Massachusetts Institute of Technology
School of Architecture and Planning
77 Massachusetts Avenue
Cambridge Massachusetts 02139
Telephone (617) 253-5960

Architecture Machine Group

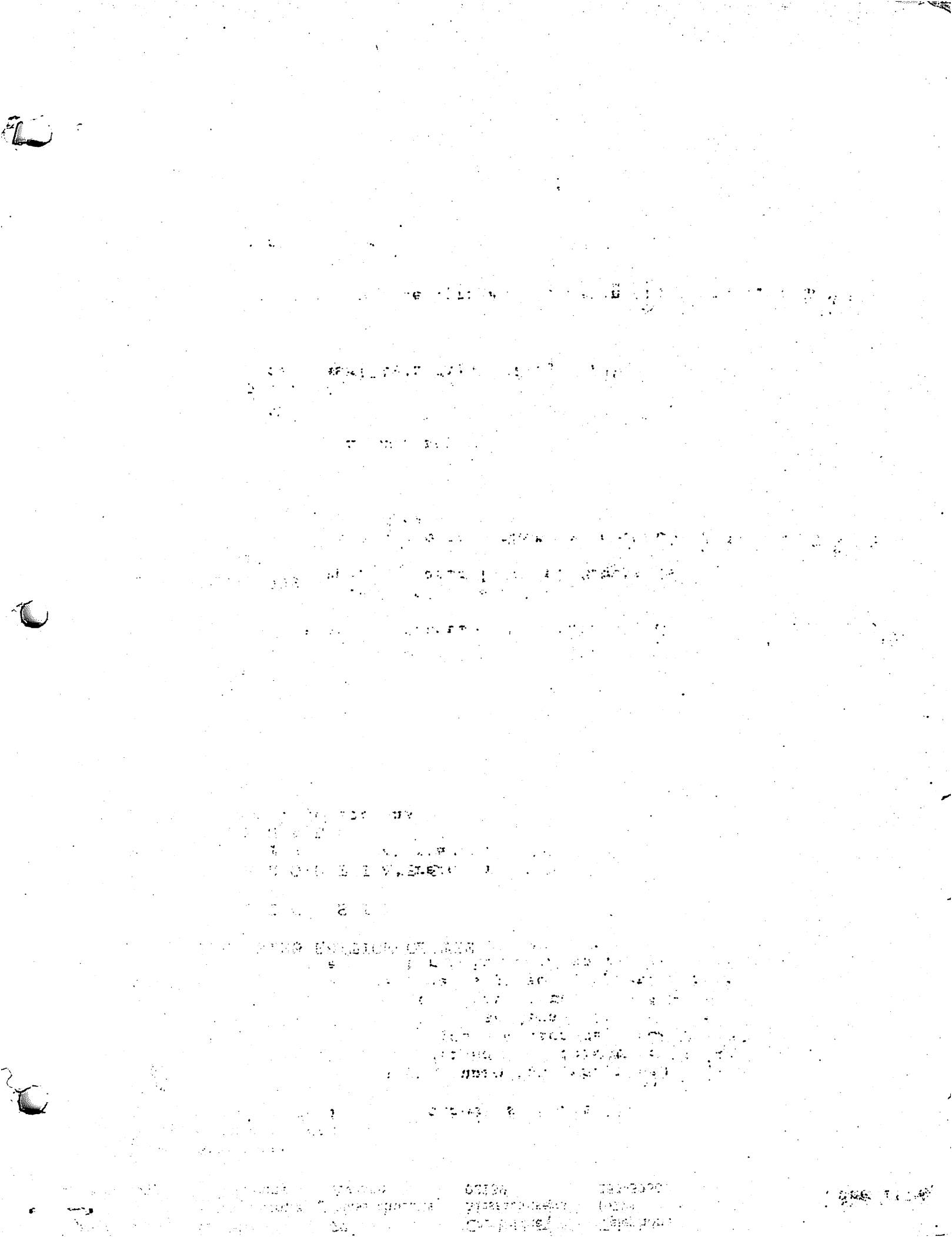
THE THIRD EDITION OF THE

M A G I C S I X

S U B R O U T I N E S

M A N U A L

June 1979



```
ask entry options(variable)
call ask(prompt, variable [, other arguments]);
```

ask is a general purpose input subroutine which can read in just about any type of data desired in a format appropriate to that type of data. In particular, each call to ask reads in one (and only ONE) variable. ask first writes out the prompt string and waits for the user to type in a line with the value on it. The line is parsed completely so that subsequent calls to ask will result in those values being used as opposed to the user being prompted for them.

Example:

```
 call ask("x,y?",x);      /* read in x */
 call ask("y?",y);        /* read in y */
```

Two sample interactions:

```
          x,y?
->        256
          y?
->        256

          x,y?
->        256 256
```

Note that in the second interaction the second prompt is not printed out.

```
dcl ask$ask_nnl entry options(variable);
call ask$ask_nnl(prompt, variable [, other arguments]);
```

ask$ask_nnl is just like ask except that it does not put a newline at the end of its prompt message

The other arguments are useful for providing information in the prompt. For example, to read in an array:

```
     do i=1 to 3;
          do j=1 to 3;
               call ask("A(^i,^i):",a(i,j),i,j);
          end;
     end;
```

would provide a prompt of the form A(2,3): when prompting for a(2,3).

Additional ask entry points.

```
ask$ask_check  entry rtns (bit(1)),
   returns '1'b iff there is something in the ask buffer.

ask$ask_look   entry options(variable) returns(fix bin(15)),
   looks at next token, prompts if none, returns token in
   second arg if possible, if not it returns -1, if
   possible it returns:

      0: nothing special, ok conversion
      1: token was quoted
      2 -> N: index(+1) of break char in parse_string

 ask$ask_look_nnl is just like ask$ask_look except that
 the former does not end its prompt with a newline
 when it does prompt

ask$ask_parse  entry options(variables)
   takes a string and makes subsequent calls to ask use those
   chars as break characters

ask$ask_clear  entry
   flushes the buffer

ask$ask_set    entry options(variable),
   takes a string and uses it as the input string.
   ask_clear is called internally; the string will
   be truncated after 131 chars and need not be
   ended by a <cr>

   -will not work with character varying!!

ask$pop_token  entry
   pops the next token in the buffer
```

bdbm stands for Big DataBase Manager. It is not really a database manager but does make it easier to manage large databases which may span many segments. The database is stored in a set of segments with the names <whatever>.001, <whatever>.002, and so on. Your program may put whatever it wants into the specified segments. (This means that they may range in size from 2048 bytes to 63488.)

```
dcl bdbm$initiate entry(char(168)vary,char(32)vary,char(2),
                    fixed bin,ptr,fixed bin(31);
call bdbm$initiate(dname,ename,indicator,count,bp,code);
```

where:

dname	is the name of the directory in which all of the segments are to be found
ename	is the <whatever> or the name of the database
indicator	is just a string unique for this database and is used to generate address space names
count	is set to the number of segments in the database (note that they must be contiguous, i.e. no segment number 3 before a segment number 2 exists)
bp	is set to a structure describing the database. This pointer is used to refer to the database to all other calls to bdbm)
code	is set to a standard system error code

bdbm$initiate initiates the database, at least one segment <whatever>.001 must exist. bdbm initiates all of the existing segments in the database in a set of address spaces and tells you how many were found and gives you a pointer as a handle. This pointer is used to refer to the database (since several may be initiated at any given time) for alll of the other calls to the database manager.

```
dcl bdbm$get_seg(ptr,fixed bin,ptr,fixed bin(31));
call bdbm$get_seg(bp,segno,sp,code);
```

where:

bp	is the pointer returned by bdbm$initiate
segno	is the number of the segment you want to get at
sp	is set to point at the segment you just got at when you are done with this segment, just terminate it normally with hcs$terminate
code	is set to a standard system error code

```
dcl bdbm$append_seg entry(ptr,fixed bin,fixed bin(31));
call bdbm$append_seg(bp,segno,code);
```

```
where:
 bp             is the pointer returned by bdbm$initiate
 segno          is the index number of the segment created
 code           is set to a standard system error code

bdbm$append_seg appends a new segment to the database and
returns its index number


 dcl bdbm$terminate entry(ptr,bit(16),fixed bin(31));
 call bdbm$terminate(bp,flags,code);

where:
 bp             is the pointer returned by bdbm$initiate
 flags          is a set of flags indicating how much work
                     terminate is to do, for now, use "0000"b4
 code           is set to a standard system error code
```

```
com_error entry options(variable)
call com_error(code,name[,pattern_string[,arguments]]);
```

Writes a standard error message of the form name:message other out on the stream error_output.

where:

code
: is a standard error code

name
: is a string which is the name of the program reporting the error, e.g. "print"

pattern_string
: is a pattern string which along with any arguments is passed to ioa to build up the rest of the error message

arguments
: other arguments for use with the pattern_string

For example:

```
call com_error(code,"print");
call com_error(code,"pl1","While trying to initiate ^c>^c.",dname,ename);
```

For a list of the currently known error messages look in >source>error_table.et.

```
error_table
*
* these are the definitions of the standard error table codes
*
* if you add a code to the file, make sure you put it in the right place
* the codes are in descending order
*
* codes 201-300 are reserved for use with the i/o system
* codes 301-350 are reserved for use with the graphics system
* codes 601-700 are reserved for use with the command processor
* codes 701-800 are reserved for conditions
* codes 801-900 are reserved for remote system stuff
*
208,oldstream,A stream already exists by the name.
5,nameset,The username was already set as requested.
4,oldrefcor,Refcor block already existed.
3,namenotascii,The username was not ascii. It was converted to ascii.
2,devowned,Device is already owned.
1,segknown,Segment is already known in this process.
0,noerror,If you see this please report it.
-1,nosegs,No free segments in domain.
-2,notfound,Entry not found.
-3,badpath,Some directory in path does not exist.
-4,badtype,Entry is not of proper type.
-5,dirnotempty,Attempt to delete a non-empty directory.
-6,badname,Invalid pathname or entry name.
-7,badlink,Illegal path name in a link.
-8,lastname,Attempt to delete the only name on an entry.
-10,badecode,Error code does not exist in the error table.
-11,incnargs,Incorrect number of arguments to procedure.
-12,badargs,Bad argument passed to procedure.
-13,linkloop,More than 16 links were chased.
-14,namedup,Name already exists in directory.
-15,argmissing,Expected argument missing.
-16,nomatch,No entries matched the specified star-name.
-17,nosuchdir,No such directory in old file system.
-18,nosuchfile,No such file in old file system.
-19,badformat,A file was not formatted correctly for it's use.
-20,bad_userid,A nonexistent userid was encountered.
-21,badring,Invalid ring number.
-22,extsists,External static data already exists.
-23,notinarchive,A reference was made to a non-existent archive entry.
-24,nosuchspace,The address space [domain] does not exist.
-25,safetyswitchon,The safety switch is on.
-26,badtruncate,Attempt to truncate a read-only segment.
-27,archtoobig,The target archive is too big to be appended to.
-28,not_multics,This system is not multics.
-29,noargsall,No arguments may be given.
-31,bactrunc,Attempt to truncate thru a link.
-32,badvolume,Attempt to use an unmounted volume.
```

```
-33,badoldseg,The specified segment number is unavailable.
-34,norefcor,Unable to find specified refcor block.
-35,refcorlok,Refcore block locked. (refcnt>0 during delete request)
-36,badsname,Sname is not the name of an SDB.
-37,bad_seg,Segment not found in domain.
-38,novolume,Volume not found.
-39,volprotected,The volume was already protected.
-40,volunprotected,The volume was already unprotected.
-47,nouser,User not logged in.
-48,userres,The username "^^@^^@^^@^^@^^@^^@^^@^^@" reserved for use by the s
-49,userdupe,User already logged in.
-50,norefptr,Attempt to use a null referencing ptr in make_ptr
-51,inputoverflow,Input line too long.
-52,outputoverflow,Return string too long (> 133).
-53,noletter,Letter not found in font.
-54,badfont,Font in bad format.
-55,notlink,Entry is not a link.
-56,not_static,External static data not found.
-63,vtoce_big,Attempt to call initiate by index on a too big vtoce.
-64,vtoce_free,Attempt to call initiate by index on a free (unallocated) vtoc
-65,vtoce_bad,Attempt to call initiate by index on a bad vtoce (type ^^= 1).
-66,vtoce_alloc,Attempt to call initiate by index on an allocation vtoce.
-67,vtoce_4,Attempt to call initiate by index on vtoce 4.
-68,vtoce_1,Attempt to call initiate by index on the vtoc.
-69,gerzso,Attempt to create a dir deeper than >u>sdms>wcd>papers>acm.
-70,norecipients,No recipients specified.
-71,badrecipients,Illegal recipient list.
-72,badnetid,Illegal network user id.
-98,callroot,Attempt to call the root.
-99,cdroot,Attempt to create_dir the root.
-100,nosuchseg,No such segment in specified address space.
-101,notobject,Not a valid object segment.
-102,noentry,Entry point not found in segment.
-103,nolines,No statement map in object segment.
-104,nosuchline,Impossible to get line number from object segment.
-105,badfs,An inconsistency was found in the file system.
*
* Error codes 201-300 are reserved for the io system.
*
-200,notiocb,Object is not an iocb.
-201,attached,Attempt to attach an already attached stream.
-202,badatt,Bad attach description.
-203,nodim,Can't find the dim.
-205,inhibit,I/O operation inhibited on stream.
-206,nostream,No such stream was found.
-207,cantwire,Unable to grab wired storage.
-208,cantunwire,Unable to unwire memory.
-217,noihroom,No room for a new interupt handler.
-218,noih,No such interupt handler.
-219,eot,End of transmission (or medium).
```

```
-220,eof,End of file mark.
-221,transmit,Transmission error.
-222,oldih,Already an interupt handler active for that device.
-249,not_diverted,Stream is not diverted.
-250,outservice,Device is out of service.
-251,DU,Device unavailable.
-252,devfree,Attempt to release a device you do not own.
-253,devattach,Device is attached by another user.
-255,badevice,Invalid device name.
-257,detached,Invalid operation on a detached stream.
-260,badsleep,Attempt to "set_sleep" on the same stream twice.
-261,nosleep,Nothing to sleep on, bad I/O order call.
-262,nodismiss,No sleep block to dismiss, bad I/O order call.
-263,noiodone,No I/O done since start of empty order call.
-264,notempty,Only some I/O done since start of I/O call.
-265,timeout,Timeout on some device.
-270,noinput,Cannot open device for input.
-271,otherattachers,Other process have devices attached.
-277,invreqt,Invalid request type on I/O operation.
-278,noinfo,Addition information required for operation.
-279,nodisplay,Device is not a display.
-280,badversion,Attempt to use a non-existant or obsolete structure version.
-282,badioreq,Illegal I/O request.
-283,notsupport,Cursor positioning request not supported by device.
-286,baddevptr,Attempt to use an invalid device pointer.
-293,badcode,Bad op-code for I/O order operation.
-294,badoperand,Bad operand for I/O order operation.
-295,clkprotect,The real time clock was protected.
-296,clkstopped,The real time clock was not running.
*
* See comment above. **********
*
*
* Error codes 301-350 are reserved for the graphics system.
*
-301,nogsetup,No graphics streams have been setup.
-302,nogcb,GCB not found.
-303,gcbexists,A GCB by that name already exists.
-304,ramdu,Ramtek device unavailable.
-305,viddu,Vidicon device unavailable.
-306,users,Vidicon user must run single user MagicSix.
*
* See comment above. **********
*
-373,notdir,Entry is not a directory.
-409,badopt,Bad option or keyword.
-410,badnum,Invalid number.
-411,noarg,Expected argument missing.
-500,topicnotfnd,No information was found for the specified topic.
*
```

```
* Error codes 601-700 are reserved for use by the cp. **********
*
-601,relerr,A release failed.
-602,starterr,A start failed.
-603,listenerr,The listener received a garbage code.
-604,badquote,Quotes do not balance.
-605,badcmd,An illegal command name was encountered.
-606,badparens,Parentheses are used incorrectly.
-607,badactvfunc,A bad call was made to an active function.
-608,noerrfr,The default error handler was called incorrectly.  Please report
-612,brakunbal,Active function brackets do not balance
-631,svc5/1,Invalid stack pointer or free pointer while gating into ring 0.
-632,svc5/2,Inhibit flag on during a fault.
-633,svc5/3,Invalid saved stack pointer.
-634,svc5/4,No standard default handler.
-635,svc5/5,Attempt to unwind completely.
-636,svc5/6,Attempt to unwind with an invalid stack pointer.
-637,svc5/7,Out of bounds fault on user's stack.
-638,svc5/8,Out of bounds fault on user's brain.
-639,svc5/9,Ast locked on crawlout.
-640,svc5/10,Unable to perform critical I/O.
-641,scc5/11,Attempt by error handler to call listener failed.
-642,svc5/12,Loop in stack frames: unable to find base of stack.
-643,svc5/13,Bad machine stack frame.
-644,svc5/14,Called core_to_index on bad vtoce.
*
* See comment above. **********
*
*
* Error codes 701-800 are conditions with reasons.
*
-701,systemii,Illegal instruction.
-702,no_write,Attempt to write into read only memory.
-703,systemao,Arithmetic overflow or zerodivide.
-704,oobounds,Attempt to touch beyond the end of a segment.
-705,no_execu,Branch to a non-executable segment.
-706,back_up,Unrecoverable memory fault. (back_up fault.)
-707,linkage,Unresolvable external reference. (Non-existent subroutine.)
-708,nonexist,Attempt to touch a non-existent segment.
-709,badnargs,Incorrect number of arguments to a procedure.
-710,gate_err,Wild branch or illegal call to a system routine. (gate error)
-711,area,Allocation in an area failed because the area was full.
-712,bad_args,Read-only segment passed illegally to a ring 0 routine.
-713,badxstat,Segment containing external data has incompatable space name.
-714,noroom,Mapping an argument into another address space failed (noroom fa
-715,recurse,Stack overflow due to uninitialized string, infinite recursion,
-716,unimplop,PL/1 program used an unimplemented operator.  Try bug pl1.
-717,badsegno,Attempt to pass a bad segment number to hcs.
-718,badalarm,The timer alarm feature was used improperly.
-719,conversi,Illegal data type conversion.
```

```
-720,askconvs,Illegal data type conversion in ask.
*
* See comment above. **********
*
*
* Error codes 800 to 900 are remote machine errors.
*
-800,nomach,Unknown machine id.
*
* See comment above. **********
*
```

```
getstar$eqstar entry(char(32),char(32));
match=getstar$eqstar(ename,sname);

where:
 ename          is the entry name to match
 sname          is the star name to match against
 match          is "0"b if they don't match, otherwise "1"b


getstar$star_init entry(char(168)varying,char(32),ptr,fixed bin(31))
call getstar$star_init(pathname,starname,magic_ptr,code);

where:
 pathname       name of the directory in which to match star
                 names
 starname       starname to match
 magic_ptr      a ptr which is set to point at a special
                 database which describes the iteration
 code           is an error code

getstar entry(ptr,char(32)varying)
call getstar(magic_ptr,name);

where:
 magic_ptr      is the ptr returned by getstar$star_init
 name           is the name which matched it
                     if this name is "" then there are no more matches

getstar$type entry(ptr,char(32)varying,bit(16))
call getstar$type(magic_ptr,name,entry_type);

where:
 magic_ptr      is as above
 name           is as above
 entry_type     is a file system type:
                     8000 - directory
                     4000 - link
                     2000 - segment

getstar$cleanup entry(ptr)
call getstar$cleanup(magic_ptr);

where:
 magic_ptr      points to the data base set up by getstar$star_init
                 and which this entry destroys
```

```
dcl hcs$add_ref_name entry(pointer,char(32) varying);
call hcs$add_ref_name(sp,rname);
```

where:
 sp   is a pointer into the segment
 rname is the reference name to add

```
dcl hcs$append_dir entry(char(168) varying,char(32) varying,fixed bin(31));
call hcs$append_dir(dname,ename,code);
```

where:
 dname is the name of the containing directory
 ename is the name of the directory to create in dname
 code is a standard error code

```
dcl hcs$append_link entry(char(168)varying,char(32)varying,char(168)varying
call hcs$append_link(dname,ename,tname,code);
```

where:
 dname is the name of the containing directory
 ename is the name of the link to create in dname
 tname is the target of the link ename
 code is a standard error code

```
dcl hcs$append_seg entry(char(168) varying,char(32) varying,fixed bin(31));
call hcs$append_seg(dname,ename,code);
```

where:
 dname is the name of the containing directory
 ename is the name of the entry to create in dname
 code is a standard error code

```
dcl hcs$get_bit_count entry(pointer,fixed bin(31));
call hcs$get_bit_count(sp,bc);
```

where:
 sp is a pointer into a segment
 bc is the length of that segment in bits

```
dcl hcs$get_seg_size entry(pointer,fixed bin);
call hcs$get_seg_size(sp,length);
```

where:
 sp is a pointer into the segment
 length is the size of the segment in bytes rounded to the nearest page(2K)

```
dcl hcs$get_sdb entry(ptr,ptr);
call hcs$get_sdb(segp,sdbp);
```

where:
 segp is a pointer to a segment
 sdbp is a pointer to the sdb for that segment (or null())

```
dcl hcs$find_sdb_space entry (char (4), ptr, ptr, bit(16),fixed bin (31));
call hcs$find_sdb_space (sname, segptr, sdbptr, segacl,code);
```

where:

sname	is the address space name
segptr	is a ptr to segment in the address space
sdbptr	is a ptr to the sdb of the above segment (output)
segacl	is the segment acl from the address space (output)
code	standard system error code

(see also: hcs$get_sdb)

```
dcl hcs$get_seg_ptr entry(ptr,char(32) varying);
call hcs$get_seg_ptr(sp,rname);

where:
 sp    is the pointer to the segment if it exists
 rname is the reference name to search for
```

```
dcl hcs$initiate entry(char(168) varying,char(32) varying,ptr,fixed bin(3
call hcs$initiate (dname,ename,ptr,code);
```

where:
dname is the name of the containing directory
ename is the name of the entry to initiate
ptr is the ptr to the segment when it is initiated
code is a standard error code

```
hcs$initiate_w_options
          entry(char(168) varying,char(32) varying,char(32) varying, bit(
          ptr,fixed bin(31));
call hcs$initiate_w_options(dname,ename,refname,place,ptr,code);
```

where:
dname is the name of the containing directory
ename is the name of the entry to initiate
refname is the refname to be added to the segment
place this bit is set if the segment is to be initiated in the
segment specifed by ptr
ptr is the ptr to the segment when it is initiated
code is a standard error code

```
hcs$initiate_in_space
     entry(char(168)vary,char(32)vary,char(4),char(4),ptr,fixed bin(31));
call hcs$initiate_in_space(dname,ename,space,nspace,segp,code);
```

initiates the segment dname>ename in the space designated by space
(see also hcs$initiate_space_options and hcs$terminate_in_space)

where:
dname is the directory name
ename is the entry name
space is the target space name
if space=" " then the space designated by
the file system is used, if space="\@\@\@\@"
then \@\@\@\a is used
if space="...." then the space designated by
the file system is used, if space="\@\@\@\@"

```
                    then the callers space is used
               if space="====" then the space in which the caller
                    is running is used (like initiate)
               otherwise the space designated by space is used
 nspace    is set to the space in which the segment was actually initiate
 segp      is set to a pointer to the base of the segment
 code      is set to a standard error code
```

```
hcs$initiate_space_options
     entry(char(168)vary,char(32)vary,char(32)vary,char(4),char(4),
          bit(1),ptr,fixed bin(31));
call hcs$initiate_space_options(dname,ename,rname,space,nspace,
          flag,segp,code);
```

initiates the segment dname>ename in the space designated by space in the segment designated by segp (if flag is "1"b) with the reference name of rname

where:

```
 dname     is the directory name
 ename     is the entry name
 rname     is the reference name, none is added if this string is ""
 space     is the space in which to initiate the segment
               if space="    " then the space indicated by the file
                    system is used
               if space="====" then the space in which the caller is
                    running is used (like initiate_w_options)
               otherwise the space designated by space is used
 nspace    is set to the space in which the segment is initiated
 flag      if "0"b then the segment is initiated in the first free slot
               otherwise the segment indicated by segp is used (if free)
 segp      if flag was "0"b is set to the segment in which was initiated
 code      is set to a standard error code
(see also: hcs$terminate)
```

```
dcl hcs$remove_dir entry(char(168) varying,char(32) varying,fixed bin(31
call hcs$remove_dir(dname,ename,code);
```

where:
 dname is the name of the containing directory
 ename is the name of the dir to delete
 code is a standard error code

```
dcl hcs$remove_link entry(char(168) varying,char(32) varying,fixed bin(3
call hcs$remove_link(dname,ename,code);
```

where:
 dname is the name of the containing directory
 ename is the name of the link to delete
 code is a standard error code

```
dcl hcs$remove_seg entry(char(168) varying,char(32) varying,fixed bin(31
call hcs$remove_seg(dname,ename,code);
```

where:
 dname is the name of the containing directory
 ename is the name of the seg to delete
 code is a standard error code

```
dcl  hcs$request_device (char (8), pointer, fixed bin (31))
call hcs$request_device (device_name, device_index, code)
```

hcs$request_device will attach the device to your user id along with its buffers and so on initialized. It will return the pointer to the devstruct.incl.pll structure in >i.

```
dcl hcs$free_device (ptr, fixed bin (31))
call hcs$free_device (index, code)
```

hcs$free_device will detach your user id from the device and etc. Note, this will be done for you when you log out if you don't do it explicity yourself.

```
dcl hcs$set_acl entry (ptr,bit(16));
call hcs$set_acl (segp,acl);

where:
 segp is a pointer to the segment
 acl  is the access bits as follow:
     byte 1 is the access when running in the system domain
     byte 2 is the access when running in the user domain
          within each byte:
          bit 0: system will take a fault when segment is executed
          bits 1-2: 00 - segment is writable, 11 - segment is read/onl
          bit 3: set after system copies linkage section on a fault
          bit 4: segment is a gate and therefor requires gate actions
          bit 5: segment can be read
          bit 6: segment can be executed
          bit 7: segment can be written

(see also: nrfs$set_acl)
```

```
dcl hcs$set_bit_count entry(ptr,fixed bin(31));
hcs$set_bit_count(segp,bc)
```

- sets the bit count on the segment to the specified number

where:

segp	is a pointer to the segment whose bit count is to be set
bc	is the bit count to set it to (eight times the number of characters in the segment

```
dcl hcs$split_name entry(char(168)varying,char(168)varying,char(32)var
call hcs$split_name(pname,dname,ename,code);
```

where:
 pname is an absolute pathname
 dname is the directory part of pname
 ename is the entry part of pname
 code is a standard error code

(see also: scs$expand_path)

```
dcl hcs$terminate entry(ptr);
call hcs$terminate(pointer);
```

where:
 ptr is a pointer into the segment to terminate

```
hcs$terminate_in_space entry(ptr,char(4),fixed bin(31))
call hcs$terminate_in_space(segp,space,code);
```

terminates the specified segment in the specified space

where:
 segp is a pointer to the segment
 space is the name of the space
 code is a standard error code

```
dcl hcs$truncate_file entry(char(168)vary,char(32)vary,fixed bin,fixed
call hcs$truncate_file(dname,ename,nblocks,code);

dcl hcs$truncate_ptr entry(ptr,fixed bin,fixed bin(31));
call hcs$truncate_ptr(seg_ptr,nblocks,code);
```

- truncates a segment to the specified size (usually zero)
  in 2048 byte pages

where:

dname	is the directory name of the segment
ename	is the entry name of the segment
seg_ptr	is a pointer to the segment to truncate
nblocks	is the number of 2KB blocks that the file should contain
code	is set to a standard system error code

file_io is the file input/output dim which parses its attach description as follows:

```
file_io filename options
```

where:

filename is the pathname of the file to read from or write to

options are as follows:

-ascii	interpret the file as ascii data so that carriage return terminates transfers
-binary -b	interpret the file as a simple string of bytes
-append -a	indicates that any old file is to appended to as opposed to reinitialized
-msf	indicates that when a file fills up that another file is to be allocated The entry name of the file being attached to has the index of the file appended to it as in: file001, file002, ... If the file name has a ? then it is replaced by the index of the file.
-max #	indicates that when a file contains # bytes on output it is to be considered full by the -msf option

```
ioa entry options(variable)
call ioa(pattern_string[,arguments ...]);

ioa$ioa_nnl entry options(variable)
call ioa$ioa_nnl(pattern_string[,arguments]);

ioa$ioars entry options(variable)
call ioa$ioars(result_string,pattern_string[,arguments]);

ioa$ioars_nnl entry options(variable);
call ioa$ioars_nnl(resul _string,pattern_string[,arguments]);
```

ioa generates a string derived by substituting the converted values of the arguments (if any) into the appropriate places in the pattern string and then outputs the result on the stream user_output.

ioa$ioa_nnl is like ioa except it doesn't put out a carriage returns at the end of the string

ioa$ioars is like ioa except that the first argument is a string in which to place the result as opposed to writing it to user_output

ioa$ioars_nnl is like ioa$ioars except that it does not put out a carriage return at the end of the string and is thus useful for conversions

where:

result_string	is a string in which the result of the conversion et alia is placed. If it is character varying then the length will be set correctly. ioa will never write past the end of the string.
pattern_string	is a string of text which is treated as follows:

Any character other than ^ is simply moved into the target string. Thus, call ioa("String!"); would type out String! and then a carriage return.

When a ^ is encountered it may be followed by either a letter or a number (decimal) followed by a letter. The letter indicates the type of conversion (or operation) while the number usually is the width of the target field. The number may be replaced by the letter v, which tells ioa to use the value of the next argument rather than a value in the pattern_string for the field width.

The following letters indicate the following:

^na inserts a character string variable into the target string. It either uses the full length of the string if n is not given otherwise it converts the string to one of the appropriate length (padding with blanks or truncating) before outputting.

^nc is like ^na except that the character string is output only up to the first blank or carriage return in the variable

^nh converts the value of the argument in question to hexadecimal and inserts that string right justified with leading zeroes removed into the field

^ni convert the value of the argument to a decimal number and inserts that string right justified in the field

^p converts the value of the argument to a pointer in the form s|oooo, where s is the segment number and oooo the offset

^r is the same as a carriage return

^nt tabs to the appropriate position, useful for lining things up

^nw like ^nh except it doesn't remove leading zeroes

^nd like ^ni except it doesn't remove leading zeros

^nx inserts n (or 1) spaces (or space)

For example:

```
 call ioa("value=^i",32);     prints
value=32

 call ioa("frotz at ^p has ^i names, first is ^va",frotz_ptr,
     frotz.names,frotz.name(1).length,frotz.name(1).string); prints
frotz at 4|0280 has 2 names, first is mumblebarf

 call ioa("pathname ^c>^c loses",dname,ename); prints
pathname >sl1>foobar loses
```

Advanced hackery:

^n(iterates the field up to the next ^) n times (n may be 0)

^) closes iteration

^n[executes the n+1th field between the ^n[and the ^]
where ^; is used to bound fields

^; field delimiter for conditionals

^* same as ^; but handles any numbers not already handled. It is an "else" clause.

^] end of conditional

^ng "goes" to the n-th argument, the first argument is number 1

^oxx does an order call xx with 0,1 or 2 arguments as specified
^noxx
^n:noxx

For example:

```
 call ioa("^4(^h ^)",val(1),val(2),val(3),val(4));

 call ioa("^v(foobar^)",5); prints
foobarfoobarfoobarfoobarfoobar

 call ioa("^ocs^1:4oxyStatus of version ^a is ^a.",version,status);
```
clears the screen, positions at 1,4 and writes the message

```
 call ioa("^v:voxy",xpos,ypos);
```
cheap way of doing a cur_pos

I/O to Streams Other Than user_output:

ioa$ioa_iocb	takes as a first argument a pointer to the stream on which output is to be performed
ioa$ioa_iocb_nnl	like ioa$ioa_iocb except that it does not put a newline (carriage return) at the end of the line
ioa$ioa_stream	takes as a first argument a character string name of the stream on which to perform output
ioa$ioa_stream_nnl	is like except that it does not put out a newline at the end

Examples:

```
call ioa$ioa_iocb(linkage$error_output,"barfage at j=^i",j);
call ioa$ioa_stream_nnl("listing_output","^40tPage ^i",page_number);
```

How to Get Iocbs

To get the iocb of one of the most commonly used streams which are initialized by the process initialization there are a number of static external pointers in the process linkage section as follows:

```
linkage$user_input ptr external
linkage$user_output ptr external
linkage$error_output ptr external
linkage$listing_output ptr external
```

In general to get an iocb one uses the following program:

```
* Subroutine iocs$find_iocb_ptr

iocs$find_iocb_ptr entry(char(32) varying,ptr,fixed bin(31))
call iocs$find_iocb_ptr(iocb_name,iocb_ptr,code);
```

Returns a pointer to the i/o control block for the stream specified.

where:

iocb_name	is the name of the i/o stream for which the iocb pointer is desired
iocb_ptr	is set to the pointer to the i/o control block
code	is a standard error code

How to Make and Unmake Iocbs

```
* Subroutine iocs$make_iocb

iocs$make_iocb entry(char(32)varying,ptr,fixed bin(31));
call iocs$make_iocb(iocb_name,iocb_ptr,code);
```

where:

iocb_name	is the name for the iocb to be created
iocb_ptr	ptr to the iocb made or found (output)
code	standard system error code

Create an iocb by the name of iocb_name if it does not already exist. If an iocb by that name already exists return that iocb ptr and a positive code.

* Subroutine iocs$destroy_iocb

```
iocs$destroy_iocb entry (ptr, fixed bin (31))
call iocs$destroy_iocb (iocb_ptr, code)
```

where:

iocb_ptr	is a pointer to the iocb to be destroyed
code	is a standard system error code

Destroys an iocb by free its storage and unchaining it from the iocb c
NOTE: The iocb must be detached to be destroyed.

How to attach and detach streams

* Subroutine iocs$attach

```
iocs$attach entry(ptr,char(168)varying,ptr,char(8),fixed bin(31));
call iocs$attach(iocb_ptr,attach_des,ref_ptr,modes,code));
```

where:

iocb_ptr	points at an I/O control block
attach_des	is a string of the form: "io_module option1 option2 ..." where io_module$io_module_attach is the name of the entry point which is called with the iocb_ptr pointing at a "filled in" iocb (see iocb.incl.pl1) and an error code
ref_ptr	is currently unused (use null())
modes	is a string describing the mode of the attached stream the following letters indicate the designated attr s stream (mandatory for now) i input allowed o output allowed
code	is a standard system error code

This entry point attaches the specified stream with the appropriate mo
to make it usable for input or output.

* Subroutine iocs$detach

```
iocs$detach entry(ptr,fixed bin(31));
call iocs$detach(iocb_ptr,code);
```

where:
 iocb_ptr   is an iocb ptr
 code   is a standard system error code

This entry detaches the specified stream.

How to do I/O

* Subroutine iocs$put_chars:

```
iocs$put_chars entry(ptr,ptr,fixed bin(31),fixed bin(31))
call iocs$put_chars(iocb_ptr,chars_ptr,chars_len,code);
```

Transmits the specified data to the device indicated
by the i/o control block designated.

where:
 iocb_ptr   is a pointer to an i/o control block
 chars_ptr   is a pointer to the data to be transmitted
 chars_len   is the number of bytes to transmit
 code   is a standard error code

* Subroutine iocs$get_chars

```
iocs$get_chars entry(ptr,ptr,fixed bin(31),fixed bin(31),fixed bin(31)
call iocs$get_chars(iocb_ptr,chars_ptr,chars_len,real_len,code);
```

Reads the specified number of bytes of data from the device
specified by the i/o control block.

where:
 iocb_ptr   is a pointer to an i/o control block
 chars_ptr   is a pointer to the input buffer
 chars_len   is the number of bytes that the input
                 buffer can hold
 real_len   is the actual number of bytes transmitted
                 by the operation
 code   is a standard error code

* Subroutine iocs$get_line

```
iocs$get_line entry(ptr,ptr,fixed bin(31),fixed bin(31),fixed bin(31))
call iocs$get_line(iocb_ptr,chars_ptr,chars_len,real_len,code);
```

Reads characters up to the first carriage return (hex 0d)

from the device specified.

where:

iocb_ptr	is a pointer to an i/o control block
chars_ptr	is a pointer to the input buffer
chars_len	is the number of bytes that the input buffer can hold
real_len	is the actual number of bytes transmitted by the operation
code	is a standard error code

How to control Dims and Devices

* Subroutine iocs$order

```
iocs$order entry(ptr,char(8),ptr,fixed bin(31))
call iocs$order(iocb_ptr,order,info_ptr,code);
```

where:

iocb_name	is an iocb pointer (as above)
order	is the operation to perform
info_ptr	is a pointer to any information or other space which the order call might need
code	is a standard system error code

This program is used to control the operation of a device or an ic through the DIM that controls that i/o stream. Specific operations are initiated by specifying order-type in the order argument. Each DIM sup a different set of operation, the tty_io module order-types are listed

Orders include:

resetrd — reset the input buffer stream to flush all buffered characters

status — give the status of the input and output tty buffers. Both the number of characters in the input and the output buffers are returned as fixed bin numbers in the following structure:

```
dcl 1 status based (info_ptr),
      2 input_status fixed bin,
      2 output_status fixed bin;
```

empty — allows the user to wait for the input, or the output buffe to be empty. It also lets the user specify a time-out afte the order call returns with status information. It info st

is:

```
dcl 1 empty based (info_ptr),
      2 type char (8),
      2 delay fixed bin (31);
```

where:

type — is either "input" or "output" and specifies which buffer is to be tested.

delay — is the time-out time in milliseconds

This order call returns as a code, 0 if the buffer is empt
error_table$devicedead if the call timed-out with no chang
buffer contents, and error_table$devicetimeout if the call
timed-out but the buffer contents changed. For example, to
wait for the output buffer to be empty you could use:

```
code = error_table$devicetimeout;
do while (code = error_table$devicetimeout)
     call iocs$order (iocb_ptr, "empty   ", addr(empty), c
end;
```

cur_pos performs cursor positioning as specified in a structure which is indicated by the info_ptr

```
dcl 1 info_structure based(info_ptr),
      2 opcode char(2),
      2 operand1 fixed bin,
      2 operand2 fixed bin;
```

opcodes are:

ho	set the cursor to the home position (1,1)
sl	set the cursor line to operand1
sc	set the cursor column to operand1
xy	sets the column to operand1 and the line to operand
rt	move the cursor to the next column
lt	move the cursor to the previous column
up	move the cursor up one line
dn	move the cursor down one line
cl	clears to the end of the current line
ce	clears from the current position to the end of the
cs	clears the entire screen (like ho,ce)
ct	clears the tab at the current position
st	sets a tab stop at the current position
ca	clears all tabs

As of February 21, 1978 NO window support exists.

makewind — makes a new window specified by the following structure:

```
dcl 1 info_structure based(info_ptr),
      2 name char(4),
      2 x_origin fixed bin,
      2 y_origin fixed bin,
      2 width fixed bin,
      2 height fixed bin,
      2 additional_info ptr; /* normally null() */
```

delwind — deletes the window indicated by the char(4) name at the info_ptr. The window " " is sacred to the system and cannot be deleted without tragic consequences. Attempting to delete it will result in a warning from the gods.

curwind — makes the designated window current (active) which means that all output, cursor positio
djs: help the compiler is stuck
inng and echoing
on the specified stream will occur within this window

getwind — fills in a window info_structure given info_structure.name

woodwind — makes beautiful mu
djs: I think that we found the problem
sic (not implemented yet)

As of February 21, 1978 gsti is used as opposed to modes.

```
iocs$modes entry(ptr,char(8),ptr,fixed bin(31))
call iocs$modes(iocb_ptr,modespec,mode_ptr,code)
```

where:

iocb_ptr	is an iocb pointer (see above)
modespec	is a specification of the action to perform (see table below)
mode_ptr	is a pointer to a structure used by the various actions (see below)
code	is a standard error code

Mode specifications work as follows:

modespec	action
getpage	fills in the following structure:

```
dcl 1 page_info,
      2 lines fixed bin,
      2 columns fixed bin;
```

setpage — sets the page and line length from the structure described for getpage

setflags — sets various bits describing the device modes as described in getflags (see below)

getflags — gets various bits describing the device modes into the following structure:

```
dcl 1 flags,
      2 c rrent bit(32),
      2 future bit(32); /* not implemented yet */
```

where the bits are as follows:

1	device in service (ro)
2	interrupt driven (ro)
3	handle_quit
4	wakeup for input (ro)
5	wakeup for output (ro)
6	output in progress (ro)
7	echo input
8	raw input
9	raw output
10	high speed (pasla only)
11	display (with cursor positioning)
12	ignore break (ro)
13	flush until input (ro)
14	more processing
15	reverse (underscore and delete)
32	shared bus device (ro)

```
/* start of ntty_info.incl.pll */

dcl 1 tty_info,
     2 version_number fixed bin init (0),
     2 devadd fixed bin (31),
     2 devname char (8),        /* device name */
     2 devtype char (8),        /* name of device type */
     2 page_size fixed,
     2 line_size fixed,
     2 padcrlf fixed(7),
     2 inhibit_break bit(1),   /* 3  */
     2 echo_input bit(1),      /* 7  */
     2 raw_input bit(1),       /* 8  */
     2 raw_output bit(1),      /* 9  */
```

```
      2 high_speed bit(1),      /* 10 */
      2 display bit(1),         /* 11 */
      2 more bit(1),            /* 14 */
      2 reverse bit(1),         /* 15 */
      2 allow_parity bit(1),    /* 17 */
      2 no_scroll bit(1),       /* 18 */
      2 no_tabs bit(1),         /* 19 */
      2 form_feed bit (1),      /* 20 */
      2 upper_case_only bit(1),/* 21 */
      2 ring_bell bit (1),      /* 22 */
      2 vis_bell bit (1),       /* 23 */
      2 lf_before_cr bit (1);   /* 24 */

/* end of ntty_info.incl.pll */
```

```
/* Begin iocb.INCL.PL1 */

dcl 1 iocb based,
     2 stream_name char(32) varying,
     2 open_modes bit (16),
               /* bit (0) = stream_input
                  bit (1) = stream_output
                  bit (15)= this stream is a syn attachment */
     2 real_iocb_ptr ptr, /* ptr to  next iocb for syn streams */
     2 effective_iocb_ptr ptr, /* ptr to the effective iocb (ie end of chai
     2 work_area_ptr ptr, /* set by io module for its own use */
     2 attach_descrip_ptr ptr, /* ptr to attach description */
     2 inhibited_operations bit (16),
               /* bit (0) = inhibit reads (if on)
                  bit (1) = inhibit writes
                  bit (2) = inhibit graphic_get operation
                  bit (3) = inhibit graphic_put
                  bit (4) = ditto for order op
                  bit (5) = ditto for modes operation */
     2 put_chars entry,
     2 get_chars entry,
     2 get_line entry,
     2 graphic_put entry,
     2 graphic_get entry,
     2 order entry,
     2 modes entry,
     2 detach entry,
     2 next_iocb_ptr ptr;
```

```
/* End iocb.INCL.PL1 */
```

IO utilities support
-- --------- -------

intended for rapid development of drivers and the like
when time prohibits immediate development of sysin
This support is also used quite a bit by users unfamiliar
with magicsix low level stuff.

```
dcl (io_util$wd,io_util$rd,
     io_util$oc,io_util$ss) entry(bit(16),bit(8));
dcl device bit(16); /* device address */
dcl param  bit(8);  /* Used as data,command,or status */

call io_util$xx(device,cmnd);
```

Ramtek Joystick

```
dcl joystick entry (fixed(31),fixed(31),fixed(31),fixed(31));
call joystick (x, y, z, code);
```

x and y range from approximately -100 to 100. z is 0 if Enter switch is depressed and -1 otherwise (corresponds to touching, and near field like a tablet) code is standard error code. In the event of no error, code is a number from 0 to 16 which describe the state of the four right switches.

The readable switches:
Visible, Blink, Track, and Enter.

The Enter switch is momentary and when depressed it halts the joystick hardware from sensing data.As a convenience, when the joystick is in this state, it will not touch the parameters x, y, z, or code. -- i.e. you will be able to know what they were last.

If you think of the code returned as a bit(4) number then Each switch corresponds to whether it is depressed or not. V, B, T are considered depressed if they are up. E is depressed if held down. i.e. 0 means none depressed, 8 means V depressed, etc.

Its not the best joystick in the world because of the oddball design. There is at most a latency of one second to get the correct value returned because the joystick was designed to interrupt the processor at a rate proportional to its deflection. Its minimum time is once / sec.

This is the old Ramtek joystick which used to be for manual control of tracking cursor has been hacked into a joystick for your use.

Send comments to b.

3/10/79.

The segment msm is a message segment manager for message queues. The following entry points are provided:

msm$send entry (ptr,char(10000)varying)

This entry takes a pointer to a message segment in the first argument position and a char varying message and sends the message via the pointed at message segment.

msm$get entry (ptr,fixed,char(10000)varying)

This entry takes a pointer to a message segment, a relative message number (the first message is number 1), and a string to return the message in, and returns the message_number'th message from the poi at message segment into the third arguement.

msm$delete entry (ptr)

This entry takes a pointer to a message segment and deletes the first message in the message queue.

msm$create entry (char(168)varying,char(32)varing,ptr,fix(31))

This entry takes a dirname, and entryname, a pointer and a code, and initiates (creating if neccessary) the given message segment, retu a pointer and a code from initiate/append_seg. This is the entry t called to get a pointer to a message segment and also to create a mess segment.

mos is a set of subroutines which are designed to make it easy to construct object segments for various purposes.

mos$make_object_segment creates the object segment, sets up the acl's and starts forming the template for the final object segment.

mos$pure and mos$impure allow data to be placed in the object segment, and mos$put_name allows entry points to be defined in the object segment.

mos$pure_block and mos$impure_block allow data to be stuffed into the object segment in big chunks

mos$cleanup finishes off the object segment and terminates everything which must be terminated.

```
mos$make_object_segment
     entry(char(168)vary,char(32)vary,fixed bin,fixed bin,bit(1),fixed
call mos$make_object_segment(dname,ename,pursize,impsize,relbits,code)
```

where:

dname — is the name of the directory in which the segment will be created

ename — is the name of the segment to create

pursize — is the number of bytes in the pure section of the object segment

impsize — is the number of bytes in the impure (linkage) section of the object segment

relbits — tells whether to generate relocation bits or not

code — is set to a standard error code

```
(mos$pure,mos$impure) entry(fixed bin,bit(16),bit(1),bit(1))
call mos$xxxxx(offset,data,purrel,imprel);
```

where:

offset — is the offset into the appropriate section at which to place the data

data — is the data to put at the specified location in the object segment

purrel — tells whether the data is a value relative to

the pure section of code

imprel tells whether the data is a value relative to the impure section of code

```
(mos$pure_block,mos$impure_block) entry(fixed bin,ptr,fixed bin)
call mos$xxxxx_block(offset,data_ptr,data_len);
```

where:

offset is the offset into the section of the object segment at which to put the data

data_ptr points at the data to be put into the object segment

data_len is the number of bytes to be put into the object segment

```
mos$put_name entry(char(32) vary,fixed bin)
call mos$put_name(ename,offset);
```

where:

ename is the name of the entry to be defined in the object segment

offset is the value (relative to the base of the segment) at which the entry is to be defined

```
mos$cleanup entry
call mos$cleanup;
```

The segment math contains a package of mathematical number-crunching r

They include: sqrt, exp, log, log10, sin, cos, asin, acos, atan, tanh,
atan2, random, and seed

those listed on the first line are declared like:
"math$sqrt entry (flt) rtns (flt)"

atan2 takes 2 args, and works like: atan(a/b)=atan2(a,b)
the advantage of atan2 over atan is that it doesn't barf if b=0, and
whereas -pi/2 <= atan <= pi/2, atan2 can be as big as pi or as small a
depending on the quadrant. "dcl math$atan2 entry (flt,flt) rtns (flt)

random and seed are parts of a random number generator. type "help ran

math$log10 is base 10, math$log is base e

ph, 3/78

math$random and math$seed -- Very Random Number Generator

declarations:

```
dcl math$random entry rtns(flt);
dcl math$seed entry (fix(31));
dcl r flt;
dcl johnny_apple fix(31);
```

usage:

to get a random number: "r=math$random();"
to set the seed: "call math$seed(johnny_apple);"

The random numbers are uniformly distributed between 0 and 1.
The period of repetition is 2^31.

The user has three options in regard to seeding: he can do nothing, in
which case the ranumber sequence will continue where it left off last
was called, it can be given a nonnegative seed, in which case that num
be used to generate the rest of the sequence, or it can be given a neg
seed, in which case the clock will be used for seeding. If math$seed
called, the first call to math$random in a process seeds from the cloc

The random numbers are generated using the linear congruential method.
See page 949 of The Handbook of Mathematical Functions by Abramowitz a

Paul Heckbert, 3/24/78

mt_io is an input/output dim for use with the magnetic tape drive. It parses its attach description as follows:

```
mt_io device options
```

Where the device is usually the tape drive "tape"

The options include:

-hbd -hdb	since some tape drives don't know how to recognize end of file marks this option makes the software perform a heuristic check for eof's
-block # -b #	sets up the drive for blocked input or output (but not both) with a block of the specified size. Blocks on the tape must actually be the specified size although the dim will chop off data (or buffer up and write out) data as needed. The output buffer will be flushed when the stream is detached or certain I/O orders are performed.
-retries # -r #	sets the number of times to retry in the event of an I/O error, if set to zero, no retry is performed. After most errors the tape is left at the end of the error block.

Order calls supported by the tape dim are:

eof	write an eof mark
rewind	rewinds the tape
backspac	backspaces one record
backfile	backspaces one file
skipfile	skips over one file

NOTE: The entry point: mt_io$losing is designed to compensate for the hardware's apparent inability to handle the skipfile order since the drive cannot always detect ends of file. The bit set by this entry causes the skipfile to be done in software complete with the hairy eof heuristics.

```
dcl name_util$copy_names entry(char(168)vary,char(32)vary,
                               char(168)vary,char(32)vary,fixed bin(31));
call name_util$copy_names(dname1,ename1,dname2,ename2,code);
```

copies all of the names from one segment to another

This utility simply tries to add each name of the first entry to the second entry. If it fails because the name is already in the directory it will keep trying to add the other names. Note that you cannot copy the names from one entry in a directory to another in the same directory. To do this you need to use name_util$move_names.

where:

dname1	is the directory containing the first entry
ename1	is the name of the first entry
dname2	is the directory containing the second entry
ename2	is the name of the second entry
code	is set to a standard system error code

```
dcl name_util$move_names entry(char(168)vary,char(32)vary,char(32)vary,
                         char(32)vary,char(168)vary,char(32)vary,fixed bin(31
call name_util$move_names(dname1,ename1,ioaname,nname,dname2,ename2,code
```

moves the names from the first entry to the second one

This utility goes through each name of the first entry and removes it from the first entry and adds it to the second one. If the name is already in the target directory (after it has been removed from the first entry) it will just go on to add the next name. Thus, it is possible to move all of the names from one segment to another segment in a given directory.

Since this entry point will remove all of the names on the first entry a residual name must be specified. This name will be the name left on the first entry. If the name contains no carets ("^"'s) it will simply be added to the first segment before its names are removed. If the name is already in the directory it may not be possible to move all of the names. If the name contains a caret it is assumed to be an ioa string and will be passed a number from 1 to 99 to generate a name until the name can be added to the first segment. Thus, if the string is "old.^i" move_names will first try to add the name old.1, then old.2 and so on until one of them wins. The name actually added is returned to the calling program.

where:

dname1	is the directory containing the first entry
ename1	is the name of the first entry
ioaname	is the ioa string used to derive the residual name

```
 nname     is set to the actual residual name
 dname2    is the directory containing the second entry
 ename2    is the name of the second entry
 code      is set to a standard system error code
```

```
dcl name_util$rename
     entry(char(168)vary,char(32)vary,char(32)vary,fixed bin(31));
call name_util$rename(dname,oname,nname,code);
```

renames the specified entry

```
where:
 dname     is the directory in which the entry to rename is found
 oname     is the old name of the entry
 nname     is the new name of the entry
 code      is set to a standard system error code
```

```
dcl nrfs$append_name entry (char(168) vary, char(32) vary, char(32) vary,
     fix(31));
call nrfs$append_name (dname, ename, nname, code);
```

where:
 dname is the name of the containing directory
 ename is the name of the entry
 nname is the name to add to the entry
 code is a standard error code

```
dcl nrfs$delete_name entry (char(168) vary, char(32) vary, fix(31));
call nrfs$delete_name (dname, ename, code);
```

where:
 dname is the name of the containing directory
 ename is the name of the entry to be deleted
 code is a standard error code

```
dcl nrfs$get_link_target
        entry(char(168)vary,char(32)vary,char(168)vary,fixed bin(31
call nrfs$get_link_target(dname,ename,target,code);
```

This entry gets the target of the specified link, where:

dname	is the name of directory containg the link
ename	is the entry name of the link
target	is set to the target of the link
code	is set to a standard system error code

```
dcl nrfs$set_acl entry (char(168) vary, char(32) vary, bit(16), fix(31));
call hcs$set_acl (dname, ename, acl, code);

where:
 dname is the name of the containing directory
 ename is the name of the entry to be modified
 acl  is the access bits as follow:
     byte 1 is the access when running in the system domain
     byte 2 is the access when running in the user domain
          within each byte:
          bit 0: system will take a fault when segment is executed
          bits 1-2: 00=do not take write faults, 11= take write faults
          bit 3: set after system copies linkage section on a fault
          bit 4: segment is a gate and therefor requires gate actions
          bit 5: segment can be read
          bit 6: segment can be executed
          bit 7: segment can be written
 code  is a standard system error code.



dcl nrfs$set_domain entry (char(168) vary, char(32) vary, fix, fix(31));
call nrfs$set_domain (dname, ename, ring, code);

where:
 dname is the name of the containing directory
 ename is the name of the entry
 ring is the new ring of the entry must be 0 or 1
 code is the standard error code



dcl nrfs$set_dtb entry (char(168) vary, char(32) vary, bit(64), fix(31));
call nrfs$set_dtb (dname, ename, dtb, code);

where:
 dname is the name of the containing directory
 ename is the name of the entry
 dtb is the time value to which the date_time_backed up is to be set.
 code  is a standard error code



dcl nrfs$set_dtc entry (char(168) vary, char(32) vary, bit(64), fix(31));
```

```
call nrfs$set_dtc (dname, ename, dtc, code);
```

where:
dname is the name of the containing directory
ename is the name of the entry
dtc is the time value to which the date_time_created up is to be set.
code is a standard error code

```
dcl nrfs$set_name entry (char(168) vary, char(32) vary, char(4), fix(31))
call nrfs$set_name (dname, ename, sname, code);
```

where:
dname is the name of the containing directory
ename is the name of the entry
sname is the name of the address space in which the segment should run
code is the standard error code.

```
dcl nrfs$set_maxlength entry (char(168) vary, char(32) vary, bit(16), fi
call nrfs$set_maxlength (dname, ename, maxlength, code);
```

where:
dname is the directory containing the entry
ename is the ename of the entry
maxlength is the value to set the maxlength to
code is a standard system error code

```
dcl nrfs$status entry (char(168) vary, char(32) vary, bit(16), ptr, ptr,
     fix(31));
call nrfs$status (dname, ename, flags, info_ptr, name_area_ptr, code);
```

where:
dname is the name of the containing directory
ename is the name of the entry
flags are cotrol bits specifying how much infomation is returned as fol
"0000"b4 returns the minimum info.
"8000"b4 looks in the vtoc for the length and bit_count of the entr
"4000"b4 returns various worthless bits of info like dtc and sname.
"2000"b4 returns the xinfo stuff with validation, safety switch, et
info_ptr is a pointer to a structure in which the info is to be returne
name_area_ptr is a ponter to and area in which all the names of the ent
returned. If null() no names are returned.
code is a standard error code

```
declare
    1 info,
      2 type bit(16),
      2 num_names fixed,
      2 name_list offset(info_area),
      2 mode bit(16),
      2 domain fixed bin(15),
      2 index_in_vtoc fixed,

/* Bit one (1) of the flags argument should be set if the length and
   bit_count are to be looked up in the vtoc.   */

      2 records fixed bin(31),
      2 bit_count fixed bin(31),

/* This is the end of the structure if the standard status call is made.
   The rest is only returned if the long bit of the flags argument is se

      2 space_name char(4),
      2 dtc bit(64),   /* used as unique index */
      2 dtm bit(64),
      2 dtu bit(64),
      2 dtb bit(64),

    1 link_info defined info,
      2 skjskhsaslkjho char(6),
      2 target offset(info_area),

    1 xinfo defined info.dtu,
```

```
2 safety_switch bit(1),
2 validation bit(1),
2 maxlength bit(16),
2 mounted bit(1),
2 volume bit(16),
2 pad bit(8);
```

```
dcl nrhcs$setih entry(char(8),char(32) varying,ptr,fixed bin(31));
call nrhcs$setih(device, handler, ip, code);

dcl nrhcs$mobyih entry(char(8),char(32) varying,fixed bin,ptr,fixed bin(
call nrhcs$mobyih(device, handler, nhandlers, ip, code);
```

sets up an interupt handler for the device(s) specified

where:

device	is a standard device name
handler	is the name of an interupt handler (see below)
nhandlers	is the number of contiguous devices which use the same handler and data base pairs
ip	is set to a pointer to the impure section of the handler
code	is set to a standard system error code

```
dcl nrhcs$clrih entry(char(8),fixed bin(31));
call nrhcs$clrih(device,code);
```

clears an interupt handler (see description above for arguments)

An interupt handler is an ordinary piece of code which must be (as must all Magic Six programs) location insensitive and typically pure. The program itself may be shared if another device uses the same handler. This sharing is done by checking for the uniqueness of the first eight letters of the handlers name so keep the signifigant porions small. The impure (or static) section of the program is unique for each setih call. The handler is arrived in with the following registers:

0:	old program status word
1:	old program counter
2:	device number
3:	device status (this is read by microcode for you)
12:	pointer to the static data so you can take advantage of midi's automatically indexing it

The handler runs in ultra-privileged mode with mapping off(!), in register set zero, and with interupts masked. Do not spend too much time in an interupt handler or the system may die or appear to die.

The interupt handler must be a normal Magic Six object segment which is small enough to fit into system free storage. If the routine has an entry named "cleanup" then that entry will be called (return address in register 14) before the handler is cleared.

Ramtek Software and Documentation

This is a general overview of the Ramtek system on MagicSix. This guid indicates where the code is located and some notes on how to use it as give pointers to the detailed documentation.

Ramtek code:

Ramtek code is in the graphics directory >g. It is in the segment "ram calls to Ramtek routines are of the form ram$<entry name>. All Ramtek subroutines run in the address space ram.

To use the Ramtek routines, add >g to your search rules.

Ramtek documentation:

Documentation for Ramtek routines is accessed via the help command. Hel ramtek gets you this stuff. Help <any of the below> gets you informati on a category of routines. Help ram$<routine name> gets you informatio on the category of routines.

ram_cursor	cursor related routines cursor, refcur
ram_driver (ram_ramt)	very low level drivers ram_device, ram_read, ram_write, ram_driver$ram_time ramdu, timeout (conditions)
ram_font (ram_text)	programmable font related routines font_fileload, fontload, init_text, load_standard_font, write_text
ram_line	routines that draw lines dejaggied_line, line, mline, rel_vector
ram_matrix	color matrix manipulation routines color, defcm, grayscale, refcm, uncolor
ram_micro	microcode manipulation and drivers dashed_vector, flood, patterned_vector, ram_call, ram_lo
ram_param	parameter setting routines background, dimension, flashing_window, foreground, handshake, init, mask, reset, scale, scan, spacing, startpt, wi_function, window

ram_rect	routines that fill rectangles clear, erase, mrect, open_rectangle, rect
ram_rw	routines that read and write pixels read_block, read_image, read_line, write_block, write_ima write_line
ram_shapes	complex shape generators filled_circle
ram_sline	routines that draw lines and take structures as paramete sdejaggied_line, sline, smline
ram_srect	routines that fill rectangles and take structures as parameters: smrect, sopen_rectangle, srect
ram_srw	routines that read and write pixels and take structures parameters: sread_block, swrite_block
ram_commands (ram_cmnds) (rcommands)	documentation on the Ramtek commands that are found in >

A brief note on the subroutine documentation. A <r> after an argument that the value of that argument is changed by the routine. Also, the "changed" section tells which (if any) of the Ramtek instruction operands (parameters) are changed by the routine.

Contrary to popular belief, there is essentially no difference in execu speed between the structure and the parameter versions of these subrout The choice to use one or the other should be made solely on readability ease of programming criteria.

ring0_wakeups is the general purpose scheduling hack for use by programs in the user ring which need to wait a specified amount of time, for a device to change state, or for a value to change

```
ring0_wakeups$signal(flavor,info_ptr,condition,handle,code)
dcl ring0_wakeups$signal entry(fixed bin,ptr,char(8),ptr,fixed bin(31))
```

where:

flavor — is the kind of wakeup:
- 0 timer
- 1 device status
- 2 value change (in segment 0 or 1)
- 3 value change (to a value) (in segment 0 to 1)

info_ptr — points at a structure of information needed for the specified kind of wakeup:
- 0 system up time (bit(64)) when timer should go
- 1 device number bit(32)
  mask for status bit(32)
  status to change from bit(32)
- 2 value to change from bit(32)
  mask for value bit(32)
  address of value to change ptr
- 3 value to wait for bit(32)
  mask for value bit(32)
  address of value to wait for ptr

condition — is the name of the condition to signal when the wake conditions are satisfied
if this is "alarm" then the set_entry hack will be supported by the process overseer to provide a standard call on a particular condition

handle — points at the wakeup block created by ring0_wakeups and is effectively the caller's handle on this wakeup

code — is a standard system error code

```
ring0_wakeups$sleep(flavor,info_ptr,handle,code)
dcl ring0_wakeups$sleep entry(fixed bin,ptr,ptr,fixed bin(31));
```

where all of the arguments are as those for ring0_wakeups$signal except that these conditions are masked until ring0_wakeups$catnap is called to put the process to sleep.

```
ring0_wakeups$dismiss
dcl ring0_wakeups$dismiss entry;
```

This routine removes the current wakeup from the chain of wakeups being tested for and reenables the wakeup mechanism. This entry must be called before any other (possibly) pending wakeups occur.

```
ring0_wakeups$reset(handle)
dcl ring0_wakeups$reset entry(ptr);
```

where:
handle is the pointer returned by either:
ring0_wakeups$signal or ring0_wakeups$sleep

ring0_wakeups$reset deletes the request specified from the list of active requests

```
ring0_wakeups$reset_all
dcl ring0_wakeups$reset_all entry;
```

ring0_wakeups$reset_all resets all pending conditions and dismisses any pending interupts

```
ring0_wakeups$catnap
dcl ring0_wakeups$catnap entry;
```

This entry enables all sleep type entries created so far and then goes to sleep. The process will be woken by either a break, or any standard wakeup going off.

```
ring0_wakeups$get_entry(entry)
dcl ring0_wakeups$get_entry entry(entry);
```

This entry point is known to the process overseer as timer_manager$get_ and probably will be until the names are changed.

where:
entry is set to a place to call to service the currently pending wakeup

```
ring0_wakeups$set_entry(handle,addr(entry))
dcl ring0_wakeups$set_entry entry(ptr,ptr);
```

where:
handle is as above
entry is an external entry point declared options(float)

For general convenience and to replace timer_manager there is an entry point which sets up a call to an entry at some time in the near future.

```
ring0_wakeups$timer(usecs,entry,handle);
dcl ring0_wakeups$timer entry(bit(64),entry,ptr);
```

where:

usecs	is the number of microseconds to delay
entry	is the entry to call
handle	is either a standard handle or null()

There is also an entry to pause for a given number of microseconds.

```
ring0_wakeups$pause(usecs);
dcl ring0_wakeups$pause entry(bit(64));
```

where:

usecs	is the number of microseconds before this entry returns

NOTE: To get an approximate number of seconds e.g. x use:
"0000000000x00000"b4

```
dcl sas$nasa entry(char(4),ptr,char(4),ptr,fixed bin(31));
call sas$nasa(ospace,op,nspace,np,code);
```

maps the specified segment into the specified space

where:

ospace	the space to snarf the segment from
op	a pointer (in ospace) to the segment
nspace	the space to move the segment to, ==== is the current one
np	is set to point at the segment in this space
code	is a standard system error code

```
dcl sas$get_line_number entry(ptr,fixed bin,fixed bin,fixed bin(31));
call sas$get_line_number(sp,lineno,lineofs,code);
```

given a pointer into an object segment this returns the source line number and the offset of the start of the line

where:

sp	is a pointer into the segment
lineno	is set to the line number
lineofs	is set to the offset of the line in the segment
code	is a standard system error code

```
dcl sas$getedp entry(ptr,fixed bin(31),ptr,fixed bin,fixed bin(31));
call sas$getedp(op,bc,ep,dif,code);
```

gives the name of the entry point nearest the specified pointer

where:

op	points at the object segment
bc	is the bit count of the object segment
ep	is set to point at the entry name (length,string)
dif	is the distance from the entry to the ptr
code	is a standard system error code

```
dcl sas$validp entry(ptr) returns(bit(1));
if sas$validp(p) then p-> ...
```

tells if a pointer is safe to reference to

where:

p	is the pointer to check out

```
dcl sas$eacalc entry(ptr,ptr,ptr);
call sas$eacalc(ip,sp,eap);
```

calculates the effective address of an instruction given a pointer to the instruction and a stack frame containing the registers to use

where:

ip	is a pointer to the instruction
sp	is a pointer to the stack frame
eap	is set to the effective address of the instruction

```
dcl sas$term_ entry(char(4),ptr,fixed bin(31));
call sas$term_(space,sp,code);
```

```
dcl sas$term_elsewhere entry(char(4),ptr,fixed bin(31));
call sas$term_elsewhere(space,sp,code);
```

terminates all instances of the specified segment in the current process and frees their linkage sections. The segment is identified by a space name and a pointer. sas$term_elsewhere is like sas$term_ except that it terminates all instances except the one in the initial space specified.

where:

space	is the name of the space
sp	is a pointer to the segment in that space
code	is set to a standard system error code

```
dcl scs$add entry(bit(64), bit(64), bit(64));
call scs$add (op1,op2,sum);
```

- double precision add

```
where:
 sum = op1+op2;
```

```
dcl scs$subtract entry(bit(64),bit(64), bit(64));
call scs$subtract (op1,op2,difference);
```

- double precision subtract operator

```
where:
 difference = op1 - op2;
```

```
dcl scs$multiply entry(bit(32), bit(32), bit(64));
call scs$multiply (op1,op2,product);
```

- semi-double precision multiply

```
where:
 product = op1*op2;
```

```
dcl scs$divide entry (bit(64), bit(32), bit(32));
call scs$divide(dividend,divisor,quotient);
```

- semi-double presision divide

```
where:
 quotient = dividend / divisor;
```

```
dcl scs$allocn entry (fixed(31), pointer, area);
call scs$allocn (size, block_ptr, foo_area);
```

where:
 size is the number of bytes to be allocated
 block_ptr is the ptr returned to the allocated block
 foo_area is the area in which to allocate the block

```
dcl scs$freen entry (fixed(31), pointer, area);
call scs$freen (size, block_ptr, foo_area);
```

where:
 size is the number of bytes to be freed
 block_ptr is the ptr to the block to be freed
 foo_area is the area in which the block was allocated

```
dcl scs$date_time entry(bit(64), char(40) varying);
call scs$date_time(dtm, dt_string);

dcl scs$date_time_short entry(bit(64), char(15) varying);
call scs$date_time_short(dtm, dt_string);
```

where:
  dtm is a standard time value such as those in the file system. If zero
     the current time is used.
  dt_string is returned as the string giving the time in character form.

```
dcl scs$hour_min entry(char(8));
call scs$hour_min (time_string);
```

where:
 time_string is the current time in character string format: "hh:mm:ss"

```
dcl scs$ch_wdir entry(char(168) varying);
call scs$ch_wdir(wname);
```

where:
 wname is the name of the new working directory

```
dcl scs$cl entry options(variable);
call scs$cl(command_line);
```

where:
command_line is a string which is parsed by the command processor

```
dcl scs$expand_path entry(char(168)varying,char(168)varying,char(32)va
call scs$expand_path(pname,dname,ename,code);
```

where:
 pname is an absolute or relative pathname
 dname is the directory part of pname
 ename is the entry part of ename
 code is a standard error code

Name: scs$get_arg_count

Function:

Returns the number of arguments a routine was called with.

Usage:

```
dcl scs$get_arg_count entry (fix);
call scs$get_arg_count (nargs);
```

Where:

nargs is the number of arguments (output)

Name: scs$get_arg_info

Function:

Returns a pointer to a given argument and the associated descriptor information.

Usage:

```
dcl scs$get_arg_info entry (fix, bit(16), fix, ptr);
call scs$get_arg_info (argno, type, al, ap);
```

Where:

```
argno     is the number of the argument (input)
type      is the descriptor for the argument (output)
al        is the length of the argument (output)
ap        is the pointer to the argument (output)
```

type may be interpreted as follows:

```
bit 2     specifies if the argument is varying
bit 3     specifies if the argument is an array
bit 4     specifies if the argument is aligned
bits 5-8  specified the type of the argument as follows:
          "0001"b   fixed binary
          "0010"b   float binary
          "0011"b   bit
          "0100"b   character
          "0101"b   label
          "0110"b   pointer
```

```
"0111"b   area
"1000"b   offset
"1001"b   entry
"1011"b   structure
all other types are in error
```

```
dcl scs$get_caller_space entry(char(4));
call scs$get_caller_space(space);
```

returns the space of the caller in the variable space

```
dcl scs$get_search_rules entry(ptr);
call scs$get_search_rules(rp);
```

where:

rp is a pointer to a search rule structure (see SRULES.INCL.PL1)

```
/* Begin SRULES.INCL.PL1 */
dcl 1 srule based,          /* This is where it is at */
     2 flags bit(32),       /* Flags for a search rule:

                               bit 0 = this entry is valid (in use)
                               bit 1 = this is the working dir */
     2 next_ptr ptr, /* ptr to next s rule in chain */
     2 path_name char (168) varying;

dcl 1 workdir based,
     2 path_name char(168) varying;

dcl 1 search_rules based,     /* this is where it is all at */
     2 search_rule (4) like srule;

dcl 1 homedir based,
     2 path_name char(168) varying;
```

```
dcl scs$get_wdir entry(char(168) varying);
call scs$get_wdir(wname);
```

where:
wname is the name of the current working directory

```
dcl scs$get_uname entry(char(8) varying);
call scs$get_uname (uname);
```

where:
uname is returned as the name of the person logged in.

```
dcl scs$make_ptr entry (char (32) varying, char (32) varying, ptr, ptr,
                        fixed bin (31));
call scs$make_ptr (ename, epname, ref_seg, seg_ptr, code);
```

where:

```
ename   is the entry name of a segment (input)
epname  is an entry point name in segment ename (input)
ref_seg is the referencing segment (input)
seg_ptr is the pointer to the entry (output)
code    standard system error code (output)
```

make_ptr is used to find and initiate a segment by means of the search rules. It is the program used by the system during dynamic linking. In general make_ptr first finds a segment by means of the search rules and then looks for the entry point epname in it by searching the segments object map. If epname is null then the obj map is not searched and a point to the base of the segment is returned. The ref_seg argument is a pointer a segment to be searched for epname before searching for segment ename. (A used in the linker when snaping a link to $foo.) Normally ref_seg is null. IE. If you don't know what ref_seg is, use null! You can bet what code is.

```
scs$make_space_ptr
     entry(char(32)vary,char(32)vary,char(4),char(4),ptr,ptr,fixed bin(31)
call scs$make_space_ptr(ename,epname,space,nspace,refptr,segptr,code);
```

using the current search rules, this entry makes a pointer to the appropriate entry point (ename$epname) in the space specified by space.

where:

```
 ename    is the reference name of the segment to look for
 epname   is the entry point in the above segment
 space    is the space in which to initiate the segment
              if space="...." then the segment will be
                   initiated in the space indicated by the file system
                   if the space name is not ^@^@^@^@ then the segment
                   will be initiated in the caller's address space
              if space="====" then the segment will be
                   initiated in the space of the caller
 nspace   is set to the space in which the segment is actually initiated
 refptr   is the "reference" pointer, if ename="" then first make_ptr
              tries to find epname in the segment indicated by refptr,
              then it tries to find epname$epname
 segptr   is set to a pointer to the designated entry
 code     is set to a standard error code
```

```
dcl scs$find_gate entry(char(32) varying,ptr,fixed bin(31));
call scs$find_gate(rname,sp,code);

dcl scs$find_gate_space entry(char(32) varying,char(4),char(4),ptr,
                    fixed bin(31));
call scs$find_gate_space(rname,wspace,nspace,sp,code);

where:
 rname    is the reference name to look for
 wspace   is the desired address space
              "====="    indicates the callers space
              "     "    indicates any space is good (never maps)
              "...."     is like "===="
 nspace   is set to space the segment is actually in
 sp       is set to a pointer to the segment
 code     is a standard system error code

scs$find_gate looks in all spaces other than the current
one and maps the segment with the specified reference name
into the current address space and returns a pointer to it

scs$find_gate_space looks in all the spaces and:

if wspace="     " it returns the space it found the segment
in and the pointer there to it

otherwise if the segment has a file system space name then
the segment is left in the space it was found in and that
space name and corresponding pointer are returned, if the
segment has no space name (a space name of ^@^@^@^@) then
it is mapped into the specified address space as for
scs$find_gate



dcl scs$make_static_variable entry(char(32)varying, char(32) varying, fix
call scs$make_static_variable (refname,data_name, size, code);

 - this entry allow data declared external to have space allocated for th
 - they are referenced as <refname>$<data_name>

where:
 refname is the name to be added to te linkage section
```

data_name is the name of the data to be used
size is the size of <data>
code is a standard error code.

```
dcl scs$make_static_external entry(char(32) varying, fixed, ptr, fixed,
call scs$make_static_external (epname,ring,data_ptr,size,block_ptr,code)
```

- this is the generalized version of make_static_variable
- Any data anywhere can be make referencable by using this routine
  (i.e. the data segment doesn't need an entry map)

where:
epname is the name of the data area to be created
ring is the ring of the program for which the data is to be created
data_ptr is a pointer to the data if known. if null() then data will be
size is the amount of space to be allocated
block_ptr is returned as a ptr to the data
code is a standard error code

(see also:scs$set_search_rules)

```
dcl scs$obj_map entry(pointer,fixed bin(31)) returns(pointer);
op=scs$obj_map(sp,bc);
where:
 sp    is a pointer into the segment
 bc    is the bit count of that segment
 op    is a pointer to the start of the definitions section of sp
```

```
dcl strip_entry_ (char (32) varying, char(32) varying, char (32) varyin
call strip_entry_ (ename, what_to_strip, stripped_ename);
```

Subroutine removes the suffix "." || what_to_strip from the end of ename if it exists and returns the stripped form in stripped_ename. For example:

```
call strip_entry_ ("foo.pl1", "pl1", stripped_entry);

stripped_entry = "foo".

call strip_entry_ ("foo", "pl1", stripped_entry);

stripped_entry = "foo.".
```

Name: tablet

Function:

This called used to be used for reading points from the summagraphics tablet. The graphics system has really replaced this call. Type "help gs" for information. This call is being kept around for your convenience.

Usage:

```
declare tablet entry (fix(31), fix(31), fix(31), fix(31));
call tablet (x, y, z, code);
```

Where:

x is the UNSCALED x coordinate
y is the UNSCALED y coordinate
z is the z coordinate
    +n is the pressure (if the tablet returns the pressure)
    0 means that the pen is touching and pressing
    -1 means that the pen is touching but not pressing
    -2 means that the pen is in the "near field" of the tablet
    -3 menas that the pen is in the "far field" fo the table
        and that the values returned for x and y are probably
        batshit.
code is a standard error code
    0 if everything went ok
    -251 if the tablet was du (off or non-existent)

Notes:
    If you're using the big tablet, it returns x in the range (0:4799)
    y in the range (0:3599). Type 'help tab' for a more specialized ta
    routine which is designed for the big tablet and ramtek.

See RK with any complaints or desires.

Name: tsd

Function:

This called used to be used for reading points from the new elographics interface.

Usage:

```
declare tsd entry (fix(31), fix(31), fix(31), fix(31));
call tsd.(x, y, a, code);
```

Where:

x is the UNSCALED x coordinate
y is the UNSCALED y coordinate
a is the tsd number that produced the point
code is a standard error code
    0 if everything went ok
    -251 if the tablet was du (off or non-existent)

Notes:

See >u>s>elo.info for more information on the interface.
See RK with any complaints or desires.

VIDEO DISK SUPPORT (N.B. Be sure you look at video$pause below.)

The segment video_disk (or video for short.) contains our library of videodisk programs. It is in >g. To use it you need >g in your search ru

Note: these may also be called by the entry video$play_frame, etc.
Other relevent commands: help video.

```
dcl video_disk$play_forward entry(fixed,fixed);
call video_disk$play_forward(number,time);
```

This routine plays the disk at user controlled rates in milliseconds/fra forward. It is the time in between frames. You must position the video_disk first by calling video_disk$search_frame first.

number is the number of frames to play.
time is the time in milliseconds.

```
dcl video_disk$play_reverse entry(fixed,fixed);
call video_disk$play_reverse(number,time);
```

Same as above but backward instead.

```
dcl video_disk$search_frame entry(fixed(31));
call video_disk$search_frame(frame)
```

This will seek for the disk frame and display it.

```
dcl video_disk$auto_stop entry(fixed(31));
call video_disk$auto_stop(frame);
```

Directs the disk to play at 30 frames/sec until it gets to the specified frame. Then it will stop. If the current frame number exceeds the frame number passed, a search will be done. The user can always bbe sure that at the end of the operation, the frame last displayed will be the one passed.

```
dcl video_disk$pause entry (fixed);
call video_disk$pause (time_to_pause);
```

This entry will pause the user's process for the specified number of milliseconds. This entry was needed because of a known and hard to fix bug of the MCA. There is always a finite amount of time needed to search and also to auto_stop. Unfortunately when future commands are sent to the MCA they will immediately overide whatever the MCA is doing (Hardware lossage by MCA) So if you search or auto_stop then your software will probably want to wait (This wait time is a function of how bad the disk is warped and how hot the MCA is --really!)

This entry may not be needed for the Magnavox disk.

```
dcl video_disk$set_times entry (fixed(31), fixed(31));
call video_disk$set_times (tmin, tmax);
```

This entry will allow the user to set the maximum search times for searches between 0-5000 frames and 5000-50,000. These will overide the internal used by the driver by default. By default they are 2000 and 10000 (2 and 10 seconds) Performance may very well degrade by setting these constants to smaller values. You may notice that you will have to send a stop command after every search so that you can overide the search seek if it is still in progress.

The following routines are available for munging with the inquire data base.

```
dcl user_info$extract_topic entry (char(32) varying, char(32)varying, pt
    fixed, fixed, fixed(31));
call user_info$extract_topic (uname,topic_name,addr(buffer),buflen,
    size,code);
```

The specified topic is read into the buffer. buflen is the length o buffer and size the a accual size of the topic's text. Code is the usual thing.

```
dcl user_info$find_topic entry (ptr,char(32)varying, ptr,fixed,fixed(31)
call user_info$find_topic (info_seg_ptr,topic_name,topic_start,size,code
```

This routine takes a pointer to an inquire data segment and returns a pointer to the start of the specified topic in that segment. size is the length of the data starting at topic_start and code is usual.

```
dcl user_info$replace_topic entry (char(32)varying, char(32)varying, buf
    fixed,bit(1),fixed(31));
call user_info$replace_topic (uname,topic_name,addr(substr(new_topic,1,1
    len,cinf,code);
```

This replaces the old value of the specified topic with the string supplied. If the specified topic is not found then the flag cinf is queried and if true the topic is created.