

The Very First Real Published (Almost) Magic Six Subroutines
Manual --- February 21, 1973

Revision ZERO (No revisions have been made, this is the
stuff the computer dumped out)

Since people are starting to use MAGIC Six and are printing out
help files I figured I would send the more or less most printed
ones down to be Xeroxed. So here they are straight off the Bright
and in cruddy condition. Things may be missing the right ends of
lines but all of this stuff is accessible by typing help topic to
the system or printing the appropriate file in greater-than-doc.
For info on ask try the old PL/I run time manual. Things get out
of date quickly so if it flows out check the latest or look in one
of the log mailboxes.

Additional ask entry points.

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_parse entry options(variable)
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>

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

```
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>.",crane,erane);
```

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

Get and Set Tty Info (gsti)

This program has three entry points which make the examination and modification of device parameters very easy.

```
dcl gsti$get_tty_devtab entry (pointer,pointer);
call gsti$get_tty_devtab (linkage$user_output,cevtab);
```

This routine returns the pointer to the device table for a certain io stream. The structure this pointer points to is in >i>devstruct.incl.pll.

```
dcl gsti$get_tty_info entry (pointer,structure);
call gsti$get_tty_info (linkage$user_output, tty_info);
```

This routine fills in the structure which infomation about the device attached to the io stream. The structure is in >i>tty_info.incl.pll.

```
dcl gsti$set_tty_info entry (pointer,structure);
call gsti$set_tty_info (linkage$user_output, tty_info);
```

This routine copies the user setable information out of the structure into the device table for the device attached to the io stream. It is recommended that you call get_tty_info, modify the bits you want to change and then call set_tty_info.

```
dc) | tty_info,
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 padctrlf fixed(7),
2 for_later_expansion1 fixed(31),
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 for_later_expansion2 fixed(3);
```

hcs\$add_ref_name

07/05/77 09:45:16

269

```
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

hcs\$append_seg

07/05/77 09:46:02

.354

```
cc1 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

hcs\$append_dir

07/05/77 09:46:17

.358

```
cc1 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

hcs\$append_link

07/05/77 09:46:32

412

```
cc) hcs$append_link entry(char(168)varying,char(32)varying,char(168)varying,fixed(31));
call hcs$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

hcs\$get_bit_count

07/05/77 09:41:52

267

```
ocl hcs$get_bit_count entry(pointer, fixed bir(31));
call hcs$get_bit_count(sp,bc);
```

where:

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

hcs\$get_sdb

07/05/77 09:47:33

262

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

where:

segp is a pointer to a segment
sdhp is a pointer to the sck for that segment (or null())

hcs\$get_seg_ptr

07/05/77 09:41:38

284

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

where:

sp is the pointer to the segment if it exists
iname is the reference name to search for

hcs\$get_seg_size

07/05/77 09:44:46

301

```
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)

hcs\$initiate

07/05/77 09:40:34

407

```
ccl hcs$initiate entry(char(168) varying,char(32) varying,ptr,fixed bin(31));
call hcs$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 where it is initiated
code is a standard error code

hcs\$initiate_w_options

07/05/77 09:40:53

626

```
dcl hcs$initiate_w_options
    entry(char(168) varying, char(32) varying, char(32) varying, bit(1),
          ptr, fixed bin(31));
call hcs$initiate_w_options(dname,ename,refname,place,ptr,ccce);
```

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 specified by ptr

ptr is the ptr to the segment when it is initiated

code is a standard error code

hcs\$make_seg

07/05/77, 09:45:01

223

```
cc1 hcs$make_seg entry(pointer);
call hcs$make_seg(sp);
```

where:

sp is a pointer to the segment created or else null();

hcs\$newproc

07/05/77 09:44:32

192

cc1 hcs\$newproc entry;
call hcs\$newproc;

- terminates this process and creates a new one

hcs\$remove_dir

07/05/77 09:46:47

343

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

hcs\$remove_seg

07/05/77 09:47:02

cc1 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

hcs\$remove_link

07/05/77 09:47:17

347

```
dcl hcs$remove_link entry(char(168) varying,char(32) varying,fixed bin(31));
call hcs$remove_link(cname,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

hcs\$set_acl

07/05/77 09:47:45

786

```
cc1 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-only

bit 3: set after system copies linkage section on a fault

bit 4: segment is a gate arc therefore requires gate actions

bit 5: segment can be read

bit 6: segment can be executed

bit 7: segment can be written

hcs\$split_name

07/05/77 09:41:24

.377

```
cc1 hcs$split_name entry(char(16) varying, char(16) varying, char(32) varying, fixed(31));
call hcs$split_name(pname,dname,ename,ccde);
```

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

hcs\$terminate

07/05/77 09:41:10

219

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

where:

ptr is a pointer into the segment to terminate

```
ioa entry options(variable)
call ioa(pattern_string[,arguments ...]);  
  
ioa$ioannl entry options(variable)
call ioa$ioannl(pattern_string[,arguments]);  
  
ioa$ioars entry options(variable)
call ioa$ioars(result_string,pattern_string[,arguments]);  
  
ioa$ioarsnnl entry options(variable);
call ioa$ioarsnnl(result_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\$ioannl is like ioa except it doesn't put out a carriage return 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\$ioarsnnl 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 \$|oooo, where \$ 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|C280 has 2 names, first is nimbblebarf

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

Advanced hackery:

- ^n[iterates the field up to the next ^] n times (n may be 0).
- ^) closes iteration
- ^n[executes the nth field between the ^[and the ^]
 where ^; is used to bound fields
- ^; field delimiter for conditionals
- ^] end of conditional

For example:

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

call ioa("^v{foobar^}",5); prints
foobarfoobarfoobarfoobar
```

```
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

```
iocs$get_chars entry(ptr,ptr,fixed bin(31),fixed bin(31))
call iocs$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.

```
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 operator
code	is a standard error code

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
```

```
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 this iocb pointer is desired
iocb_ptr	is set to the pointer to the i/o control block
code	is a standard error code

```
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

Orders include:

resetrd reset the input buffer stream to flush all
 buffered characters

cur_pos performs cursor positioning as specified in
 a structure which is specified 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 operand2
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 screen
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.

- cutwind makes the designated window current (active) which means that all output, cursor positioning and echoing on the specific stream will occur within this window
- getwind fills in a window info structure given `info_structure.name`
- woodwind makes beautiful music (not implemented yet)

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

```
iocs$modes entry(ptr,char(8),ptr,fixed bin(3))  
call iocs$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:

- | <code>modespec</code> | <code>action</code> |
|-----------------------|--|
| <code>getpage</code> | fills in the following structure:
dcl l page_info,
2 lines fixed bin,
2 columns fixed bin; |
| <code>setpage</code> | sets the page and line length from the structure described for <code>getpage</code> |
| <code>setflags</code> | sets various bits describing this device modes as described in <code>getflags</code> (see below) |
| <code>getflags</code> | gets various bits describing the device modes into the following structure:
dcl l flags,
2 current bit(32),
2 future bit(32); /* not implemented yet */ |

where the bits are as follows:

- 1 device in service (ro)
- 2 interrupt quiver (rc)
- 3 handle_quit
- 4 wakeup for input (rc)
- 5 wakeup for output (ro)
- 6 output in progress (ro)
- 7 echo input
- 8 raw input
- 9 raw output
- 10 high speed (rasle only)
- 11 display (with cursor positioning)

12 ignore break (rc)
13 flush until input (ro)
14 more processing
15 reverse (underscore and delete)
32 shared bus device (ro)

nrfs\$append_name

07/05/77 09:58:15

394

```
dcl nrfs$append_name entry(char(168) varying,char(32) varying,char(32) varying,fixed(31));
call nrfs$append_name(cname,ename,dname,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

nrlfs\$delete_name

07/05/77 09:58:28

350

```
dcl nrlfs$delete_name entry(char(168) varying, char(32) varying, fixed(31));
call nrlfs$delete_name(cname,ename,code);
```

where:

cname is the name of the containing directory
ename is the name of the entry to be deleted
code is a standard error code

nrfs\$get_link_target

10/18/77 17:22:42

428

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

where:

dname	is the name of the directory containing the link
ename	is the name of the link itself
tname	is the target of the link
code	is a standard system error code

mtf\$set_acl

07/05/77 :09:58:41

941

```
dc1 mtf$set_acl entry (char(168) varying, char(32) varying, bit(16), fixed(31));  
call hcs$set_acl (iname,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 therefore 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.

nrfss\$set_domain

07/05/77 09:59:38

396

```
dcl nrfss$set_domain entry(char(168) vary, char(32) varying, fixed, fixed(31));
call nrfss$set_domain (cname,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

rifs\$set_dtb

07/05/77 09:58:55

413

```
dcl rifs$set_dtb entry(char(168) varying, char(32) varying, bit(64), fixed(31));
call nrfs$set_dtb (iname,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_stacked up is to be set.

 code is a standard error code

nrf\$set_dtc

07/05/77 09:59:10

414

```
dc1 nrf$set_dtc entry(char(168) varying, char(32) varying, bit(64), fixed(31));
call nrf$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

nrfss\$set_dtc

07/05/77 09:59:10

414

```
dcl nrfss$set_dtc entry(char(168) varying, char(32) varying, bit(64), fixed(31));
call nrfss$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

nrfss\$set_name

07/05/77 09:59:52

421

```
dcl nrfss$set_name entry(char(168)varying, char(32) varying, char(4), fixed(31));
call nrfss$set_name (dname,ename,sname,ccde);
```

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.

nrfs\$status

07/05/77 09:59:24

891

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

- see the incl files in >incl for more info

where:

 dname is the name of the containing directory

 ename is the name of the entry

 flags are control bits specifying how much information is returned as follows:

 "0000'b4 returns the minimum.info.

 "E000'b4 looks in the vtoc for the length and bit count of the entry.

 "4000'b4 returns various worthless bits of info like dtc and sname.

 info_ptr is a pointer to a structure in which the .info is to be returned.

 name_area_ptr is a pointer to and area in which all the names of the entry are returned. If null() no names are returned.

 code is a standard error code

```
declare
  l 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 vtcc. */
  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 set. */
  2 space_name char(4),
  2 dtc bit(64), /* used as unique index */
  2 dtm bit(64),
  2 dti bit(64),
  2 dtb bit(64),
  l link_info defined info,
  2 skjshsaskjh char(6),
  2 target offset(info_area);
```

scs\$add

07/05/77 10:09:19

224

```
dc1 scs$add entry(bit(64), bit(64), bit(64));  
call scs$addc (op1, op2, sum);
```

- double precision acc

where:

sum = op1+op2;

scs\$allocn

07/05/77 10:07:32

355

```
dcl scs$allocn entry (fixec(31), painter, ^area);
call scs$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

scs\$date_time

02/21/78 15:11:28

475

```
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.

scsSch_wdir

07/05/77 10:02:08

224

dc) scsSch_wdir entry(char(168) var);
call scsSch_wdir(wname);

where:

wname is :the name of the new working directory

scsScl

07/05/77 10:01:38

240

```
dcl scsScl entry(char(255) varying);
call scsScl(command_line);
```

where:

command_line is a string which is parsed by the command processor

scs\$date_time

07/05/77 10:07:02

380

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

where:

dt\$ 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.

scs\$divide

07/05/77 10:09:49

268

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

- semi-double precision divide

where:

```
quotient = dividend / divisor;
```

sc\$eqstar

07/05/77 10:08:18

336

dc] sc\$eqstar entry(char(32), char(32)) returns(bit(1));
equal = sc\$eqstar(starname, ename);

where:

starname is the starname to be matched.
ename is the name to compare to the starname
equal is "1"b if ename matched the starname

scs\$expand_path

07/05/77 10:05:36

390

```
dcl scs$expand_path entry(char(168) varying,char(168) varying,char(32) varying,fixed(31));
call scs$expand_path(pname,cname,ename,code);
```

where:

- pname is an absolute or relative pathname
- cname is the directory part of pname
- ename is the entry part of ename
- code is a standard error code

scs\$freeen

07/05/77 10:07:48

344

```
dcl scs$freeen entry (fixed(31), pcharter, area);
call scs$freeen (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

scs\$get_arg_count

07/05/77 10:05:49

211

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

where:

nargs is the number of arguments

scs\$get_arg_info

07/05/77 10:06:02

369

```
dcl scs$get_arg_info entry(fixed,bit(16) aligned,fixed,pointer);
call scs$get_arg_info(argno,type,a1,ap);
```

where:

argno is the number of the argument
type is the descriptor for the argument
a1 is the length of the argument
ap is the pointer to the argument

`scs$get_search_rules`

07/05/77 10:20:26

257

```
dcl scs$get_search_rules entry(ptr);
call scs$get_search_rules(prp);
where:
    prp    is a pointer to a search rule structure
          (see SRULFS.INCL.PL1)
```

scs\$get_wdir

07/05/77 10:01:53

230

```
dcl scs$get_wdir entry(char(168) varying);
call scs$get_wdir(wname);
where:
wname is the name of the current working directory
```

scs\$get_uname

07/05/77 10:09:04

235

```
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.

scsShow_min

07/05/77 :10:07:17

247

```
dc1 scsShow_min entry(char(18));
call scsShow_min (time_string);
```

where:

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

scs\$ioa

07/05/77 10:08:03

320

```
dcl scs$ioa entry options(variable);
call scs$ioa(control_string, arg1, arg2, ... argn);
```

- see ioa.order for more complete information

where:

control_string describes the number and then printing format of the args

scs\$make_ptr

09/25/77 02:59:15 1182

```
icl 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 pointer to the base of the segment is returned. The ref_seg argument is a pointer to a segment to be searched for epname before searching for segment ename. (As used in the linker when snapping a link to \$fcc.) Normally ref_seg is null. If you don't know what ref_seg is, use null! You can bet what code is.

scs\$make_space_ptr

11/07/77 18:21:30

1075

```
scs$make_space_ptr
    entry (char(32)vary,char(32)var),char(4),char(4),ptr,ptr,fixed bin(31))
    call scs$make_space_ptr(ename,epname,space,rspcptr,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 lock 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 space="====" then the segment will be initiated in the space of the caller
rspcptr	is set to the space in which the segment is actually initiated
refptr	is the "reference" pointer, if ename="" then first makeptr 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

scs\$make_static_variable

07/05/77 10:08:34

547

dcl scs\$make_static_variable entry(char(32)varying, char(32) varying, fixed,fixed(31));
call scs\$make_static_variable (refname,data_name, size, code);

- this entry allow data declarec external to have space allocated for them.
- 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.

scs\$multiply

07/05/77 10:10:04

252

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

- semi-double precision multiply

where:

product = op1*op2;

scs\$obj_map

07/05/77 10:06:46

338

```
dcl scs$obj_map entry(pointer,fixed bit(31)) returns(pointer);
op=scs$obj_map(sp,dc);
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
```

scs\$subtract

07/15/77 10:05:34

263

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

- double precision subtract operator

Where:

```
difference = op1 - op2;
```