#define sob_halloc(x,y) struct sob_hstruct (x) [y]
0
```

```
->
Goodbye
```



```
-> (exec rm -f sob_vdemo.lh)
0
-> (setq *C-definition-code-port* (outfile 'sob_vdemo.lh))
#<port sob_vdemo.lh>
-> (comment (allow this demo to be run many times by unbinding a-vfoo))
comment
-> (makunbound 'a-vfoo)
a-vfoo
-> (declare-vector-type (a-vfoo sob_vfoo sob_v)
                        has-C-plist-vector-element-name nil
                        is-read-init-write a-value
                        (has-fiddle nil sob_vfiddle) a-char (has-fee 9)
                        a-short (has-fum 101 sob_vfum) is-read-init a-value
                        (has-fi nil sob_vfi))
a-vfoo
-> (setq x (a-vfoo has-fi 'my-fi has-fee 100))
(a-vfoo has-fee 100 has-fum 101 has-fi my-fi)
-> (has-fiddle x)
nil
-> (setf (has-fiddle x) 100)
100
-> x
(a-vfoo has-fiddle 100 has-fee 100 has-fum 101 has-fi my-fi)
-> (has-fi x)
my-fi
-> (macroexpand '(has-fiddle (a-vfoo x)))
(cxr 2 (vprop x))
-> (macroexpand '(setf (has-fiddle (a-vfoo x)) 100))
(  (lambda (o v) (vseti-value-function v nil 1 o) (rplacx 2 (vprop o) v)) x
   100)
-> (macroexpand '(setf (has-fi (a-vfoo x)) 'hi-there))
(never-set-function
   'hi-there
   (progn
      (declare (special *descriptor*a-vfoo*has-fi*1*))
      (cond
         (  (boundp '*descriptor*a-vfoo*has-fi*1*)
            *descriptor*a-vfoo*has-fi*1*)
         (t (get-attribute-descriptor-function '*descriptor*a-vfoo*has-fi*1*
                                              a-vfoo has-fi 1))))
   has-fi x)
-> (makunbound 'a-vfop)
a-vfop
-> (makunbound 'an-allocate-vfop)
an-allocate-vfop
-> (declare-vector-type (a-vfop sob_vfop sob_vf)
                        has-C-type-vector-element-name nil
                        has allocate-C-type an-allocate-vfop
                        is-read-init-write in-vector a-short 3
                        (has-short '(66 77 88) sob_vfshort) a-ubit
                        (has-ubit1 nil sob_vfubit1) an-lbit has-ubit2 a-ubit
                        (has-ubit3 nil sob_vfubit3) a-long
                        (has-long nil sob_vflong) a-value
                        (has-value nil sob_vfvalue))
a-vfop
(compute-time - 0.2 seconds / gc-time - 1.46667 seconds for 1 gcs)
```

```
-> an-allocate-vfop
(a-type has-name an-allocate-vfop has-size 128 has-vector-type a-vfop)
-> (macroexpand '(has-long (a-vfop x)))
((lambda (v) (cond ((=& v _SAT_IMISSING) nil) (t v))) (vrefi-long x 2))
-> (macroexpand '(setf (has-value (a-vfop x)) 'horray))
(vseti-value-function 'horray nil 3 x)
-> (macroexpand '(a-vfop has-value 'valant))
(  (lambda (T00018)
           (vseti-value-function 'valant nil 3 T00018)
           (set-vector-attribute-list '(66 77 88) (cxr 25 a-short) 0 3 1
                                      T00018)
           (rplacx 4 a-vfop (|1+| (cxr 4 a-vfop)))
           T00018)
   (copy-vector (cxr 0 (cxr 6 a-vfop))))
-> (macroexpand '(has-short (a-vfop x)))
(get-vector-attribute-list (cxr 25 a-short) 0 3 1 x)
-> (macroexpand '(has-short (a-vfop x) 2))
((lambda (v) (cond ((=& v -32768) nil) (t v))) (vrefi-word x 2))
-> (macroexpand '(setf (has-short (a-vfop x)) '(2 3 4)))
(set-vector-attribute-list '(2 3 4) (cxr 25 a-short) 0 3 1 x)
-> (macroexpand '(setf (has-short (a-vfop x) 2) 99))
(vseti-word x 2 (or 99 -32768))
-> (setq x (a-vfop has-value 'valant has-short '(3 4 5)))
(a-vfop has-short (3 4 5) has-ubit1 0 has-ubit3 0 has-value valant)
-> (has-short (a-vfop x))
(3 4 5)
-> (setf (has-short x 1) 9)
9
-> x
(a-vfop has-short (3 9 5) has-ubit1 0 has-ubit3 0 has-value valant)
-> (setf (has-short x) '(7 8))
(7 8)
-> (has-short x)
(7 8)
-> (has-short x 1)
8
-> (setf (has-short x) '(88))
(88)
-> (has-short x)
(88)
-> (makunbound 'a-vfum)
a-vfum
-> (declare-vector-type (a-vfum sob_vfum sob_vu)
                        has-C-vsize-vector-element-name sob_vuSIZE
                        is-read-init-write in-vector an-allocate-vfop 3
                        (has-x nil sob_vux) a-long (has-y nil sob_vuy)
                        an-allocate-vfop (has-z nil sob_vuz) a-vfop
                        (has-w nil sob_vuw) is-read-private a-long 3
                        has-pad)
a-vfum
-> (macroexpand '(a-vfum has-x (list (a-vfop has-long 123456789))))
(  (lambda (T00022)
      (set-vector-attribute-list
         (list (  (lambda (T00024)
                           (vseti-long T00024 2 (or 123456789 _SAT_IMISSING))
                           (set-vector-attribute-list '(66 77 88)
```



```
                                                           (cxr 25 a-short) 0 3 1
                                                           T00024)
                                   (rplacx 4 a-vfop (|1+| (cxr 4 a-vfop)))
                                   T00024)
                           (copy-vector (cxr 0 (cxr 6 a-vfop)))))
                    (cxr 25 an-allocate-vfop) 1 3 4 T00022)
         (rplacx 4 a-vfum (|1+| (cxr 4 a-vfum)))
         T00022)
   (copy-vector (cxr 0 (cxr 6 a-vfum))))
-> (macroexpand '(setf (has-z (a-vfum x))
                          (a-vfop has-short nil has-ubit1 1)))
(set-vector-element-attribute-function
   (  (lambda (T00026)
                (set-vector-attribute-list nil (cxr 25 a-short) 0 3 1 T00026)
                (vseti-ubit-function 1 nil 48 T00026)
                (rplacx 4 a-vfop (|1+| (cxr 4 a-vfop)))
                T00026)
      (copy-vector (cxr 0 (cxr 6 a-vfop))))
   (cxr 25 an-allocate-vfop) 14 x)
-> (macroexpand '(setf (has-z (a-vfum x) has-ubit2) t))
(vseti-1bit-function t nil (+ 49 (* 32 14)) x)
-> (setq x (a-vfum has-x (list (a-vfop has-long 123456789))))
(a-vfum has-x
         (  (a-vfop has-short (66 77 88) has-ubit1 0 has-ubit3 0
                    has-long 123456789))
        has-z (a-vfop has-ubit1 0 has-ubit3 0))
-> (setf (has-z (a-vfum x)) (a-vfop has-short nil has-ubit1 1))
(a-vfop has-ubit1 1 has-ubit3 0)
-> (setf (has-z (a-vfum x) has-ubit2) t)
t
-> x
(a-vfum has-x
         (  (a-vfop has-short (66 77 88) has-ubit1 0 has-ubit3 0
                    has-long 123456789))
        has-z (a-vfop has-ubit1 1 has-ubit2 t has-ubit3 0))
-> (close *C-definition-code-port*)
t
-> (setq *C-definition-code-port* nil)
nil
-> (exec cat sob_vdemo.lh)
typedef struct sob_vstruct * sob_vfoo;
struct sob_vstruct {
        union { int SOB_VSIZE [1];
                sat_lvalue * SOB_VPLIST [1];
                sob_type SOB_VTYPE; } SOB_VFIRST;
#       define sob_vtype SOB_VFIRST.SOB_VTYPE
#       define sob_vvsize SOB_VFIRST.SOB_VSIZE[-2]
        sat_lvalue sob_vfiddle;
        unsigned char sob_vPAD64 [2];
        short sob_vfum;
        sat_lvalue sob_vfi;
        };
#define sob_valloc(x,y) struct sob_vstruct (x) [y]
typedef struct sob_vfstruct * sob_vfop;
struct sob_vfstruct {
        short sob_vfshort [3];
```

```
        unsigned char sob_vfubit1:1,
                :1,
                sob_vfubit3:1,
                :5;
        unsigned char sob_vfPAD56 [1];
        long sob_vflong;
        sat_lvalue sob_vfvalue;
        ];
#define sob_vfalloc(x,y) struct sob_vfstruct (x) [y]
typedef struct sob_vustruct * sob_vfum;
struct sob_vustruct {
        union { int SOB_VSIZE [1];
                sat_lvalue * SOB_VPLIST [1];
                sob_type SOB_VTYPE; ] SOB_VFIRST;
#       define sob_vutype SOB_VFIRST.SOB_VTYPE
#       define sob_vuplist SOB_VFIRST.SOB_VPLIST[-1][0]
#       define sob_vuSIZE SOB_VFIRST.SOB_VSIZE[-2]
        sob_vfalloc (sob_vux, 3);
        long sob_vuy;
        sob_vfalloc (sob_vuz, 1);
        sob_vfop sob_vuw;
        unsigned char sob_vuPAD608 [12];
        ];
#define sob_vualloc(x,y) struct sob_vustruct (x) [y]
0
-> (declare-vector-type (a-vfum sob_vfum sob_vu)
                        has-C-vsize-vector-element-name sob_vuSIZE
                        is-read-init-write in-vector an-allocate-vfop 3
                        (has-x nil sob_vux) a-long (has-y nil sob_vuy)
                        an-allocate-vfop (has-z nil sob_vuz) a-vfop
                        (has-w nil sob_vuw) is-read-private a-long 3
                        has-pad)
a-vfum
->
Goodbye
```



```
-> (sstatus data-search-path (|.|))
(|.|)
-> (exec rm -f |\\#sca_demo.1\\#|)
0
-> (exec echo |>\\#sca_demo.1\\#|)
0
-> (setq port (get-random-port nil '|#sca_demo.1#| 'write))
#<port #sca_demo.1#>
-> (do ((i 1 (|1+| i)) (r '(0) (cons (fseek port 0 1) r)))
       ((>& i 10) (setq positions (nreverse r)))
       (print (list 'hello 'there '- 'I 'am 'message i) port)
       (terpri port))
(0 31 62 93 124 155 186 217 248 279 311)
-> (drain port)
nil
-> (exec cat |\\#sca_demo.1\\#|)
(hello there - I am message 1)
(hello there - I am message 2)
(hello there - I am message 3)
(hello there - I am message 4)
(hello there - I am message 5)
(hello there - I am message 6)
(hello there - I am message 7)
(hello there - I am message 8)
(hello there - I am message 9)
(hello there - I am message 10)
0
-> (do l positions
       (restl l)
       (null l)
       (setq port
             (get-random-port port (list '|#sca_demo.1#| (first l))
                                   'read))
       (print (read port 'end-of-file))
       (terpri))
(hello there - I am message 1)
(hello there - I am message 2)
(hello there - I am message 3)
(hello there - I am message 4)
(hello there - I am message 5)
(hello there - I am message 6)
(hello there - I am message 7)
(hello there - I am message 8)
(hello there - I am message 9)
(hello there - I am message 10)
end-of-file
nil
-> (print (get-random-port-location port))
(|#sca_demo.1#| 311)nil
-> (setq port
         (get-random-port port (list '|#sca_demo.1#| (sixth positions))
                               'write))
#<port #sca_demo.1#>
-> (do (   (i 6 (|1+| i))
           (r (nreverse (copy-list positions 6)) (cons (fseek port 0 1) r)))
       ((>& i 10) (setq positions (nreverse r)))
```

```
       (print (cons 'hello
                    (cons 'there
                          (cons '-
                                (cons 'I
                                      (cons 'am
                                            (cons 'message
                                                  (cons i '(+))))))))
              port)
       (terpri port))
(0 31 62 93 124 155 188 221 254 287 321)
-> (setq port (get-random-port port '(|#sca_demo.1#| end-of-file) 'write))
#<port #sca_demo.1#>
-> (print '(I am the last message) port)
nil
-> (drain port)
nil
-> (exec cat |\\#sca_demo.1\\#|)
(hello there - I am message 1)
(hello there - I am message 2)
(hello there - I am message 3)
(hello there - I am message 4)
(hello there - I am message 5)
(hello there - I am message 6 +)
(hello there - I am message 7 +)
(hello there - I am message 8 +)
(hello there - I am message 9 +)
(hello there - I am message 10 +)
(I am the last message)0
-> (do i positions
       (restl l)
       (null l)
       (setq port
             (get-random-port port (list '|#sca_demo.1#| (first l))
                              'read))
       (print (read port 'end-of-file))
       (terpri))
(hello there - I am message 1)
(hello there - I am message 2)
(hello there - I am message 3)
(hello there - I am message 4)
(hello there - I am message 5)
(hello there - I am message 6 +)
(hello there - I am message 7 +)
(hello there - I am message 8 +)
(hello there - I am message 9 +)
(hello there - I am message 10 +)
(I am the last message)
nil
-> (progn (print (read port 'end-of-file)) (terpri))
end-of-file
nil
-> (get-random-port port)
nil
-> (sstatus catalog-search-path (|.|))
(|.|)
-> (exec rm -f |\\#sca_demo.ca|)
```



```
0
-> (setq c (new-catalog '|#sca_demo.ca|))
(a-catalog has-file |#sca_demo.ca|)
-> (write-catalog c '(hello there this is message 1))
nil
-> (get-catalog-location c)
(0 0)
-> (write-catalog c
                  (a-man has-weight 99 has-name 'Jim
                         has-hand-sizes '(93 95)))
nil
-> (get-catalog-location c)
(33 0)
-> (close-catalog c)
nil
-> (exec cat |\\#sca_demo.ca|)
'(hello there this is message 1)
(a-man has-weight 99 has-name 'Jim has-hand-sizes '(93 95))
0
-> (read-catalog c)
(hello there this is message 1)
-> (get-catalog-location c)
(0 0)
-> (read-catalog c)
Jim
-> (get-catalog-location c)
(32 0)
-> (read-catalog c)
end-of-catalog
-> (close-catalog c)
nil
-> (exec rm -f |\\#sca_demo2.ca| |\\#sca_demo2.ci|)
0
-> (setq c2 (new-catalog '|#sca_demo2.ca|))
(a-catalog has-file |#sca_demo2.ca|)
-> (write-catalog c2 '(the first message in c2))
nil
-> (write-catalog c2
                  (list 'please-include
                        (a-catalog has-file '|#sca_demo.ca|)))
nil
-> (read-catalog c2)
(the first message in c2)
-> (write-catalog c2 '(the last message in c2))
nil
-> (exec cat |\\#sca_demo2.ca|)
'(the first message in c2)
`(please-include ,(a-catalog has-file '|#sca_demo.ca|))
'(the last message in c2)
0
-> (read-catalog c2)
(the first message in c2)
-> (setq l1 (get-catalog-location c2))
(0 0)
-> (read-catalog c2)
(hello there this is message 1)
```

```
-> (setq 12 (get-catalog-location c2))
(included-catalog-location (26 0) (0 0))
-> (read-catalog c2)
Jim
-> (setq 13 (get-catalog-location c2))
(included-catalog-location (26 0) (32 0))
-> (read-catalog c2)
(the last message in c2)
-> (setq 14 (get-catalog-location c2))
(82 0)
-> (read-catalog c2)
end-of-catalog
-> (setq 15 (get-catalog-location c2))
(108 0)
-> (read-catalog c2 11)
(the first message in c2)
-> (read-catalog c2 12)
(hello there this is message 1)
-> (read-catalog c2 13)
Jim
-> (read-catalog c2 14)
(the last message in c2)
-> (read-catalog c2 15)
end-of-catalog
-> (setq c3 (a-catalog is-index-of c2 has-index-function '(lambda (x y) y)))
(a-catalog is-index-of
              (a-catalog has-file |#sca_demo2.ca| has-function file-catalog)
          has-index-function (lambda (x y) y))
-> (read-catalog c3 3)
Jim
-> (read-catalog c3 1)
(the first message in c2)
-> (get-catalog-location c3)
1
-> (read-catalog c3)
(hello there this is message 1)
-> (get-catalog-location c3)
2
-> (read-catalog c3)
Jim
-> (get-catalog-location c3)
3
-> (read-catalog c3)
(the last message in c2)
-> (get-catalog-location c3)
4
-> (read-catalog c3)
end-of-catalog
-> (get-catalog-location c3)
end-of-catalog
-> (read-catalog c3)
end-of-catalog
-> (get-catalog-location c3)
end-of-catalog
-> (read-catalog c3 4)
(the last message in c2)
```



```
-> (get-catalog-location c3)
4
-> (close-catalog c3)
nil
-> (read-catalog c3)
(the first message in c2)
-> (get-catalog-location c3)
1
-> (read-catalog c3)
(hello there this is message 1)
-> (get-catalog-location c3)
2
-> (exec "cat \\#sca_demo2.ci; ls -l \\#sca_demo2.ci; sleep 2")
(lambda (x y) y)
(1 1 (0 0))
(2 2 (included-catalog-location (26 0) (0 0)))
(3 3 (included-catalog-location (26 0) (32 0)))
(4 4 (82 0))
(end-of-catalog 5 (109 0))

-rw-r--r--  1 walton        165 Apr 27 15:36 #sca_demo2.ci
0
-> (close-catalog c3)
nil
-> (exec ls -l |\\#sca_demo2.ci|)
-rw-r--r--  1 walton        165 Apr 27 15:36 #sca_demo2.ci
0
-> (exec rm -f |\\#sca_demo3.ca| |\\#sca_demo3.ci|)
0
-> (exec echo catalog-number |>\\#sca_demo3.ci|)
0
-> (copy-catalog '|#sca_demo2.ca| '|#sca_demo3.ca|)
4
(compute-time - 0.216667 seconds / gc-time - 1.51667 seconds for 1 gcs)
-> (make-catalog-index '|#sca_demo3.ca|)
4
-> (exec "cat \\#sca_demo3.ca; ls -l \\#sca_demo3.ca; sleep 2")
'(the first message in c2)
'(hello there this is message 1)
(a-man has-weight 99 has-name 'Jim has-hand-sizes '(93 95))
'(the last message in c2)
-rw-r--r--  1 walton        146 Apr 27 15:36 #sca_demo3.ca
0
-> (exec "cat \\#sca_demo3.ci; ls -l \\#sca_demo3.ci; sleep 2")
catalog-number
(1 1 (0 0))
(2 2 (26 0))
(3 3 (59 0))
(4 4 (119 0))
(end-of-catalog 5 (145 0))

-rw-r--r--  1 walton         95 Apr 27 15:36 #sca_demo3.ci
0
-> (get-catalog-keys c3)
(1 2 3 4 end-of-catalog)
-> (copy-catalog '|#sca_demo3.ca| '|#sca_demo4.ca| '(4 3 1))
```

```
3
(compute-time - 1.68333 seconds)
-> (append-catalog '|#sca_demo3.ca| '|#sca_demo4.ca| '(2))
1
-> (exec "cat \\#sca_demo4.ca; ls -l \\#sca_demo4.ca; sleep 2")
'(catalog-key 4)
'(the last message in c2)
'(catalog-key 3)
(a-man has-weight 99 has-name 'Jim has-hand-sizes '(93 95))
'(catalog-key 1)
'(the first message in c2)
'(catalog-key 2)
'(hello there this is message 1)
-rw-r--r--  1 walton        214 Apr 27 15:36 #sca_demo4.ca
0
-> (exec "cat \\#sca_demo4.ci; ls -l \\#sca_demo4.ci; sleep 2")
nil
(4 1 (0 0))
(3 3 (42 0))
(1 5 (119 0))
(2 7 (163 0))
(end-of-catalog 9 (213 0))

-rw-r--r--  1 walton         85 Apr 27 15:36 #sca_demo4.ci
0
->
Goodbye
```



```
-> (setq x  (an-array has-sizes '(9 9)
                      by-expression '(plus Y (product 10 X))))
(an-array by-expression (plus Y (product 10 X)) has-element-type a-long
          has-exponent -16 has-sizes (9 9))
-> x
(an-array by-expression (plus Y (product 10 X)) has-element-type a-long
          has-exponent -16 has-sizes (9 9))
-> (allocate-array x)
(an-array has-element-type a-long has-exponent -16 has-sizes (9 9))
-> x
(an-array has-element-type a-long has-exponent -16 has-sizes (9 9))
-> (has-parent-sizes x)
(9 9 1 1 1 1)
-> (has-desired-sizes x)
(9 9 1 1 1 1)
-> (has-sizes x)
(9 9 1 1 1 1)
-> (has-parent-increments x)
(1 9 81 81 81 81)
-> (has-increments x)
(1 9 81 81 81 81)
-> (has-element-type x)
a-long
-> (has-exponent x)
-16
-> (print-array x)
       0       10       20       30       40       50       60       70       80
       1       11       21       31       41       51       61       71       81
       2       12       22       32       42       52       62       72       82
       3       13       23       33       43       53       63       73       83
       4       14       24       34       44       54       64       74       84
       5       15       25       35       45       55       65       75       85
       6       16       26       36       46       56       66       76       86
       7       17       27       37       47       57       67       77       87
       8       18       28       38       48       58       68       78       88
nil
-> (sstatus catalog-search-path (|.|))
(|.|)
-> (sstatus data-search-path (|.|))
(|.|)
-> (exec rm -f |\\#cache.ar| |\\#sar_demo.ca|)
0
-> (setq *default-array-file* '(|#cache.ar| end-of-file))
(|#cache.ar| end-of-file)
-> (setq c (new-catalog '|#sar_demo.ca|))
(a-catalog has-file |#sar_demo.ca|)
-> (has-been-changed x)
t
-> (setf (has-desired-sizes x) '(5 5))
(5 5)
-> (setf (has-desired-origins x) '(5 5))
(5 5)
-> (write-catalog c x)
nil
(compute-time - 0.25 seconds / gc-time - 1.48333 seconds for 1 gcs)
-> (has-been-changed x)
```

```
nil
-> (setq z (read-catalog c))
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (5 5) has-origins (5 5))
-> (has-been-changed z)
nil
-> (allocate-array z)
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (5 5) has-origins (5 5))
-> (has-been-changed z)
nil
-> (setf (has-been-changed z) t)
t
-> (has-been-changed z)
t
-> z
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (5 5) has-origins (5 5))
-> (print-array z)
      55      65      75      85
      56      66      76      86
      57      67      77      87
      58      68      78      88
nil
-> (reset-array z)
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9))
-> (print-array z)
       0      10      20      30      40      50      60      70      80
       1      11      21      31      41      51      61      71      81
       2      12      22      32      42      52      62      72      82
       3      13      23      33      43      53      63      73      83
       4      14      24      34      44      54      64      74      84
       5      15      25      35      45      55      65      75      85
       6      16      26      36      46      56      66      76      86
       7      17      27      37      47      57      67      77      87
       8      18      28      38      48      58      68      78      88
nil
-> (reset-array x)
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9))
-> (inspect-array x '(5 5))

       0      10      20      30      40
       1      11      21      31      41
       2      12      22      32      42
       3      13      23      33      43
       4      14      24      34      44
(0 0 0 0 0 0) x
       2      12      22      32      42
       3      13      23      33      43
       4      14      24      34      44
       5      15      25      35      45
```



```
       6      16      26      36      46
(0 2 0 0 0 0) D
      52      62      72      82
      53      63      73      83
      54      64      74      84
      55      65      75      85
      56      66      76      86
(5 2 0 0 0 0) ce
empty array
(9 -2 0 0 0 0) ( 2  3  )
      23      33      43      53      63
      24      34      44      54      64
      25      35      45      55      65
      26      36      46      56      66
      27      37      47      57      67
(2 3 0 0 0 0) $
nil
-> (place-array x '(3 3) '(1 4))
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (3 3) has-origins (1 4))
-> (has-sizes x)
(3 3 1 1 1 1)
-> (has-parent-sizes x)
(9 9 1 1 1 1)
-> (has-origins x)
(1 4 0 0 0 0)
-> (print-array x)
      14      24      34
      15      25      35
      16      26      36
nil
-> (place-array x nil '(-2 -1))
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (3 3) has-origins (-2 -1))
-> (has-desired-origins x)
(-2 -1 0 0 0 0)
-> (has-origins x)
(0 0 0 0 0 0)
-> (has-desired-sizes x)
(3 3 1 1 1 1)
-> (has-sizes x)
(1 2 1 1 1 1)
-> (print-array x)
       0
       1
nil
-> (reset-array x)
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9))
-> (setf (has-steps x) '(3 4))
(3 4)
-> (print-array x)
       0      30      60
       4      34      64
```

```
      8       38       68
nil
-> (reset-array x)
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9))
-> (place-array x '(4 4) '(4 4))
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (4 4) has-origins (4 4))
-> (set-array-by-expression x 0)
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (4 4) has-origins (4 4))
-> (place-array x '(6 6) '(3 3))
(an-array has-array-file (|#cache.ar| 0) has-element-type a-long
          has-exponent -16 has-array-format motorola has-sizes (9 9)
          has-desired-sizes (6 6) has-origins (3 3))
-> (print-array x)
     33      43      53      63      73      83
     34       0       0       0       0      84
     35       0       0       0       0      85
     36       0       0       0       0      86
     37       0       0       0       0      87
     38      48      58      68      78      88
nil
-> (setq y (an-array has-sizes '(2 2) by-value '((1 -1) (-1 1))))
(an-array by-value ((1 -1) (-1 1)) has-element-type a-long has-exponent -16
          has-sizes (2 2))
-> (print-array y)
      1      -1
     -1       1
nil
-> (setq z (mirror-of-array y '(2 2)))
(an-array has-element-type a-long has-exponent -16 has-sizes (6 6))
-> (print-array z)
      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
nil
-> (setq x (an-array has-sizes '(5 5) by-expression '(diff X Y)))
(an-array by-expression (diff X Y) has-element-type a-long has-exponent -16
          has-sizes (5 5))
-> (summary-of-array x)
(an-array-summary has-count 25 has-missing-count 0 has-minimum -4.0
                  has-maximum 4.0 has-sum 0.0 has-sum-squares 100.0
                  has-mean 0.0 has-standard-deviation 2.04124)
-> (bounds-of-array x 0.0)
(-4.0 4.0)
-> (bounds-of-array x)
(-4.00008 4.00008)
-> (log-bound-of-array x)
3
-> (setq z (prepare-array x '(1 2) a-float))
```



```
(an-array has-element-type a-float has-sizes (7 7))
-> (print-array z)
      0       0       1       2       3       4       4
      0       0       1       2       3       4       4
     -1      -1       0       1       2       3       3
     -2      -2      -1       0       1       2       2
     -3      -3      -2      -1       0       1       1
     -4      -4      -3      -2      -1       0       0
     -4      -4      -3      -2      -1       0       0
nil
-> (has-element-type z)
a-float
-> (setq zz (prepare-array z nil a-long))
(an-array has-element-type a-long has-exponent -21 has-sizes (7 7))
-> (print-array zz)
      0       0       1       2       3       4       4
      0       0       1       2       3       4       4
     -1      -1       0       1       2       3       3
     -2      -2      -1       0       1       2       2
     -3      -3      -2      -1       0       1       1
     -4      -4      -3      -2      -1       0       0
     -4      -4      -3      -2      -1       0       0
nil
-> (has-element-type zz)
a-long
-> (has-exponent zz)
-21
-> (setq x  (an-array has-sizes '(9) by-expression 'X
                      has-desired-sizes '(4)))
(an-array by-expression X has-element-type a-long has-exponent -16
          has-sizes (9) has-desired-sizes (4))
-> (print-array x)
      0       1       2       3
nil
-> (has-exponent x)
-16
-> (setq y (an-array x has-element-type a-short))
(an-array has-element-type a-short has-exponent -11 has-sizes (9)
          has-desired-sizes (4))
-> (has-exponent y)
-11
-> (print-array y)
      0       1       2       3
nil
-> (reset-array y)
(an-array has-element-type a-short has-exponent -11 has-sizes (9))
-> (print-array y)
      0       1       2       3       4       5       6       7       8
nil
-> (normalize-ruler '(3 (4.5 10.5)))
((-0.5 2.5) (4.5 10.5) 2.0)
-> (normalize-ruler '(3 (4.5 9.5) (1)))
((-0.5 2.5) (4.5 10.5) 2.0)
-> (normalize-ruler '(3 (4.5 9.5) (1 ((t t) (nil t)))))
((-0.5 2.5) (3.5 9.5) 2.0)
-> (normalize-ruler '(nil (4.5 10.5) 2))
```

```
((-0.5 2.5) (4.5 10.5) 2.0)
-> (integerize-ruler '((2 3.9) (4 nil) 2))
(5 (-1.0 9.0) 2.0)
-> (declare-vector-type (a-complex sar_complex sar_com) has-allocate-C-type
                        an-allocate-complex a-float has-real has-imaginary)
a-complex
-> (setq x (an-array has-element-type an-allocate-complex has-sizes '(2 2)))
(an-array has-element-type an-allocate-complex has-sizes (2 2))
-> (setf (has-element x 0 0) (a-complex has-real 0.0 has-imaginary 0.0))
(a-complex has-real 0.0 has-imaginary 0.0)
-> (setf (has-element x 1 0) (a-complex has-real 1.0 has-imaginary 0.0))
(a-complex has-real 1.0 has-imaginary 0.0)
-> (setf (has-element x 0 1) (a-complex has-real 0.0 has-imaginary 1.0))
(a-complex has-real 0.0 has-imaginary 1.0)
-> (setf (has-element x 1 1) (a-complex has-real 1.0 has-imaginary 1.0))
(a-complex has-real 1.0 has-imaginary 1.0)
-> (has-element x 0 0)
(a-complex has-real 0.0 has-imaginary 0.0)
-> (has-element x 1 0)
(a-complex has-real 1.0 has-imaginary 0.0)
-> (has-element x 0 1)
(a-complex has-real 0.0 has-imaginary 1.0)
-> (has-element x 1 1)
(a-complex has-real 1.0 has-imaginary 1.0)
-> (setq x (an-array has-sizes '(9 4) by-expression '(+ X (* 10 Y))))
(an-array by-expression (+ X (* 10 Y)) has-element-type a-long
        has-exponent -16 has-sizes (9 4))
-> (print-array x)
      0       1       2       3       4       5       6       7       8
     10      11      12      13      14      15      16      17      18
     20      21      22      23      24      25      26      27      28
     30      31      32      33      34      35      36      37      38
nil
-> (setq y (reorganization-of-array x '(0 0) '(3 12)))
(an-array has-element-type a-long has-exponent -16 has-sizes (3 12))
-> (print-array y)
      0       1       2
      3       4       5
      6       7       8
     10      11      12
     13      14      15
     16      17      18
     20      21      22
     23      24      25
     26      27      28
     30      31      32
     33      34      35
     36      37      38
nil
-> (setq z (reorganization-of-array x '(0 2) '(18)))
(an-array has-element-type a-long has-exponent -16 has-sizes (18))
-> (print-array z)
     20      21      22      23      24      25      26      27      28
      30      31      32      33      34      35      36      37      38
nil
-> (comment '(the following is an error))
```



```
comment
-> (reorganization-of-array x '(0 1) '(3 12))
ERROR - (sar_reorganize - new array will not fit inside some ancestor of
                        array argument)
WHILE EVALUATING - (ccheck (0))
WHILE EVALUATING - (reorganization-of-array x '(0 1) '(3 12))

BREAK -
<1> ^D
[Return to Top Level]
->
Goodbye
```

```
-> (setq x (an-array has-sizes '(100)))
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
-> (allocate-array x)
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
-> (has-array-id x)
16
-> (collect-array-blocks)
nil
(compute-time - 0.0 seconds / sweep-time - 1.13333 seconds for sweeping 0
              bytes in 1 sweeps / compact-time - 0.0166667 seconds for
              moving 0 bytes in 1 compactifications)
-> (allocate-array x)
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
-> (has-array-id x)
16
-> (setq *collect-blocks-test* (copy-of-array x))
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
-> (allocate-array *collect-blocks-test*)
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
-> (has-array-id *collect-blocks-test*)
17
-> (collect-array-blocks)
nil
-> (allocate-array *collect-blocks-test*)
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
-> (has-array-id *collect-blocks-test*)
17
-> (setq y (collect-array-blocks (copy-of-array x)))
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
(compute-time - 0.0166667 seconds / sweep-time - 1.05 seconds for sweeping 0
              bytes in 1 sweeps / compact-time - 0.0166667 seconds for
              moving 0 bytes in 1 compactifications)
-> (allocate-array y)
(an-array has-element-type a-long has-exponent -16 has-sizes (100))
-> (has-array-id y)
18
-> (setq l1 nil)
nil
-> (do ((i 0 (|1+| i)) (l2 nil)) ((> i 1000))
       (let (l2) (setq x (an-array has-sizes (list (+ 100 (mod i 21))))))
       (assert (not (log-bound-of-array x))
               (cons 'array (cons i '(was non-zero when allocated))))
       (caseq (mod i 3) (1 (push x l1)) (2 (push x l2)))
       (caseq (mod i 19) (7 (collect-array-blocks))
              (18 (dolist (y l1)
                          (assert (not (log-bound-of-array y))
                                  '(array in l1 became non-zero)))
                  (dolist (y l2)
                          (assert (not (log-bound-of-array y))
                                  '(array in l2 became non-zero)))
                  (setq l1 nil l2 nil))))
nil
(compute-time - 35.5333 seconds / gc-time - 16.8333 seconds for 11 gcs /
              sweep-time - 56.3333 seconds for sweeping 75624 bytes in 53
              sweeps / compact-time - 0.4 seconds for moving 142752 bytes in
              53 compactifications)
```



```
-> (do ((i 0 (|1+| i)) (l nil)) ((> i 1000))
       (setq x (an-array has-sizes (list (+ 100 (mod i 19)))))
       (push x l)
       (assert (not (log-bound-of-array x))
               '(failed to allocate zero array when region grown)))
nil
(compute-time - 28.85 seconds / gc-time - 20.5667 seconds for 11 gcs /
              sweep-time - 2.2 seconds for sweeping 1824 bytes in 2 sweeps /
              compact-time - 0.0 seconds for moving 13512 bytes in 2
              compactifications)
-> *array-blocks-history*
(  (compute 0.483333 seconds) (sweep 1.13333 seconds)
   (compact 0.0166667 seconds) (compute 0.631333 seconds)
   (sweep 0.966667 seconds) (compute 0.2 seconds) (sweep 1.05 seconds)
   (compact 0.0166667 seconds) (compute 1.58333 seconds)
   (sweep 1.08333 seconds) (compute 0.683333 seconds) (sweep 1.05 seconds)
   (compute 0.683333 seconds) (sweep 1.1 seconds) (compute 0.681333 seconds)
   (sweep 1.05 seconds) (compact 0.0333333 seconds)
   (compute 0.683333 seconds) (sweep 1.06667 seconds)
   (compact 0.0333333 seconds) (compute 0.666667 seconds)
   (sweep 1.08333 seconds) (compute 0.683333 seconds)
   (sweep 1.08333 seconds) (compact 0.0166667 seconds)
   (compute 0.666667 seconds) (sweep 1.05 seconds)
   (compact 0.0166667 seconds) (compute 0.683333 seconds)
   (sweep 1.08333 seconds) (compute 0.683333 seconds)
   (sweep 1.06667 seconds) (compact 0.0333333 seconds)
   (compute 0.65 seconds) (sweep 1.05 seconds) (compact 0.0166667 seconds)
   (compute 0.65 seconds) (sweep 1.1 seconds) (compact 0.0333333 seconds)
   (compute 0.65 seconds) (sweep 1.08333 seconds)
   (compact 0.0166667 seconds) (compute 0.666667 seconds)
   (sweep 0.983333 seconds) (compute 0.0166667 seconds)
   (compute 0.683333 seconds) (sweep 1.08333 seconds)
   (compact 0.0166667 seconds) (compute 0.666667 seconds)
   (sweep 1.06667 seconds))
->
Goodbye
```

```
-> (setq x (an-array has-sizes '(3 4) by-expression 'X))
(an-array by-expression X has-element-type a-long has-exponent -16
          has-sizes (3 4))
-> (setq y (an-array has-sizes '(3 4) by-expression 'Y))
(an-array by-expression Y has-element-type a-long has-exponent -16
          has-sizes (3 4))
-> (setq z (an-array has-sizes '(3 4)))
(an-array has-element-type a-long has-exponent -16 has-sizes (3 4))
-> (print-array (add-to-array-elements (copy-array z x) 5))
        5        6        7
        5        6        7
        5        6        7
        5        6        7
nil
(compute-time - 0.0666667 seconds / gc-time - 1.55 seconds for 1 gcs)
-> (print-array (multiply-array-elements-by (copy-array z x) 5))
        0        5       10
        0        5       10
        0        5       10
        0        5       10
nil
-> (print-array (add-arrays z x y))
        0        1        2
        1        2        3
        2        3        4
        3        4        5
nil
-> (print-array (subtract-arrays z x y))
        0        1        2
       -1        0        1
       -2       -1        0
       -3       -2       -1
nil
-> (print-array (minimize-arrays z x y))
        0        0        0
        0        1        1
        0        1        2
        0        1        2
nil
-> (print-array (maximize-arrays z x y))
        0        1        2
        1        1        2
        2        2        2
        3        3        3
nil
-> (print-array (multiply-array-elements z x y))
        0        0        0
        0        1        2
        0        2        4
        0        3        6
nil
-> (setq x (an-array has-sizes '(9) by-expression '(diff X 4)))
(an-array by-expression (diff X 4) has-element-type a-long has-exponent -16
          has-sizes (9))
-> (setq z (an-array has-sizes '(9)))
(an-array has-element-type a-long has-exponent -16 has-sizes (9))
```



```
-> (print-array (absolute-value-array-elements z x))
       4        3        2        1        0        1        2        3        4
nil
-> (print-array (log-array-elements z x))
     nil      nil      nil      nil      nil        0   0.6931   1.0986   1.3863
nil
-> (print-array (exponentiate-array-elements z x))
  0.0183   0.0498   0.1353   0.3679        1   2.7183   7.3891  20.0855  54.5981
nil
-> (print-array (square-root-array-elements z x))
     nil      nil      nil      nil        0        1   1.4142   1.7321        2
nil
-> (print-array (power-array-elements z x 0.5))
     nil      nil      nil      nil        0        1   1.4142   1.7321        2
nil
-> (print-array (sin-array-elements z x))
  0.7568  -0.1411  -0.9093  -0.8415        0   0.8415   0.9093   0.1411  -0.7568
nil
-> (print-array (cos-array-elements z x))
 -0.6536    -0.99  -0.4162   0.5403        1   0.5403  -0.4162    -0.99  -0.6536
nil
-> (print-array (arcsin-array-elements z x))
     nil      nil      nil  -1.5708        0   1.5708      nil      nil      nil
nil
-> (print-array (arccos-array-elements z x))
     nil      nil      nil   3.1416   1.5708        0      nil      nil      nil
nil
-> (print-array (gaussian-kernel '(1.5 1.5)))
        0  0.00004  0.00022  0.00055  0.00055  0.00022  0.00004        0
  0.00004  0.00055  0.00324  0.00787  0.00787  0.00324  0.00055  0.00004
  0.00022  0.00324  0.01915  0.04657  0.04657  0.01915  0.00324  0.00022
  0.00055  0.00787  0.04657  0.11328  0.11328  0.04657  0.00787  0.00055
  0.00055  0.00787  0.04657  0.11328  0.11328  0.04657  0.00787  0.00055
  0.00022  0.00324  0.01915  0.04657  0.04657  0.01915  0.00324  0.00022
  0.00004  0.00055  0.00324  0.00787  0.00787  0.00324  0.00055  0.00004
        0  0.00004  0.00022  0.00055  0.00055  0.00022  0.00004        0
nil
-> (print-array (del2g-kernel '(1 1)))
        0        0-0.00001-0.00004-0.00004-0.00001        0        0
        0-0.00004-0.00132-0.00715-0.00715-0.00132-0.00004        0
-0.00001-0.00132-0.03364-0.10654-0.10654-0.03364-0.00132-0.00001
-0.00004-0.00715-0.10654   0.2624   0.2624-0.10654-0.00715-0.00004
-0.00004-0.00715-0.10654   0.2624   0.2624-0.10654-0.00715-0.00004
-0.00001-0.00132-0.03364-0.10654-0.10654-0.03364-0.00132-0.00001
        0-0.00004-0.00132-0.00715-0.00715-0.00132-0.00004        0
        0        0-0.00001-0.00004-0.00004-0.00001        0        0
nil
-> (print-array (dxg-kernel '(1 1)))
        0        0        0        0        0        0        0        0
        0-0.00001-0.00034-0.00085  0.00085  0.00034  0.00001        0
        0-0.00057  -0.0188-0.04631  0.04631   0.0188  0.00057        0
-0.00001-0.00424-0.13893  -0.3422   0.3422  0.13893  0.00424  0.00001
-0.00001-0.00424-0.13893  -0.3422   0.3422  0.13893  0.00424  0.00001
        0-0.00057  -0.0188-0.04631  0.04631   0.0188  0.00057        0
        0-0.00001-0.00034-0.00085  0.00085  0.00034  0.00001        0
        0        0        0        0        0        0        0        0
```

```
nil
-> (setq ac (an-array has-sizes '(9) has-exponent 0))
(an-array has-element-type a-long has-exponent 0 has-sizes (9))
-> (set-array-elements ac 1)
(an-array has-element-type a-long has-exponent 0 has-sizes (9))
-> (print-array ac)
      1      1      1      1      1      1      1      1      1
nil
-> (setq mv (an-array has-sizes '(9 5)
                  by-value
                    '(  (99 99 4 2 3 4 8 9 6) (99 99 99 3 4 6 7 99 9)
                        (1 99 99 24 5 99 19 8 5) (3 3 99 3 5 8 7 4 8)
                        (2 4 3 4 6 7 8 6 7))))
(an-array by-value
          (  (99 99 4 2 3 4 8 9 6) (99 99 99 3 4 6 7 99 9)
             (1 99 99 24 5 99 19 8 5) (3 3 99 3 5 8 7 4 8)
             (2 4 3 4 6 7 8 6 7))
        has-element-type a-long has-exponent -16 has-sizes (9 5))
-> (print-array mv)
     99     99      4      2      3      4      8      9      6
     99     99     99      3      4      6      7     99      9
      1     99     99     24      5     99     19      8      5
      3      3     99      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (print-array (setq markmv (mark-missing-of mv '(-98 98) '(-10 10))))
    nil    nil      4      2      3      4      8      9      6
    nil    nil    nil      3      4      6      7    nil      9
      1    nil    nil    nil      5    nil    nil      8      5
      3      3    nil      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (print-array (shrink-missing-of markmv 4))
    nil    nil      4      2      3      4      8      9      6
    nil    nil    nil      3      4      6      7      8      9
      1    nil    nil      3      5      7      7      8      5
      3      3      3      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (print-array (setq shrinkmv (shrink-missing-of markmv 2)))
    nil      4      4      2      3      4      8      9      6
      1    2.5    3.5      3      4      6      7      8      9
      1      3      3      3      5      7      7      8      5
      3      3      3      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (print-array (expand-missing-of shrinkmv markmv nil nil 1))
    nil    nil      4      2      3      4      8      9      6
    nil    nil    3.5      3      4      6      7      8      9
      1      3      3      3      5      7      7      8      5
      3      3      3      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (print-array (expand-missing-of shrinkmv markmv nil nil 2))
    nil    nil      4      2      3      4      8      9      6
    nil    nil    nil      3      4      6      7      8      9
```



```
      1    nil    nil      3      5      7      7      8      5
      3      3      3      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (print-array (expand-missing-of shrinkmv markmv '(2 2) 1 1))
    nil    nil      4      2      3      4      8      9      6
    nil    nil    nil      3      4      6      7      8      9
      1    nil    nil      3      5      7      7      8      5
      3      3      3      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (print-array (expand-missing-of shrinkmv markmv))
    nil    nil      4      2      3      4      8      9      6
    nil    nil    nil      3      4      6      7      8      9
      1    nil    nil    nil      5      7      7      8      5
      3      3    nil      3      5      8      7      4      8
      2      4      3      4      6      7      8      6      7
nil
-> (setq dx (an-array has-sizes '(3 3)
                         by-value '((-1 0 1) (-1 0 1) (-1 0 1))))
(an-array by-value ((-1 0 1) (-1 0 1) (-1 0 1)) has-element-type a-long
        has-exponent -16 has-sizes (3 3))
-> (setq dy (an-array has-sizes '(3 3)
                         by-value '((-1 -1 -1) (0 0 0) (1 1 1))))
(an-array by-value ((-1 -1 -1) (0 0 0) (1 1 1)) has-element-type a-long
        has-exponent -16 has-sizes (3 3))
-> (scalar-product dx dx)
6.0
-> (scalar-product dx dy)
0.0
-> (scalar-product dy dy)
6.0
-> (setq ix (an-array has-sizes '(8 8) by-expression 'X))
(an-array by-expression X has-element-type a-long has-exponent -16
        has-sizes (8 8))
-> (print-array (convolution-of ix dx))
      3      6      6      6      6      6      6      3
      3      6      6      6      6      6      6      3
      3      6      6      6      6      6      6      3
      3      6      6      6      6      6      6      3
      3      6      6      6      6      6      6      3
      3      6      6      6      6      6      6      3
      3      6      6      6      6      6      6      3
      3      6      6      6      6      6      6      3
nil
-> (print-array (convolution-of ix dy))
      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
      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0
nil
-> (setq ixy (an-array has-sizes '(8 8) by-expression '(plus X Y)))
```

```
(an-array by-expression (plus X Y) has-element-type a-long has-exponent -16
        has-sizes (8 8))
-> (print-array (convolution-of ixy dx))
       3      6      6      6      6      6      6      3
       3      6      6      6      6      6      6      3
       3      6      6      6      6      6      6      3
       3      6      6      6      6      6      6      3
       3      6      6      6      6      6      6      3
       3      6      6      6      6      6      6      3
       3      6      6      6      6      6      6      3
       3      6      6      6      6      6      6      3
nil
-> (print-array (convolution-of ixy dy))
       3      3      3      3      3      3      3      3
       6      6      6      6      6      6      6      6
       6      6      6      6      6      6      6      6
       6      6      6      6      6      6      6      6
       6      6      6      6      6      6      6      6
       6      6      6      6      6      6      6      6
       6      6      6      6      6      6      6      6
       3      3      3      3      3      3      3      3
nil
-> (print-array (cached-dither 2))
       0      2
       3      1
nil
-> (print-array (cached-dither 2))
       0      2
       3      1
nil
-> (print-array (cached-dither *default-dither-size*))
       0     32      8     40      2     34     10     42
      48     16     56     24     50     18     58     26
      12     44      4     36     14     46      6     38
      60     28     52     20     62     30     54     22
       3     35     11     43      1     33      9     41
      51     19     59     27     49     17     57     25
      15     47      7     39     13     45      5     37
      63     31     55     23     61     29     53     21
nil
-> (print-array (sum-filter (copy-of-array ixy) Y-dimension 3))
       1      2      3      4      5      6      7      8
       2      3      4      5      6      7      8      9
       3      4      5      6      7      8      9     10
       4      5      6      7      8      9     10     11
       5      6      7      8      9     10     11     12
       6      7      8      9     10     11     12     13
nil
-> (print-array (accumulate-filter (copy-of-array ixy) Y-dimension))
       0      1      2      3      4      5      6      7
       1      3      5      7      9     11     13     15
       3      6      9     12     15     18     21     24
       6     10     14     18     22     26     30     34
      10     15     20     25     30     35     40     45
      15     21     27     33     39     45     51     57
      21     28     35     42     49     56     63     70
```



```
      28     36     44     52     60     68     76     84
nil
-> (print-array (maximum-filter (copy-of-array ixy) Y-dimension 3))
       2      3      4      5      6      7      8      9
       3      4      5      6      7      8      9     10
       4      5      6      7      8      9     10     11
       5      6      7      8      9     10     11     12
       6      7      8      9     10     11     12     13
       7      8      9     10     11     12     13     14
nil
-> (print-array (minimum-filter (copy-of-array ixy) Y-dimension 3))
       0      1      2      3      4      5      6      7
       1      2      3      4      5      6      7      8
       2      3      4      5      6      7      8      9
       3      4      5      6      7      8      9     10
       4      5      6      7      8      9     10     11
       5      6      7      8      9     10     11     12
nil
-> (print-array (derivative-filter (copy-of-array ixy) Y-dimension 3))
       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
nil
-> (print-array (interpolation-filter (copy-of-array ixy) Y-dimension 1.5
                                      0.5))
     0.5    1.5    2.5    3.5    4.5    5.5    6.5    7.5
       2      3      4      5      6      7      8      9
     3.5    4.5    5.5    6.5    7.5    8.5    9.5   10.5
       5      6      7      8      9     10     11     12
     6.5    7.5    8.5    9.5   10.5   11.5   12.5   13.5
nil
-> (print-array (contrast-of ixy '(1 1) 1 0))
 -0.6667 -1.3333 -2.3333 -3.3333 -4.3333 -5.3333 -6.3333      -7
 -1.3333      -2      -3      -4      -5      -6      -7 -7.6667
 -2.3333      -3      -4      -5      -6      -7      -8 -8.6667
 -3.3333      -4      -5      -6      -7      -8      -9 -9.6667
 -4.3333      -5      -6      -7      -8      -9     -10-10.6667
 -5.3333      -6      -7      -8      -9     -10     -11-11.6667
 -6.3333      -7      -8      -9     -10     -11     -12-12.6667
      -7 -7.6667 -8.6667 -9.6667-10.6667-11.6667-12.6667-13.3333
nil
-> (print-array (contrast-of ixy '(1 1) 0 1))
       0      1      2      3      4      5      6      7
       1      2      3      4      5      6      7      8
       2      3      4      5      6      7      8      9
       3      4      5      6      7      8      9     10
       4      5      6      7      8      9     10     11
       5      6      7      8      9     10     11     12
       6      7      8      9     10     11     12     13
       7      8      9     10     11     12     13     14
nil
(compute-time - 0.15 seconds / gc-time - 1.53333 seconds for 1 gcs)
-> (print-array (contrast-of ixy '(1 1)))
```

```
 -0.6667 -0.3333 -0.3333 -0.3333 -0.3333 -0.3333 -0.3333       0
 -0.3333       0       0       0       0       0       0  0.3333
 -0.3333       0       0       0       0       0       0  0.3333
 -0.3333       0       0       0       0       0       0  0.3333
 -0.3333       0       0       0       0       0       0  0.3333
 -0.3333       0       0       0       0       0       0  0.3333
 -0.3333       0       0       0       0       0       0  0.3333
       0  0.3333  0.3333  0.3333  0.3333  0.3333  0.3333  0.6667
nil
-> (print-array (interpolation-of (an-array has-sizes '(9 9)
                                            by-expression '(plus X Y))
                                  '(7 11)))
       0  1.3333  2.6667       4  5.3333  6.6667       8
     0.8  2.1333  3.4667     4.8  6.1333  7.4667     8.8
     1.6  2.9333  4.2667     5.6  6.9333  8.2667     9.6
     2.4  3.7333  5.0667     6.4  7.7333  9.0667    10.4
     3.2  4.5333  5.8667     7.2  8.5333  9.8667    11.2
       4  5.3333  6.6667       8  9.3333 10.6667      12
     4.8  6.1333  7.4667     8.8 10.1333 11.4667    12.8
     5.6  6.9333  8.2667     9.6 10.9333 12.2667    13.6
     6.4  7.7333  9.0667    10.4 11.7333 13.0667    14.4
     7.2  8.5333  9.8667    11.2 12.5333 13.8667    15.2
       8  9.3333 10.6667      12 13.3333 14.6667      16
nil
->
Goodbye
```



```
-> (setq
     m  (an-array has-element-type a-ubit has-sizes '(76 13)
                  by-expression
                    '(if (greaterp Y
                                   (plus 6
                                         (product 6
                                                  (sin (product X 0.2)))))
                         1 0)))
(an-array by-expression
          (if (greaterp Y (plus 6 (product 6 (sin (product X 0.2))))) 1
             0)
        has-element-type a-ubit has-sizes (76 13))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000001111110000000000000000000000000011111000000000000000000
0000000000000000000011111111000000000000000000000001111111110000000000000000
0000000000000000000111111111100000000000000000000011111111111000000000000000
0000000000000000001111111111110000000000000000000111111111111100000000000000
0000000000000000011111111111111000000000000000001111111111111100000000000000
0000000000000000111111111111111100000000000000001111111111111110000000000000
1000000000000001111111111111111110000000000000011111111111111111000000000000
1100000000000001111111111111111111000000000000111111111111111111100000000000
1110000000000011111111111111111111100000000001111111111111111111110000000000
1111000000000111111111111111111111110000000011111111111111111111111000000001
1111100000011111111111111111111111111000000111111111111111111111111100000011
1111111111111111111111111111111111111111111111111111111111111111111111111111
nil
(compute-time - 4.01667 seconds)
-> (setq m  (an-array has-sizes '(9 9) has-element-type a-ubit
                      by-value
                        '(  (0 0 0 0 0 0 0 0 0) (0 0 0 0 1 0 0 0 0)
                            (0 0 0 1 1 1 0 0 0) (0 0 1 1 1 1 1 0 0)
                            (0 1 1 1 1 1 1 1 0) (0 0 1 1 1 1 1 0 0)
                            (0 0 0 1 1 1 0 0 0) (0 0 0 0 1 0 0 0 0)
                            (0 0 0 0 0 0 0 0 0))))
(an-array by-value
          (  (0 0 0 0 0 0 0 0 0) (0 0 0 0 1 0 0 0 0) (0 0 0 1 1 1 0 0 0)
             (0 0 1 1 1 1 1 0 0) (0 1 1 1 1 1 1 1 0) (0 0 1 1 1 1 1 0 0)
             (0 0 0 1 1 1 0 0 0) (0 0 0 0 1 0 0 0 0) (0 0 0 0 0 0 0 0 0))
        has-element-type a-ubit has-sizes (9 9))
-> (print-array m)
000000000
000010000
000111000
001111100
011111110
001111100
000111000
000010000
000000000
nil
-> (setq m (an-array has-sizes '(76 10) has-element-type a-ubit))
(an-array has-element-type a-ubit has-sizes (76 10))
-> (bitgraph-box m 5 10 -1 9)
(an-array has-element-type a-ubit has-sizes (76 10))
(compute-time - 0.0166667 seconds / gc-time - 1.48333 seconds for 1 gcs)
```

```
-> (bitgraph-box m 11 69 4 6)
(an-array has-element-type a-ubit has-sizes (76 10))
-> (bitgraph-box m 70 75 0 11)
(an-array has-element-type a-ubit has-sizes (76 10))
-> (print-array m)
0000011111100000000000000000000000000000000000000000000000000000000000111111
0000011111100000000000000000000000000000000000000000000000000000000000111111
0000011111100000000000000000000000000000000000000000000000000000000000111111
0000011111100000000000000000000000000000000000000000000000000000000000111111
0000011111111111111111111111111111111111111111111111111111111111111111111111
0000011111111111111111111111111111111111111111111111111111111111111111111111
0000011111111111111111111111111111111111111111111111111111111111111111111111
0000011111100000000000000000000000000000000000000000000000000000000000111111
0000011111100000000000000000000000000000000000000000000000000000000000111111
0000011111100000000000000000000000000000000000000000000000000000000000111111
nil
-> (setq m (an-array has-sizes '(76 10) has-element-type a-ubit))
(an-array has-element-type a-ubit has-sizes (76 10))
-> (bitgraph-ruler m '((0 60) (10 nil) 0.2) 10 70 9 do-reverse t)
(an-array has-element-type a-ubit has-sizes (76 10))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000100000000000000000000000000000000000000000000000001000000000000000
0000000000100000000000000000000000000000000000000000000000001000000000000000
0000000000100000000000000000000000001000000000000000000000001000000000000000
0000000000100000000000000000000000001000000000000000000000001000000000000000
0000000000100001000010000100001000010000100001000010000100001000010000100000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 9)))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-parallelogram m 4 -0.5 -4.5 4.5 4.5 4.5)
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-parallelogram m 9.5 2.5 0 3 10 0)
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-parallelogram m 20.5 4 10 -4.5 10 4.5)
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-parallelogram m 42.5 3.5 0 1 10 -4)
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-parallelogram m 42.5 3.5 0 1 10 4)
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-parallelogram m 51.5 -0.5 0 1 10 4)
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-parallelogram m 51.5 7.5 0 1 10 -4)
(an-array has-element-type a-ubit has-sizes (76 9))
-> (print-array m)
0000100000000000000000000000000110000000000000000000100000000000000000000000
0001110000000000000000000000111111000000000000000111011100000000000000000000
0011111000000000000000000111111111111000000000011000000011000000000000000000
0111111100111111111100011111111111111110000011100000000000111000000000000000
1111111110111111111101111111111111111111100100000000000000000100000000000000
0111111100111111111100011111111111111110000011100000000000111000000000000000
0011111000000000000000000111111111111000000000011000000011000000000000000000
```



```
0001110000000000000000000000111111000000000000000111011100000000000000000000
0000100000000000000000000000000110000000000000000000100000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 22)))
(an-array has-element-type a-ubit has-sizes (76 22))
-> (bitgraph-parallelogram m 3 3 0 1 5 5)
(an-array has-element-type a-ubit has-sizes (76 22))
-> (bitgraph-parallelogram m 8 9 0 1 5 5)
(an-array has-element-type a-ubit has-sizes (76 22))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0001000000000000000000000000000000000000000000000000000000000000000000000000
0000100000000000000000000000000000000000000000000000000000000000000000000000
0000010000000000000000000000000000000000000000000000000000000000000000000000
0000001000000000000000000000000000000000000000000000000000000000000000000000
0000000100000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000010000000000000000000000000000000000000000000000000000000000000000000
0000000001000000000000000000000000000000000000000000000000000000000000000000
0000000000100000000000000000000000000000000000000000000000000000000000000000
0000000000010000000000000000000000000000000000000000000000000000000000000000
0000000000001000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 22)))
(an-array has-element-type a-ubit has-sizes (76 22))
-> (bitgraph-line m 3 3 13 3)
(an-array has-element-type a-ubit has-sizes (76 22))
-> (bitgraph-line m 13 4 23 4)
(an-array has-element-type a-ubit has-sizes (76 22))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0001111111111100000000000000000000000000000000000000000000000000000000000000
0000000000000111111111110000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 22)))
(an-array has-element-type a-ubit has-sizes (76 22))
-> (bitgraph-line m 3 3 13 3 2)
(an-array has-element-type a-ubit has-sizes (76 22))
-> (bitgraph-line m 13 4 23 4 2)
(an-array has-element-type a-ubit has-sizes (76 22))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0011111111111100000000000000000000000000000000000000000000000000000000000000
0011111111111111111111100000000000000000000000000000000000000000000000000000
0000000000011111111111100000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq x (an-array has-sizes '(19) by-expression 'X))
(an-array by-expression X has-element-type a-long has-exponent -16
          has-sizes (19))
-> (setf (has-ruler x X-dimension) '(19 (-0.005 nil) 0.01))
((-0.5 18.5) (-0.005 0.185) 0.01)
-> (setq b (bar-graph-of x 19 4))
(an-array has-element-type a-ubit has-sizes (76 19) has-offsets (-0.005 0.0)
          has-scales (0.04 1.0))
-> (print-array b)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000001111
0000000000000000000000000000000000000000000000000000000000000000000001111111
0000000000000000000000000000000000000000000000000000000000000000011111111111
0000000000000000000000000000000000000000000000000000000000000111111111111111
0000000000000000000000000000000000000000000000000000000001111111111111111111
0000000000000000000000000000000000000000000000000000011111111111111111111111
0000000000000000000000000000000000000000000000000111111111111111111111111111
00C0000000000000000000000000000000000000000001111111111111111111111111111111
0000000000000000000000000000000000000000011111111111111111111111111111111111
```



```
0000000000000000000000000000000000001111111111111111111111111111111111111111
0000000000000000000000000000000011111111111111111111111111111111111111111111
0000000000000000000000000000111111111111111111111111111111111111111111111111
0000000000000000000000001111111111111111111111111111111111111111111111111111
0000000000000000000011111111111111111111111111111111111111111111111111111111
0000000000000000111111111111111111111111111111111111111111111111111111111111
0000000000001111111111111111111111111111111111111111111111111111111111111111
0000000011111111111111111111111111111111111111111111111111111111111111111111
0000111111111111111111111111111111111111111111111111111111111111111111111111
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 9)))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m '(4 0) '(0 4) '(4 8) '(8 4) '(4 0))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 3 '(12 4) '(21 4) nil '(17 -100) '(17 100))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 5 '(30 3) '(34 5))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 4 '(45 3) '(49 5))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 3 '(55 2) '(59 6))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 2 '(65 2) '(69 6))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 2 '(70 -2) '(82 4) '(70 10))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (print-array m)
0000100000000000111000000000000000000000000000000000000000000000000000000011
0001010000000000111000000000000011100000000000000100001110000000100000000011
0010001000000000111000000000111111000000000001111000001111000011100000000000
0100000100011111111111000001111111110000000111111100001111100000011100000000
1000000010011111111111000001111111110000000111111100001111100000011100000000
0100000100011111111111000001111111110000000111111100001111100000011100000000
0010001000000000111000000000111111000000000001111000001111000011100000000000
0001010000000000111000000000000011100000000000000000000000001110000000000011
0000100000000000111000000000000000000000000000000000000000000000000000000011
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 9)))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 5 '(5 4) '(15 4))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 4 '(25 4) '(35 4))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 3 '(40 4) '(50 4))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (bitgraph-lines m 2 '(65 4) '(75 4))
(an-array has-element-type a-ubit has-sizes (76 9))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000111111111111000000000111111111100000000000000000000000000000000000000000
0001111111111111100000001111111111111001111111111111000000000000111111111110
0001111111111111100000011111111111111001111111111111000000000000111111111111
0001111111111111100000001111111111111001111111111111000000000000000000000000
0000111111111111000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(10 10)))
(an-array has-element-type a-ubit has-sizes (10 10))
-> (bitgraph-lines m has-origins '(-10.0 -20.0) '(12 22) '(18 28) nil
                 '(18 22) '(11.5 28.5))
(an-array has-element-type a-ubit has-sizes (10 10))
-> (print-array m)
0000000000
0000000000
0010000010
0001000100
0000101000
0000010000
0000101000
0001000100
0010000010
0000000000
nil
-> (setq pi-over-180 (quotient pi 180))
0.0174533
-> (setq circle
         (an-array has-sizes '(2 361)
                   by-expression
                      '(if (=& X 0) (sin (product pi-over-180 Y))
                           (cos (product pi-over-180 Y)))))
(an-array by-expression
          (if (=& X 0) (sin (product pi-over-180 Y))
              (cos (product pi-over-180 Y)))
      has-element-type a-long has-exponent -16 has-sizes (2 361))
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 21)))
(an-array has-element-type a-ubit has-sizes (76 21))
-> (bitgraph-lines m has-origins '(15 10) has-zooms '(9 9) '(0 0) circle nil
                 '(1.2 -1) '(1.2 1))
(an-array has-element-type a-ubit has-sizes (76 21))
(compute-time - 2.38333 seconds / gc-time - 1.51667 seconds for 1 gcs)
-> (bitgraph-lines m 3 has-origins '(45 10) has-zooms '(10 10) circle)
(an-array has-element-type a-ubit has-sizes (76 21))
-> (bitgraph-lines m 3 has-origins '(65 -4) has-zooms '(10 10) circle)
(an-array has-element-type a-ubit has-sizes (76 21))
-> (bitgraph-lines m 3 has-origins '(65 25) has-zooms '(10 10) circle)
(an-array has-element-type a-ubit has-sizes (76 21))
-> (print-array m)
0000000000000000000000000000000000000011100000011110001110000000000000001110
0000000000111111110000001000000000001111100000111110011110000000000000001110
0000000001100000001100001000000000001111100000111111011111000000000000011110
0000000011000000000110001000000000001110000000000011111011111000000001111100
0000000110000000000001101000000000001110000000000000011110011111111111111100
0000000100000000000001100100000000001110000000000000001111001111111111110000
0000001000000000000000100100000000011100000000000000001110000111111111100000
0000001000000000000000100100000000011100000000000000001110000000001110000000
0000001000000000000000100100000000011100000000000000001110000000000000000000
0000001000000000000000100100000000011100000000000000001111000000000000000000
0000001000000001000000110100000000011100000000000000001111000000000000000000
0000001000000001000000100100000000011100000000000000001111000000000000000000
0000001000000001000000100100000000011100000000000000001110000000000000000000
```



```
0000001000000001000000100100000000011100000000000000001110000000000000000000
0000001100000001000001100100000000011110000000000000001111000011111111100000
0000000110000001000001000100000000011110000000000000001110001111111111110000
0000000011000001000001000010000000001111100000000000011110011111111111111000
0000000001100001000110000010000000001111100000000011111001111110000001111100
0000000000111111111000000100000000011111100001111111001111000000000000111110
0000000000000001000000000010000000001111100001111100011100000000000000011110
0000000000000000000000000000000000000111000011100000110000000000000000001110
nil
-> (setq c6 (a-character-set has-file 'vfonts/times.r.6
                             has-font 'times-roman))
(a-character-set has-font times-roman has-file vfonts/times.r.6)
-> (setq c10(a-character-set has-file 'vfonts/times.r.10
                             has-font 'times-roman))
(a-character-set has-font times-roman has-file vfonts/times.r.10)
-> (read-character-set c6)
(a-character-set has-sizes (14 19) has-x-range (-1 16) has-y-range (-14 3)
                 has-width-range (2 17) has-width-estimate 14
                 has-font times-roman has-file vfonts/times.r.6
                 has-dispatch-array
                    (an-array has-element-type
                                 an-allocate-bitgraph-character
                              has-sizes (128))
                 has-bitgraph-array
                    (an-array has-element-type a-ubit has-sizes (32 4028)))
(compute-time - 0.916667 seconds / sweep-time - 1.06667 seconds for sweeping
             3384 bytes in 1 sweeps / compact-time - 0.0166667 seconds for
             moving 8040 bytes in 1 compactifications)
-> (read-character-set c10)
(a-character-set has-sizes (23 32) has-x-range (-1 27) has-y-range (-23 6)
                 has-width-range (4 28) has-width-estimate 23
                 has-font times-roman has-file vfonts/times.r.10
                 has-dispatch-array
                    (an-array has-element-type
                                 an-allocate-bitgraph-character
                              has-sizes (128))
                 has-bitgraph-array
                    (an-array has-element-type a-ubit has-sizes (32 6590)))
-> (get-character-display c6 'A)
(a-bitgraph-character has-offset 1546 has-origins (0 -12) has-sizes (14 12)
                      has-width 14)
-> (setq A (get-character-bitgraph c6 'A))
(an-array has-element-type a-ubit has-sizes (14 12) has-increments (1 32))
-> (setq Ar (get-character-bitgraph c6 'A 'left-rotate))
(an-array has-element-type a-ubit has-sizes (12 14) has-increments (1 32))
-> (print-array A)
00000010000000
00000011000000
00000111000000
00000111100000
00000101100000
00001001110000
00001000110000
00011111111000
00010000011000
00100000011100
```

```
00100000001100
11110000011110
nil
-> (print-array Ar)
10000000000
10000000000
11100000000
10011000000
00001110000
00001001110
00001000111
00001011111
00001111100
10111110000
11111000000
11100000000
10000000000
00000000000
nil
-> (get-character-display c10 'A)
(a-bitgraph-character has-offset 2524 has-origins (0 -20) has-sizes (23 20)
                      has-width 23)
-> (setq A (get-character-bitgraph c10 'A))
(an-array has-element-type a-ubit has-sizes (23 20) has-increments (1 32))
-> (setq Ar (get-character-bitgraph c10 'A 'left-rotate))
(an-array has-element-type a-ubit has-sizes (20 23) has-increments (1 32))
-> (print-array A)
00000000000100000000000
00000000000110000000000
00000000001110000000000
00000000001111000000000
00000000011111000000000
00000000010111000000000
00000000110111100000000
00000000100011100000000
00000001100011110000000
00000001100001110000000
00000011000001111000000
00000011000001111000000
00000111111111111100000
00000111111111111100000
00001100000000011110000
00001100000000011110000
00011000000000001111000
00011000000000001111000
00111000000000001111100
11111110000000111111111
nil
-> (print-array Ar)
10000000000000000000
10000000000000000000
11000000000000000000
11110000000000000000
11111100000000000000
10001111000000000000
10000011110000000000
```



```
00000011111100000000
00000011001111000000
00000011000001110000
00000011000000011100
00000011000001111111
00000011000111111110
00000011111111111000
10000011111111000000
10001111111100000000
11111111110000000000
11111111000000000000
11111100000000000000
11110000000000000000
11000000000000000000
10000000000000000000
10000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 25)))
(an-array has-element-type a-ubit has-sizes (76 25))
-> (bitgraph-text m '(40 0) 'over c10 'hello)
nil
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000110000000000000000000000000001100000110000000000000000000000000
0000000000001111100000000000000000000000111110011111000000000000000000000000
0000000000000111000000000000000000000000011100001110000000000000000000000000
0000000000000111000000000000000000000000011100001110000000000000000000000000
0000000000000111000000000000000000000000011100001110000000000000000000000000
0000000000000111000000000000000000000000011100001110000000000000000000000000
0000000000000111000000000000000000000000011100001110000000011000000000000000
0000000000000111011111110000000111111000011100001110000001111111100000000000
0000000000000111111111110000011000011100011100001110000110000011100000000000
0000000000000111100001110000011000001110011100001110000110000001110000000000
0000000000000111000001110000110000001110011100001110001110000000111000000000
0000000000000111000001110000111111111110011100001110001110000000111000000000
0000000000000111000001110000110000000000011100001110001110000000111000000000
0000000000000111000001110000110000000000011100001110001110000000111000000000
0000000000000111000001110000111000000000011100001110001110000000111000000000
0000000000000111000001110000111000000001011100001110001110000000111000000000
0000000000000111000001110000111100000011011100001110000111000000110000000000
0000000000000111000001110000011110001100011100001110000011100001110000000000
0000000000000111000001110000011111111100011100001110000011100011100000000000
0000000000001111111011111111000111111001111111111111100000111111000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (bitgraph-text m '(40 22) 'over 'left c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'left 'left-rotate c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'left 'top-rotate c6 'ho)
nil
```

Apr 27 15:42 1989 sbg/sbg_demo.ou Page 10

```
-> (bitgraph-text m '(40 22) 'over 'left 'right-rotate c6 'ho)
nil
-> (bitgraph-lines m '(40 22))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000001110000000000000000000000000000
0000000000000000000000000000000000000000000001111100000000000000000000000000
0000000000000000000000000000000000000000000011100011000000000000000000000000
0000000000000000000000000000000000000000000010000010000000000000000000000000
0000000000000000000000001111100111011100000010000010000000000000000000000000
0000000000000000000000011000110011000110000010000010000000000000000000000000
0000000000000000000000010000110011000110000011000111000000000000000000000000
0000000000000000000000110000011011000110000001111110000000000000000000000000
0000000000000000000000110000011011000110000000011000000000000000000000000000
0000000000000000000000111000010011000110000000000010000000000000000000000000
0000000000000000000000011000110011001110000001111110000000000000000000000000
0000000000000000000000001111100001110110000011111111000000000000000000000000
0000000000000000000000000000000000011000000010000010000000000000000000000000
0000000000000000000000000000000000011000000010000000000000000000000000000000
0000000000000000000000000000000000011000000010000010000000000000000000000000
0000000000000000000000000000000000011000011111111110000000000000000000000000
0000000000000000000000000000000000000000011111111110000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000010000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000011111111111000000000000000000000000000000000000000
0000000000000000000000000011111111111000110000000000000000000000000000000000
0000000000000000000000000010000010000000110000000000000000000000000000000000
0000000000000000000000000000000010000000110000000000000000000000000000000000
0000000000000000000000000010000010000000110000000000000000000000000000000000
0000000000000000000000000011111110000000110111000111110000000000000000000000
0000000000000000000000000011111110000000111001100110001100000000000000000000
0000000000000000000000000010000000000000110001100100001110000000000000000000
0000000000000000000000000000011000000000110001101100000110000000000000000000
0000000000000000000000000001111110000000110001101100000110000000000000000000
0000000000000000000000000011100110000000110001100110000100000000000000000000
0000000000000000000000000010000010000000110001100110001100000000000000000000
0000000000000000000000000010000010000000111011110011111100000000000000000000
0000000000000000000000000010000010000000000000000000000000000000000000000000
0000000000000000000000000011000111000000000000000000000000000000000000000000
0000000000000000000000000001111110000000000000000000000000000000000000000000
0000000000000000000000000000111000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (bitgraph-text m '(40 22) 'over 'left 'mirror c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'left 'left-mirror c6 'ho)
```

Apr 27 15:42 1989 sbg/sbg_demo.ou Page 11

```
nil
-> (bitgraph-text m '(40 22) 'over 'left 'top-mirror c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'left 'right-mirror c6 'ho)
nil
-> (bitgraph-lines m '(40 22))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000011111111111000000000000000000000000000000000000
0000000000000000000000000000011111111111000000000000000000000000000000000000
0000000000000000000000000000010000010000000000000000000000000000000000000000
0000000000000000000000000000000000010000000011111001111011100000000000000000
0000000000000000000000000000010000010000000110001100110001100000000000000000
0000000000000000000000000000011111110000000100001100110001100000000000000000
0000000000000000000000000000011111100000001100000110110001100000000000000000
0000000000000000000000000000010000000000001100000110110001100000000000000000
0000000000000000000000000000011000000000001110000100110001100000000000000000
0000000000000000000000000000001111100000001100011001100111000000000000000000
0000000000000000000000000000011100011000000011111000011101100000000000000000
0000000000000000000000000000010000010000000000000000000110000000000000000000
0000000000000000000000000000010000010000000000000000000110000000000000000000
0000000000000000000000000000010000010000000000000000000110000000000000000000
0000000000000000000000000000011000111000000000000000000110000000000000000000
0000000000000000000000000000001111100000000000000000000000000000000000000000
0000000000000000000000000000000011100000000000000000000000000000000000000000
0000000000000000000000000000000000000000010000000000000000000000000000000000
0000000000000000000000000000000000000000000000000111000000000000000000000000
0000000000000000000000000000000000000000000000001111110000000000000000000000
0000000000000000000000000011000000000000000000011100011000000000000000000000
0000000000000000000000000011000000000000000000010000001000000000000000000000
0000000000000000000000000011000000000000000000010000001000000000000000000000
0000000000000000000000000011000000000000000000010000001000000000000000000000
0000000000000000000000000011011100001111100000011000111000000000000000000000
0000000000000000000000000011100110011000110000011111110000000000000000000000
0000000000000000000000000011000110010000111000001100000000000000000000000000
0000000000000000000000000011000110110000011000000000001000000000000000000000
0000000000000000000000000011000110110000011000011111111000000000000000000000
0000000000000000000000000011000110011000010000011111111000000000000000000000
0000000000000000000000000011000110011000110000010000001000000000000000000000
0000000000000000000000000011101111001111100000010000000000000000000000000000
0000000000000000000000000000000000000000000000010000001000000000000000000000
0000000000000000000000000000000000000000000111111111110000000000000000000000
0000000000000000000000000000000000000000000111111111110000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
```

```
-> (bitgraph-text m '(40 22) 'over 'right c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'right 'left-rotate c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'right 'top-rotate c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'right 'right-rotate c6 'ho)
nil
-> (bitgraph-lines m '(40 22))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000011111111111100000000000000000000000000000000000000
0000000000000000000000000011111111111100000000000000000000000000000000000000
0000000000000000000000000010000010000000000000000000000000000000000000000000
0000000000000000000000000000000001000000000111110011101110000000000000000000
0000000000000000000000000010000001000000011000110011000110000000000000000000
0000000000000000000000000011111110000000010000110011000110000000000000000000
0000000000000000000000000011111110000000011000011011000110000000000000000000
0000000000000000000000000010000000000000011000011011000110000000000000000000
0000000000000000000000000000011000000000011100010011000110000000000000000000
0000000000000000000000000001111110000000011000110011001110000000000000000000
0000000000000000000000000011100011000000000111110001110110000000000000000000
0000000000000000000000000010000001000000000000000000000110000000000000000000
0000000000000000000000000010000001000000000000000000000110000000000000000000
0000000000000000000000000010000001000000000000000000000110000000000000000000
0000000000000000000000000011000111000000000000000000000110000000000000000000
0000000000000000000000000001111110000000000000000000000000000000000000000000
0000000000000000000000000000011100000000000000000000000000000000000000000000
0000000000000000000000000000000000000000100000000000000000000000000000000000
0000000000000000000000000000000000000000000000000111000000000000000000000000
0000000000000000000000000000000000000000000000001111110000000000000000000000
0000000000000000000000011000000000000000000000011100110000000000000000000000
0000000000000000000000011000000000000000000000010000010000000000000000000000
0000000000000000000000011000000000000000000000010000010000000000000000000000
0000000000000000000000011000000000000000000000010000010000000000000000000000
0000000000000000000000011011100001111100000000011001110000000000000000000000
0000000000000000000000011100110011000110000000001111110000000000000000000000
0000000000000000000000011000110010000111000000000011000000000000000000000000
0000000000000000000000011000110110000110000000000000100000000000000000000000
0000000000000000000000011000110110000011000000001111111000000000000000000000
0000000000000000000000011000110011000100000000011111110000000000000000000000
0000000000000000000000011000110011000110000000010000010000000000000000000000
0000000000000000000000011101111001111100000000010000000000000000000000000000
0000000000000000000000000000000000000000000000010000010000000000000000000000
0000000000000000000000000000000000000000000111111111110000000000000000000000
0000000000000000000000000000000000000000000011111111110000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```



```
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (bitgraph-text m '(40 20) c6 'hello '|  there  |)
nil
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000001100000000000000001100110000000000000000000000000000
0000000000000000000000001100000000000000001100110000000000000000000000000000
0000000000000000000000001100000000000000001100110000000000000000000000000000
0000000000000000000000001100000000000000001100110000000000000000000000000000
0000000000000000000000001101111000011110001100110001111100000000000000000000
0000000000000000000000001110011001100110001100110001100000000000000000000000
0000000000000000000000001100011001000110001100110010000111000000000000000000
0000000000000000000000001100011011111110011001101100000110000000000000000000
0000000000000000000000001100011011000000011001101100000110000000000000000000
0000000000000000000000001100011001100001011001100110000100000000000000000000
0000000000000000000000001100011001110010011001100110001100000000000000000000
0000000000000000000000001110111100111100011101110011111000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000110000000000000000000000000000000000000000000000
0000000000000000000000000000110000000000000000000000000000000000000000000000
0000000000000000000000001000110000000000000000000000000000000000000000000000
0000000000000000000000011000110000000000000000000000000000000000000000000000
0000000000000000000000111110110111000011110001101100111000000000000000000000
0000000000000000000000011000111001100110011101011001100000000000000000000000
0000000000000000000000011000110001100100011000100001000110000000000000000000
0000000000000000000000011000110001101111111100010001111111000000000000000000
0000000000000000000000011000110001101100000000100011000000000000000000000000
0000000000000000000000011000110001100110000100100001100001000000000000000000
0000000000000000000000011000110001100111001000100001110010000000000000000000
0000000000000000000000011101110111100111100011100001111100000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (bitgraph-text m '(-1 -1) 'over 'left c6 'hello '|  there  |)
nil
```

```
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0110000000000000001100110000000000000000000000000000000000000000000000000000
0110000000000000001100110000000000000000000000000000000000000000000000000000
0110000000000000001100110000000000000000000000000000000000000000000000000000
0110000000000000001100110000000000000000000000000000000000000000000000000000
0110111000011110001100110001111100000000000000000000000000000000000000000000
0111001100110011001100110011000110000000000000000000000000000000000000000000
0110001100100011001100110010000111000000000000000000000000000000000000000000
0110001101111111001100110110000011000000000000000000000000000000000000000000
0110001101100000001100110110000011000000000000000000000000000000000000000000
0110001100110000101100110011000010000000000000000000000000000000000000000000
0110001100111001001100110011000110000000000000000000000000000000000000000000
0111011110011110001110111001111100000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000001100000000000000000000000000000000000000000000000000
0000000000000000000000001100000000000000000000000000000000000000000000000000
0000000000000000000010001100000000000000000000000000000000000000000000000000
0000000000000000000110001100000000000000000000000000000000000000000000000000
0000000000000000001111101101110000111100011011001111000000000000000000000000
0000000000000000001100011100110011001100111010110011000000000000000000000000
0000000000000000001100011000110010001100010000100011000000000000000000000000
0000000000000000001100011000110111111110001000111111000000000000000000000000
0000000000000000001100011000110110000000010001100000000000000000000000000000
0000000000000000001100011000110011000010010000110000100000000000000000000000
0000000000000000001100011000110011100100010000111001000000000000000000000000
0000000000000000000011101110111100111100011100001111000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (bitgraph-text m '(76 47) 'under 'right c6 'hello '|  there  |)
nil
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```



```
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000110000000000000001100110000000000
0000000000000000000000000000000000000000000110000000000000001100110000000000
0000000000000000000000000000000000000000000110000000000000001100110000000000
0000000000000000000000000000000000000000000110000000000000001100110000000000
0000000000000000000000000000000000000000000110111000011110001100110001111100
0000000000000000000000000000000000000000000111001100110011001100110011000110
0000000000000000000000000000000000000000000110001100100011001100110010000111
0000000000000000000000000000000000000000000110001101111111001100110110000011
0000000000000000000000000000000000000000000110001101100000001100110110000011
0000000000000000000000000000000000000000000110001100110000101100110011000010
0000000000000000000000000000000000000000000110001100111001001100110011000110
0000000000000000000000000000000000000000000111011110011110001110111001111100
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000110000000000000000000000000000000000000000
0000000000000000000000000000000000110000000000000000000000000000000000000000
0000000000000000000000000000001000110000000000000000000000000000000000000000
0000000000000000000000000000011000110000000000000000000000000000000000000000
0000000000000000000000000000111110110111000011110001101100111100000000000000
0000000000000000000000000000110001110011001100110011101011001100000000000000
0000000000000000000000000000110001100011001000110001000010001100000000000000
0000000000000000000000000000110001100011011111111000100011111100000000000000
0000000000000000000000000000110001100011011000000001000110000000000000000000
0000000000000000000000000000110001100011001100001001000011000010000000000000
0000000000000000000000000000110001100011001110010001000011100100000000000000
0000000000000000000000000000001110111011110011110001110000111100000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (bitgraph-text m '(40 -1) 'over c6 'MIW)
nil
-> (bitgraph-text m '(-1 20) 'left c6 'O)
nil
-> (bitgraph-text m '(76 20) 'right c6 'O)
nil
-> (bitgraph-text m '(40 45) 'under c6 'gjpqy)
nil
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000001111000000001111111111111101111100111000000000000000000
```

```
0000000000000000000000110000000111000110001110001100000110000000000000000000
0000000000000000000000111000000111000110000110001110000100000000000000000000
0000000000000000000000111000001111000110000110000110000100000000000000000000
0000000000000000000000111100001011000110000111000111001100000000000000000000
0000000000000000000000101100011011000110000011001111001000000000000000000000
0000000000000000000000101110010011000110000011101011001000000000000000000000
0000000000000000000000100110110011000110000001101011111000000000000000000000
0000000000000000000000100111100011000110000001110001110000000000000000000000
0000000000000000000000100011100011000110000001110001110000000000000000000000
0000000000000000000000100011000011000110000000110001100000000000000000000000
0000111111000000000000111110010011111111111000001000001000000000000111111000
0001100001100000000000000000000000000000000000000000000000000000001100001100
0011000000110000000000000000000000000000000000000000000000000000011000000110
0111000000111000000000000000000000000000000000000000000000000001110000000111
0110000000011000000000000000000000000000000000000000000000000000110000000011
0110000000011000000000000000000000000000000000000000000000000000110000000011
0110000000011000000000000000000000000000000000000000000000000000110000000011
0110000000011000000000000000000000000000000000000000000000000000110000000011
0110000000111000000000000000000000000000000000000000000000000000110000000111
0011000000110000000000000000000000000000000000000000000000000000011000000110
0011100001110000000000000000000000000000000000000000000000000000011100001110
0000111111000000000000000000000000000000000000000000000000000000000111111000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000110000000000000000000000000000000000000000000
0000000000000000000000000000000110000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000111110011001111111100001111101111011100000000000000000
0000000000000000000000110011000110011100110011000110011000100000000000000000
0000000000000000000000110011000110011000111010000110011100100000000000000000
0000000000000000000000110011000110011000011010000110001100100000000000000000
0000000000000000000000110011000110011000011110000110001101000000000000000000
0000000000000000000000111000001100110000100110001100001110000000000000000000
0000000000000000000000110000000110011000110011000110000111000000000000000000
0000000000000000000000111111001100111111000111111100000100000000000000000000
0000000000000000000000100111100110011000000000001100000100000000000000000000
0000000000000000000000110000010011001100000000001100001000000000000000000000
0000000000000000000000110000101111001100000000001100111000000000000000000000
0000000000000000000000111110011100111000000000001100110000000000000000000000
nil
-> (setq m (an-array has-element-type a-ubit has-sizes '(76 45)))
(an-array has-element-type a-ubit has-sizes (76 45))
-> (bitgraph-text m '(40 22) 'over 'left c6 'hi)
nil
-> (bitgraph-text m '(40 22) 'over 'left 'left-rotate c6 'ho)
nil
-> (bitgraph-text m '(40 22) 'over 'left 'top-rotate c6 'hi)
nil
-> (bitgraph-text m '(40 22) 'over 'left 'right-rotate c6 'hi)
nil
-> (print-array m)
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```



```
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000001110111101110000000000000000000000000000000000000
0000000000000000000000000000110011000110000000000000001000000000000000000000
0000000000000000000000000000110011000110000110011111111000000000000000000000
0000000000000000000000000000110011000110000110011111111000000000000000000000
0000000000000000000000000000110011000110000000000000000000000000000000000000
0000000000000000000000000000110011000110000000000000001000000000000000000000
0000000000000000000000000000110011001110000000001111111000000000000000000000
0000000000000000000000000000110001101100000000011111111000000000000000000000
0000000000000000000000000000000000000110000000010000001000000000000000000000
0000000000000000000000000000000000000110000000010000000000000000000000000000
0000000000000000000000000000110000000110000000010000001000000000000000000000
0000000000000000000000000000110000000110000111111111111000000000000000000000
0000000000000000000000000000000000000000000111111111111000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000011111111111100000000000000000000000000000000000000
0000000000000000000000000011111111111100001100000001100000000000000000000000
0000000000000000000000000010000010000000001100000001100000000000000000000000
0000000000000000000000000000000010000000001100000000000000000000000000000000
0000000000000000000000000010000010000000001100000000000000000000000000000000
0000000000000000000000000011111110000000001101110001100000000000000000000000
0000000000000000000000000001111110000000001110011001100000000000000000000000
0000000000000000000000000010000000000000001100011001100000000000000000000000
0000000000000000000000000001100000000000001100011001100000000000000000000000
0000000000000000000000000001111100000000001100011001100000000000000000000000
0000000000000000000000000001110011000000001100011001100000000000000000000000
0000000000000000000000000010000010000000001100011001100000000000000000000000
0000000000000000000000000010000010000000001110111101110000000000000000000000
0000000000000000000000000010000010000000000000000000000000000000000000000000
0000000000000000000000000001100011100000000000000000000000000000000000000000
0000000000000000000000000000111110000000000000000000000000000000000000000000
0000000000000000000000000000011100000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
nil
-> (comment (set-mask m has-sizes '(76 14))
         (set-array i has-sizes '(17 2) by-expression 'X)
         (bitgraph-half-tone m i has-width 4 has-dither (cached-dither 4)
                  has-offset 0 has-factor 1 dont-rule)
         (print-array m) (set-mask m has-sizes '(76 26))
         (set-array ac has-sizes '(16) by-expression 'X has-scales
              '((0 1)))
         (bitgraph-bar-graph m ac dont-rule has-range-space 2
                  has-range-width 3 has-range '(0 5) has-range
                  '(10 15))
         (print-array m)
         (set-array i has-sizes '(10 3) by-value
```

```
                    '(  (-1 0 -1 -1 0 -1 0 0 1 0) (0 1 0 0 0 0 0 0 0 0)
                        (-1 0 -1 -1 0 -1 0 0 1 0)))
        (set-mask m has-sizes '(72 16)) (contour m i 0 1 0 0)
        (print-array i) (print-array m))
comment
->
Goodbye
```



```
-> (setq *default-array-file* '(|#cache.ar| end-of-file))
(|#cache.ar| end-of-file)
-> (exec rm -f |#cache.ar|)
0
-> (setq x (a-vector has-x 1 has-y 2))
(a-vector has-x 1.0 has-y 2.0)
-> (has-x x)
1.0
-> (has-y x)
2.0
-> (has-z x)
nil
-> (has-dimension x)
2
-> (has-length x)
2.23607
-> (uneval-object x)
(a-vector has-x 1.0 has-y 2.0)
-> (pretty-print x)
(a-vector has-x 1.0 has-y 2.0)0
-> (setq y (a-vector has-x 10 has-y 20))
(a-vector has-x 10.0 has-y 20.0)
-> (distance-between-points x y)
20.1246
-> (setq z (move-vector x 100 200))
(a-vector has-x 101.0 has-y 202.0)
-> (point-between-points x (a-vector has-x -1 has-y -2) 0.5)
(a-vector has-x 0.0 has-y 0.0)
-> (product-of-scalar-and-vector 10 x)
(a-vector has-x 10.0 has-y 20.0)
-> (scalar-product-of-vectors x x)
5.0
-> (scalar-product-of-vectors y y)
500.0
-> (scalar-product-of-vectors x y)
50.0
-> (sum-of-vectors x y)
(a-vector has-x 11.0 has-y 22.0)
-> (vector-product-of-vectors (a-vector has-x 1 has-y 1 has-z 0)
                              (a-vector has-x -1 has-y 1 has-z 0))
(a-vector has-x 0.0 has-y 0.0 has-z 2.0)
-> (setq ix (a-vector has-x 1 has-y 0 has-z 0 has-length 1))
(a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0)
-> (setq iy (a-vector has-x 0 has-y 1 has-z 0 has-length 1))
(a-vector has-x 0.0 has-y 1.0 has-z 0.0 has-length 1.0)
-> (setq iz (a-vector has-x 0 has-y 0 has-z 1 has-length 1))
(a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0)
-> (uneval-object ix)
(a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0)
-> (pretty-print ix)
(a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0)0
-> (scalar-product-of-vectors ix ix)
1.0
-> (scalar-product-of-vectors iy iy)
1.0
-> (scalar-product-of-vectors iz iz)
```

```
1.0
-> (scalar-product-of-vectors ix iy)
0.0
-> (scalar-product-of-vectors iy iz)
0.0
-> (scalar-product-of-vectors iz ix)
0.0
-> (vector-product-of-vectors ix ix)
(a-vector has-x 0.0 has-y 0.0 has-z 0.0)
-> (vector-product-of-vectors iy iy)
(a-vector has-x 0.0 has-y 0.0 has-z 0.0)
-> (vector-product-of-vectors iz iz)
(a-vector has-x 0.0 has-y 0.0 has-z 0.0)
-> (vector-product-of-vectors ix iy)
(a-vector has-x 0.0 has-y 0.0 has-z 1.0)
-> (vector-product-of-vectors iy iz)
(a-vector has-x 1.0 has-y 0.0 has-z 0.0)
-> (vector-product-of-vectors iz ix)
(a-vector has-x 0.0 has-y 1.0 has-z 0.0)
-> *1d-zero-vector*
(a-vector has-x 0.0)
-> *2d-zero-vector*
(a-vector has-x 0.0 has-y 0.0)
-> *3d-zero-vector*
(a-vector has-x 0.0 has-y 0.0 has-z 0.0)
-> *1d-x-unit-vector*
(a-vector has-x 1.0 has-length 1.0)
-> *2d-x-unit-vector*
(a-vector has-x 1.0 has-y 0.0 has-length 1.0)
-> *2d-y-unit-vector*
(a-vector has-x 0.0 has-y 1.0 has-length 1.0)
-> *3d-x-unit-vector*
(a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0)
-> *3d-y-unit-vector*
(a-vector has-x 0.0 has-y 1.0 has-z 0.0 has-length 1.0)
-> *3d-z-unit-vector*
(a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0)
-> (a-vector has-x 1 has-y 1 has-z 1 has-length 1)
(a-vector has-x 0.57735 has-y 0.57735 has-z 0.57735 has-length 1.0)
-> (setq x  (a-line has-start *3d-zero-vector*
                    has-end (a-vector has-x 1 has-y 1 has-z 1)))
(a-line has-length 1.73205
        has-start (a-vector has-x 0.0 has-y 0.0 has-z 0.0)
        has-direction
           (a-vector has-x 0.57735 has-y 0.57735 has-z 0.57735
                     has-length 1.0))
-> (is-infinite x)
nil
-> (has-length x)
1.73205
-> (has-transform x)
nil
-> (has-end x)
(a-vector has-x 1.0 has-y 1.0 has-z 1.0)
-> (setq tr1(a-transform has-xx 1 has-xy 0 has-yx 0 has-yy 1
                         is-orthogonal t))
```



```
(a-transform is-orthogonal t is-linear t has-xx 1.0 has-xy 0.0 has-yx 0.0
             has-yy 1.0)
-> (transform-point *2d-y-unit-vector* tr1)
(a-vector has-x 0.0 has-y 1.0)
-> (has-inverse tr1)
(a-transform is-orthogonal t is-linear t has-xx 1.0 has-xy 0.0 has-yx 0.0
             has-yy 1.0)
-> (has-determinant tr1)
1.0
-> (has-input-dimension tr1)
2
-> (has-output-dimension tr1)
2
-> (is-linear tr1)
t
-> (is-affine tr1)
nil
-> (is-orthogonal tr1)
t
-> (has-displacement tr1)
nil
-> (setq tr2 (a-transform has-axis *3d-x-unit-vector* has-angle pi))
(a-transform is-orthogonal t is-linear t has-angle 3.14159
             has-axis
                (a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0))
-> (transform-point *3d-x-unit-vector* tr2)
(a-vector has-x 1.0 has-y 0.0 has-z 0.0)
-> (transform-point *3d-y-unit-vector* tr2)
(a-vector has-x 0.0 has-y -1.0 has-z -3.21629e-16)
-> (transform-point *3d-z-unit-vector* tr2)
(a-vector has-x 0.0 has-y 3.21629e-16 has-z -1.0)
-> (compose-transforms tr2 tr2)
(a-transform is-orthogonal t is-linear t has-xx 1.0 has-xy 0.0 has-xz 0.0
             has-yx 0.0 has-yy 1.0 has-yz 6.43257e-16 has-zx 0.0
             has-zy -6.43257e-16 has-zz 1.0)
-> (setq tr3 (a-transform tr2 has-displacement *3d-z-unit-vector*))
(a-transform is-orthogonal t is-affine t has-angle 3.14159
             has-axis
                (a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0)
             has-displacement
                (a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0))
-> (transform-point *3d-x-unit-vector* tr3)
(a-vector has-x 1.0 has-y 0.0 has-z 1.0)
-> (transform-point *3d-y-unit-vector* tr3)
(a-vector has-x 0.0 has-y -1.0 has-z 1.0)
-> (transform-point *3d-z-unit-vector* tr3)
(a-vector has-x 0.0 has-y 3.21629e-16 has-z 0.0)
-> (compose-transforms tr2 tr3)
(a-transform is-orthogonal t has-xx 1.0 has-xy 0.0 has-xz 0.0 has-xt 0.0
             has-yx 0.0 has-yy 1.0 has-yz 6.43257e-16 has-yt 0.0 has-zx 0.0
             has-zy -6.43257e-16 has-zz 1.0 has-zt 0.0 has-tx 0.0 has-ty 0.0
             has-tz 1.0 has-tt 1.0)
-> (compose-transforms tr3 tr2)
(a-transform is-orthogonal t has-xx 1.0 has-xy 0.0 has-xz 0.0 has-xt 0.0
             has-yx 0.0 has-yy 1.0 has-yz 6.43257e-16 has-yt 0.0 has-zx 0.0
             has-zy -6.43257e-16 has-zz 1.0 has-zt 0.0 has-tx 0.0
```

```
            has-ty 3.21629e-16 has-tz -1.0 has-tt 1.0)
-> (setq tr4(a-transform has-axis *3d-z-unit-vector*
                         has-angle (quotient pi 2)
                         has-displacement *3d-z-unit-vector*))
(a-transform is-orthogonal t is-affine t has-angle 1.5708
             has-axis
                (a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0)
             has-displacement
                (a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0))
-> (setq ltr4 (linearize-transform tr4))
(a-transform is-orthogonal t is-linear t has-angle 1.5708
             has-axis
                (a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0))
-> (transpose-transform ltr4)
(a-transform is-orthogonal t is-linear t has-angle -1.5708
             has-axis
                (a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0))
-> (has-inverse tr4)
(a-transform is-orthogonal t is-affine t has-angle -1.5708
             has-axis
                (a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0)
             has-displacement (a-vector has-x 0.0 has-y 0.0 has-z -1.0))
-> (has-determinant tr4)
1.0
-> (transform-point *3d-x-unit-vector* tr4)
(a-vector has-x -1.60814e-16 has-y 1.0 has-z 1.0)
-> (transform-vector *3d-x-unit-vector* tr4)
(a-vector has-x -1.60814e-16 has-y 1.0 has-z 0.0)
-> (transform-covector tr4 *3d-x-unit-vector*)
(a-vector has-x -1.60814e-16 has-y -1.0 has-z 0.0)
-> (transform-point *3d-y-unit-vector* tr4)
(a-vector has-x -1.0 has-y -1.60814e-16 has-z 1.0)
-> (transform-vector *3d-y-unit-vector* tr4)
(a-vector has-x -1.0 has-y -1.60814e-16 has-z 0.0)
-> (transform-covector tr4 *3d-y-unit-vector*)
(a-vector has-x 1.0 has-y -1.60814e-16 has-z 0.0)
-> (transform-vector *3d-y-unit-vector* (transpose-transform ltr4))
(a-vector has-x 1.0 has-y -1.60814e-16 has-z 0.0)
-> (setq tr5 (a-transform has-input-dimension 2))
(a-transform is-linear t has-input-dimension 2 has-output-dimension 0)
-> (transform-point *2d-x-unit-vector* tr5)
(a-vector)
-> (setq tr6 (a-transform has-displacement *3d-z-unit-vector*))
(a-transform is-affine t has-input-dimension 0 has-output-dimension 3
             has-displacement
                (a-vector has-x 0.0 has-y 0.0 has-z 1.0 has-length 1.0))
-> (transform-point (a-vector) tr6)
(a-vector has-x 0.0 has-y 0.0 has-z 1.0)
-> (has-angle tr6)
nil
-> (has-axis tr6)
nil
-> (setq l1 (a-line has-start (a-vector has-x 1.0 has-y 3.0 has-z 2.0)
                    has-end (a-vector has-x -1.0 has-y -3.0 has-z -2.0)))
(a-line has-length 7.48331
        has-start (a-vector has-x 1.0 has-y 3.0 has-z 2.0)
```



```
        has-direction
           (a-vector has-x -0.267261 has-y -0.801784 has-z -0.534522
                     has-length 1.0))
-> (has-segment l1)
(a-vector has-x -2.0 has-y -6.0 has-z -4.0)
-> (has-end l1)
(a-vector has-x -1.0 has-y -3.0 has-z -2.0)
-> (is-infinite l1)
nil
-> (setq l1i (a-line l1 is-infinite t))
(a-line has-start
           (a-vector has-x 5.96046e-08 has-y 2.38419e-07 has-z 1.19209e-07)
        has-direction
           (a-vector has-x -0.267261 has-y -0.801784 has-z -0.534522
                     has-length 1.0))
-> (setq l2 (a-line has-start
                    (difference-of-vectors (has-start l1)
                                           *3d-x-unit-vector*)
                 has-direction *3d-x-unit-vector* has-length 5.0))
(a-line has-length 5.0 has-start (a-vector has-x 0.0 has-y 3.0 has-z 2.0)
        has-direction
           (a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0))
(compute-time - 0.0166667 seconds / gc-time - 1.5 seconds for 1 gcs)
-> (setq l2i (a-line l2 is-infinite t))
(a-line has-start (a-vector has-x 0.0 has-y 3.0 has-z 2.0)
        has-direction
           (a-vector has-x 1.0 has-y 0.0 has-z 0.0 has-length 1.0))
-> (angle-between-lines l1 l2)
1.84135
-> (distance-between-point-and-line *3d-zero-vector* l1)
2.73143e-07
-> (distance-between-point-and-line *3d-zero-vector* l2)
3.60555
-> (distance-between-lines l1 l2)
0.0
-> (setq x  (a-cluster has-point-list
                       (list (a-vector has-x 2.0 has-y 4.0)
                             (a-vector has-x 6.0 has-y 9.0)
                             (a-vector has-x 2.0 has-y 4.0))
                       is-chain t))
(a-cluster is-chain t)
-> (has-dimension x)
2
-> (has-count x)
3
-> (is-closed x)
t
-> (center-of-gravity-of-cluster x)
(a-vector has-x 3.33333 has-y 5.66667)
-> (setq y  (a-cluster has-point-array
                       (an-array has-sizes '(2 3)
                                 by-value
                                    '((2.0 4.0) (6.0 9.0) (2.0 4.0)))
                       is-chain t))
(a-cluster has-point-array
           (an-array has-element-type a-short has-exponent -11
```

```
                   has-sizes (2 3))
          is-chain t)
(compute-time - 0.0333333 seconds / gc-time - 1.61667 seconds for 1 gcs)
-> (has-dimension y)
2
-> (has-count y)
3
-> (is-closed y)
t
-> (center-of-gravity-of-cluster y)
(a-vector has-x 3.33333 has-y 5.66667)
-> (print-array (has-point-array y))
       2       4
       6       9
       2       4
nil
-> (equal (has-point-list x) (has-point-list y))
t
-> (uneval-object x)
(a-cluster has-point-array
             (an-array has-array-file '(|#cache.ar| 12)
                       has-element-type (a-type has-name 'a-short)
                       has-exponent -11 has-array-format 'motorola
                       has-sizes '(2 3))
          is-chain 't)
-> (print-array (has-point-array x))
       2       4
       6       9
       2       4
nil
-> (setq z (a-cluster x is-chain nil))
(a-cluster has-point-array
             (an-array has-array-file (|#cache.ar| 12)
                       has-element-type a-short has-exponent -11
                       has-array-format motorola has-sizes (2 3)))
-> (is-closed z)
nil
-> (setq w (a-cluster z is-chain t is-maximum-polygon t))
(a-cluster has-point-array
             (an-array has-array-file (|#cache.ar| 12)
                       has-element-type a-short has-exponent -11
                       has-array-format motorola has-sizes (2 3))
          is-chain t is-maximum-polygon t)
-> (is-closed w)
t
-> (setf (is-maximum-polygon x) t)
t
-> x
(a-cluster has-point-array
             (an-array has-array-file (|#cache.ar| 12)
                       has-element-type a-short has-exponent -11
                       has-array-format motorola has-sizes (2 3))
          is-chain t is-maximum-polygon t)
-> (setq w (a-cluster is-chain t))
(a-cluster is-chain t)
-> (has-point-list w)
```



```
nil
-> (has-point-array w)
nil
-> (has-dimension w)
nil
-> (has-count w)
0
->
Goodbye
```

```
-> (if (not *current-display-window*) (load 'sketch.rc))
nil
-> (new-data-file '|#sdi_demo.ar|)
|#sdi_demo.ar|
-> (setq *default-array-file* '(|#sdi_demo.ar| end-of-file))
(|#sdi_demo.ar| end-of-file)
-> (setq cat(a-catalog is-index-of (new-catalog 'sdi_demo.ca)
                     has-index-function 'catalog-number))
(a-catalog is-index-of (a-catalog has-file sdi_demo.ca)
          has-index-function catalog-number)
-> (clear-display)
nil
(compute-time - 1.81667 seconds / gc-time - 4.66667 seconds for 3 gcs /
              sweep-time - 2.15 seconds for sweeping 0 bytes in 2 sweeps /
              compact-time - 0.0 seconds for moving 0 bytes in 2
              compactifications)
-> (setq x (an-array has-sizes '(256 1) by-expression 'X))
(an-array by-expression X has-element-type a-long has-exponent -16
          has-sizes (256))
-> (display-image x '(0 0) has-zooms '(2 30) has-bounds '(-0.5 255.5))
nil
(compute-time - 1.01667 seconds)
-> (display-image x '(0 40) has-zooms '(2 30) has-bounds '(-0.5 255.5)
                  do-pseudocolor)
nil
-> (display-image (slice-of-array x X-dimension 32) '(0 80) has-zooms
                  '(16 30) has-bounds '(-0.5 255.5))
nil
-> (display-image (slice-of-array x X-dimension 32) '(0 120) has-zooms
                  '(16 30) has-bounds '(-0.5 255.5) do-pseudocolor)
nil
-> (display-image (slice-of-array x X-dimension 32 224) '(0 160) has-zooms
                  '(16 30) has-bounds '(-0.5 255.5))
nil
-> (display-image (slice-of-array x X-dimension 32 224) '(0 200) has-zooms
                  '(16 30) has-bounds '(-0.5 255.5) do-pseudocolor)
nil
-> (add-to-array-elements x -128)
(an-array has-element-type a-long has-exponent -16 has-sizes (256))
-> (display-image x '(0 240) has-zooms '(2 30) has-bounds
                  '(-128.5 negative -0.5 positive 127.5))
nil
-> (display-image (slice-of-array x X-dimension 128 128) '(0 280) has-zooms
                  '(4 30) has-bounds
                  '(-0.5 magenta 15.5 red 31.5 brown 47.5 yellow 63.5 green
                        79.5 cyan 95.5 turquoise 111.5 blue 127.5))
nil
-> (display-patom "this is sample text" '(0 320) 1 'left 'white)
"this is sample text"
-> (display-patom "this is sample text" '(0 332) 1.2 'left 'turquoise)
"this is sample text"
(compute-time - 0.7 seconds / sweep-time - 1.18333 seconds for sweeping 552
              bytes in 1 sweeps / compact-time - 0.0166667 seconds for
              moving 12120 bytes in 1 compactifications)
-> (display-patom "this is sample text" '(0 345) 1.5 'left 'red)
"this is sample text"
```



```
-> (display-patom "this is sample text" '(0 365) 2.0 'left 'green)
"this is sample text"
-> (display-patom "this is sample text" '(0 390) 2.7 'left 'yellow)
"this is sample text"
-> (display-patom "this is sample text" '(0 420) 3.2 'left 'magenta)
"this is sample text"
-> (display-patom "this is sample text" '(0 450) 4.2 'left 'cyan)
"this is sample text"
-> (display-pretty-print 10 '(a (b c) (d e) (f g h j)) '(384 320) 2 'left
                         'over 'brown)
3
-> (write-display cat)
nil
(compute-time - 1.9 seconds)
-> (clear-display)
nil
-> (setq m (quotient pi 180.0))
0.0174533
-> (setq c  (an-array has-sizes '(2 361)
                      by-expression
                        '(if (equal X 0) (sin (product m Y))
                             (cos (product m Y)))))
(an-array by-expression
          (if (equal X 0) (sin (product m Y)) (cos (product m Y)))
          has-element-type a-long has-exponent -16 has-sizes (2 361))
-> (multiply-array-elements-by c 50)
(an-array has-element-type a-long has-exponent -16 has-sizes (2 361))
(compute-time - 2.35 seconds / gc-time - 1.7 seconds for 1 gcs)
-> (move-display-cursor-to '(100 100))
(100 100)
-> (display-lines 'white c)
nil
-> (setq m  (an-array has-sizes '(16 32) by-expression 1
                      has-element-type a-ubit))
(an-array by-expression 1 has-element-type a-ubit has-sizes (16 32))
-> (display-bitgraph m 'white '(0 256))
nil
-> (display-bitgraph m 'red '(16 256))
nil
-> (display-bitgraph m 'green '(32 256))
nil
-> (display-bitgraph m 'turquoise '(48 256))
nil
-> (setq m  (an-array has-sizes '(64 8) by-expression 1
                      has-element-type a-ubit))
(an-array by-expression 1 has-element-type a-ubit has-sizes (64 8))
-> (display-bitgraph m 'white '(0 256))
nil
-> (display-bitgraph m 'red '(0 264))
nil
-> (display-bitgraph m 'green '(0 272))
nil
-> (display-bitgraph m 'turquoise '(0 280))
nil
-> (setq m (an-array has-sizes '(256 32) has-element-type a-ubit))
(an-array has-element-type a-ubit has-sizes (256 32))
```

```
-> (bitgraph-ruler m '(256 (-0.5 nil) 1.0) 0 255 16 do-reverse t)
(an-array has-element-type a-ubit has-sizes (256 32))
-> (display-bitgraph m 'turquoise '(128 256))
nil
-> (setq m (an-array has-sizes '(32 32) has-element-type a-ubit))
(an-array has-element-type a-ubit has-sizes (32 32))
-> (bitgraph-line m 0 0 31 31 nil 'reverse)
(an-array has-element-type a-ubit has-sizes (32 32))
-> (bitgraph-line m 0 31 31 0 nil 'reverse)
(an-array has-element-type a-ubit has-sizes (32 32))
-> (display-bitgraph m 'green '(390 256))
nil
-> (setq m (an-array has-sizes '(16 16) has-element-type a-ubit))
(an-array has-element-type a-ubit has-sizes (16 16))
-> (bitgraph-line m 0 8 15 8 2 'reverse)
(an-array has-element-type a-ubit has-sizes (16 16))
-> (bitgraph-line m 8 0 8 15 2 'reverse)
(an-array has-element-type a-ubit has-sizes (16 16))
-> (display-bitgraph m 'red '(92 92))
nil
-> (write-display cat)
nil
(compute-time - 1.73333 seconds)
-> (close-catalog cat)
nil
->
Goodbye
```



```
-> (find-display-map 'standard-bitgraph)
(a-display-map has-ids (standard-bitgraph)
               has-plane-types
                  (  (white (a-plane-type has-color white has-line-width 1.0
                                          has-character-sizes (6 12)
                                          has-character-font display))
                     (magenta (a-plane-type has-color magenta
                                            has-line-width 1.0
                                            has-character-sizes (6 12)
                                            has-character-font display))
                     (red (a-plane-type has-color red has-line-width 1.0
                                        has-character-sizes (6 12)
                                        has-character-font display))
                     (brown (a-plane-type has-color brown has-line-width 1.0
                                          has-character-sizes (6 12)
                                          has-character-font display))
                     (yellow (a-plane-type has-color yellow
                                           has-line-width 1.0
                                           has-character-sizes (6 12)
                                           has-character-font display))
                     (green (a-plane-type has-color green has-line-width 1.0
                                          has-character-sizes (6 12)
                                          has-character-font display))
                     (cyan (a-plane-type has-color cyan has-line-width 1.0
                                         has-character-sizes (6 12)
                                         has-character-font display))
                     (turquoise (a-plane-type has-color turquoise
                                              has-line-width 1.0
                                              has-character-sizes (6 12)
                                              has-character-font display))
                     (blue (a-plane-type has-color blue has-line-width 1.0
                                         has-character-sizes (6 12)
                                         has-character-font display))))
-> (setq d1 (a-display has-sizes '(10 10) has-map 'standard
                       has-bitgraph-planes '(red green blue)
                       has-bitgraph-programs t))
(a-display has-sizes (10 10) has-map standard
           has-intensity-array
              (an-array has-element-type a-uchar has-sizes (10 10))
           has-bitgraph-planes (red green blue)
           has-bitgraph-array
              (an-array has-element-type a-ubit has-sizes (10 10 8)
                        has-increments (8 80 1))
           has-bitgraph-programs [nil nil nil])
(compute-time - 0.05 seconds / gc-time - 1.45 seconds for 1 gcs)
-> (has-range d1)
256
-> (has-colors d1)
(  (black 0) (white 127) (magenta 143) (red 159) (brown 175) (yellow 191)
   (green 207) (cyan 223) (turquoise 239) (blue 255))
-> (has-primary-colors d1)
nil
-> (setq w1 (a-display-window has-parent d1))
(a-display-window has-sizes (10.0 10.0) has-zooms (1.0 1.0)
                  has-upper-left (0.0 0.0) has-cursor (0.0 0.0)
                  has-line-plane 0 has-area-plane 0 has-text-plane 0
```

```
                 has-parent
                    (a-display has-sizes (10 10) has-map standard
                               has-intensity-array
                                  (an-array has-element-type a-uchar
                                            has-sizes (10 10))
                               has-bitgraph-planes (red green blue)
                               has-bitgraph-array
                                  (an-array has-element-type a-ubit
                                            has-sizes (10 10 8)
                                            has-increments (8 80 1))
                               has-bitgraph-programs [nil nil nil]))
(compute-time - 0.05 seconds / gc-time - 1.56667 seconds for 1 gcs)
-> (a-display-window has-parent d1 has-sizes '(4 5) has-origins '(7 3)
                    has-orientation 'left-rotate)
(a-display-window has-sizes (4.0 5.0) has-zooms (1.0 1.0)
                 has-upper-left (3.0 3.0) has-cursor (0.0 0.0)
                 has-line-plane 0 has-area-plane 0 has-text-plane 0
                 has-parent
                    (a-display has-sizes (10 10) has-map standard
                               has-intensity-array
                                  (an-array has-element-type a-uchar
                                            has-sizes (10 10))
                               has-bitgraph-planes (red green blue)
                               has-bitgraph-array
                                  (an-array has-element-type a-ubit
                                            has-sizes (10 10 8)
                                            has-increments (8 80 1))
                               has-bitgraph-programs [nil nil nil])
                 has-orientation left-rotate)
-> (a-display-window has-parent w1 has-sizes '(4 5)
                    has-origins (a-vector has-x 7 has-y 3)
                    has-orientation 'left-rotate)
(a-display-window has-sizes (4.0 5.0) has-zooms (1.0 1.0)
                 has-upper-left (3.0 3.0) has-cursor (0.0 0.0)
                 has-line-plane 0 has-area-plane 0 has-text-plane 0
                 has-parent
                    (a-display has-sizes (10 10) has-map standard
                               has-intensity-array
                                  (an-array has-element-type a-uchar
                                            has-sizes (10 10))
                               has-bitgraph-planes (red green blue)
                               has-bitgraph-array
                                  (an-array has-element-type a-ubit
                                            has-sizes (10 10 8)
                                            has-increments (8 80 1))
                               has-bitgraph-programs [nil nil nil])
                 has-orientation left-rotate)
-> (setq w2 (a-display-window w1 has-transform
                               (a-transform has-xx 1 has-xy 0 has-yx 0
                                            has-yy 1 has-tx 7 has-ty 3)))
(a-display-window has-sizes (10.0 10.0) has-zooms (1.0 1.0)
                 has-upper-left (0.0 0.0) has-cursor (0.0 0.0)
                 has-line-plane 0 has-area-plane 0 has-text-plane 0
                 has-parent
                    (a-display has-sizes (10 10) has-map standard
                               has-intensity-array
```



```
                                  (an-array has-element-type a-uchar
                                            has-sizes (10 10))
                               has-bitgraph-planes (red green blue)
                               has-bitgraph-array
                                  (an-array has-element-type a-ubit
                                            has-sizes (10 10 8)
                                            has-increments (8 80 1))
                               has-bitgraph-programs [nil nil nil])
                 has-transform
                    (a-transform is-affine t
                                 has-displacement
                                    (a-vector has-x 7.0 has-y 3.0)
                                 has-xx 1.0 has-xy 0.0 has-yx 0.0
                                 has-yy 1.0))
-> (setq w3 (a-display-window has-parent w2 has-sizes '(4 5)
                              has-origins *2d-zero-vector*
                              has-orientation 'left-rotate))
(a-display-window has-sizes (4.0 5.0) has-zooms (1.0 1.0)
                 has-upper-left (3.0 3.0) has-cursor (0.0 0.0)
                 has-line-plane 0 has-area-plane 0 has-text-plane 0
                 has-parent
                    (a-display has-sizes (10 10) has-map standard
                               has-intensity-array
                                  (an-array has-element-type a-uchar
                                            has-sizes (10 10))
                               has-bitgraph-planes (red green blue)
                               has-bitgraph-array
                                  (an-array has-element-type a-ubit
                                            has-sizes (10 10 8)
                                            has-increments (8 80 1))
                               has-bitgraph-programs [nil nil nil])
                 has-orientation left-rotate)
-> (display-image (an-array has-sizes '(7 7) by-expression '(+ X (* 10 Y)))
                 w3)
nil
-> (print-array (has-intensity-array d1))
    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   77   58   39   20    1    0    0
    0    0    0   79   60   41   22    2    0    0
    0    0    0   81   62   43   24    4    0    0
    0    0    0   83   64   45   26    6    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
nil
-> (setf (has-cursor w1) (a-vector has-x -10 has-y -20))
(-10.0 -20.0)
-> (has-cursor w1)
(-10.0 -20.0)
-> (setf (has-plane w1) 'green)
1
-> (display-lines w1 '(12 22) '(18 28) nil '(18 22) '(12 28))
nil
-> (has-bitgraph-programs d1)
```

```
(nil (  (w (10.0 10.0) (0.0 0.0) (1.0 1.0) nil
         (1 1.0 (-10.0 -20.0) (12 22) (18 28) nil (18 22) (12 28))))
    nil)
-> (print-array (slice-of-array (has-bitgraph-array d1) '(nil nil 3)))
(* * 0):
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
(* * 1):
0000000000
0000000000
0010000010
0001000100
0000101000
0000010000
0000101000
0001000100
0010000010
0000000000
(* * 2):
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
nil
-> (setq *current-display-window*
          (a-display-window has-parent
                            (a-display has-sizes '(76 40)
                                       has-map 'standard
                                       has-bitgraph-planes
                                          '(red green blue)
                                       has-bitgraph-array t
                                       has-bitgraph-programs t)))
(a-display-window has-sizes (76.0 40.0) has-zooms (1.0 1.0)
                  has-upper-left (0.0 0.0) has-cursor (0.0 0.0)
                  has-line-plane 0 has-area-plane 0 has-text-plane 0
                  has-parent
                     (a-display has-sizes (76 40) has-map standard
                                has-intensity-array
                                   (an-array has-element-type a-uchar
                                             has-sizes (76 40))
                                has-bitgraph-planes (red green blue)
                                has-bitgraph-array
```



```
                                   (an-array has-element-type a-ubit
                                             has-sizes (76 40 8)
                                             has-increments (8 608 1))
                                has-bitgraph-programs (nil nil nil)))
-> (has-plane *current-display-window*)
0
-> (move-display-cursor-to '(40 30))
(40 30)
-> (display-text (10 1.0) (print 'x))
nil
(compute-time - 1.33333 seconds / sweep-time - 1.06667 seconds for sweeping
             240 bytes in 1 sweeps / compact-time - 0.0166667 seconds for
             moving 7704 bytes in 1 compactifications)
-> (print-array
      (slice-of-array (has-bitgraph-array *current-display-window*)
                      '(nil nil 1) '(nil nil 0)))
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000010001000000000000000000000000000000000
0000000000000000000000000000000000000001010000000000000000000000000000000000
0000000000000000000000000000000000000000100000000000000000000000000000000000
0000000000000000000000000000000000000001010000000000000000000000000000000000
0000000000000000000000000000000000000010001000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
nil
->
Goodbye
```



```
-> (setq i1 (an-array has-sizes '(5 5) by-expression '(plus X Y)))
(an-array by-expression (plus X Y) has-element-type a-long has-exponent -16
        has-sizes (5 5))
-> (setq i2 (an-array has-sizes '(5 5) by-expression '(diff X Y)))
(an-array by-expression (diff X Y) has-element-type a-long has-exponent -16
        has-sizes (5 5))
-> (print-array i1)
      0      1      2      3      4
      1      2      3      4      5
      2      3      4      5      6
      3      4      5      6      7
      4      5      6      7      8
nil
-> (print-array i2)
      0      1      2      3      4
     -1      0      1      2      3
     -2     -1      0      1      2
     -3     -2     -1      0      1
     -4     -3     -2     -1      0
nil
-> (setq h1 (histogram-of i1 '(nil nil 1)))
(an-array has-element-type a-long has-exponent 0 has-sizes (9)
        has-offsets (-8e-05) has-scales (1.0))
(compute-time - 0.0333333 seconds / gc-time - 1.48333 seconds for 1 gcs)
-> h1
(an-array has-element-type a-long has-exponent 0 has-sizes (9)
        has-offsets (-8e-05) has-scales (1.0))
-> (print-array h1)
      1      2      3      4      5      4      3      2      1
nil
-> (setq h2 (histogram-of i1 '(nil nil 1) i2 '(nil nil 1)))
(an-array has-element-type a-long has-exponent 0 has-sizes (9 9)
        has-offsets (-8e-05 -4.00008) has-scales (1.0 1.0))
-> h2
(an-array has-element-type a-long has-exponent 0 has-sizes (9 9)
        has-offsets (-8e-05 -4.00008) has-scales (1.0 1.0))
-> (print-array h2)
      0      0      0      0      1      0      0      0      0
      0      0      0      1      0      1      0      0      0
      0      0      1      0      1      0      1      0      0
      0      1      0      1      0      1      0      1      0
      1      0      1      0      1      0      1      0      1
      0      1      0      1      0      1      0      1      0
      0      0      1      0      1      0      1      0      0
      0      0      0      1      0      1      0      0      0
      0      0      0      0      1      0      0      0      0
nil
-> (auto-clip i1 4 2)
(1.27492 6.52492)
->
Goodbye
```

```
-> (setq c (an-array has-sizes '(7 7) by-expression '(diff X 3)))
(an-array by-expression (diff X 3) has-element-type a-long has-exponent -16
          has-sizes (7 7))
-> (desetq (s e) (zero-edges-of c '(2 2)))
(  (an-array has-element-type a-long has-exponent -16 has-sizes (6 9)
             has-desired-sizes (6 6) has-increments (1 9 81 81 81 81))
   (an-array has-element-type a-char has-sizes (6 6 2)))
(compute-time - 0.3 seconds / gc-time - 2.96667 seconds for 2 gcs)
-> (print-array s)
       0       0       0       3       0       0
       0       0       0       3       0       0
       0       0       0       3       0       0
       0       0       0       3       0       0
       0       0       0       3       0       0
       0       0       0       3       0       0
nil
-> (print-array e)
(* * 0):
 nil nil nil  24 nil nil
 nil nil nil  24 nil nil
 nil nil nil  24 nil nil
 nil nil nil  24 nil nil
 nil nil nil  24 nil nil
 nil nil nil  24 nil nil
(* * 1):
 nil nil nil   0 nil nil
 nil nil nil   0 nil nil
 nil nil nil   0 nil nil
 nil nil nil   0 nil nil
 nil nil nil   0 nil nil
 nil nil nil   0 nil nil
nil
->
Goodbye
```



```
-> (setq x  (an-array has-sizes '(9 9)
                      by-expression
                         '(plus X Y
                                (product 0.1
                                         (sin (product 10 (plus X Y)))))))
(an-array by-expression
             (plus X Y (product 0.1 (sin (product 10 (plus X Y)))))
          has-element-type a-long has-exponent -16 has-sizes (9 9))
-> (setf (has-element x '(2 4)) nil)
nil
-> (setf (has-element x '(3 1)) nil)
nil
-> (setf (has-element x '(4 6)) nil)
nil
-> (setf (has-element x '(8 1)) nil)
nil
-> (setf (has-element x '(4 7)) nil)
nil
-> (setf (has-element x '(6 4)) nil)
nil
-> (print-array x)
        0  0.9456  2.0913  2.9012  4.0745  4.9738  5.9695  7.0774  7.9006
   0.9456  2.0913  2.9012     nil  4.9738  5.9695  7.0774  7.9006     nil
   2.0913  2.9012  4.0745  4.9738  5.9695  7.0774  7.9006  9.0894  9.9494
   2.9012  4.0745  4.9738  5.9695  7.0774  7.9006  9.0894  9.9494 10.9956
   4.0745  4.9738     nil  7.0774  7.9006  9.0894     nil 10.9956 12.0581
   4.9738  5.9695  7.0774  7.9006  9.0894  9.9494 10.9956 12.0581  12.907
   5.9695  7.0774  7.9006  9.0894     nil 10.9956 12.0581  12.907  14.098
   7.0774  7.9006  9.0894  9.9494     nil 12.0581  12.907  14.098 14.9285
   7.9006  9.0894  9.9494 10.9956 12.0581  12.907  14.098 14.9285 16.0219
nil
-> (let (*array-expander*) (setq y (box-linear-fit-of x '(3 5) '(2 4))))
(an-array has-element-type a-long has-exponent -16 has-sizes (4 2 4))
-> (print-array y)
(* * 0):
   3.0034  4.9971  7.0035  8.9974
    7.003  9.0068 11.0087  13.005
(* * 1):
   1.0039  0.9956   1.005  0.9784
   1.0042   1.006   0.999  0.9939
(* * 2):
   1.0019  0.9989  1.0003  1.0132
   0.9929  1.0013  1.0028  0.9977
(* * 3):
   0.0844  0.0805  0.0816  0.0779
   0.0819  0.0904  0.0847  0.0828
nil
->
Goodbye
```

```
-> (setq x (an-array has-sizes '(9 9) by-expression '(plus X Y)))
(an-array by-expression (plus X Y) has-element-type a-long has-exponent -16
          has-sizes (9 9))
-> (print-array x)
         0       1       2       3       4       5       6       7       8
         1       2       3       4       5       6       7       8       9
         2       3       4       5       6       7       8       9      10
         3       4       5       6       7       8       9      10      11
         4       5       6       7       8       9      10      11      12
         5       6       7       8       9      10      11      12      13
         6       7       8       9      10      11      12      13      14
         7       8       9      10      11      12      13      14      15
         8       9      10      11      12      13      14      15      16
nil
-> (let (*array-expander*)
        (setq y (box-standard-deviation-of x '(3 5) '(2 4))))
(an-array has-element-type a-long has-exponent -16 has-sizes (4 2 2))
-> (print-array y)
(* * 0):
         3       5       7       9
         7       9      11      13
(* * 1):
   1.6903  1.6903  1.6903  1.6903
   1.6903  1.6903  1.6903  1.6903
nil
-> (setq x  (an-array has-sizes '(9 9)
                      by-expression
                        '(product 100 (sin (+ (* 10 X) (* 100 Y))))))
(an-array by-expression (product 100 (sin (+ (* 10 X) (* 100 Y))))
          has-element-type a-long has-exponent -16 has-sizes (9 9))
-> (setf (has-element x '(2 4)) nil)
nil
(compute-time - 0.233333 seconds / gc-time - 1.41667 seconds for 1 gcs)
-> (setf (has-element x '(3 1)) nil)
nil
-> (setf (has-element x '(4 6)) nil)
nil
-> (setf (has-element x '(8 1)) nil)
nil
-> (setf (has-element x '(4 7)) nil)
nil
-> (setf (has-element x '(6 4)) nil)
nil
-> (print-array x)
         0-54.4021 91.2945-98.8032 74.5113-26.2375-30.4811 77.3891-99.3889
-50.6366 -4.4243 58.0611     nil  98.024-71.4876 21.9425 34.6649     nil
-87.3297 46.7719  8.8399-61.6064 94.5445-97.0528  68.324-17.6046-38.7809
-99.9756 85.0888-42.8155-13.2382 65.0311-95.8933 95.8916-65.0265 13.2322
-85.0919 99.9754     nil 38.7754 17.6105-68.3284     nil-94.5426 61.6017
-46.7772 87.3327-99.7795 80.1116-34.6593-21.9484 71.4919-98.0252 93.0084
  4.4182 50.6418-89.4024 99.3882     nil 30.4753 26.2433-74.5153 98.8041
  54.397   0.006-54.4072  91.297     nil 74.5073-26.2317-30.4868 77.3929
  89.397-50.6314 -4.4303  58.066-93.0128 98.0228-71.4834 21.9366 34.6706
nil
-> (let (*array-expander*)
        (setq y (box-standard-deviation-of x '(3 5) '(2 4))))
```



```
(an-array has-element-type a-long has-exponent -16 has-sizes (4 2 2))
-> (print-array y)
(* * 0):
 -2.4746 25.4022  10.457  2.0939
  -3.168  0.7964  0.2239  6.4189
(* * 1):
 69.8127 63.9682 74.1433 65.3429
 70.1303 74.2171  62.522 71.2151
nil
-> (setq x (an-array has-sizes '(3 3) by-expression '(plus X Y)))
(an-array by-expression (plus X Y) has-element-type a-long has-exponent -16
          has-sizes (3 3))
-> (print-array x)
       0       1       2
       1       2       3
       2       3       4
nil
-> (setq y (box-standard-deviation-of x '(3 3) '(1 1)))
(an-array has-element-type a-long has-exponent -16 has-sizes (3 3 2))
-> (print-array y)
(* * 0):
  0.6667  1.3333       2
  1.3333       2  2.6667
       2  2.6667  3.3333
(* * 1):
  0.7071       1  0.7071
       1  1.2247       1
  0.7071       1  0.7071
nil
-> (setq y (box-horizontal-total-variation-of x '(3 3) '(1 1)))
(an-array has-element-type a-long has-exponent -16 has-sizes (3 3))
-> (print-array y)
     0.5       1     0.5
     0.5       1     0.5
     0.5       1     0.5
nil
-> (setq y (box-vertical-total-variation-of x '(3 3) '(1 1)))
(an-array has-element-type a-long has-exponent -16 has-sizes (3 3))
-> (print-array y)
     0.5     0.5     0.5
       1       1       1
     0.5     0.5     0.5
nil
-> (setq y (box-minimum-total-variation-of x '(3 3) '(1 1)))
(an-array has-element-type a-long has-exponent -16 has-sizes (3 3))
-> (print-array y)
     0.5     0.5     0.5
     0.5       1     0.5
     0.5     0.5     0.5
nil
->
Goodbye
```

SOURCE FILES OF
A REPRESENTATIVE SKETCH PACKAGE

BASIC ARITHMETIC PACKAGE: SBA

VERSION 4B

April 1989

```
     18      39     535 makefile.mk
     48     228    1439 sba_cdither.c
    593    2760   19842 sba_celem.c
    359    1701   10868 sba_cfilter.c
      0       0       0 sba_chap.ma
     49     247    1838 sba_ckernel.c
    361    1586   11176 sba_cmiss.c
      4       8      95 sba_compile.1
    243    1147    7928 sba_cprod.c
      9      13     113 sba_defs.h
     81     388    3081 sba_demo.1
     58     301    1889 sba_dither.1
    436    1981   15475 sba_elem.1
    429    2366   15930 sba_filter.1
    253    1620    9223 sba_kernel.1
     11      19     191 sba_load.1
    395    2034   14850 sba_miss.1
     77     411    3233 sba_prod.1
   3424   16849  117706 total

lrwxrwxrwx  1 walton        18 Apr 27 05:30 make -> ../package_make.sh
```

```
PREFIX=sba
CHAPTER=8
TITLE=BASIC ARITHMETIC
MKFILES=makefile.mk
MAFILES=sba_chap.ma
LOAD_LFILES=sba_load.l sba_elem.l sba_miss.l sba_prod.l \
        sba_dither.l sba_filter.l sba_kernel.l
COMPILE_LFILES=sba_compile.l
DEMO_LFILES=sba_demo.l
DEMO_OUFILES=sba_demo.ou
HFILES=sba_defs.h
CFILES=sba_celem.c sba_cmiss.c sba_cprod.c sba_ckernel.c \
        sba_cdither.c sba_cfilter.c
OFILES=sba_celem.o sba_cmiss.o sba_cprod.o sba_ckernel.o \
        sba_cdither.o sba_cfilter.o \
        sba_compile.o sba_load.o
LISP=psearch sar/sar_lisp
LISZT=psearch sar/sar_liszt
```



```
#include <sba/sba_defs.h>

sba_dither (matrix)
        register sar_array matrix;
        {
        register int size;
        int s;

        sfe_assert (matrix->sar_etype == SOB_LONG, "\
(sba_dither - matrix element type is not a-long)");

        size = matrix->sar_xsize;

        sfe_assert (matrix->sar_ysize == size, "\
(sba_dither - matrix is not square)");
        sfe_assert (matrix->sar_exponent == -16, "\
(sba_dither - matrix exponent is not -16)");
        sfe_assert (size > 1 && ((size - 1) & size) == 0, "\
(sba_dither - matrix size not a positive power of two)");


        /* Make 2 x 2 dither matrix.                                    */
        sar_place (matrix, SAR_X, 2, 0, 1);
        sar_place (matrix, SAR_Y, 2, 0, 1);
        {       register long * p = matrix->sar_lbase;
                p [0] = (0 << 16);
                p [matrix->sar_xincrement] = (2 << 16);
                p [matrix->sar_yincrement] = (3 << 16);
                p [matrix->sar_xincrement + matrix->sar_yincrement] =
                        (1 << 16); }

        for (s = 2; s < size; s *= 2) {
                /*      Double the size of the matrix.                  */
                register int xstep =
                        matrix->sar_xincrement * matrix->sar_xsize;
                register int ystep =
                        matrix->sar_yincrement * matrix->sar_ysize;
                register int xystep = xstep + ystep;
                { sar_xfor_matrix_elements (matrix, long *,
                                matrix->sar_lbase) {
                        xp [0] <<= 2;
                        xp [xstep] = xp [0] + (2 << 16);
                        xp [ystep] = xp [0] + (3 << 16);
                        xp [xystep] = xp [0] + (1 << 16); }}
                sar_place (matrix, SAR_X, 2 * matrix->sar_xsize, 0, 1);
                sar_place (matrix, SAR_Y, 2 * matrix->sar_ysize, 0, 1); }

        return (0); }
```

```
#define SBA_CELEM_C

#include <sba/sba_defs.h>

/*      Function to add one array to another and store in a third.      */

sba_aadd (output, input1, input2)
        register sar_array output, input1, input2;
        {
        sfe_assert (sar_similar (output, input1),
                "(sba_aadd - output and first input arrays are not similar)");
        sfe_assert (sar_similar (output, input2),
                "(sba_aadd - output and second input arrays are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG,
                "(sba_aadd - output array element type is not a-long)");
        sfe_assert (input1->sar_etype == SOB_LONG,
                "(sba_aadd - first input array element type is not a-long)");
        sfe_assert (input2->sar_etype == SOB_LONG,
                "(sba_aadd - second input array element type is not a-long)");
        sfe_assert (output->sar_exponent == input1->sar_exponent,
          "(sba_aadd - output and first input arrays have unequal exponents)");
        sfe_assert (output->sar_exponent == input2->sar_exponent,
          "(sba_aadd - output and second input arrays have unequal exponents)");
        sfe_assert (sar_write (output),
                "(sfb_aadd - cannot write output array)");

        {       sar_for_3_elements (output, input1, input2, long *) {
                * x1p = (sat_lmissing (* x2p) || sat_lmissing (* x3p)) ?
                        SAT_LMISSING : * x2p + * x3p; }}

        return (0); }

/*      Function to add scalar to block floating array.         */

sba_add (array, scalar)
        register sar_array array;
        double scalar;
        {
        register long addend = sat_round (scalar, - array->sar_exponent);
        sfe_assert (array->sar_etype == SOB_LONG,
                "(sba_add - array element type is not a-long)");
        sfe_assert (sar_write (array),
                "(sfb_add - cannot write array)");


        {       sar_for_elements (array, long *)
                        if (! sat_lmissing (* xp))
                                * xp += addend; }

        return (0); }

/*      Function to subtract one array to another and store in a third. */

sba_asubtract (output, input1, input2)
        register sar_array output, input1, input2;
        {
```



```
        sfe_assert (sar_similar (output, input1), "\
(sba_asubtract - output and first input arrays are not similar)");
        sfe_assert (sar_similar (output, input2), "\
(sba_asubtract - output and second input arrays are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_asubtract - output array element type is not a-long)");
        sfe_assert (input1->sar_etype == SOB_LONG, "\
(sba_asubtract - first input array element type is not a-long)");
        sfe_assert (input2->sar_etype == SOB_LONG, "\
(sba_asubtract - second input array element type is not a-long)");
        sfe_assert (output->sar_exponent == input1->sar_exponent, "\
(sba_asubtract - output and first input arrays have unequal exponents)");
        sfe_assert (output->sar_exponent == input2->sar_exponent, "\
(sba_asubtract - output and second input arrays have unequal exponents)");
        sfe_assert (sar_write (output), "\
(sfb_asubtract - cannot write output array)");

        {       sar_for_3_elements (output, input1, input2, long *) {
                * x1p = (sat_lmissing (* x2p) || sat_lmissing (* x3p)) ?
                        SAT_LMISSING : * x2p - * x3p; }}

        return (0); }

/*      Function to multiply one array by another and store in a third. */

sba_amultiply (output, input1, input2)
        register sar_array output, input1, input2;
        {
        register sat_rdeclare;
        sfe_assert (sar_similar (output, input1), "\
(sba_amultiply - output array and first input array are not similar)");
        sfe_assert (sar_similar (output, input2), "\
(sba_amultiply - output array and second input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_amultiply - output array element type is not a-long)");
        sfe_assert (input1->sar_etype == SOB_LONG, "\
(sba_amultiply - first input array element type is not a-long)");
        sfe_assert (input2->sar_etype == SOB_LONG, "\
(sba_amultiply - second input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_amultiply - cannot write output array)");

        sat_rset (input1->sar_exponent + input2->sar_exponent
                - output->sar_exponent);

        { sar_for_3_matrices (output, input1, input2, long *) {
            if (shift < 0) {
                sar_for_3_matrix_elements (output, input1, input2, long *) {
                        if (sat_lmissing (* x2p) || sat_lmissing (* x3p))
                                * x1p = SAT_LMISSING;
                        else * x1p = sat_rmas (* x2p, * x3p); }}
            else if (shift > 0) {
                sar_for_3_matrix_elements (output, input1, input2, long *) {
                        if (sat_lmissing (* x2p) || sat_lmissing (* x3p))
                                * x1p = SAT_LMISSING;
                        else * x1p = (* x2p * * x3p) << shift; }}
```

```
        else {
            sar_for_3_matrix_elements (output, input1, input2, long *) {
                    if (sat_lmissing (* x2p) || sat_lmissing (* x3p))
                            * x1p = SAT_LMISSING;
                    else * x1p = * x2p * * x3p; }}}}

        return (0); }

/*      Function to multiply block floating array by a scalar.          */

sba_multiply (array, scalar)
        register sar_array array;
        double scalar;
        {
        register long factor;
        sat_rdeclare;
        sfe_assert (array->sar_etype == SOB_LONG,
                "(sba_multiply - array element type is not a-long)");
        sfe_assert (sar_write (array),
                "(sfb_multiply - cannot write array)");

        if (scalar != 0) {
                register int exponent = sat_log (scalar) - 30;
                factor = sat_round (scalar, - exponent);
                sat_rset (exponent); }
        else {
                factor = 0; sat_rset (0);}

        {       sar_for_matrices (array, long *) {
                        if (shift == 0) {
                                sar_for_matrix_elements (array, long *)
                                        if (! sat_lmissing (* xp))
                                                * xp *= factor; }

                        else if (shift > 0) {
                                sar_for_matrix_elements (array, long *)
                                        if (! sat_lmissing (* xp))
                                                * xp = (* xp * factor) << shift; }

                        else if (shift < 0) {
                                sar_for_matrix_elements (array, long *)
                                        if (! sat_lmissing (* xp))
                                                * xp = sat_rmas (* xp, factor); }
                        }}

        return (0); }

/*      Function to maximize one array with another and store in a third.*/

sba_amaximize (output, input1, input2)
        register sar_array output, input1, input2;
        {
        sfe_assert (sar_similar (output, input1), "\
(sba_amaximize - output and first input arrays are not similar)");
        sfe_assert (sar_similar (output, input2), "\
(sba_amaximize - output and second input arrays are not similar)");
```



```
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_amaximize - output array element type is not a-long)");
        sfe_assert (input1->sar_etype == SOB_LONG, "\
(sba_amaximize - first input array element type is not a-long)");
        sfe_assert (input2->sar_etype == SOB_LONG, "\
(sba_amaximize - second input array element type is not a-long)");
        sfe_assert (output->sar_exponent == input1->sar_exponent, "\
(sba_amaximize - output and first input arrays have unequal exponents)");
        sfe_assert (output->sar_exponent == input2->sar_exponent, "\
(sba_amaximize - output and second input arrays have unequal exponents)");
        sfe_assert (sar_write (output), "\
(sfb_amaximize - cannot write output array)");

        {       sar_for_3_elements (output, input1, input2, long *) {
                * x1p = sat_lmissing (* x2p) ? SAT_LMISSING :
                        sat_lmissing (* x3p) ? SAT_LMISSING :
                        * x2p > * x3p ? * x2p : * x3p; }}

        return (0); }

/*      Function to replace each element of a block floating array      */
/*      with the maximum of that element and a scalar.                  */

sba_maximize (array, scalar)
        register sar_array array;
        double scalar;
        {
        register long maxend = sat_round (scalar, - array->sar_exponent);
        sfe_assert (array->sar_etype == SOB_LONG,
                "(sba_maximize - array element type is not a-long)");
        sfe_assert (sar_write (array),
                "(sfb_maximize - cannot write array)");


        {       sar_for_elements (array, long *) {
                        if (! sat_lmissing (* xp) && * xp < maxend)
                                * xp = maxend; }}

        return (0); }

/*      Function to minimize one array with another and store in a third.*/

sba_aminimize (output, input1, input2)
        register sar_array output, input1, input2;
        {
        sfe_assert (sar_similar (output, input1), "\
(sba_aminimize - output and first input arrays are not similar)");
        sfe_assert (sar_similar (output, input2), "\
(sba_aminimize - output and second input arrays are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_aminimize - output array element type is not a-long)");
        sfe_assert (input1->sar_etype == SOB_LONG, "\
(sba_aminimize - first input array element type is not a-long)");
        sfe_assert (input2->sar_etype == SOB_LONG, "\
(sba_aminimize - second input array element type is not a-long)");
        sfe_assert (output->sar_exponent == input1->sar_exponent, "\
```

```
(sba_aminimize - output and first input arrays have unequal exponents)");
        sfe_assert (output->sar_exponent == input2->sar_exponent, "\
(sba_aminimize - output and second input arrays have unequal exponents)");
        sfe_assert (sar_write (output), "\
(sfb_aminimize - cannot write output array)");

        {       sar_for_3_elements (output, input1, input2, long *) {
                * x1p = sat_lmissing (* x2p) ? SAT_LMISSING :
                        sat_lmissing (* x3p) ? SAT_LMISSING :
                        * x2p < * x3p ? * x2p : * x3p; }}

        return (0); }

/*      Function to replace each element of a block floating array      */
/*      with the minimum of that element and a scalar.                  */

sba_minimize (array, scalar)
        register sar_array array;
        double scalar;
        {
        register long minend = sat_round (scalar, - array->sar_exponent);
        sfe_assert (array->sar_etype == SOB_LONG,
                "(sba_minimize - array element type is not a-long)");
        sfe_assert (sar_write (array),
                "(sfb_minimize - cannot write array)");


        {       sar_for_elements (array, long *) {
                        if (! sat_lmissing (* xp) && * xp > minend)
                                * xp = minend; }}

        return (0); }

/*      Function to set all elements of a block floating array          */
/*      to a scalar.                                                    */

sba_set (array, scalar)
        register sar_array array;
        double scalar;
        {
        register long value = sat_dmissing (scalar) ? SAT_LMISSING :
                sat_round (scalar, - array->sar_exponent);
        sfe_assert (array->sar_etype == SOB_LONG,
                "(sba_set - array element type is not a-long)");
        sfe_assert (sar_write (array),
                "(sfb_set - cannot write array)");


        {       sar_for_elements (array, long *)
                        * xp = value; }

        return (0); }

/*      Function to take the absolute value of each array element       */
/*      and store in a second array.                                    */
```



```
sba_aabsolute (output, input)
        register sar_array output, input;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_aabsolute - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_aabsolute - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_aabsolute - input array element type is not a-long)");
        sfe_assert (output->sar_exponent == input->sar_exponent, "\
(sba_aabsolute - output and input arrays have unequal exponents)");
        sfe_assert (sar_write (output), "\
(sfb_aabsolute - cannot write output array)");


        {       register long v;
                sar_for_2_elements (output, input, long *) {
                        v = * x2p;
                        * x1p = sat_lmissing (v) ? SAT_LMISSING :
                                v >= 0 ? v :
                                - v; }}
        return (0); }

/*      Function to take the logrithm of each array element and store   */
/*      in a second array.                                               */

sba_alog (output, input)
        register sar_array output, input;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_alog - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_alog - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_alog - input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_alog - cannot write output array)");


        {       double maximum = ldexp ((double) SAT_LMAXIMUM,
                                        output->sar_exponent);
                double minimum = ldexp ((double) SAT_LMINIMUM,
                                        output->sar_exponent);

                sar_for_2_elements (output, input, long *) {
                        if (sat_lmissing (* x2p) || * x2p <= 0)
                                * x1p = SAT_LMISSING;
                        else {
                                register double value =
                                  log (ldexp ((double) * x2p,
                                                input->sar_exponent));
                                if (value >= maximum)
                                        * x1p = SAT_LMAXIMUM;
                                else if (value <= minimum)
                                        * x1p = SAT_LMINIMUM;
                                else * x1p = sat_round (value,
```

```
                                                  - output->sar_exponent); }}}
        return (0); }

/*      Function to take the exponent of each array element and store    */
/*      in a second array.                                                */

sba_aexp (output, input)
        register sar_array output, input;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_aexp - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_aexp - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_aexp - input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_aexp - cannot write output array)");


        {       double maximum = ldexp ((double) SAT_LMAXIMUM,
                                                output->sar_exponent);
                double minimum = ldexp ((double) SAT_LMINIMUM,
                                                output->sar_exponent);

                sar_for_2_elements (output, input, long *) {
                        if (sat_lmissing (* x2p)) * xlp = SAT_LMISSING;
                        else {
                                register double value =
                                  exp (ldexp ((double) * x2p,
                                                input->sar_exponent));
                                if (value >= maximum)
                                        * xlp = SAT_LMAXIMUM;
                                else if (value <= minimum)
                                        * xlp = SAT_LMINIMUM;
                                else * xlp = sat_round (value,
                                                - output->sar_exponent); }}}
        return (0); }

/*      Function to take the square root of each array element           */
/*      and store in a second array.                                      */

sba_asqrt (output, input)
        register sar_array output, input;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_asqrt - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_asqrt - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_asqrt - input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_asqrt - cannot write output array)");


        {       double maximum = ldexp ((double) SAT_LMAXIMUM,
                                                output->sar_exponent);
```



```
                double minimum = ldexp ((double) SAT_LMINIMUM,
                                                output->sar_exponent);

                sar_for_2_elements (output, input, long *) {
                        if (sat_lmissing (* x2p) || * x2p < 0)
                                * xlp = SAT_LMISSING;
                        else {
                                register double value =
                                  sqrt (ldexp ((double) * x2p,
                                                input->sar_exponent));
                                if (value >= maximum)
                                        * xlp = SAT_LMAXIMUM;
                                else if (value <= minimum)
                                        * xlp = SAT_LMINIMUM;
                                else * xlp = sat_round (value,
                                                - output->sar_exponent); }}}
        return (0); }

/*      Function to take the power of each array element by a scalar     */
/*      and store in a second array.                                      */

sba_apower (output, input, exponent)
        register sar_array output, input;
        double exponent;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_apower - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_apower - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_apower - input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_apower - cannot write output array)");


        {       double maximum = ldexp ((double) SAT_LMAXIMUM,
                                                output->sar_exponent);
                double minimum = ldexp ((double) SAT_LMINIMUM,
                                                output->sar_exponent);

                sar_for_2_elements (output, input, long *) {
                        if (sat_lmissing (* x2p) || * x2p < 0)
                                * xlp = SAT_LMISSING;
                        else {
                                register double value =
                                  pow (ldexp ((double) * x2p,
                                     input->sar_exponent), exponent);
                                if (value >= maximum)
                                        * xlp = SAT_LMAXIMUM;
                                else if (value <= minimum)
                                        * xlp = SAT_LMINIMUM;
                                else * xlp = sat_round (value,
                                                - output->sar_exponent); }}}
        return (0); }

/*      Function to take the sine of each array element                   */
```

```
/*      and store in a second array.                                    */

sba_asin (output, input)
        register sar_array output, input;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_asin - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_asin - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_asin - input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_asin - cannot write output array)");


        {       double maximum = ldexp ((double) SAT_LMAXIMUM,
                                        output->sar_exponent);
                double minimum = ldexp ((double) SAT_LMINIMUM,
                                        output->sar_exponent);

                sar_for_2_elements (output, input, long *) {
                        if (sat_lmissing (* x2p)) * xlp = SAT_LMISSING;
                        else {
                                register double value =
                                  sin (ldexp ((double) * x2p,
                                              input->sar_exponent));
                                if (value >= maximum)
                                        * xlp = SAT_LMAXIMUM;
                                else if (value <= minimum)
                                        * xlp = SAT_LMINIMUM;
                                else * xlp = sat_round (value,
                                             - output->sar_exponent); }}}
        return (0); }

/*      Function to take the cosine of each array element               */
/*      and store in a second array.                                    */

sba_acos (output, input)
        register sar_array output, input;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_acos - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_acos - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_acos - input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_acos - cannot write output array)");


        {       double maximum = ldexp ((double) SAT_LMAXIMUM,
                                        output->sar_exponent);
                double minimum = ldexp ((double) SAT_LMINIMUM,
                                        output->sar_exponent);

                sar_for_2_elements (output, input, long *) {
```



```
                        if (sat_lmissing (* x2p)) * xlp = SAT_LMISSING;
                        else {
                                register double value =
                                  cos (ldexp ((double) * x2p,
                                              input->sar_exponent));
                                if (value >= maximum)
                                        * xlp = SAT_LMAXIMUM;
                                else if (value <= minimum)
                                        * xlp = SAT_LMINIMUM;
                                else * xlp = sat_round (value,
                                             - output->sar_exponent); }}}
        return (0); }

/*      Function to take the arc sine of each array element             */
/*      and store in a second array.                                    */

sba_aarcsin (output, input)
        register sar_array output, input;
        {
        sfe_assert (sar_similar (output, input), "\
(sba_aarcsin - output array and input array are not similar)");
        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_aarcsin - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_aarcsin - input array element type is not a-long)");
        sfe_assert (sar_write (output), "\
(sfb_aarcsin - cannot write output array)");


        {       double maximum = ldexp ((double) SAT_LMAXIMUM,
                                        output->sar_exponent);
                double minimum = ldexp ((double) SAT_LMINIMUM,
                                        output->sar_exponent);
                register long unit = sat_round (1.0, - input->sar_exponent);

                sar_for_2_elements (output, input, long *) {
                        if (sat_lmissing (* x2p) ||
                                * x2p > unit || * x2p < - unit)
                                * xlp = SAT_LMISSING;
                        else {
                                register double value =
                                  asin (ldexp ((double) * x2p,
                                              input->sar_exponent));
                                if (value >= maximum)
                                        * xlp = SAT_LMAXIMUM;
                                else if (value <= minimum)
                                        * xlp = SAT_LMINIMUM;
                                else * xlp = sat_round (value,
                                             - output->sar_exponent); }}}
        return (0); }

/*      Function to take the arc cosine of each array element           */
/*      and store in a second array.                                    */

sba_aarccos (output, input)
        register sar_array output, input;
```

```
	{
	sfe_assert (sar_similar (output, input), "\
(sba_aarccos - output array and input array are not similar)");
	sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_aarccos - output array element type is not a-long)");
	sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_aarccos - input array element type is not a-long)");
	sfe_assert (sar_write (output), "\
(sfb_aarccos - cannot write output array)");

	{	double maximum = ldexp ((double) SAT_LMAXIMUM,
					output->sar_exponent);
		double minimum = ldexp ((double) SAT_LMINIMUM,
					output->sar_exponent);

		register long unit = sat_round (1.0, - input->sar_exponent);

		sar_for_2_elements (output, input, long *) {
			if (sat_lmissing (* x2p) ||
				* x2p > unit || * x2p < - unit)
				* x1p = SAT_LMISSING;
			else {
				register double value =
				  acos (ldexp ((double) * x2p,
						input->sar_exponent));
				if (value >= maximum)
					* x1p = SAT_LMAXIMUM;
				else if (value <= minimum)
					* x1p = SAT_LMINIMUM;
				else * x1p = sat_round (value,
						- output->sar_exponent); }}}
	return (0); }
```



```
#include <sba/sba_defs.h>

sba_dfilter (array, dimension, width)
	register sar_array array;
	int dimension;
	int width;
	{
	int width_minus_1 = width - 1;
	int divisor = ((width * width - 1) * width) / 6;
	int half_divisor = divisor / 2;

	sfe_assert (array->sar_etype == SOB_LONG, "\
(sba_dfilter - array argument element type is not a-long)");
	sfe_assert (0 <= dimension && dimension < SAR_MDIMENSIONS, "\
(sba_dfilter - dimension argument out of range)");
	sfe_assert (sar_write (array), "\
(sba_dfilter - array argument is not writable)");
	sfe_assert (0 < width && width <= 128, "\
(sba_dfilter - width is not > 0 and <= 128)");

	sar_transpose (array, dimension, 0);

	sfe_assert (width <= array->sar_xsize, "\
(sba_dfilter - width is > dimension size)");

	{ sar_for_matrices (array, long *) {
	    int Y; long * yp;
	    for (Y = 0, yp = zp;
		Y < array->sar_ysize;
		++ Y, yp += array->sar_yincrement) {
		int X = 0;
		register long * xp = yp;
		register long sum = 0;
		register long difference = 0;
		register long * oldxp = yp;

		for (; X < width; ++ X, xp += array->sar_xincrement) {
			difference += * xp * width_minus_1 - (sum << 1);
			sum += * xp; }
		for ( ; ; ++ X, xp += array->sar_xincrement,
			oldxp += array->sar_xincrement) {
			register long temp = * oldxp;
			* oldxp = (difference +
					(difference >= 0 ? half_divisor :
						- half_divisor))
				/ divisor;
			if (X >= array->sar_xsize) break;
			difference += (temp + * xp) * width_minus_1;
			sum -= temp;
			difference -= sum << 1;
			sum += * xp; }}}}

	sar_place (array, SAR_X, array->sar_xsize - width + 1, 0, 1);
	sar_transpose (array, dimension, 0);
	return (0); }
```

```
sba_hfilter (array, dimension, width)
        register sar_array array;
        int dimension;
        int width;
        {
        sfe_assert (array->sar_etype == SOB_LONG, "\
(sba_hfilter - array argument element type is not a-long)");
        sfe_assert (0 <= dimension && dimension < SAR_MDIMENSIONS, "\
(sba_hfilter - dimension argument out of range)");
        sfe_assert (sar_write (array), "\
(sba_hfilter - array argument is not writable)");
        sfe_assert (0 < width, "\
(sba_hfilter - width is not > 0)");

        sar_transpose (array, dimension, 0);

        sfe_assert (width <= array->sar_xsize, "\
(sba_hfilter - width is > dimension size)");

        { sar_for_matrices (array, long *) {
            int Y; long * yp;
            for (Y = 0, yp = zp;
                Y < array->sar_ysize;
                ++ Y, yp += array->sar_yincrement) {
                int X = 0;
                register long * xp = yp;
                register long maximum = SAT_LMINIMUM;
                register long * oldxp = yp;

                for (; X < width; ++ X, xp += array->sar_xincrement) {
                        if (* xp > maximum) maximum = * xp; }
                for ( ; ; ++ X, xp += array->sar_xincrement,
                        oldxp += array->sar_xincrement) {
                        if (* oldxp == maximum) {
                                register long * xp2 = oldxp;
                                register long oldmaximum = maximum;
                                maximum = SAT_LMINIMUM;
                                while ( xp2 += array->sar_xincrement,
                                        xp2 != xp) {
                                        if (* xp2 > maximum) {
                                                maximum = * xp2;
                                                if (maximum == oldmaximum)
                                                        break; }}}
                        else * oldxp = maximum;
                        if (X >= array->sar_xsize) break;
                        if (* xp > maximum) maximum = * xp; }}}}

        sar_place (array, SAR_X, array->sar_xsize - width + 1, 0, 1);
        sar_transpose (array, dimension, 0);
        return (0); }

sba_lfilter (array, dimension, width)
        register sar_array array;
        int dimension;
        int width;
        {
```



```
        sfe_assert (array->sar_etype == SOB_LONG, "\
(sba_lfilter - array argument element type is not a-long)");
        sfe_assert (0 <= dimension && dimension < SAR_MDIMENSIONS, "\
(sba_lfilter - dimension argument out of range)");
        sfe_assert (sar_write (array), "\
(sba_lfilter - array argument is not writable)");
        sfe_assert (0 < width, "\
(sba_lfilter - width is not > 0)");

        sar_transpose (array, dimension, 0);

        sfe_assert (width <= array->sar_xsize, "\
(sba_lfilter - width is > dimension size)");

        { sar_for_matrices (array, long *) {
            int Y; long * yp;
            for (Y = 0, yp = zp;
                Y < array->sar_ysize;
                ++ Y, yp += array->sar_yincrement) {
                int X = 0;
                register long * xp = yp;
                register long minimum = SAT_LMAXIMUM;
                register long * oldxp = yp;

                for (; X < width; ++ X, xp += array->sar_xincrement) {
                        if (* xp < minimum) minimum = * xp; }
                for ( ; ; ++ X, xp += array->sar_xincrement,
                        oldxp += array->sar_xincrement) {
                        if (* oldxp == minimum) {
                                register long * xp2 = oldxp;
                                register long oldminimum = minimum;
                                minimum = SAT_LMAXIMUM;
                                while ( xp2 += array->sar_xincrement,
                                        xp2 != xp) {
                                        if (* xp2 < minimum) {
                                                minimum = * xp2;
                                                if (minimum == oldminimum)
                                                        break; }}}
                        else * oldxp = minimum;
                        if (X >= array->sar_xsize) break;
                        if (* xp < minimum) minimum = * xp; }}}}

        sar_place (array, SAR_X, array->sar_xsize - width + 1, 0, 1);
        sar_transpose (array, dimension, 0);
        return (0); }

sba_sfilter (array, dimension, width)
        register sar_array array;
        int dimension;
        int width;
        {
        sfe_assert (array->sar_etype == SOB_LONG, "\
(sba_sfilter - array argument element type is not a-long)");
        sfe_assert (0 <= dimension && dimension < SAR_MDIMENSIONS, "\
(sba_sfilter - dimension argument out of range)");
        sfe_assert (sar_write (array), "\
```

```
(sba_sfilter - array argument is not writable)");
        sfe_assert (0 < width, "\
(sba_sfilter - width is not > 0)");

        sar_transpose (array, dimension, 0);

        sfe_assert (width <= array->sar_xsize, "\
(sba_sfilter - width is > dimension size)");

        { sar_for_matrices (array, long *) {
            int Y; long * yp;
            for (Y = 0, yp = zp;
                Y < array->sar_ysize;
                ++ Y, yp += array->sar_yincrement) {
                int X = 0;
                register long * xp = yp;
                register long sum = 0;
                register long * oldxp = yp;

                for (; X < width; ++ X, xp += array->sar_xincrement)
                        sum += * xp;
                for ( ; ; ++ X, xp += array->sar_xincrement,
                        oldxp += array->sar_xincrement) {
                        register long temp = * oldxp;
                        * oldxp = sum / width;
                        if (X >= array->sar_xsize) break;
                        sum += * xp - temp; }}}}

        sar_place (array, SAR_X, array->sar_xsize - width + 1, 0, 1);
        sar_transpose (array, dimension, 0);
        return (0); }

sba_contrast (output, input, widths, background, center)
        register sar_array output, input;
        register sat_lvalue widths;
        double background, center;
        {
        {       register int dimension;
                register sar_dimension dim;
                for (dimension = 0, dim = input->sar_dimensions;
                        widths != sat_nil;
                        ++ dimension, widths = widths->sat_lrest) {
                        register int w = widths->sat_lfirst->sat_lint;
                        sfe_assert (dimension < SAR_MDIMENSIONS, "\
(sba_contrast - widths argument list is too long)");
                        sba_sfilter (output, dimension, 2 * w + 1);
                        sfe_check ();
                        sar_place (input, dimension,
                                dim->sar_size - 2 * w, w, 1);
                        sfe_check (); }}

        {       register int exponent;
                register sat_rdeclare;
                register long b, c;
                {       register int blog = sat_log (background);
                        register int clog = sat_log (center);
```



```
                        exponent = -30 + (blog > clog ? blog : clog); }
                b = sat_round (background, - exponent);
                c = sat_round (center, - exponent);
                sat_rset (exponent);

                {       sar_for_2_elements (output, input, long *) {
                                if (sat_lmissing (* xlp)) /* do nothing */;
                                else if (sat_lmissing (* x2p))
                                        * xlp = SAT_LMISSING;
                                else * xlp = sat_rmas (* x2p, c)
                                                - sat_rmas (* xlp, b); }}}
        return (0); }

sba_afilter (array, dimension)
        register sar_array array;
        int dimension;
        {
        sfe_assert (array->sar_etype == SOB_LONG, "\
(sba_afilter - array argument element type is not a-long)");
        sfe_assert (0 <= dimension && dimension < SAR_MDIMENSIONS, "\
(sba_afilter - dimension argument out of range)");
        sfe_assert (sar_write (array), "\
(sba_afilter - array argument is not writable)");

        sar_transpose (array, dimension, 0);

        {       sar_for_matrices (array, long *) {
                        register long * xp, * yp;
                        register long total;
                        register int nx, ny;
                        for (   ny = array->sar_ysize, yp = zp;
                                ny; yp += array->sar_yincrement, -- ny)
                        for (   nx = array->sar_xsize,
                                        xp = yp, total = 0;
                                nx; xp += array->sar_xincrement, -- nx)
                                if (sat_lmissing (total)
                                        || sat_lmissing (* xp))
                                        total = * xp = SAT_LMISSING;
                                else * xp = (total += * xp); }}
        sar_transpose (array, dimension, 0);
        return (0); }

/* Kludge.  The following variables have been moved from their natural  */
/* place as local variables because the SUN3 compiler has trouble with   */
/* too many local variables.                                             *

/* Group 1 */
static  long temp [SAR_MSIZE];
static  int xsize;
/* Group 2 */
static    int offset_unit;
static    long offset_fraction;
static    int factor_unit;
static    long factor_fraction;

sba_ifilter (array, dimension, factor, offset)
```

```
        register sar_array array;
        int dimension;
        double factor;
        double offset;
        {
        /* Group 1 local variables belong here. */
        xsize = 0;

        sfe_assert (array->sar_etype == SOB_LONG, "\
(sba_ifilter - array argument element type is not a-long)");
        sfe_assert (0 <= dimension && dimension < SAR_MDIMENSIONS, "\
(sba_ifilter - dimension argument out of range)");
        sfe_assert (sar_write (array), "\
(sba_ifilter - array argument is not writable)");
        sfe_assert (- SAR_MSIZE <= factor && factor <= SAR_MSIZE, "\
(sba_ifilter - factor is out of range)");

        sar_transpose (array, dimension, SAR_X);

        sfe_assert (array->sar_xsize <= SAR_MSIZE, "\
(sba_ifilter - array argument dimension size is above maximum dimension size)");

        if (0 <= offset && offset <= array->sar_xsize - 1) {
          sat_rdeclare;
          /* Group 2 local varibles belong here. */
          offset_unit = (int) floor (offset);
          offset_fraction = sat_round (offset - offset_unit, 30);
          factor_unit = (int) floor (factor);
          factor_fraction = sat_round (factor - factor_unit, 30);

          sat_rset (-30);

          { sar_for_matrices (array, long *) {
              int Y; long * yp;
              for (Y = 0, yp = zp;
                Y < array->sar_ysize;
                ++ Y, yp += array->sar_yincrement) {
                register int X;
                register long * xp;
                register long * tp;
                register int unit = offset_unit;
                register long fraction = offset_fraction;


                for (xp = yp, X = 0, tp = temp; X < array->sar_xsize;
                        ++ X, xp += array->sar_xincrement) * tp ++ = * xp;

                for (X = 0, xp = yp, tp = temp + unit;
                        X < array->sar_xsize && unit < array->sar_xsize;
                        ++ X, xp += array->sar_xincrement) {
                        if (unit < 0) {
                                if (unit == -1 &&
                                        fraction >= (1 << 30)
                                                - (1 - (30 - 10)))
                                        * xp = tp [1];
                                else break; }
```



```
                        if (fraction == 0) * xp = * tp;
                        else if (unit + 1 >= array->sar_xsize) {
                                if (fraction <= (1 << (30 - 10)))
                                        * xp = * tp;
                                else break; }
                        else if (sat_lmissing (tp [0])
                                        || sat_lmissing (tp [1]))
                                * xp = SAT_LMISSING;
                        else * xp = sat_rmas (tp [0], (1 << 30) - fraction)
                                        + sat_rmas (tp [1], fraction);
                        tp += factor_unit;
                        unit += factor_unit;
                        fraction += factor_fraction;
                        if (fraction >= (1 << 30)) {
                                fraction -= (1 << 30);
                                ++ tp;
                                ++ unit; }}

                xsize = X; }}}}

        sar_place (array, SAR_X, xsize, 0, 1);
        sar_transpose (array, dimension, SAR_X);
        return (0); }
```

```
#include <sar/sar_defs.h>

sba_gkernel (kernel, amplitude, xwidth, ywidth, xorigin, yorigin)
	register sar_array kernel;
	double amplitude, xwidth, ywidth, xorigin, yorigin;
	{
	sfe_assert (kernel->sar_etype == SOB_LONG, "\
(sba_gkernel - kernel array argument element type is not a-long)");

	{	sar_xfor_matrix_elements (kernel, long *, kernel->sar_lbase) {
		double xexponent = (X - xorigin) / xwidth;
		double yexponent = (Y - yorigin) / ywidth;
		double exponent =
			- (xexponent * xexponent + yexponent * yexponent);
		* xp = sat_round (amplitude * exp (exponent),
					- kernel->sar_exponent); }}
	return (0); }

sba_d2gkernel (kernel, amplitude, xwidth, ywidth, xorigin, yorigin)
	register sar_array kernel;
	double amplitude, xwidth, ywidth, xorigin, yorigin;
	{
	sfe_assert (kernel->sar_etype == SOB_LONG, "\
(sba_d2gkernel - kernel array argument element type is not a-long)");

	{	sar_xfor_matrix_elements (kernel, long *, kernel->sar_lbase) {
		double xexponent = (X - xorigin) / xwidth;
		double yexponent = (Y - yorigin) / ywidth;
		double exponent =
			1.0 - (xexponent * xexponent + yexponent * yexponent);
		* xp = sat_round (amplitude * exponent * exp (exponent),
					- kernel->sar_exponent); }}
	return (0); }

sba_dxgkernel (kernel, amplitude, xwidth, ywidth, xorigin, yorigin)
	register sar_array kernel;
	double amplitude, xwidth, ywidth, xorigin, yorigin;
	{
	sfe_assert (kernel->sar_etype == SOB_LONG, "\
(sba_dxgkernel - kernel array argument element type is not a-long)");

	{	sar_xfor_matrix_elements (kernel, long *, kernel->sar_lbase) {
		double xexponent = (X - xorigin) / xwidth;
		double yexponent = (Y - yorigin) / ywidth;
		double exponent =
			- xexponent * xexponent - yexponent * yexponent;
		* xp = sat_round (amplitude * xexponent * exp (exponent),
					- kernel->sar_exponent); }}
	return (0); }
```



```
#include <sba/sba_defs.h>

sba_mmissing (output, input, minimum, maximum, lower, upper,
			xsize, ysize, count)
	register sar_array output, input;
	double minimum, maximum, lower, upper;
	int xsize, ysize, count;
	{
	long lminimum, lmaximum, lupper, llower;
	int missing_count = 0;

	if (! sat_dmissing (lower) || ! sat_dmissing (upper)) {
		sfe_assert (sar_xsimilar (output, input, 03), "\
(sba_mmissing - output and input arrays are not similar for dimensions\
 other than X and Y)");
		sfe_assert (input->sar_xsize == output->sar_xsize + 2 * xsize,
			"\
(sba_mmissing - input X size != output X size + 2 * xsize)");
		sfe_assert (input->sar_ysize == output->sar_ysize + 2 * ysize,
			"\
(sba_mmissing - input Y size != output Y size + 2 * ysize)"); }

	else sfe_assert (sar_similar (output, input),
		"(sba_mmissing - output and input arrays are not similar)");

	sfe_assert (output->sar_etype == SOB_LONG,
		"(sba_mmissing - output array element type is not a-long)");
	sfe_assert (input->sar_etype == SOB_LONG,
		"(sba_mmissing - input array element type is not a-long)");
	sfe_assert (output->sar_exponent == input->sar_exponent,
	  "(sba_mmissing - input and output arrays have different exponents)");
	sfe_assert (sat_dmissing (minimum) || sat_dmissing (maximum)
			|| minimum <= maximum,
		"(sba_mmissing - minimum > maximum)");
	sfe_assert (sat_dmissing (upper) || sat_dmissing (lower)
			|| lower <= upper,
		"(sba_mmissing - lower > upper)");
	sfe_assert (sat_imissing (xsize) || xsize >= 0,
		"(sba_mmissing - xsize < 0)");
	sfe_assert (sat_imissing (ysize) || ysize >= 0,
		"(sba_mmissing - ysize < 0)");
	sfe_assert (sat_imissing (count) || count >= 0,
		"(sba_mmissing - count < 0)");
	sfe_assert (sar_write (output),
		"(sba_mmissing - output array is not writable)")

	lminimum = sat_dmissing (minimum) ? SAT_LMINIMUM :
		sat_round (minimum, - input->sar_exponent);
	lmaximum = sat_dmissing (maximum) ? SAT_LMAXIMUM :
		sat_round (maximum, - input->sar_exponent);
	llower = sat_dmissing (lower) ? SAT_LMISSING :
		sat_round (lower, - input->sar_exponent);
	lupper = sat_dmissing (upper) ? SAT_LMISSING :
		sat_round (upper, - input->sar_exponent);

	{	register long value;
```

```
		register int wcount =
			sat_lmissing (llower) && sat_lmissing (lupper) ? 0 :
			count + 1;
		int wxsize = wcount == 0 ? 0 : 2 * xsize + 1;
		int wysize = wcount == 0 ? 0 : 2 * ysize + 1;
		register int center = wcount == 0 ? 0 :
			xsize * input->sar_xincrement +
				ysize * input->sar_yincrement;

		sar_for_2_elements (output, input, long *) {


			value = x2p [center];
			if (sat_lmissing (value)
				|| value < lminimum || value > lmaximum
				|| (wcount != 0 && SBA_MMISSING (
					x2p, input->sar_xincrement,
						input->sar_yincrement,
					wxsize, wysize, wcount,
					lminimum, lmaximum,
					(long) (sat_lmissing (llower) ?
						SAT_LMINIMUM :
						value+llower),
					(long) (sat_lmissing (lupper) ?
						SAT_LMAXIMUM :
						value+lupper)))) {
				* xlp = SAT_LMISSING;
				++ missing_count; }
			else * xlp = value; }}

	return (missing_count); }

	static
SBA_MMISSING (yp, xincrement, yincrement, xsize, ysize, count,
			lminimum, lmaximum, llower, lupper)
	long * yp;
	register int xincrement;
	int yincrement;
	int xsize;
	int ysize;
	register int count;
	long lminimum, lmaximum;
	register long llower, lupper;
	{
	register long v;
	register long * xp;
	register int xs;

	if (lminimum > llower) llower = lminimum;
	if (lmaximum < lupper) lupper = lmaximum;
	for (; ysize > 0; -- ysize, yp += yincrement)
		for (xp = yp, xs = xsize; xs > 0; -- xs, xp += xincrement) {
			v = * xp;
			if (! sat_lmissing (v) && llower <= v && v <= lupper
				&& -- count == 0) return (0); }
	return (1); }
```



```
sba_cmissing (output, input, original, xsize, ysize, count)
	register sar_array output, input, original;
	int xsize, ysize, count;
	{
	int replacement_count = 0;

	sfc_assert (sar_xsimilar (output, input, 03), "\
(sba_emissing - input array and output array arguments are not similar\
 for dimensions other than X and Y)");
	sfc_assert (xsize >= 0, "(sba_emissing - xsize < 0)");
	sfc_assert (ysize >= 0, "(sba_emissing - ysize < 0)");
	sfc_assert (input->sar_xsize == output->sar_xsize + 2 * xsize, "\
(sba_emissing - input array X size != output array X size + 2 * xsize)");
	sfc_assert (input->sar_ysize == output->sar_ysize + 2 * ysize, "\
(sba_emissing - input array Y size != output array Y size + 2 * ysize)");
	sfc_assert (sar_similar (output, original), "\
(sba_emissing - original array and output array arguments are not similar)");
	sfc_assert (output->sar_etype == SOR_LONG,
		"(sba_emissing - output array element type is not a-long)");
	sfc_assert (input->sar_etype == SOR_LONG,
		"(sba_emissing - input array element type is not a-long)");
	sfc_assert (original->sar_etype == SOR_LONG,
		"(sba_emissing - original array element type is not a-long)");
	sfc_assert (output->sar_exponent == input->sar_exponent,
	  "(sba_emissing - input and output arrays have different exponents)");
	sfc_assert (output->sar_exponent == original->sar_exponent, "\
(sba_emissing - original and output arrays have different exponents)");
	sfc_assert (count >= 0, "(sba_emissing - count < 0)");
	sfc_assert (sar_write (output),
		"(sba_emissing - output array is not writable)");

	{
		register int wxsize = 2 * xsize + 1;
		register int wysize = 2 * ysize + 1;
		register int center =
			xsize * input->sar_xincrement +
				ysize * input->sar_yincrement;
		sar_for_3_elements (output, input, original, long *) {

			register long v = x2p [center];

			if (sat_lmissing (v) || ! sat_lmissing (* x3p)) {
				* xlp = v;
				continue; }

			if (count == 0 || SBA_EMISSING (
					x2p, input->sar_xincrement,
					input->sar_yincrement,
					wxsize, wysize, count)) {
				* xlp = SAT_LMISSING;
				++ replacement_count; }
			else * xlp = v; }}

	return (replacement_count); }
```

```
        static
SBA_EMISSING (yp, xincrement, yincrement, xsize, ysize, count)
        long * yp;
        register int xincrement;
        int yincrement;
        int xsize;
        int ysize;
        register int count;
        {
        register long * xp;
        register int xs;

        for (; ysize > 0; -- ysize, yp += yincrement)
                for (xp = yp, xs = xsize; xs > 0; -- xs, xp += xincrement)
                        if (sat_lmissing (* xp) && -- count == 0) return (1);
        return (0); }

sba_smissing (output, input, count)
        register sar_array output, input;
        int count;
        {
        int missing_count = 0;

        sfe_assert (sar_xsimilar (output, input, 03), "\
(sba_smissing - input array and output array arguments are not similar\
 for dimensions other than X and Y)");
        sfe_assert (input->sar_xsize == output->sar_xsize + 2, "\
(sba_smissing - input array X size != output array X size + 2)");
        sfe_assert (input->sar_ysize == output->sar_ysize + 2, "\
(sba_smissing - input array Y size != output array Y size + 2)");
        sfe_assert (output->sar_etype == SOB_LONG,
                "(sba_smissing - output array element type is not a-long)");
        sfe_assert (input->sar_etype == SOB_LONG,
                "(sba_smissing - input array element type is not a-long)");
        sfe_assert (output->sar_exponent == input->sar_exponent,
          "(sba_smissing - input and output arrays have different exponents)");
        sfe_assert (count >= 2,
          "(sba_smissing - count < 2)");
        sfe_assert (sar_write (output),
                "(sba_smissing - output array is not writable)");

        {       register long value;
                register long temp;
                register long first;
                register long second;
                register long range;

                sar_xfor_2_elements (output, long *, output->sar_lbase,
                                        input, long *,
                                        input->sar_lbase
                                                + input->sar_xincrement
                                                + input->sar_yincrement) {

#                       define down (input->sar_yincrement)
#                       define up (- down)
#                       define right (input->sar_xincrement)
```



```
#                       define left (- right)

                        value = * x2p;
                        if (! sat_lmissing (value)) {
                                * x1p = value;
                                continue; }

#                       define try(a)                                   \
                          if (! sat_lmissing (x2p [a])                  \
                                && -- temp == 0) goto ok1;              \
                          else
                        temp = count;
                        try (left);
                        try (left + up);
                        try (up);
                        try (up + right);
                        try (right);
                        try (down + right);
                        try (down);
                        try (down + left);

                        /* Not enough non-missing neighbors */
                        goto failure;
                ok1:
                        range = SAT_LMAXIMUM;
#                       define set_first(a)                             \
                          (! sat_lmissing (first = x2p [a]))
#                       define try_second(b)                            \
                          if (! sat_lmissing (second = x2p [b])         \
                            && ((temp = first - second) < 0 ?           \
                                (temp = - temp) : temp) < range) {      \
                                  range = temp;                         \
                                  value = (first + second) >> 1; }      \
                          else

                        /* Straight Lines */
                        if (set_first (left)) try_second (right);
                        if (set_first (up)) try_second (down);
                        if (set_first (up + left)) try_second (down + right);
                        if (set_first (up + right)) try_second (down + left);

                        /* 45 Degree Bends */
                        if (set_first (up + left)) {
                                try_second (right);
                                try_second (down); }
                        if (set_first (up + right)) {
                                try_second (down);
                                try_second (left); }
                        if (set_first (down + right)) {
                                try_second (left);
                                try_second (up); }
                        if (set_first (down + left)) {
                                try_second (up);
                                try_second (right); }

                        /* 90 Degree Bends */
```

```
			if (set_first (up + left)) [
				try_second (up + right);
				try_second (down + left); ]
			if (set_first (up)) [
				try_second (right);
				try_second (left); ]
			if (set_first (down + right)) [
				try_second (up + right);
				try_second (down + left); ]
			if (set_first (down)) [
				try_second (right);
				try_second (left); ]

			/* 135 Degree Bends */
			if (set_first (up + left)) [
				try_second (up);
				try_second (left); ]
			if (set_first (up + right)) [
				try_second (up);
				try_second (right); ]
			if (set_first (down + right)) [
				try_second (right);
				try_second (down); ]
			if (set_first (down + left)) [
				try_second (down);
				try_second (left); ]

			if (! sat_lmissing (value)) [
				* x1p = value;
				continue; ]

			/* Failure */
		failure:
			* x1p = SAT_LMISSING;
			++ missing_count;

#			undef try
#			undef set_first
#			undef try_second
#			undef down
#			undef up
#			undef right
#			undef left
			]]

	return (missing_count); ]

sba_cmissing (output, input)
	register sar_array output, input;
	[
	sfe_assert (sar_similar (output, input),
		"(sba_cmissing - output and input arrays are not similar)");
	sfe_assert (output->sar_etype == SOB_LONG,
		"(sba_cmissing - output array element type is not a-long)");
	sfe_assert (input->sar_etype == SOB_LONG,
		"(sba_cmissing - input array element type is not a-long)");
```



```
	sfe_assert (output->sar_exponent == input->sar_exponent,
	  "(sba_cmissing - input and output arrays have different exponents)");
	sfe_assert (sar_write (output),
		"(sba_cmissing - output array is not writable)")

	[	sar_for_2_elements (output, input, long *) [
			if (sat_lmissing (* x1p)) * x1p = * x2p; ]]

	return (0); ]

sba_setmissing (array, scalar)
	register sar_array array;
	double scalar;
	[
	register long lscalar = sat_round (scalar, - array->sar_exponent);
	sfe_assert (array->sar_etype == SOB_LONG,
		"(sba_setmissing - array element type is not a-long)");
	sfe_assert (sar_write (array),
		"(sba_setmissing - array is not writable)");


	[	sar_for_elements (array, long *) [
			if (sat_lmissing (* xp)) * xp = lscalar; ]]

	return (0); ]
```

```
(defvar *default-dither-size*)
(defvar *kernel-cutoff*)

(putprop 'sba/sba_compile t 'version)
```



```
#include <sba/sba_defs.h>

/*      Assembly language function to take the scalar product           */
/*      of two 2 dimensional block floating arrays.                     */

/* ARGSUSED */
        long
sba_lsproduct2 (base1, yincrement1, xincrement1,
        base2, yincrement2, xincrement2,
        ysize, xsize, shift)
        register long * base1;          /* r11 or a5                    */
        register int yincrement1;       /* r10 or d7                    */
        register int xincrement1;       /* r9 or d6                     */
        register long * base2;          /* r8 or a4                     */
        register int yincrement2;       /* r7 or d5 [a1]                */
        register int xincrement2;       /* r6 or d4 [a0]                */
        int ysize;                      /* [r5] or [d5]                 */
        int xsize;                      /* [r4] or [d4]                 */
        int shift;
        {
        if (xsize == 0 || ysize == 0) return (0);
        if (shift < -128) return (0);
        assert (shift < 64);
        yincrement1 <<= 2;
        xincrement1 <<= 2;
        yincrement1 -= xincrement1 * xsize;
        yincrement2 <<= 2;
        xincrement2 <<= 2;
        yincrement2 -= xincrement2 * xsize;
#       if( SFE_VAX && ! SFE_LINT
                asm ("clrl r2; clrl r3");
                asm ("movl 28(ap), r5");
                asm ("lloop1: movl 32(ap), r4");
                        asm ("lloop2: movl (r11), r0; movl (r8), r1");
                        asm ("cmpl r0, $0x80000000; beql lmissing")
                        asm ("cmpl r1, $0x80000000; bneq lnot_missing")
                        asm ("lmissing: movl $0x8000000, r0; brb lexit")
                        asm ("lnot_missing: emul r0, r1, $0, r0");
                        asm ("addl2 r0, r2; adwc r1, r3");
                                asm ("addl2 r6, r8; addl2 r9, r11");
                                asm ("sobgtr r4, lloop2");
                        asm ("addl2 r7, r8; addl2 r10, r11");
                        asm ("sobgtr r5, lloop1");
                asm ("ashq 36(ap), r2, r0");
                asm ("bvc lexit; movl $0x7FFFFFFF, r0");
                        asm ("tstl r2; bgeq lexit; mcoml r0, r0; lexit:;");
#       else
#           if SFE_68XXX >= 68020 && ! SFE_LINT
                asm ("movl d3,sp@-; movl d2,sp@-");
                asm ("clrl d0; clrl d1");
                asm ("movl d5,a1; movl d4,a0; movl a6@(32), d5");
                asm ("lloop1: movl a6@(36), d4");
                        asm ("lloop2: movl a5@,d2; movl a4@,d3");
                        asm ("cmpl #0x80000000,d2; jeq lmissing");
                        asm ("cmpl #0x80000000,d3; jne lnot_missing");
                        asm ("lmissing: movl #0x8000000,d0; jra lexit");
```

```
                asm ("lnot_missing: mulsl d2,d2:d3");
                asm ("addl d3,d1; addxl d2,d0");
                        asm ("addl a0, a4; addl d6, a5");
                        asm ("subql #1,d4; jgt lloop2");
                asm ("addl a1, a4; addl d7, a5");
                asm ("subql #1,d5; jgt lloop1");
        asm ("movl a6@(40),d2; jlt 9f");
        asm ("tstl d0; jlt 5f");
        asm ("jne 2f; movl d1,d0; jlt 2f; asll d2,d0; jvc lexit");
        asm ("2: movl #0x7FFFFFFF,d0; jra lexit");
        asm ("5: cmpl #-1,d0; jne 3f; movl d1,d0; jge 3f");
        asm ("asll d2,d0; jvc lexit");
        asm ("3: movl #-0x7FFFFFFF,d0; jra lexit");
        asm ("9: addl #32,d2; jgt 5f");
        asm ("movl d1,d0; negl d2; asrl d2,d0; jra lexit");
        asm ("5: movl a6@(40),d3; negl d3; subql #1,d3");
        asm ("movl d0,d4; asrl d3,d4; jgt 2b; jeq 1f");
        asm ("cmpl #-1,d4; jne 3b; 1:;");
        asm ("lsll d2,d0; addql #1,d3; lsrl d3,d1; addl d1,d0");
        asm ("lexit: movl sp@+,d2; movl sp@+,d3;");
#       else
        yincrement1 >>= 2;
        xincrement1 >>= 2;
        yincrement2 >>= 2;
        xincrement2 >>= 2;
        {       /* huge */ int sum = 0;
                while (ysize--) {
                        int xn = xsize;
                        while (xn --) {
                                if (sat_lmissing (* base1)
                                        || sat_lmissing (* base2))
                                        return (SAT_IMISSING);
                                sum += /* (huge) */ * base1 * * base2;
                                base1 += xincrement1;
                                base2 += xincrement2; }
                        base1 += yincrement1;
                        base2 += yincrement2; }
                if (shift > 0 && (sum >= (1L << (63 - shift))))
                        return (SAT_LMAXIMUM);
                else if (shift > 0 && (sum < (-1L << (63 - shift))))
                        return (SAT_IMINIMUM);
                else return ((long) (sum << (shift - 32))); }
#       endif
#       endif
        }

/*      Assembly language function to take the scalar product          */
/*      of two 2 dimensional block floating arrays.                    */

        static
double sba_sproduct2_missing;

        double
sba_sproduct2 (base1, yincrement1, xincrement1,
        base2, yincrement2, xincrement2,
        ysize, xsize, exponent)
```



```
        register long * base1;          /* r11                          */
        register int yincrement1;       /* r10                          */
        register int xincrement1;       /* r9                           */
        register long * base2;          /* r8                           */
        register int yincrement2;       /* r7                           */
        register int xincrement2;       /* r6                           */
        int ysize;                      /* r5                           */
        int xsize;                      /* r4                           */
        {
        if (xsize == 0 || ysize == 0) return (0);
        yincrement1 <<= 2;
        xincrement1 <<= 2;
        yincrement1 -= xincrement1 * xsize;
        yincrement2 <<= 2;
        xincrement2 <<= 2;
        yincrement2 -= xincrement2 * xsize;
        sba_sproduct2_missing = SAT_DMISSING;
#       if SFE_VAX && ! SFE_LINT
                asm ("clrd r0");
                asm ("movl 28(ap), r5");
                asm ("loop1: movl 32(ap), r4");
                        asm ("loop2: movl (r11), r2; movl (r8), r3")
                                asm ("cmpl $0x80000000, r2; beql missing");
                                asm ("cmpl $0x80000000, r3; bneq not_missing");
                                asm ("missing:")
                                asm ("movd _sba_sproduct2_missing, r0")
                                asm ("brb exit")
                                asm ("not_missing: cvtld r3, -(sp)");
                                asm ("cvtld r2, r2")
                                asm ("muld2 (sp)+, r2");
                                asm ("addd2 r2, r0");
                                asm ("addl2 r6, r8; addl2 r9, r11");
                                asm ("sobgtr r4, loop2");
                        asm ("addl2 r7, r8; addl2 r10, r11");
                        asm ("sobgtr r5, loop1");
                asm ("pushl 36(ap); movd r0, -(sp); calls $3, _ldexp; exit:");
#       else
                yincrement1 >>= 2;
                xincrement1 >>= 2;
                yincrement2 >>= 2;
                xincrement2 >>= 2;
                {       double sum = 0;
                        while (ysize--) {
                                int xn = xsize;
                                while (xn --) {
                                        if (sat_lmissing (* base1)
                                                || sat_lmissing (* base2))
                                                return (sba_sproduct2_missing);
                                        sum += (double) * base1 * * base2;
                                        base1 += xincrement1;
                                        base2 += xincrement2; }
                                base1 += yincrement1;
                                base2 += yincrement2; }
                        return (ldexp (sum, exponent)); }
#               endif
        }
```

```
        double
sba_sproduct (array1, array2)
        register sar_array array1, array2;
        {
        register double sum = 0;
        register double value;
        sfe_assert (sar_similar (array2, array1),
                "(sba_sproduct - input arrays are not similar)");
        sfe_assert (array1->sar_etype == SOB_LONG, "\
(sba_sproduct - first input array does not have element type a-long)");
        sfe_assert (array2->sar_etype == SOB_LONG, "\
(sba_sproduct - second input array does not have element type a-long)");

        {       sar_for_2_matrices (array1, array2, long *) {
                value = sba_sproduct2 (
                        z1p, array1->sar_yincrement, array1->sar_xincrement,
                        z2p, array2->sar_yincrement, array2->sar_xincrement,
                        array1->sar_ysize, array1->sar_xsize,
                        array1->sar_exponent + array2->sar_exponent);
                if (sat_dmissing (value)) return (SAT_DMISSING);
                else sum += value; }}

        return (sum); }

sba_convolve (output, input, kernel)
        register sar_array output, input, kernel;
        {

        register long * oxp, * ixp;
        register int xn;
        register int shift;
        register long * oyp, * iyp;
        register int yn;

        sfe_assert (output->sar_etype == SOB_LONG, "\
(sba_convolve - output array does not have element type a-long)");
        sfe_assert (input->sar_etype == SOB_LONG, "\
(sba_convolve - input array does not have element type a-long)");
        sfe_assert (kernel->sar_etype == SOB_LONG, "\
(sba_convolve - kernel array does not have element type a-long)");
        sfe_assert (output->sar_xsize ==
                input->sar_xsize + 1 - kernel->sar_xsize, "\
(sba_convolve - output size != input size + 1 - kernel size for X dimension)");
        sfe_assert (output->sar_ysize ==
                input->sar_ysize + 1 - kernel->sar_ysize, "\
(sba_convolve - output size != input size + 1 - kernel size for Y dimension)");
        sfe_assert (sar_write (output), "\
(sba_convolve - output array is not writable)");

        shift = kernel->sar_exponent + input->sar_exponent
                - output->sar_exponent;
        sfe_assert (shift < 64, "\
(sba_convolve - output array exponent much too small relative to input exponent\
 + kernel exponent)");
```



```
        for (yn = 0, oyp = output->sar_lbase, iyp = input->sar_lbase;
                yn < output->sar_ysize;
                ++ yn, oyp += output->sar_yincrement,
                        iyp += input->sar_yincrement)
        for (xn = 0, oxp = oyp, ixp = iyp;
                xn < output->sar_xsize;
                ++ xn, oxp += output->sar_xincrement,
                        ixp += input->sar_xincrement)
                * oxp = sba_lsproduct2 (
                        kernel->sar_lbase,
                        kernel->sar_yincrement,
                        kernel->sar_xincrement,
                        ixp,
                        input->sar_yincrement,
                        input->sar_xincrement,
                        kernel->sar_ysize,
                        kernel->sar_xsize,
                        shift);
        return (0); }
```

```
#ifndef SBA_DEFS_H

#define SBA_DEFS_H

#ifndef SAR_DEFS_H
#       include <sar/sar_defs.h>
#       endif

#endif SBA_DEFS_H
```



```
(setq x (an-array has-sizes '(3 4) by-expression 'X))
(setq y (an-array has-sizes '(3 4) by-expression 'Y))
(setq z (an-array has-sizes '(3 4)))
(print-array (add-to-array-elements (copy-array z x) 5))
(print-array (multiply-array-elements-by (copy-array z x) 5))
(print-array (add-arrays z x y))
(print-array (subtract-arrays z x y))
(print-array (minimize-arrays z x y))
(print-array (maximize-arrays z x y))
(print-array (multiply-array-elements z x y))

(setq x (an-array has-sizes '(9) by-expression '(diff X 4)))
(setq z (an-array has-sizes '(9)))
(print-array (absolute-value-array-elements z x))
(print-array (log-array-elements z x))
(print-array (exponentiate-array-elements z x))
(print-array (square-root-array-elements z x))
(print-array (power-array-elements z x 0.5))
(print-array (sin-array-elements z x))
(print-array (cos-array-elements z x))
(print-array (arcsin-array-elements z x))
(print-array (arccos-array-elements z x))

(print-array (gaussian-kernel '(1.5 1.5)))
(print-array (del2g-kernel '(1 1)))
(print-array (dxg-kernel '(1 1)))

(setq ac (an-array has-sizes '(9) has-exponent 0))
(set-array-elements ac 1)
(print-array ac)

(setq mv (an-array has-sizes '(9 5)
                   by-value '(
                             (99 99   4  2  3  4  8  9  6)
                             (99 99  99  3  4  6  7 99  9)
                             ( 1 99  99 24  5 99 19  8  5)
                             ( 3  3  99  3  5  8  7  4  8)
                             ( 2  4   3  4  6  7  8  6  7))))

(print-array mv)
(print-array (setq markmv (mark-missing-of mv '(-98 98) '(-10 10))))
(print-array (shrink-missing-of markmv 4))
(print-array (setq shrinkmv (shrink-missing-of markmv 2)))
(print-array (expand-missing-of shrinkmv markmv nil nil 1))
(print-array (expand-missing-of shrinkmv markmv nil nil 2))
(print-array (expand-missing-of shrinkmv markmv '(2 2) 1 1))
(print-array (expand-missing-of shrinkmv markmv))

(setq dx (an-array has-sizes '(3 3) by-value '((-1 0 1) (-1 0 1) (-1 0 1))))
(setq dy (an-array has-sizes '(3 3) by-value '((-1 -1 -1) (0 0 0) (1 1 1))))

(scalar-product dx dx)
(scalar-product dx dy)
(scalar-product dy dy)

(setq ix (an-array has-sizes '(8 8) by-expression 'X))
```

```
(print-array (convolution-of ix dx))
(print-array (convolution-of ix dy))

(setq ixy (an-array has-sizes '(8 8) by-expression '(plus X Y)))
(print-array (convolution-of ixy dx))
(print-array (convolution-of ixy dy))

(print-array (cached-dither 2))
(print-array (cached-dither 2))
(print-array (cached-dither *default-dither-size*))

(print-array (sum-filter (copy-of-array ixy) Y-dimension 3))
(print-array (accumulate-filter (copy-of-array ixy) Y-dimension))
(print-array (maximum-filter (copy-of-array ixy) Y-dimension 3))
(print-array (minimum-filter (copy-of-array ixy) Y-dimension 3))
(print-array (derivative-filter (copy-of-array ixy) Y-dimension 3))
(print-array (interpolation-filter (copy-of-array ixy) Y-dimension 1.5 0.5))

(print-array (contrast-of ixy '(1 1) 1 0))
(print-array (contrast-of ixy '(1 1) 0 1))
(print-array (contrast-of ixy '(1 1)))

(print-array (interpolation-of (an-array has-sizes '(9 9)
                                          by-expression '(plus X Y))
                               '(7 11)))
```



```
;       .En ( dither " 'x_size)" "[LISP Function]"
;       .Cn ( cached-dither " 'x_size)"
;       .Cn "" *default-dither-size* ""
;       .Pa WHERE
;       X size is a power of two.
;       .Pa RETURNS
;       .i Dither
;       and
;       .i cached-dither
;       return the Dither Matrix of the given size: the matrices
;       D\*(n\*) of the paper Jarvis, J.F., Judice, C.N.,
;       and Ninke, W.H.,
;       .i "A Survey of Tehniques for the Display of Continuous Tone Pictures"
;       .i "on Bilevel Displays" ,
;       Computer Graphics and Image Processing,
;       .b 5 ,
;       13-40 (1976), where n is the matrix size, x_size.
;       The matrix is square.
;
;       .i Cached-dither
;       remembers all dither matrices it has computed, saving them.  It returns
;       previously computed matrices without recomputing them.
;       Only those of size <= 64 are saved at the moment.
;
;       .i *default-dither-size*
;       is a global variable set to a default value suitable for
;       the size parameter.  It itself defaults to 8.

(defvar *default-dither-size* 8)

(cload '(_sba_dither) 'sba/sba_cdither)

(defun dither (size &aux the-result)
  (assert (and (fixp size)
               (>& size 0))
          '(size argument is not a fixnum > 0))
  (setq the-result (an-array has-sizes (list size size)))
  (ccheck (_sba_dither (allocate-array the-result)))
  the-result)

; Vector cache whose i'th entry has the name of a symbol whose value is
; (dither i), or else the i'th entry is nil if the entry has never
; before been needed.  The symbol is needed to prevent garbage collection.
(defvar *cached-dither* (new-vector 65))
;
(defun cached-dither (size &aux the-symbol)
  (assert (and (fixp size)
               (>& size 0))
          '(size argument is not a fixnum > 0))
  (cond
   ((>& size 64) (dither size))
   ((setq the-symbol (vref *cached-dither* size))
    (symeval the-symbol))
   (t
    (setq the-symbol (gentemp))
    (set the-symbol (dither size))
```

```
    (setf (vref *cached-dither* size) the-symbol)
    (symeval the-symbol))))
```



```
(cload '(_sba_aadd _sba_add _sba_asubtract _sba_amultiply _sba_multiply
                   _sba_amaximize _sba_maximize
                   _sba_aminimize _sba_minimize
                   _sba_set _sba_aabsolute
                   _sba_alog _sba_aexp _sba_asqrt _sba_apower
                   _sba_asin _sba_acos _sba_aarcsin _sba_aarccos)
  'sba/sba_celem)

;       .En ( set-array-elements " 'lar_array 'n_value)" "[LISP Function]"
;       .Cn ( set-array-elements " 'lar_array \fInil\fP)" "[LISP Function]"
;       .Pa RETURNS
;       Lar_array after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets all elements of lar_array to n_value or \fInil\fP.

(defun set-array-elements (the-array the-value)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (or (not the-value) (numberp the-value))
          '(value argument is not number or nil))
  (ccheck (_sba_set (allocate-array the-array)
                    (if the-value (float the-value) _SAT_DMISSING)))
  the-array)

;       .En ( add-to-array-elements " 'lar_array 'n_addend)" "[LISP Function]"
;       .Pa RETURNS
;       Lar_array after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Adds n_addend to all elements of lar_array.
;       .Pa BUG
;       The addition is done using modulo arithmetic in the event of
;       overflow.

(defun add-to-array-elements (the-array the-addend)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (numberp the-addend)
          '(addend argument is not number))
  (ccheck (_sba_add (allocate-array the-array)
                    (float the-addend)))
  the-array)

;       .En ( multiply-array-elements-by " 'lar_array 'n_multiplier)" \
;       "[LISP Function]"
;       .Pa RETURNS
;       Lar_array after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Multiplies each element of lar_array by n_multiplier.
;       .Pa BUG
;       The multiplication is done using modulo arithmetic in event of
;       overflow.

(defun multiply-array-elements-by (the-array the-multiplier)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (numberp the-multiplier)
```

```
          '(multiplier argument is not number))
  (ccheck (_sba_multiply (allocate-array the-array)
                         (float the-multiplier)))
  the-array)

;       .En ( maximize-array-elements-with " 'lar_array 'n_number)" \
;       "[LISP Function]"
;       .Pa RETURNS
;       Lar_array after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Each element of the array is replaced by the maximum of the element
;       value and n_number.  In other words, elements with values below
;       n_number are replaced by n_number.

(defun maximize-array-elements-with (the-array the-number)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (numberp the-number)
          '(number argument is not number))
  (ccheck (_sba_maximize (allocate-array the-array)
                         (float the-number)))
  the-array)

;       .En ( minimize-array-elements-with " 'lar_array 'n_number)" \
;       "[LISP Function]"
;       .Pa RETURNS
;       Lar_array after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Each element of the array is replaced by the minimum of the element
;       value and n_number.  In other words, elements with values above
;       n_number are replaced by n_number.

(defun minimize-array-elements-with (the-array the-number)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (numberp the-number)
          '(number argument is not number))
  (ccheck (_sba_minimize (allocate-array the-array)
                         (float the-number)))
  the-array)

;       .En ( add-arrays " 'lar_output 'lar_input-1 ['lar_input-2])" \
;       "[LISP Function]"
;       .Pa WHERE
;       The arrays are similar and have the same exponent, and
;       lar_input-2 defaults to lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Adds each element of lar_input-1 to the corresponding element
;       of lar_input-2 and stores the result in the corresponding
;       element of lar_output.
;       .Pa BUG
;       The addition is done using modulo arithmetic in event of overflow.

(defun add-arrays (the-output the-first-input
```



```
                            &optional (the-second-input the-output))
  (assert (object-is an-array the-first-input)
          '(first input array argument is not an-array))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-second-input)
          '(second input array argument is not an-array))
  (ccheck (_sba_aadd (allocate-array the-output)
                     (allocate-array the-first-input)
                     (allocate-array the-second-input)))
  the-output)

;       .En ( subtract-arrays " 'lar_output ['lar_input-1] 'lar_input-2)" \
;       "[LISP Function]"
;       .Pa WHERE
;       The arrays are similar and have the same exponent,
;       and lar_input-1 defaults to lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Subtracts each element of lar_input-2 from the corresponding element
;       of lar_input-1 and stores the result in the corresponding
;       element of lar_output.
;       .Pa BUG
;       The subtraction is done using modulo arithmetic in event of overflow.

(defun subtract-arrays (the-output the-first-input &optional the-second-input)
  (assert (object-is an-array the-first-input)
          '(first input array argument is not an-array))
  (assert (or (not the-second-input)
              (object-is an-array the-second-input))
          '(second input array argument is not an-array))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (cond
   ((not the-second-input)
    (setq the-second-input the-first-input)
    (setq the-first-input the-output)))
  (ccheck (_sba_asubtract (allocate-array the-output)
                          (allocate-array the-first-input)
                          (allocate-array the-second-input)))
  the-output)

;       .En ( maximize-arrays " 'lar_output 'lar_input-1 ['lar_input-2])" \
;       "[LISP Function]"
;       .Pa WHERE
;       The arrays are similar and have the same exponent, and
;       lar_input-2 defaults to lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Takes the maximum of each
;       element of lar_input-1 with the corresponding element
;       of lar_input-2 and stores the result in the corresponding
;       element of lar_output.
```

```
(defun maximize-arrays (the-output the-first-input
                                   &optional (the-second-input the-output))
  (assert (object-is an-array the-first-input)
          '(first input array argument is not an-array))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-second-input)
          '(second input array argument is not an-array))
  (ccheck (_sba_amaximize (allocate-array the-output)
                          (allocate-array the-first-input)
                          (allocate-array the-second-input)))
  the-output)

;       .En ( minimize-arrays " 'lar_output 'lar_input-1 ['lar_input-2])" \
;       "[LISP Function]"
;       .Pa WHERE
;       The arrays are similar and have the same exponent, and
;       lar_input-2 defaults to lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been modified.
;       .Pa SIDE\ EFFECT
;       Takes the minimum of
;       each element of lar_input-1 with the corresponding element
;       of lar_input-2 and stores the result in the corresponding
;       element of lar_output.

(defun minimize-arrays (the-output the-first-input
                                   &optional (the-second-input the-output))
  (assert (object-is an-array the-first-input)
          '(first input array argument is not an-array))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-second-input)
          '(second input array argument is not an-array))
  (ccheck (_sba_aminimize (allocate-array the-output)
                          (allocate-array the-first-input)
                          (allocate-array the-second-input)))
  the-output)

;       .En ( multiply-array-elements " \kI'lar_output" "[LISP Function]"
;       .Xn "'lar_input-1 ['lar_input-2])"
;       .Pa WHERE
;       All three arrays are similar and lar_input-2 defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the product of the corresponding
;       elements in the two inputs.  The three arrays may have different
;       exponents.
;       .Pa BUG
;       The multiplication is done using modulo arithmetic in event of
;       overflow.

(defun multiply-array-elements (the-output the-first-input
                                           &optional
```



```
                                           (the-second-input the-output))
  (assert (object-is an-array the-first-input)
          '(first input array argument is not an-array))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-second-input)
          '(second input array argument is not an-array))
  (ccheck (_sba_amultiply (allocate-array the-output)
                          (allocate-array the-first-input)
                          (allocate-array the-second-input)))
  the-output)

;       .En ( absolute-value-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the absolute value
;       of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun absolute-value-array-elements (the-output
                                      &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_aabsolute (allocate-array the-output)
                          (allocate-array the-input)))
  the-output)

;       .En ( log-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the logrithm of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun log-array-elements (the-output &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_alog (allocate-array the-output)
                     (allocate-array the-input)))
  the-output)
```

```
;       .En ( exponentiate-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the exponential function
;       of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun exponentiate-array-elements (the-output
                                    &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_aexp (allocate-array the-output)
                     (allocate-array the-input)))
  the-output)

;       .En ( square-root-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the square root of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun square-root-array-elements (the-output &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_asqrt (allocate-array the-output)
                      (allocate-array the-input)))
  the-output)

;       .En ( power-array-elements " 'lar_output ['lar_input] 'n_exponent)" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the n_exponent power
;       of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.
```



```
(defun power-array-elements (the-output the-input &optional the-exponent)
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (cond
   ((not the-exponent)
    (setq the-exponent the-input)
    (setq the-input the-output)))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (assert (numberp the-exponent)
          '(exponent argument is not a number))
  (ccheck (_sba_apower (allocate-array the-output)
                       (allocate-array the-input)
                       (float the-exponent)))
  the-output)

;       .En ( sin-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the sine of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun sin-array-elements (the-output &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_asin (allocate-array the-output)
                     (allocate-array the-input)))
  the-output)

;       .En ( cos-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the cosine of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun cos-array-elements (the-output &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_acos (allocate-array the-output)
```

```
                    (allocate-array the-input)))
  the-output)

;       .En ( arcsin-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the arc sine of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun arcsin-array-elements (the-output &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_aarcsin (allocate-array the-output)
                        (allocate-array the-input)))
  the-output)

;       .En ( arccos-array-elements " 'lar_output ['lar_input])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Both arrays are similar and lar_input defaults to
;       lar_output.
;       .Pa RETURNS
;       Lar_output after its elements have been set.
;       .Pa SIDE\ EFFECT
;       Sets each element in lar_output to the arc cosine of the corresponding
;       element in lar_input.  The two arrays may have different
;       exponents.

(defun arccos-array-elements (the-output &optional (the-input the-output))
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (ccheck (_sba_aarccos (allocate-array the-output)
                        (allocate-array the-input)))
  the-output)
```



```
(cload '(_sba_dfilter _sba_hfilter _sba_lfilter _sba_sfilter _sba_afilter
                      _sba_ifilter _sba_contrast)
  'sba/sba_cfilter)

;       .En ( derivative-filter " 'lar_array 'x_dimension 'x_width)" \
;       "[LISP Function]"
;       .Pa RETURNS
;       A slice of lar_array is returned after the lar_array elements are
;       modified so that
;       the slice holds the desired result.
;       The size of the given dimension for the slice is x_width-1
;       less than the size of that dimension for lar_array,
;       and the slice origin is 0
;       for that dimension.  Other dimensions are not affected.
;       .Pa SIDE\ EFFECT
;       Applies a derivative filter of the given x_width for the given
;       x_dimension of the given lar_array.
;       The derivative filter forms the sum of the terms -
;
;       (6 / (x_width ** 3 - x_width)) * (- x_width + 1 + 2 * i) * x (i)
;
;       for i = 0, 1, ..., x_width - 1.
;       The normalization constant is chosen so that if x(i) = i the result
;       will be 1.
;       The output for subscript j in the given dimension
;       is computed by letting x(i) equal the input for subscript j+i.
;       .Pa NOTE
;       The lar_array elements that are not in the returned slice are
;       modified in undefined ways.
;       .Pa BUGS
;       Missing values are not handled.
;       Overflow is handled by doing modulo arithmetic.

(defun derivative-filter (the-array the-dimension the-width &aux the-result)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (fixp the-dimension)
          '(dimension argument is not a fixnum))
  (assert (fixp the-width)
          '(width argument is not a fixnum))
  (setq the-result (slice-of-array the-array))
  (ccheck (_sba_dfilter (allocate-array the-result)
                        the-dimension the-width))
  the-result)

;       .En ( maximum-filter " 'lar_array 'x_dimension 'x_width)" \
;       "[LISP Function]"
;       .Pa RETURNS
;       A slice of lar_array is returned after the lar_array elements are
;       modified so that
;       the slice holds the desired result.
;       The size of the given dimension for the slice is x_width-1
;       less than the size of that dimension for lar_array,
;       and the slice origin is 0
;       for that dimension.  Other dimensions are not affected.
;       .Pa SIDE\ EFFECT
```

```
;       Applies a filter which forms the maximum of the last x_width
;       input values for the given dimension of the given lar_array.
;       Thus the output for subscript j in the given dimension
;       is the maximum of the input for subscripts j, j+1, ..., j+x_width-1.
;       .Pa NOTE
;       The lar_array elements that are not in the returned slice are
;       modified in undefined ways.
;       .Pa BUG
;       Missing values are not handled.

(defun maximum-filter (the-array the-dimension the-width &aux the-result)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (fixp the-dimension)
          '(dimension argument is not a fixnum))
  (assert (fixp the-width)
          '(width argument is not a fixnum))
  (setq the-result (slice-of-array the-array))
  (ccheck (_sba_hfilter (allocate-array the-result)
                        the-dimension the-width))
  the-result)

;       .En ( local-maxima-of " \kI'ar_input '(x_size ...))" \
;       "[LISP Function]"
;       .Pa WHERE
;       The x_size ... are non-negative.
;       .Pa RETURNS
;       An output array, lar_output, whose elements are
;       the maxima of the elements of a rectangular box centered
;       at the corresponding point of lar_input.
;       The sizes of the box are (2*x_size+1,\ ...).
;       The maximum is computed successively along each dimension of ar_input
;       by calling the function maximum-filter for that dimension.
;       The dimensions of lar_output are made identical to the dimensions
;       of ar_input, by first expanding lar_input by appropriate amounts.
;       This, and the conversion of element type to a-long, are accomplished
;       by passing the input array to the function prepare-array.
;       The input array is returned only when it has type a-long and the
;       x_size\ ... arguments are all zero.

(defun local-maxima-of (array ranges)
  (assert (object-is an-array array)
          '(input array argument is not an-array))
  (let ((nranges (check-list ranges #'(lambda (x) (and (fixp x)
                                                    (not (<& x 0)))))))
    (assert (and (>& nranges -1 )
                 (not (>& nranges _SAR_MDIMENSIONS)))
          `(ranges argument is not a list of 0 to ,_SAR_MDIMENSIONS
                non-negative fixnums))
    )
  (let* ((double-ranges (mapcar #'(lambda (x) (* 2 x)) ranges))
         (the-maximum-array
          (prepare-array array double-ranges a-long must-copy)))
    (do ((rest-ranges ranges (cdr rest-ranges))
         (dimension 0 (1+ dimension))
         (range))
```



```
        ((null rest-ranges) the-maximum-array)
        (if (>& (setq range (first rest-ranges)) 0)
            (setq the-maximum-array
                  (maximum-filter the-maximum-array
                                  dimension
                                  (1+ (* 2 range))
                                  )
                  )
        )
      )
    )
)

;       .En ( minimum-filter " 'lar_array 'x_dimension 'x_width)" \
;       "[LISP Function]"
;       .Pa RETURNS
;       A slice of lar_array is returned after the lar_array elements are
;       modified so that
;       the slice holds the desired result.
;       The size of the given dimension for the slice is x_width-1
;       less than the size of that dimension for lar_array,
;       and the slice origin is 0
;       for that dimension.  Other dimensions are not affected.
;       .Pa SIDE\ EFFECT
;       Applies a filter which forms the minimum of the last x_width
;       input values for the given dimension of the given lar_array.
;       Thus the output for subscript j in the given dimension
;       is the minimum of the input for subscripts j, j+1, ..., j+x_width-1.
;       .Pa NOTE
;       The lar_array elements that are not in the returned slice are
;       modified in undefined ways.
;       .Pa BUG
;       Missing values are not handled.

(defun minimum-filter (the-array the-dimension the-width &aux the-result)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (fixp the-dimension)
          '(dimension argument is not a fixnum))
  (assert (fixp the-width)
          '(width argument is not a fixnum))
  (setq the-result (slice-of-array the-array))
  (ccheck (_sba_lfilter (allocate-array the-result)
                        the-dimension the-width))
  the-result)

;       .En ( local-minima-of " \kI'ar_input '(x_size ...))" \
;       "[LISP Function]"
;       .Pa WHERE
;       The x_size ... are non-negative.
;       .Pa RETURNS
;       An output array, lar_output, whose elements are
;       the minima of the elements of a rectangular box centered
;       at the corresponding point of lar_input.
```

```
;       The sizes of the box are (2*x_size+1,\ ...).
;       The minimum is computed successively along each dimension of ar_input
;       by calling the function minimum-filter for that dimension.
;       The dimensions of lar_output are made identical to the dimensions
;       of ar_input, by first expanding lar_input by appropriate amounts.
;       This, and the conversion of element type to a-long, are accomplished
;       by passing the input array to the function prepare-array.
;       The input array itself is returned only when it has type a-long and the
;       x_size\ ... arguments are all zero.

(defun local-minima-of (array ranges)
  (assert (object-is an-array array)
          '(input array argument is not an-array))
  (let ((nranges (check-list ranges  #'(lambda (x) (and (fixp x)
                                                       (not (<& x 0)))))))
    (assert (and (>& nranges -1 )
                 (not (>& nranges _SAR_MDIMENSIONS)))
            `(ranges argument is not a list of 0 to ,_SAR_MDIMENSIONS
                     non-negative fixnums))
    )
  (let* ((double-ranges (mapcar #'(lambda (x) (* 2 x)) ranges))
         (the-minimum-array
          (prepare-array array double-ranges a-long)))
    (do ((rest-ranges ranges (cdr rest-ranges))
         (dimension 0 (1+ dimension))
         (range))
        ((null rest-ranges) the-minimum-array)
        (if (>& (setq range (first rest-ranges)) 0)
            (setq the-minimum-array
                  (minimum-filter the-minimum-array
                                  dimension
                                  (1+ (* 2 range))
                                  )
                  )
            )
        )
    )
)


;       .En ( sum-filter " 'lar_array 'x_dimension 'x_width)" \
;       "[LISP Function]"
;       .Pa RETURNS
;       A slice of lar_array is returned after the lar_array elements are
;       modified so that
;       the slice holds the desired result.
;       The size of the given dimension for the slice is x_width-1
;       less than the size of that dimension for lar_array,
;       and the slice origin is 0
;       for that dimension.  Other dimensions are not affected.
;       .Pa SIDE\ EFFECT
;       Applies a filter that computes the mean of the last x_width
;       input values for the given dimension of the given lar_array.
;       Thus the output for subscript j in the given dimension
;       is the mean of the input for subscripts j, j+1, ..., j+x_width-1.
;       .Pa NOTE
```



```
;       The lar_array elements that are not in the returned slice are
;       modified in undefined ways.
;       .Pa BUGS
;       Missing values are not handled.
;       Overflow is handled by doing modulo arithmetic.

(defun sum-filter (the-array the-dimension the-width &aux the-result)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (fixp the-dimension)
          '(dimension argument is not a fixnum))
  (assert (fixp the-width)
          '(width argument is not a fixnum))
  (setq the-result (slice-of-array the-array))
  (ccheck (_sba_sfilter (allocate-array the-result)
                        the-dimension the-width))
  the-result)

;       .En ( contrast-of " \kI'ar_input '(x_width ...)" "[LISP Function]"
;       .Xn "['n_background ['n_center]])"
;       .Pa WHERE
;       The x_width are non-negative integers and
;       both n_background and n_center default to 1.0.
;       .Pa RETURNS
;       An output array, lar_output, whose elements are
;       the "convolution" in the sense of the
;       .i convolve
;       function of ar_input and a kernel with a special form which is
;       parametrized by (x_width\ ...), n_background, and n_center.
;       The kernel is of size-
;
;       (2 * x_width + 1, ...)
;
;       and has for
;       all its elements except the center element the value-
;
;       - n_background / ((2 * x_width + 1) * ...).
;
;       where the denominator is the area of the kernel.
;       The center element has the value-
;
;       n_center - n_background / ((2 * x_width + 1) * ...).
;
;       The effect is to output into each lar_output element (X, ...)
;       n_center times the ar_input element (X+x_width,\ ...)
;       minus n_background times the average of the ar_input elements
;       in a (2*x_width+1\ ...) rectangular
;       box centered on the ar_input element just indicated.
;       .Pa BUGS
;       Missing values are not handled.
;       Overflow is handled by doing modulo arithmetic.

(defun contrast-of (the-input the-widths
                              &optional (the-background 1.0) (the-center 1.0)
                              &aux the-output the-size-differences)
  (assert (object-is an-array the-input)
```

```
          '(input array argument is not an-array))
  (assert (<& -1 (check-list the-widths
                             #'(lambda (x) (and (fixp x) (not (<& x 0))))))
          '(widths argument is not a list of non-negative fixnums))
  (assert (numberp the-background)
          '(background argument is not a number))
  (assert (numberp the-center)
          '(center argument is not a number))
  (setq the-size-differences (mapcar #'(lambda (x) (product 2 x))
                                     the-widths))
  (setq the-input (prepare-array the-input the-size-differences a-long))
  (setq the-output (copy-of-array the-input))

  (ccheck (_sba_contrast (allocate-array the-output)
                         (allocate-array (slice-of-array the-input))
                         the-widths
                         (float the-background) (float the-center)))
  the-output)

;        .En ( accumulate-filter " 'lar_array 'x_dimension)" \
;        "[LISP Function]"
;        .Pa RETURNS
;        Lar_array is returned after the lar_array elements are
;        modified to hold the desired result.
;        .Pa SIDE\ EFFECT
;        Applies a filter that computes the sum of all values with
;        equal or lower subscripts
;        for the given dimension of the given lar_array.
;        Thus the output for subscript j in the given dimension
;        is the sum of the input for subscripts 0, 1, ..., j.
;        .Pa BUGS
;        Overflow is handled by doing modulo arithmetic.

(defun accumulate-filter (the-array the-dimension)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (fixp the-dimension)
          '(dimension argument is not a fixnum))
  (ccheck (_sba_afilter (allocate-array the-array)
                        the-dimension))
  the-array)

;        .En ( interpolation-filter " \kI'lar_array 'x_dimension 'n_factor" \
;        "[LISP Function]"
;        .Xn "['n_offset])"
;        .Pa RETURNS
;        A slice of lar_array is returned after the lar_array elements are
;        modified so that
;        the slice holds the desired result.
;        The size of the given dimension for the slice is
;        as large as possible subject to the conditions that for all
;        slice subscripts j
;
;        .nf
;        - epsilon \kI\(<= n_offset + j * n_factor
;        \h'\nIu'\(<= x_input-dimension-size - 1 + epsilon.
```



```
;
;        .fi
;        where epsilon = 2\*[-10\*] is included to compensate for rounding
;        errors.
;        The slice origin is 0
;        for that dimension.  Other dimensions are not affected.
;        .Pa SIDE\ EFFECT
;        Interpolates the input values for the given dimension so that
;        the output has the given size.  Linear interpolition is used.
;        Output subscript 0 has the value associated with input
;        subscript n_offset, and in general output subscript j has
;        the value associated with input subscript
;
;        n_offset + j * n_factor
;
;        Non-integer input subscripts given by this formula are handled by
;        linearly interpolating input elements with the next lower and higher
;        integer subscripts.  Input subscripts less than 0 by an amount less
;        than or equal to epsilon are treated as 0, and similarly input
;        subscripts larger than the maximum input subscript by an amount less
;        than or equal to epsilon are treated as the maximum input subscript.
;        .Pa NOTE
;        The lar_array elements that are not in the returned slice are
;        modified in undefined ways.

(defun interpolation-filter (the-array the-dimension the-factor
                                       &optional (the-offset 0)
                                       &aux the-result)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (fixp the-dimension)
          '(dimension argument is not a fixnum))
  (assert (numberp the-factor)
          '(factor argument is not a number))
  (assert (numberp the-offset)
          '(offset argument is not a number))
  (setq the-result (slice-of-array the-array))
  (ccheck (_sba_ifilter (allocate-array the-result)
                        the-dimension (float the-factor) (float the-offset)))
  the-result)

;        .En ( interpolation-of " \kI'ar_input '(x_size ...))" "[LISP Function]"
;        .Pa WHERE
;        The x_size are non-negative.
;        .Pa RETURNS
;        An output array, lar_output, whose elements are
;        the linear interpolation, in the sense of
;        the \fIinterpolation-filter\fP function, of the elements of ar_input.
;        The dimension sizes of lar_output are (x_size\ ...).
;        The expansion factors for
;        \fIinterpolation-filter\fP are automatically chosen to be positive
;        and to give the right lar_output sizes (and the offsets are chosen
;        to be zero).
;        .Pa BUG
;        Overflow is handled by doing modulo arithmetic.
```

```
(defun interpolation-of (the-input the-sizes &aux the-output l)
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (assert (<& -1 (setq l (check-list the-sizes
                                     #'(lambda (x) (and (fixp x)
                                                        (not (<& x 0)))))))
          '(sizes argument is not a list of non-negative fixnums))
  (assert (not (>& l _SAR_MDIMENSIONS))
          '(sizes argument is too long a list))
  (setq the-output
        (an-array has-sizes
                  (mapcar 'max
                          (copy-list the-sizes _SAR_MDIMENSIONS 1)
                          (has-sizes (an-array the-input)))
                  has-exponent (if (eq a-long (has-element-type
                                               (an-array the-input)))
                                   (has-exponent (an-array the-input))
                                   (if (setq l (log-bound-of-array the-input))
                                       (+ -24 l) -16))))
  (copy-array (slice-of-array the-output (has-sizes (an-array the-input)))
              the-input)
  (do ((the-output-sizes the-sizes (restl the-output-sizes))
       (output-size)
       (the-input-sizes (has-sizes (an-array the-input))
                        (restl the-input-sizes))
       (input-size)
       (factor)
       (dimension 0 (1+ dimension)))
      ((null the-output-sizes))
      (setq output-size (first the-output-sizes)
            input-size (first the-input-sizes))
      (setq factor (if (>& output-size 1)
                       (quotient (float (diff input-size 1))
                                 (diff output-size 1))
                       input-size))
      (setq the-output (interpolation-filter the-output dimension factor)))
  the-output)
```



```
(cload '(_sba_gkernel _sba_d2gkernel _sba_dxgkernel) 'sba/sba_ckernel)

;       .En "" *kernel-cutoff* "" "[LISP Global Variable]"
;       .Pa VALUE
;       A \fIflonum\fP, default 0.01.
;       The amount of a kernel that may be discarded in order
;       to make a kernel of infinite extent fit into a small finite array.
;       Measured as a fraction bounding the integral of the discarded part
;       of the kernel divided by the integral of the kernel over the
;       part of the space where the kernel has the same sign it does
;       in the discarded
;       part.  The measurement is generally made by using integration of the
;       continuous kernel function; not its desecrete representation.
;;
(defvar *kernel-cutoff* 0.01)

;       .En ( gaussian-kernel " \kI'(n_xwidth n_ywidth)" "[LISP Function]"
;       .Xn "[ '(n_xoffset n_yoffset)])"
;       .Pa WHERE
;       N_xoffset and n_yoffset are \(>= 0 and default to 0.5.
;       .Pa RETURNS
;       A block floating point array with
;       exponent -24 representing the kernel computed by the
;       Gaussian function.
;       The sizes of the X and Y dimensions are
;
;       2 * xsize + (\fIceiling\fP n_xoffset) + 1
;
;       and
;
;       2 * ysize + (\fIceiling\fP n_yoffset) + 1
;
;       where xsize and ysize are choosen as indicated below.
;       The value of the point with subscripts (X\ Y) is-
;
;       .nf
;       \h'\w'* 'u'(1 / (pi * n_xwidth * n_ywidth))
;       * (\fIexp\fP (- (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2))
;       .fi
;
;       where
;
;       n_X = X - xsize - n_xoffset
;
;       and
;
;       n_Y = Y - ysize - n_yoffset
;
;       Xsize and ysize are chosen to be large enough so that
;       the integral of the
;       continuous function over all points (X\ Y)
;       outside the kernel returned by \fIguassian-kernel\fP
;       is less than the value of the global variable
;       .i *kernel-cutoff* .
;       In computing these sizes, n_xoffset and n_yoffset are assumed
;       to be 0, as a worst case.
```

```
;
;       The normalization constant is chosen so that the integral of
;       the kernel would be 1 if it where a continuous function extending
;       to infinity.

(defun gaussian-kernel (widths &optional (offsets '(0.5 0.5))
                               &aux sizes s full-sizes origins the-result)
  (assert (=& 2 (check-list widths #'(lambda (x) (and (numberp x)
                                                       (greaterp x 0)))))
          '(widths argument is not a pair of numbers > 0))
  (assert (=& 2 (check-list offsets #'(lambda (x) (and (numberp x)
                                                        (not (lessp x 0))))))
          '(offsets argument is not a pair of numbers >= 0))

  ; The integral of the kernel function in the area outside
  ; a circle of radius s is exp (- s ** 2) when the widths are unity.
  ; We reduce s by 10% because we have a square instead of a circle.
  (setq s (product 0.9 (sqrt (minus (log *kernel-cutoff*)))))
  (setq sizes (mapcar #'(lambda (x) (ceiling (product s x)))
                      widths))
  (setq full-sizes (mapcar 'plus '(1 1) sizes sizes
                           (mapcar 'ceiling offsets)))
  (setq origins (mapcar 'plus sizes offsets))
  (setq the-result (an-array has-sizes full-sizes has-exponent -24))
  (ccheck (_sba_gkernel (allocate-array the-result)
                        (quotient 1.00 pi (first widths) (second widths))
                        (float (first widths)) (float (second widths))
                        (float (first origins)) (float (second origins))))
  the-result)

;       .En ( del2g-kernel " \kI'(n_xwidth n_ywidth)" "[LISP Function]"
;       .Xn "['(n_xoffset n_yoffset)])"
;       .Pa WHERE
;       N_xoffset and n_yoffset are \(>= 0 and default to 0.5.
;       .Pa RETURNS
;       A block floating point array with
;       exponent -24 representing the kernel computed as the minus of the
;       Laplacian operator applied to the Gaussian function.
;       The Laplacian operator is assumed to be scaled by n_xwidth
;       and n_ywidth, so actually the second derivative with
;       respect to x is multiplied by n_xwidth\ **\ 2, and the
;       second derivative with respect to y by n_ywidth\ **\ 2.
;       The sizes of the X and Y dimensions are
;
;       2 * xsize + (\fIceiling\fP n_xoffset) + 1
;
;       and
;
;       2 * ysize + (\fIceiling\fP n_yoffset) + 1
;
;       where xsize and ysize are choosen as indicated below.
;       The value of the point with subscripts (X\ Y) is-
;
;       .nf
;       \h'\w'* 'u'(1 / (pi * n_xwidth * n_ywidth))
;       * (1 - (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2)
```



```
;       * (\fIexp\fP (1 - (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2))
;       .fi
;
;       where
;
;       n_X = X - xsize - n_xoffset
;
;       and
;
;       n_Y = Y - ysize - n_yoffset.
;
;       This function is positive inside the ellipse
;
;       (n_X / n_xwidth) ** 2 + (n_Y / n_ywidth) ** 2 \(<= 1,
;
;       zero on that ellipse, and negative outside the ellipse.
;       The total integral of the function is 0.  The function is
;       normalized to have the integral +1 inside the ellipse and
;       the integral -1 outside the ellipse, where for these purposes
;       the function is assumed to be continuous and extend to infinity.
;
;       Xsize and ysize are chosen to be large enough so that
;       the integral of the
;       continuous function over all points (X\ Y)
;       outside the kernel returned by \fIdel2g-kernel\fP
;       is less than the value of the global variable
;       .i *kernel-cutoff* .
;       In computing these sizes, n_xoffset and n_yoffset are assumed
;       to be 0, as a worst case.

(defun del2g-kernel (widths &optional (offsets '(0.5 0.5))
                            &aux sizes s full-sizes origins the-result
                            (log-cutoff (log *kernel-cutoff*)))
  (assert (=& 2 (check-list widths #'(lambda (x) (and (numberp x)
                                                       (greaterp x 0)))))
          '(widths argument is not a pair of numbers > 0))
  (assert (=& 2 (check-list offsets #'(lambda (x) (and (numberp x)
                                                        (not (lessp x 0))))))
          '(offsets argument is not a pair of numbers >= 0))

  ; The integral of the kernel function in the area outside
  ; a circle of radius s is - s ** 2 * exp (1 - s ** 2)
  ; when the widths are unity.
  ; We solve for s by Newton's method with always gives a value higher
  ; than the true one because the function
  ; s2 + (log *kernel-cutoff) - 1 - (log s2)
  ; (which is to be made 0 so that s = (sqrt s2)) is convex for s2 > 1.
  ; We reduce s by 10% because we have a square instead of a circle.
  (setq s
        (product 0.9
                 (sqrt (do ((s2 (diff 1 log-cutoff)
                                (plus s2 dels2))
                            (dels2 1.0))
                           ((lessp (abs dels2) 0.01) s2)
                           (setq dels2
                                 (quotient (diff (log s2) -1 log-cutoff s2)
```

```
                                        (diff 1 (quotient 1.0 s2))))))))
  (setq sizes (mapcar #'(lambda (x) (ceiling (product s x)))
                       widths))
  (setq full-sizes (mapcar 'plus '(1 1) sizes sizes
                            (mapcar 'ceiling offsets)))
  (setq origins (mapcar 'plus sizes offsets))
  (setq the-result (an-array has-sizes full-sizes has-exponent -24))
  (ccheck (_sba_d2gkernel (allocate-array the-result)
                           (quotient 1.00 pi (first widths) (second widths))
                           (float (first widths)) (float (second widths))
                           (float (first origins)) (float (second origins))))
  the-result)

;       .En ( dxg-kernel " \kI'(n_xwidth n_ywidth)" "[LISP Function]"
;       .Xn "['(n_xoffset n_yoffset)])"
;       .Pa WHERE
;       N_xoffset and n_yoffset are \(>= 0 and default to 0.5.
;       .Pa RETURNS
;       A block floating point array with
;       exponent -24 representing the kernel computed as the minus of the
;       x partial derivative applied to the Gaussian function.
;       The sizes of the X and Y dimensions are
;
;       2 * xsize + (\fIceiling\fP n_xoffset) + 1
;
;       and
;
;       2 * ysize + (\fIceiling\fP n_yoffset) + 1
;
;       where xsize and ysize are choosen as indicated below.
;       The value of the point with subscripts (X\ Y) is-
;
;       .nf
;       \h'\w'* 'u'(2 / ((\fIsqrt\fP pi) * n_xwidth * n_ywidth))
;       * (n_X / n_xwidth)
;       * (\fIexp\fP (- (n_X / n_xwidth) ** 2 - (n_Y / n_ywidth) ** 2))
;       .fi
;
;       where
;
;       n_X = X - xsize - n_xoffset
;
;       and
;
;       n_Y = Y - ysize - n_yoffset.
;
;       This function is positive for X > 0 and negative for X < 0.
;
;       The total integral of the function is 0.  The function is
;       normalized to have the integral +1 on the halfplane X > 0
;       and the integral -1 on the halfplane X < 0, where for these purposes
;       the function is assumed to be continuous and extend to infinity.
;
;       Xsize and ysize are chosen to be large enough so that
;       the integral of the
;       continuous function over all points (X\ Y)
```



```
;       outside the kernel returned by \fIdxg-kernel\fP
;       is less than the value of the global variable
;       .i *kernel-cutoff* .
;       In computing these sizes, n_xoffset and n_yoffset are assumed
;       to be 0, as a worst case.

(defun dxg-kernel (widths &optional (offsets '(0.5 0.5))
                          &aux sizes s full-sizes origins the-result)
  (assert (=& 2 (check-list widths #'(lambda (x) (and (numberp x)
                                                       (greaterp x 0)))))
          '(widths argument is not a pair of numbers > 0))
  (assert (=& 2 (check-list offsets #'(lambda (x) (and (numberp x)
                                                        (not (lessp x 0))))))
          '(offsets argument is not a pair of numbers >= 0))

  ; The integral of the kernel function in the area where X > s or Y > s
  ; is about exp (- s ** 2) when the widths are unity.
  (setq s (sqrt (minus (log *kernel-cutoff*))))
  (setq sizes (mapcar #'(lambda (x) (ceiling (product s x)))
                       widths))
  (setq full-sizes (mapcar 'plus '(1 1) sizes sizes
                            (mapcar 'ceiling offsets)))
  (setq origins (mapcar 'plus sizes offsets))
  (setq the-result (an-array has-sizes full-sizes has-exponent -24))
  (ccheck (_sba_dxgkernel (allocate-array the-result)
                           (quotient 2.00 sqrt-pi (first widths) (second widths))
                           (float (first widths)) (float (second widths))
                           (float (first origins)) (float (second origins))))
  the-result)
```

```
(declare (macros t))

(include sba_elem.l)
(include sba_miss.l)
(include sba_prod.l)
(include sba_dither.l)
(include sba_filter.l)
(include sba_kernel.l)


(putprop 'sba/sba_load t 'version)
```



```
(cload '(_sba_mmissing _sba_emissing _sba_smissing _sba_omissing
                       _sba_setmissing)
  'sba/sba_cmiss)

;        .En ( mark-missing " \kl'lar_output 'ar_input" "[LISP Function]"
;        .Xn "'(n_minimum n_maximum)"
;        .Xn "['(n_lower n_upper) ['(x_xsize x_ysize) ['x_count]]])"
;        .Pa WHERE
;        lar_output and lar_input have
;        the same exponent and the same dimension sizes except for the X and
;        Y dimensions.  The X and Y dimensions of lar_input are
;        respectively 2*x_xsize and 2*x_ysize larger than the
;        X and Y dimensions of lar_output.
;        N_minumum, n_maximum, x_xsize, x_ysize, x_count, n_lower, and n_upper
;        may be given as
;        \fInil\fP if they are missing.  X_xsize and X_ysize default to 1,
;        and x_count defaults to 2.
;        N_minimum and n_lower default to negative infinity,
;        while n_maximum and n_upper default to
;        positive infinity.
;        .Pa RETURNS
;        The number of missing values in lar_output.
;        .Pa SIDE\ EFFECT
;        Copies ar_input to lar_output replacing bad pixel values
;        by the missing value
;        .i nil .
;        Values outside the range from minimum to maximum, inclusive,
;        are bad.
;
;        If either n_lower or n_upper is given,
;        then a value is replaced by \fInil\fP unless the
;
;                (2*x_xsize+1 2*x_ysize+1)
;
;        box centered on the pixel contains
;        at least x_count pixels (not counting the center pixel)
;        in the range
;
;                (pixel-value+n_lower pixel-value+n_upper).
;
;        inclusive.

(defun mark-missing (the-output the-input the-absolute-range
                                &optional the-relative-range
                                the-sizes the-count
                                &aux minimum maximum lower upper xsize ysize)
  (assert (object-is an-array the-output)
          '(output array is not an-array))
  (assert (object-is an-array the-input)
          '(input array is not an-array))
  (assert (<& -1 (check-list the-absolute-range
                             #'(lambda (x) (or (not x) (numberp x))))
              3)
          '((minimum maximum) argument is not a list of <=2 numbers))
  (desetq (minimum maximum) the-absolute-range)
  (if minimum
```

```
     (assert (numberp minimum) '(minimum argument is not a number))
     (setq minimum _SAT_DMISSING))
 (if maximum
     (assert (numberp maximum) '(maximum argument is not a number))
     (setq maximum _SAT_DMISSING))
 (assert (<& -1 (check-list the-relative-range
                            #'(lambda (x) (or (not x) (numberp x))))
             3)
         '((lower upper) argument is not a list of <=2 numbers))
 (desetq (lower upper) the-relative-range)
 (if lower
     (assert (numberp lower) '(lower argument is not a number))
     (setq lower _SAT_DMISSING))
 (it upper
     (assert (numberp upper) '(upper argument is not a number))
     (setq upper _SAT_DMISSING))
 (assert (<& -1 (check-list the-sizes
                            #'(lambda (x) (or (not x) (fixp x))))
             3)
         '((xsize ysize) argument is not a list of <=2 fixnums))
 (desetq (xsize ysize) the-sizes)
 (if xsize
     (assert (and (fixp xsize) (>& xsize -1)) '(xsize is not a fixnum >= 0))
     (setq xsize 1))
 (if ysize
     (assert (and (fixp ysize) (>& ysize -1)) '(ysize is not a fixnum >= 0))
     (setq ysize 1))
 (if the-count
     (assert (and (fixp the-count) (>& the-count -1))
             '(the-count is not a fixnum >= 0))
     (setq the-count 2))
 (setq the-input (prepare-array the-input nil a-long))

 (ccheck (_sba_mmissing (allocate-array the-output)
                        (allocate-array the-input)
                        (float minimum) (float maximum)
                        (float lower) (float upper)
                        xsize ysize the-count)))

;       .En ( mark-missing-of " \kI'ar_input '(n_minimum n_maximum)" "\
;       [LISP Function]"
;       .Xn "['(n_lower n_upper) ['(x_xsize x_ysize) ['x_count]]])"
;       .Pa RETURNS
;       A new array lar_output which is computed by passing
;       it and the other parameters to the
;       .i mark-missing
;       function.  \fIPrepare-array\fP is applied to ar_input.

(defun mark-missing-of (the-input the-absolute-range
                                  &optional the-relative-range
                                  the-sizes the-count
                                  &aux the-output lower upper xsize ysize
                                  the-size-changes)
  (assert (object-is an-array the-input)
          '(input array is not an-array))
  (assert (<& -1 (check-list the-relative-range
```



```
                            #'(lambda (x) (or (not x) (numberp x))))
             3)
         '((lower upper) argument is not a list of <=2 numbers))
 (desetq (lower upper) the-relative-range)
 (assert (<& -1 (check-list the-sizes
                            #'(lambda (x) (or (not x) (fixp x))))
             3)
         '((xsize ysize) argument is not a list of <=2 fixnums))
 (desetq (xsize ysize) the-sizes)
 (if xsize
     (assert (and (fixp xsize) (>& xsize -1)) '(xsize is not a fixnum >= 0))
     (setq xsize 1))
 (if ysize
     (assert (and (fixp ysize) (>& ysize -1)) '(ysize is not a fixnum >= 0))
     (setq ysize 1))
 (setq the-size-changes (if (or lower upper) `(,(* 2 xsize) ,(* 2 ysize))))
 (setq the-input (prepare-array the-input the-size-changes a-long))
 (setq the-output (an-array has-sizes (mapcar #'diff
                                              (has-sizes (an-array the-input))
                                              (copy-list the-size-changes
                                                         _SAR_MDIMENSIONS 0))
                            has-exponent (has-exponent (an-array the-input))))
 (mark-missing the-output the-input
               the-absolute-range the-relative-range the-sizes the-count)
 the-output)

;       .En ( expand-missing " \kI'lar_output 'ar_input 'ar_original" \
;       "[LISP Function]"
;       .Xn "['(x_xsize x_ysize) ['x_count]])"
;       .Pa WHERE
;       Lar_output, ar_input, and ar_original all have
;       the same exponent and the same dimension sizes except for the X and
;       Y dimensions.  Lar_output and ar_original have the same X and Y
;       dimensions, while the X and Y dimensions of ar_input are
;       respectively 2*x_xsize and 2*x_ysize larger than the
;       X and Y dimensions of lar_output.
;
;       X_xsize, X_ysize, and x_count all default to 1.
;       .Pa RETURNS
;       The number of non-missing elements of ar_input replaced by
;       missing values in lar_output.
;       .Pa SIDE\ EFFECT
;       Copies the elements of ar_input to lar_output,
;       replacing some of the non-missing values by missing values.
;       The purpose of this is to expand a sky region (region of all missing
;       values in a laser radar image) which has been shrunk by
;       \fIshrink-missing\fP.  Ar_original is the original array before
;       it was shrunk by \fIshrink-missing\fP.
;
;       An element is replaced by a missing value when it is copied if
;       (1) the element is not missing in ar_input, (2) the element
;       is missing in ar_original, and (3) there are x_count or more
;       missing values in the box in ar_input of size
;
;               (2*x_xsize+1  2*x_ysize+1)
;
```

```
;       centered on the element, not counting the element itself.

(defun expand-missing (the-output the-input the-original
                                  &optional the-sizes the-count
                                  &aux xsize ysize)
  (assert (object-is an-array the-output)
          '(output array is not an-array))
  (assert (object-is an-array the-input)
          '(input array is not an-array))
  (assert (object-is an-array the-original)
          '(original array is not an-array))
  (assert (<& -1 (check-list the-sizes
                             #'(lambda (x) (or (not x) (fixp x))))
               3)
          '((xsize ysize) argument is not a list of <=2 fixnums))
  (desetq (xsize ysize) the-sizes)
  (if xsize
      (assert (and (fixp xsize) (>& xsize -1)) '(xsize is not a fixnum >= 0))
      (setq xsize 1))
  (if ysize
      (assert (and (fixp ysize) (>& ysize -1)) '(ysize is not a fixnum >= 0))
      (setq ysize 1))
  (if the-count
      (assert (fixp the-count) '(count argument is not a fixnum))
      (setq the-count 1))

  (setq the-input (prepare-array the-input nil a-long))
  (setq the-original (prepare-array the-original nil a-long))

  (ccheck (_sba_emissing (allocate-array the-output)
                         (allocate-array the-input)
                         (allocate-array the-original)
                         xsize ysize the-count)))

;       .En ( expand-missing-of " \kI'ar_input ar_original" "[LISP Function]"
;       .Xn "['(x_xsize x_ysize) ['x_count ['x_repeat ...]]])"
;       .Pa WHERE
;       Ar_input and ar_original should have the same dimension sizes.
;       X_repeat defaults to infinity, and may also be given as \fInil\fP to
;       specify infinity.
;
;       The part of the argument list beginning with '(x_xsize x_ysize)
;       may be repeated as long as any repetition begins
;       with a non-empty list value.
;       .Pa RETURNS
;       A new array lar_output which is computed by passing
;       ar_input and the other parameters through \fIexpand-missing\fP
;       x_repeat times.  As an optimization, the process stops when
;       no more replacement is possible, so x_repeat can
;       be a very large number.
;
;       If no replacement is done in computing lar_output, ar_input is
;       returned in place of lar_output.
;
;       \fIPrepare-array\fP is applied to ar_input.
;
```



```
;       If more than one set of size/count/repeat parameters is
;       given, these parameters are removed from the parameter list as
;       they are used, and the process is repeated with the last lar_output
;       substituted for ar_input.

(defun expand-missing-of (the-input the-original
                                    &rest other-parameters)
  (assert (object-is an-array the-input)
          '(input array is not an-array))
  (do ((the-sizes)
       (the-count nil nil)
       (the-repeat nil nil)
       (the-output the-input) (the-size-changes) (xsize) (ysize)
       (done nil (null other-parameters)))
      (done the-output)

      (setq the-sizes (pop other-parameters))
      (if (and other-parameters (not (dtpr (first other-parameters))))
          (setq the-count (pop other-parameters)))
      (if (and other-parameters (not (dtpr (first other-parameters))))
          (setq the-repeat (pop other-parameters)))
      (assert (<& -1 (check-list the-sizes
                                 #'(lambda (x) (or (not x) (fixp x))))
                   3)
              '((xsize ysize) argument is not a list of <=2 fixnums))
      (desetq (xsize ysize) the-sizes)
      (if xsize
          (assert (and (fixp xsize) (>& xsize -1))
                  `(xsize argument ,xsize is not a fixnum >= 0))
          (setq xsize 1))
      (if ysize
          (assert (and (fixp ysize) (>& ysize -1))
                  `(ysize argument ,ysize is not a fixnum >= 0))
          (setq ysize 1))
      (setq the-size-changes `(,(* 2 xsize) ,(* 2 ysize)))
      (assert (or (not the-repeat)
                  (and (fixp the-repeat) (>& the-repeat -1)))
              `(repeat argument ,the-repeat is not a fixnum >= 0))

      (do ((r the-repeat (if r (1- r)))
           (the-current-input the-output the-output)
           (the-prepared-input))
          ((and r (=& 0 r)))
          (setq the-prepared-input
                (prepare-array the-current-input the-size-changes a-long))
          (setq the-output
                (an-array has-sizes (mapcar #'diff
                                            (has-sizes
                                             (an-array the-prepared-input))
                                            (copy-list the-size-changes
                                                       _SAR_MDIMENSIONS 0))
                          has-exponent (has-exponent
                                        (an-array the-prepared-input))))
          (cond
           ((=& 0 (expand-missing the-output the-prepared-input
                                  the-original
```

```
                                the-sizes the-count))
         (setq the-output the-current-input)
         (return))))))

;       .En ( shrink-missing " \kI'lar_output 'ar_input ['x_count])" \
;       "[LISP Function]"
;       .Pa WHERE
;       Lar_output and ar_input have
;       the same exponent and the same dimension sizes except for the X and
;       Y dimensions.  The X and Y dimensions of ar_input are
;       larger by 2 than the X and Y dimensions of lar_output.
;       X_count defaults to 2 and must be >= 2.
;       .Pa RETURNS
;       The number of missing values left in lar_output.
;       .Pa SIDE\ EFFECT
;       Copies the elements of ar_input to lar_output,
;       replacing some of the
;       missing values by estimates.  In general the output element
;       equals the input element unless the input value is missing
;       and has at least x_count non-missing 2-dimensional 8-neighbors.
;
;       Missing values are replaced by values obtained through inspection
;       of the 2-dimensional 8-neighbors of the missing value.
;
;       Given a pair of non-missing neighbors, the missing
;       value is replaced by their average.
;       Generally there are many such pairs, from which one is chosen
;       whose two values are closest together.  If there are many closest
;       pairs, one is choosen which has the lest bend in the line from
;       one of the pair points to the missing value point to the
;       other pair point.

(defun shrink-missing (the-output the-input &optional the-count)
  (assert (object-is an-array the-output)
          '(output array is not an-array))
  (assert (object-is an-array the-input)
          '(input array is not an-array))
  (if the-count
      (assert (fixp the-count) '(count argument is not a fixnum))
      (setq the-count 2))
  (setq the-input (prepare-array the-input nil a-long))
  (ccheck (_sba_smissing (allocate-array the-output)
                         (allocate-array the-input)
                         the-count)))

;       .En ( shrink-missing-of " \kI'ar_input" "[LISP Function]"
;       .Xn "['x_count ...])"
;       .Pa WHERE
;       More than one x_count value may be given.
;       .Pa RETURNS
;       A new array lar_output which is computed by passing
;       ar_input and x_count through \fIshrink-missing\fP.
;       \fIPrepare-array\fP is applied to ar_input.
;
;       If more than one x_count parameter is
;       given, each x_count parameter is removed from the parameter list as
```



```
;       it is used, and the process is repeated with the last lar_output
;       substituted for ar_input.
;       As an optimization, the process stops when there are no missing
;       values left in lar_output.

(defun shrink-missing-of (the-input &rest other-parameters)
  (assert (object-is an-array the-input)
          '(input array is not an-array))
  (do ((the-count)
       (the-output) (the-missing-count)
       (the-current-input the-input the-output)
       (done nil (or (null other-parameters) (=& 0 the-missing-count))))
      (done the-output)

      (setq the-count (pop other-parameters))
      (setq the-current-input
            (prepare-array the-current-input '(2 2) a-long))
      (setq the-output
            (an-array has-sizes (mapcar #'diff
                                        (has-sizes (an-array the-current-input))
                                        (copy-list '(2 2)
                                                   _SAR_MDIMENSIONS 0))
                      has-exponent (has-exponent (an-array the-current-input))))
      (setq the-missing-count
            (shrink-missing the-output the-current-input the-count))))

;       .En ( overlay-missing " 'lar_output 'lar_input)" "[LISP Function]"
;       .Pa WHERE
;       Lar_output and lar_input must have
;       the same dimension sizes and exponent.
;       .Pa RETURNS
;       Lar_output after setting some of its elements.
;       .Pa SIDE\ EFFECT
;       Replaces every missing element value
;       in lar_output by the corresponding element value in lar_input.

(defun overlay-missing (the-output the-input)
  (assert (object-is an-array the-output)
          '(output array is not an-array))
  (assert (object-is an-array the-input)
          '(input array is not an-array))
  (ccheck (_sba_omissing (allocate-array the-output)
                         (allocate-array the-input)))
  the-output)

;       .En ( set-missing-to " 'lar_array 'n_value)" "[LISP Function]"
;       .Pa RETURNS
;       Lar_array after setting some of its elements.
;       .Pa SIDE\ EFFECT
;       Replaces every missing value in lar_array by n_value.

(defun set-missing-to (the-array the-value)
  (assert (object-is an-array the-array)
          '(array argument is not an-array))
  (assert (numberp the-value)
          '(value argument is not a number))
```

```
 (ccheck (_sba_setmissing (allocate-array the-array)
                          (float the-value)))
 the-array)
```



```
(cload '(double-c-function _sba_sproduct
                           c-function _sba_convolve
                           _sba_lsproduct2 _sba_sproduct2) 'sba/sba_cprod)

;       .En ( scalar-product " 'ar_input-1 'ar_input-2)" "[LISP Function]"
;       .Pa WHERE
;       The two arrays are similar.
;       .Pa RETURNS
;       A number.  The scalar product of the two arrays.
;       More precisely, the sum of the products of corresponding
;       elements in the arrays.  \fINil\fP is returned if either array
;       has any missing values.

(defun scalar-product (the-first-input the-second-input &aux the-result)
  (assert (object-is an-array the-first-input)
          '(first array argument is not an-array))
  (assert (object-is an-array the-second-input)
          '(second array argument is not an-array))
  (setq the-first-input (prepare-array the-first-input nil a-long))
  (setq the-second-input (prepare-array the-second-input nil a-long))
  (setq the-result
        (ccheck
         (_sba_sproduct (allocate-array the-first-input)
                        (allocate-array the-second-input))))
  (if (equal the-result _SAT_DMISSING) nil the-result))

;       .En ( convolve " 'lar_output 'lar_input 'ar_kernel)" "[LISP Function]"
;       .Pa WHERE
;       The arrays are treated as 2 dimensional, and the sizes of the
;       lar_output dimensions must be one more than
;       sizes of the corresponding lar_input dimensions minus the sizes
;       of the corresponding ar_kernel dimensions.
;       .Pa RETURNS
;       Lar_output after its elements are set.
;       .Pa SIDE\ EFFECT
;       Stores the "convolution" of lar_input and ar_kernel in lar_output.
;       To be more precise, what is atored is the convolution of lar_input
;       and the matrix whose (X, Y)'th element is the (-X, -Y)'th element of
;       ar_kernel; or in other words, the (X, Y)'th element of lar_output
;       is the
;       .i scalar-product
;       of the ar_kernel and a alice of lar_input with origins (X, Y).

(defun convolve (the-output the-input the-kernel)
  (assert (object-is an-array the-output)
          '(output array argument is not an-array))
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (assert (object-is an-array the-kernel)
          '(kernel array argument is not an-array))
  (setq the-kernel (prepare-array the-kernel nil a-long))
  (ccheck (_sba_convolve (allocate-array the-output)
                         (allocate-array the-input)
                         (allocate-array the-kernel)))
  the-output)
```

```
;       .En ( convolution-of " 'ar_input 'ar_kernel)" "[LISP Function]"
;       .Pa RETURNS
;       A new 2 dimensional array lar_output with exponent the same as
;       ar_input (after its elements are converted to \fIa-long\fP's)
;       whose elements are computed by passing it and the other parameters to
;       the \fIconvolution\fP function.  \fIPrepare-array\fP is applied
;       to ar_input.

(defun convolution-of (the-input the-kernel &aux the-size-difference)
  (assert (object-is an-array the-input)
          '(input array argument is not an-array))
  (assert (object-is an-array the-kernel)
          '(kernel array argument is not an-array))
  (setq the-size-difference (mapcar 'sub1 (has-sizes (an-array the-kernel) 2)))
  (setq the-input (prepare-array the-input the-size-difference a-long))
  (convolve (an-array has-sizes (mapcar 'diff
                                        (has-sizes (an-array the-input) 2)
                                        the-size-difference)
                      has-exponent (has-exponent (an-array the-input)))
            the-input the-kernel))
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) LM-163	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-89-118
6a. NAME OF PERFORMING ORGANIZATION Lincoln Laboratory, MIT 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Electronic Systems Division
6c. ADDRESS (City, State, and Zip Code) P.O. Box 73 Lexington, MA 02173-0073	7b. ADDRESS (City, State, and Zip Code) Hanscom AFB, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-85-C-0002
8c. ADDRESS (City, State, and Zip Code) 1400 Wilson Boulevard Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62702E PROJECT NO. 306 TASK NO. WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Sketch 4B, An Image Understanding Operating System

12. PERSONAL AUTHOR(S)

Robert L. Walton, Jacques G. Verly, and Patrick Van Hove

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Lincoln Manual	FROM ______ TO ______	1989, June 14	502

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

LISP and C computer launguages; software development environment; image processing; image understanding; computer vision; artificial intelligence (AI); Automatic target recognition (ATR); signal processing

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

SKETCH is an image understanding operating system that can be used for the development of image processing, image understanding, and computer vision systems. It is actively being used at MIT Lincoln Laboratory for the implementation and testing of automatic target recognition algorithms. SKETCH is an excellent tool for the rapid development of complex end-to-end systems. It emphasizes flexible and efficient interactive computing rather than "real-time" performance. These features are achieved through the simultaneous use of Franz LISP for high-level control and symbolic computation and C for extensive numeric processing. Array management facilities are provided to manipulate images and other multidimensional arrays both on disk and in central memory. A catalog system is provided to store LISP values and abstract SKETCH objects, including arrays. Typically, one starts with an input catalog containing tens or hundreds of sets of images and ancillary data. Each set is then read from the input catalog, a sequence of algorithms is applied to it, and the corresponding results are appended to an output catalog, which can be used, in turn, as the input catalog for the next round of computation; the read-compute-write cycle can be repeated as many times as necessary. This mechanism allows a researcher to apply a complex algorithm to a large data set overnight and to review the results the next day using the extensive display tools provided by SKETCH. One particularly convenient aspect of the display system is that whole displays are SKETCH objects that can be stored in a catalog. Browsing through pictorial results is thus rapid and straightforward.

This manual contains the complete description of the SKETCH system, examples its functions, and excerpts from the actual SKETCH code. SKETCH is organized by packages (Objects, Arrays, Catalogs, etc.) and readily accepts user-constructed packages to adapt the working environment to specific applications. Dozens of such packages have been created by the authors and their colleagues in the Machine Intelligence Technology Group at Lincoln Laboratory. These packages are not formally part of SKETCH, but are otherwise indistinguishable from the official SKETCH packages. Examples of topics covered by these add-on packages are the Hough Transform, Mathematical Morphology, and Synthetic Scene Generation. One of the more appreciated features is the relative ease with which new packages can be documented. The only requirement on the part of the user is to add a documentation header to each important new function. Then, through a simple command, these headers are automatically extracted, and formatted documentation is produced.

SKETCH is currently running on Sun 3 workstations and VAX computers under the UNIX operating system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Lt. Col. Hugh L. Southall, USAF	22b. TELEPHONE (Include Area Code) (617) 981-2330 22c. OFFICE SYMBOL ESD/TML

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE