ANNOTATED
INTERNALS

PART THREE - The CAL Time-Sharing System

Chapter 1 Running Programs under TSS

1.1 Time-Sharing Defined

1.2 At the Console: Job Definition and Entry

1.3 Sample Session

1.4 Input / Output

Chapter 2 - System Architecture

2.1 Files

2. Directories ~~System~~ Calling the system

3 Processes

4 Subprocesses

5 operations

Chapter 3 User Subsystems

3.2 Line Editor

3.1 ~~File~~ EDITOR (File Editor)

3. 3 SCOPE Simulator

3. 4 Tape I/O (GETTPE & DUMPTPE) → Under I/O section

3.5 BEAD Directory list (DLIST)

3.6 [illegible]

Ch. 4 System ~~Calls~~ Actions

BASIC ~~BCPL~~ goes under Programming Languages.

Ch 5 I/O Interfaces

Ch. 6 Debugging Aids

9 Jan '70

ECS Test

The basic philosophy is to ~~create~~ have available test code ~~to run at any time~~ which checks as many ECS features as possible and which can be incarnated as a user process and run at any time. For this purpose, the system is considered to be defined by the user document, a copy of which is appended to the test document. ~~This user document is gone through operation by operation.~~ Each operation in the 4D is thoroughly exercised to verify that the description is accurate in every respect. Some features of the ECS code, such as arrival of interrupts at certain times during processing, can't be tested ~~here~~ in this manner, & such features are noted & the testing documented separately as special tests.

Because the current executive is temporary, the test has been written in 2 pieces, the test program & the interface. The test program is (hopefully rid of the) coded & mainly in macros. It depends on the interface to provide it with an appropriate environment, including a C list & macros as specified

9 Jan '70

Requirements on the interface (ISUBP)

1) a Clist the first part of which is identical to that part of IPROC's Clist which contains ~~all the ops~~ caps for all ECS actions (as defined by OPNAMES).

In addition:

K.MALLOC a cap for an allocation block to which things can be charged

2) Macros:

NEWCAP
DOPER
TOPER
SAY
CALL

3) Misc

UNIQNM - a cell containing one of the ~~pro~~ test process
SLVNM - " " " unique name of a slave process

Interface

The current interface runs as a subprocess under the BEAD & ~~m~~ uses the BEAD for all IO functions.① It is full of hideous kludges, such as

1) UNIQNM is obtained from an event & hence is only as reliable as the event channel operation (which is what its used to test!)

∃ a file which exercises the test macros. =TTEST

① The test code is on a separate file & is ~~pulled into the interface during compilation with appropriate XTEXT card(s)~~ called as a subroutine by the interface (cause COMPASS bombed out if ∃ more than about 20 TWER-R macros).

Test Proper

The test proper is split into files according to the ECS action types, event channels, files, etc.

Current Files

TMACROS MACROS1 CLIST OPNAMES, XTEXT	used as XTEXT files in almost all assemblies
HOWOUT	used as XTEXT file in ISUBPS
ISUBPS LAST	provide the ~~operator~~ interface + call the actual tests
TEVCH	binary decks for event channel test
TEVCH1 2 3 4 5	event channel test source

TCLST
TCLST1

TALOC1
TTEST

Errors in manual

1) Numbering foul-up in Process section
do this 2) jump call
~~3) where is the return of events descriptor~~
do this 4) page 5 .. how are error class + number passed to subprocess
5) explain # change words
6) what actions have had their specs changed?

4 Feb 7) p 37 Sovereign warning
9 " 8) p 7 AB creation errors, 2,4, idx & 2,5, idx
9 Feb 9) p 49 2,0 # channels ≤ 0
16 Feb 10) p 10 down at bottom in D
20 Feb 11) p 11 Change 4H fig
23 12) p 68 2,4 - index masked in
16 Mar 13) p 20 errors
" 14) p 35 bottom, Word 0, Word 1, etc
" 15) p 21 Temp part of new class code = 0
" 16) p 22 IP3, the 0 .... (0 half stuff
" 17) p 27 tell where process state flags are
" 18) p 29 explain Subproc low core
" 19) p 4, p 33, new XS format for new para types
" 20) the 2, ? errors are nowhere consistent
27 Mar 21) changes to DAE stuff
31 " 22) delete file → file copy (p 19 & p 60)
9 June 23) p 32 - params passed at 6, not 5.
30 June 24) p 68 - pointer of f-counter is ....

30 June 25) p54 – IP3 D ..
p64 same

8 July 26) p34 etc: M
p61 same

22 Sept 27 p52 IP4 ....

1 Dec 28) p49 "packed arscale"

all above given to MAB ~1 Dec '70.

16 Dec '70 29) p7 – no 6,2 error – likewise elsewhere

" 30) New open display AB

12 Jan '70 31) p19 – 3,2 error on match

1 Feb '71 32) p10 – delete "in top 42 bits ... "

1 Feb '71 33) p10 – delete DSPARB

34) p11 – 8,0,1 error

35) p11 – is MAB's new UN correct?

36) p18 – file length

8 Feb '71 37 p42 error number mixup

19 38) p55 no 6,0 error, 223 ← 232 etc

24 39) p38 ESM format

Zero a capability

entry pt:: cpzro
ecs resident
deck: capab

IP1 D: Index in full clist of the capability

Action:

Fill the two words of the indicated capability with zeroes.

Errors

2	4	1	neg clist index
2	5	1	clist index too large
2	6	1	not a clist capability (indirection)
8	0	1	clist gone from mot (indirection)

Find Nth Son of a given Subprocess

entry pt: fson
ecs resident
deck: subproc

Action

Returns a class code for the son with the same option bits as in the original class code for the father.
Does an F-return if alleged father has no Nth son.

Errors

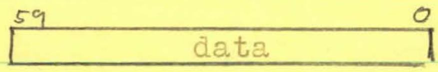
4	1	Alleged father does not exist

10Aug 70

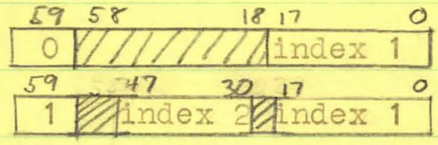
DESCRIPTION OF NEW (AND OLD) PARAMETER TYPES. WILL GO ON PAGE 2 OR SO OF USER MANUAL.

Each entry in the input parameter (IP) list is one of the following types:

1) type D - a 60-bit data item



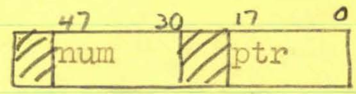
2) type C - a ~~Cxxixtxindex~~ capability specifier



the direct form specifies the capability at index 1 in the local C-list.

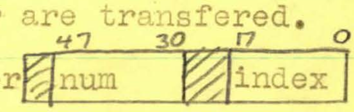
the indirect form requires that index 1 be the index ~~of~~ in the full C-list of a capability for a C-list; the specified capability is at index2 in the C-list given by index 1.

3) type BD - a block data specifier



num 60-bit data items starting at ptr are transfered.

4) type BC - a block capability specifier



num capabilities starting at index are transfered.

10 Aug 70

NEW FORM OF SYSTEM CALL. ALTERS PAGE 4 OR SO OF USER MANUAL.

59	51	47	30	18	17 0
CEJ	///////	IP list pointer	///////	BIT	P-counter offset

When BIT = 0, no return of data or capabilities is authorized.
When BIT = 1, the word ~~fxxx~~ following the CEJ must contain a pointer to a return authorization* as follows:

59	30	17 0
unconditional jump	///////	PTR →

59	45 44	30	17 0
# returned	# allowed	///////	ptr to data buffer
# returned	# allowed	///////	index to return capabilities

The # returned fields are both set by the system to the actual number of data (caps) returned.

* USED WHEN THE RETURN IS VIA A "RETURN WITH PARAMETERS" OPERATION; SEE ~P. 34.

10 Aug 70

DESCRIPTION OF NEW OPERATIONS TO CREATE CAPABILITIES OF SPECIFIED TYPE. WILL MODIFY USER MANUAL, PAGE 12 OR SO.

H. Create a Capability Creating Authorization

IP1 D: C-list index for returned authorization

A capability creating authorization is a special type of capability. The second word of the capability contains the type of capability which may be manufactured under the authorization.

Possible errors while creating an authorization:

Class	#	Description
8	2	No more capability types are available*

I. Create a Capability of the Authorized Type

IP1 D: C-list index for returned capability
IP2 C: A capability creating authorization
IP3 D: Data for second word of returned capability

A capability of the type specified by IP2, with all option bits on, and with second word equal to IP3, is returned at the specified index in the caller's C-list.

Only the entry/exit errors are possible.

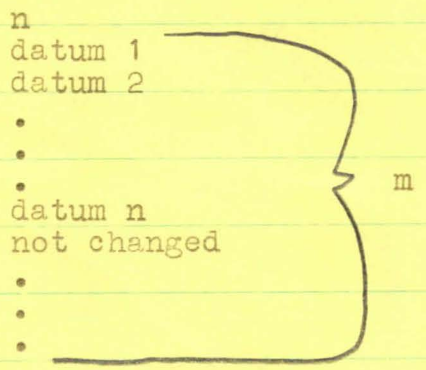
*Note that since there are only about 48000 different capability types, unrestricted use of this operation would allow one user to exhaust the supply, thus making those that wanted a special capability type later on very unhappy.

12 Aug 70

DESCRIPTION OF HOW BLOCK DATA AND BLOCK CAPABILITY PARAMETERS LAND IN THE ADDRESS SPACE OF THE CALLED SUBPROCESS. PAGE 33 OR SO OF THE USER MANUAL NEEDS LOTS OF HELP IN THIS REGARD.

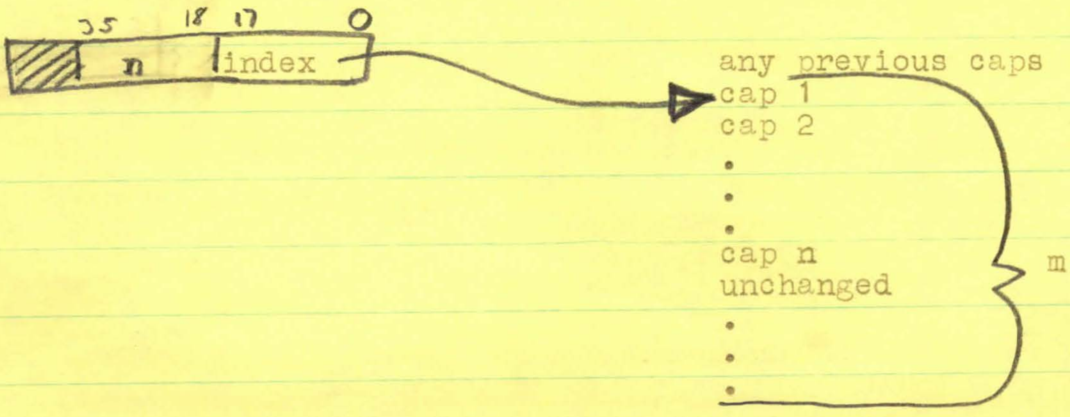
Block data

Let m be the ~~number of~~ maximum number of data that can be passed, as specified by the operation. Then m+1 cells are reserved for passing the block of data (the cells immediately follow any previous information passed to the subprocess as part of the call (or start at cell 6 if there isn't any previous info)). The actual number of data passed is set in the first of the m+1 words, immediately followed by the n data inorder. Unused cells at the end are unchanged when n is less than m.



Block capabilities

Let m be the maximum number that can be passed, as specified by the operation. Then m slots in the full C-list are reserved for passing the block of capabilities (the slots immediately follow any previous capabilities passed, or start at 0 if there were none) and the next available cell of the address space is reserved for an indicator. The number of capabilities actually passed, n, and the index of the first one are stored in the indicator. The n capabilities are copied into the C-list starting at the indicated index. When n is less than m, unused C-list slots are unchainged.

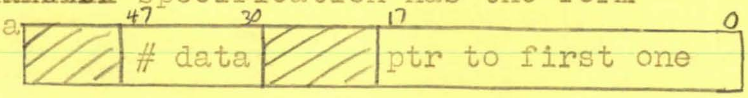


DESCRIPTION OF RETURN WITH PARAMETERS OPERATION. GOES ON PAGE 34 OR SO OF USER MANUAL.

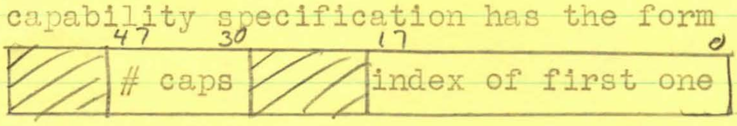
M. <u>Return with Parameters</u>

IP1 D: Data specification
IP2 D: Capability specification

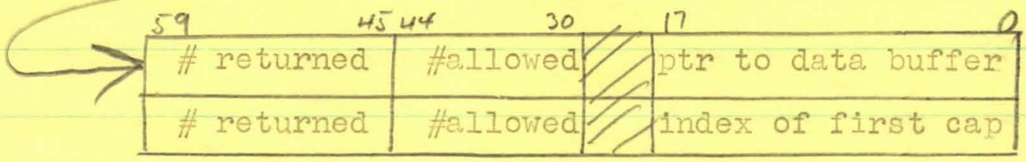
The ~~parameter~~ data specification has the form



and the capability specification has the form



Data from the full address space and capabilities from the full C-list of the returning subprocess are returned to the full address space and full C-list of the subprocess determined by the top of the stack. Provided that all pointers, indices, counts, etc. in both subprocesses are legal. In particular, the P-counter in the stack must point to a CEJ which indicates a return authorization as described on page 4. Just so you won't have to turn the page, the PTR after the CEJ points to



The # returned fields are set by the system to the actual number returned; the other fields are prescribed by the subprocess which points to them and are not disturbed.

The error list is not in yet, but trying to return more data or caps than the authorization allows is not an error; the operation merely returns the max number and says nothing. If the CEJ doesn't specify a return authorization, that isn't an error either; needless to say, nothing is returned in that case.

10 Aug 70

HOW TO SET UP BLOCK DATA AND BLOCK CAPABILITY PARAMETER TYPES IN AN OPERATION. PAGE 57 OR SO OF USER MANUAL.

6. <u>CHANGE PARAMETER FROM "NONE" to"block data"</u>

IP1 C: Capability for an operation (OB.CHTYP)
IP2 D: ~~Pxx~~ Index of parameter to be changed
IP3 D: Maximum size of block that will be passed

Sorry, no error list right now.

7. <u>Change parameter from "none" to "block capability"</u>

IP1 C: Capability for an operation (OB.CHTYP)
IP2 D: Index of parameter to be changed
IP3 D: Maximum number of capabilities that will be passed

Sorry, no error list right now.

CAL Time-Sharing System Users Guide

November 1969

Computer Center
University of California
Berkeley

TABLE OF CONTENTS

User-System Interaction

The ECS portion of the CAL Time Sharing System provides a number of actions which are available to the user so that he can interact with the system. The actions apply to the objects created and maintained by the ECS system: files, maps, allocation blocks, event channels, capability lists (C-lists), operations, processes, subprocesses, and class codes. A record is kept in a table in ECS, called the Master Object Table (MOT), of all objects existing at any given time in the system. Each entry in MOT gives the name of the object and its ECS location.

The user makes a call upon the system by setting up the appropriate parameter list for the action he wants to initiate, prior to passing control to the system entry/exit routines by executing a CEJ instruction. (The CEJ, Central Exchange Jump, causes the current contents of the 6400 central processor's registers to be exchanged with a similar 16 word package in Central memory.) The system entry/exit routines determine the nature of the user's call, collect and check the parameters needed for the action, transfer control to the proper system action routine, and finally, return control to the user (by another CEJ, which restores the registers) after the system action is completed.

Requesting a System Action

The CEJ instruction used to call the system supplies the information required to initiate the action and return to the user. (See Figure 1.) In particular, it is expected that the CEJ was in the upper 30 bits of the instruction word; of these 30 bits, the lower 18 bits are used by the system to locate the user's input parameter (IP) list. If this 18 bit field is negative, the complement of the low order 4 bits specify which register in the user's exchange package contains the input parameter list pointer (e.g., $-3 \rightarrow B3$; $-10 \rightarrow X2$). Otherwise, the 18 bit field itself is taken to be the IP list pointer. This pointer is checked for legality (i.e., it must be positive and less than the user's field length) and an error is generated if appropriate.

Each entry in the input parameter (IP) list is either a 60-bit data item or an index into the user's capability list (C-list), designating a capability. Each capability residing in a C-list authorizes access to a particular object in addition to giving the object's type (file, process, event channel, etc.) and the set of actions allowed on that particular object (option bits).

The first parameter, that is, the first word of the IP list (called IP0), is always expected to be an index into the user's capability list. This parameter, after being checked for legality (i.e., it must be positive and within the range of the full C-list), is used to fetch the capability for the operation which specifies the action to be performed, and the nature of the parameters of the action. (If the capability is not for an operation, an error is generated.)

Operations are ECS objects which direct the transfer of control from the user to the system when the user calls upon the system. They identify the action(s) to be taken by the system and direct the passing of parameters to the system or between user subprocesses (see SUBPROCESSES). An operation consists of one or more orders, each of which designates a particular system action, and a set of parameter specifications which indicate the type (capability or datum) of each parameter required for the action and the required options for each capability parameter. Basically, the parameter specifications in the operation are of two genere — parameters which are permanently "fixed" within the operation and those that are to be provided by the user. When an operation is first created, before any of its parameters have been specified, all parameter specifications are typed "none". Before the operation can be used, all of its parameters must be specified using the actions provided for specifying operation parameters (see p. 50). If the action is parameterless, the operation contains no parameter information.

The system entry/exit routine reads the first order of the operation and uses the parameter specifications to construct an actual parameter list. This list consists of parameters which are "fixed" in the operation and of user-supplied parameters drawn from the IP list. The IP list should contain, in successive words, datum parameters (indicated below by "D:") which are transferred directly to the actual parameter list, and C-list indices (indicated below by "C:") which designate capabilities in the user's full C-list. Two words are copied to the actual parameter list for each capability parameter (capabilities are two words long) and one word is copied for each datum parameter. During the construction of the actual parameter list, errors will be generated if 1) a C-list index is bad (i.e., is negative or outside the full C-list); 2) if the type and options (indicated below by "OB.x") in the capability do not correspond to those specified by the parameter information in the operation (this checking is not performed if the parameter specification is "any capability"); or 3) if a "none" parameter specification is encountered, in which case parameter processing terminates.

After the actual parameter (AP) list is completed the operation is checked to see if the action is a subprocess call or jump. If so, a flag bit will indicate the presence of a class code (the subprocess name) in the operation. In this case, the operation also contains a parameter type bit mask indicating the type (capability or datum) of each parameter. The system entry/exit routine places the class code from the operation, the number of parameters, and the bit mask into the user's process descriptor in the actual parameter list area.

Finally, the ECS action number is extracted from the operation and is used as an index to a jump to the proper entry point for the desired ECS action. When the action is completed, control returns to the user.

Under some conditions, when the normal function of an ECS system action cannot be carried out but the condition is not serious enough to warrant the generation of an error, an F-return will result. If this occurs, the count of F-returns initiated for the operation is increased, and the operation is checked to see if it contains any more orders (which are specified as alternative actions). If so, the next order of the operation is interpreted. This process is identical to the one just described, except that the actual parameter list contains the parameters for all orders up to and including the current one. If the F-return count reaches the number of orders in the operation, control is returned to the user.

There are two different ways in which control is returned to the user depending upon whether an action completed normally (possibly after one or more F-returns) or the F-return count became equal to the number of orders in the operation. The normal return causes the user's P-counter to be incremented by the number supplied by the user in the low order 18 bits of the CEJ instruction word originally used to call the system. The new P-counter must be positive and less than the user's field length; otherwise an error is generated. When the return to the user results from an ultimate F-return, the user's P-counter is left unchanged.

Figure 1 System Calling Instruction

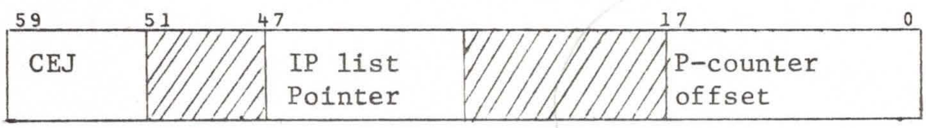
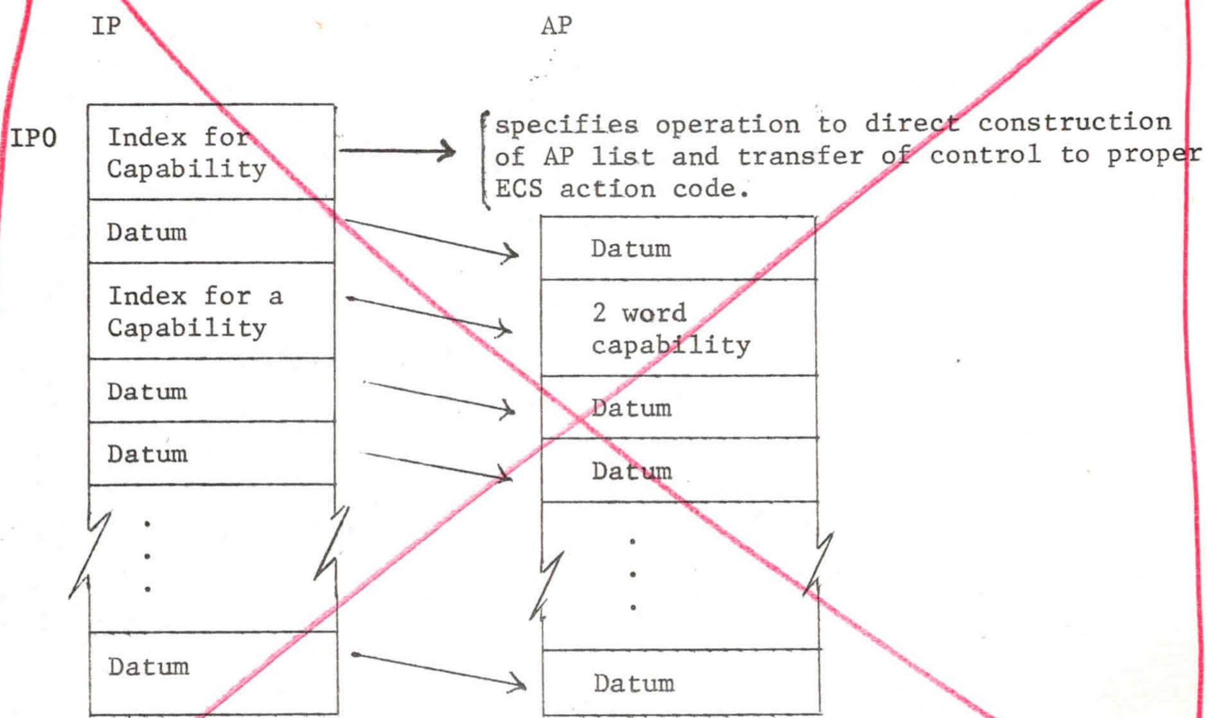


Figure 2 Example of Input Parameter (IP) list and Actual Parameter (AP) list Interaction (assuming no fixed parameters*)



* Fixed parameters will be inserted into the AP as they are encountered according to the parameter type bits.

Errors: The use of improper parameters in making an ECS system call is considered to be an error on the part of the process which is making the call. When an error is detected, it is first assigned an error class and number. The class identifies the type of the error, while the number pinpoints the particular error within a type. Furthermore, associated with each subprocess within a process is an error selection mask (ESM) indicating the classes of errors the subprocess is prepared to handle. The "ancestors" of the current subprocess (see p. 31) are checked (starting with the current subprocess) to find a subprocess whose ESM indicates it is willing to handle this class of errors. The subprocess which accepts the error is called and is passed the error class and number as the. Execution in the error processing subprocess is initiated at the normal entry point-1. A precaution is taken against error loops; the subprocess which accepts the error is temporarily disqualified from accepting any more occurrences of the errors in the same class.

Possible Errors during System entry/exit processing of an ECS system action call

Error Class	Error #	Error Description
2	2	The IP list pointer address is negative
2	3	The IP list pointer address is greater than the user's field length
2	4+idx	C-list index negative
2	5+idx	C-list index too large (not within full C-list)
7	0	First parameter (IP0) does not point to a capability for an operation
7	1	The operation does not exist
7	3	"NONE" parameter specification encountered
7	2	Type or options bad for a capability parameter
8	0	C-list does not exist
7	7	IP list extends passed user's field length
2	0	The new P-counter is negative on return to user
2	1	The new P-counter exceeds the user's field length
10	0	No subprocess to take error class

2 3 is really P-counter ≥ FL

System Actions

All system actions which can currently be requested by the user are described below. All actions are calls upon the ECS system except for the subprocess call and return actions. A summary of required parameters and possible errors appears in the Appendices.

I Allocation Blocks

An allocation block is an ECS object which regulates allocation of ECS space and CPU usage. An allocation block is provided with a sum of money and a portion of ECS space, which can only be obtained from another allocation block. (At system initialization a Master Allocation Block is created and provided with an infinite amount of money and all of the space in ECS.) Every object is associated with an allocation block; the objects associated with each allocation block are linked to that allocation block in a two-way circular list headed by the allocation block itself. The objects of ECS, therefore, form a tree whose root is the Master Allocation Block.

Each allocation block is billed for CPU-time used by its descendant processes and will be charged rent on the ECS space occupied by its descendant objects. There are four actions which the user can invoke to manipulate allocation blocks. He can 1) create an allocation block, 2) transfer funds from one allocation block to another, 3) request the capability (with all option bits set) for the *n*-th object in the list of an allocation block, and 4) destroy an allocation block.

AB10 A. Create an Allocation Block C-CRALBK

IP1 C: Allocation Block (OB.CREAB) 100
IP2 D: C-list Index for returned capability

When creating an allocation block, the user must first specify the index of the allocation block which is to provide the ECS space occupied by the new allocation block. The second parameter provides a C-list index where the system can return the capability for the newly created allocation block.

4 Jan '71
proposed User's manual section for allocation blocks

I Allocation Blocks

Allocation blocks are ECS system objects designed to serve three purposes:

1) To control the distribution of certain system resources - ECS space, MOT space, and CPU time
2) To provide a mechanism whereby the use of these resources can be accounted (and charged)
3) To provide an orderly structure on the objects maintained in the system so that a given code can recover the space consumed by a subordinate code, even if the subordinate code has gone awry and lost the capabilities for its objects.

Fig ? - Allocation Block

RESERVED SPACE	SPACE IN USE
HEAD PTR	TAIL PTR
TIME OF LAST BILL	CHARGE RATE
CONTINUOUS CHARGE ~~ECS~~ METER	
DISCONTINUOUS CHARGE METER	
CP us AVAILABLE	
CP us CONSUMED	
MOT RESERVED SLOTS	MOT IN USE ~~AVAILABLE~~

SPACE IN USE - the number of cells of ECS occupied by objects charged to this AB

RESERVED SPACE - the maximum number of cells which may be occupied by objects charged to this AB

HEAD PTR - the MOT index of the oldest extant object charged to this AB

TAIL PTR - the MOT index of the newest extant object charged to this AB

TIME OF LAST BILL - the time when the meters were last updated, reckoned in us/1024 since the last system deadstart

CHARGE RATE - the rate at which the charge meters grow. When CHARGE RATE = RESERVED SPACE the meters give the amount of space*time tied up by this AB.

CONTINUOUS CHARGE METER - this field starts at 0 & grows at the CHARGE RATE throughout the life of the AB. Units are (words × μs)/1024

DISCONTINUOUS CHARGE METER - this field is like the CONTINUOUS one except that an operation to increment it by an arbitrary amount is provided.

CPU μs AVAILABLE - the number of μs available to be put into a ~~[illegible]~~ process timer or dispensed to descendant AB's.

CPU μs CONSUMED - this field starts at 0. It is incremented when~~ever~~ a process or AB owned by this AB is destroyed~~, or when an~~

MOT SLOTS RESERVED ~~AVAILABLE~~ - the number of objects which may be charged to this AB ~~(in addition to those already charged to it).~~

MOT SLOTS IN USE - the number of objects currently charged to this AB

Whenever an object is created, a capability, with adequate option bits, for an AB must be presented. The AB must have enough space reserved, but not yet in use, to accomodate the object and must have an MOT slot available for the object. Thus, every object created by the ECS system is charged to an AB, refered to as the "owning AB" or "father AB" of the object. Each AB contains pointers to a two-way circular list of the objects charged to it. In this way, the descendents of a given AB are organized in a tree structure with the AB as the root of the tree. Actions are provided which give a code suitable access to the descendents of an AB for which the code has a capability with the correct option bits. Actions to move resources between an AB and its father AB are also provided.

The structure of ABs and the actions on them are such that a code can establish an allocation block, ABX, allow other code access to the resources in ABX and still maintain control over all the resources commanded by ABX. The control can only be abrogated by a code which has suitable access to an ancestor of ABX.

Since all objects, including allocation blocks, must be charged to an AB, a Master Allocation Block is created as part of the system initialization process and given all the system resources. The MAB is thus at the root of a tree containing all ECS system objects and a code with suitable access to the MAB has ultimate control over all the resources of the system.

A. <u>Create Allocation Block</u>

IP1 C: Allocation Block (OB.CREAB)
IP2 D: index for returned AB capability

If IP1 has an MOT slot + sufficient space available, an AB is created + a capability, with all option bits on, is returned at IP2.

6	0	AB gone
6	1	Not enough reserved space
6	2	No MOT slot available
2	4	C-list index is negative
2	5	" " exceeds full C-list

B. Destroy AB   C.DELAB

IPI C: AB to be destroyed (OB.DSTRY)

An AB cannot be destroyed if objects are still charged to it. If objects are charged to it, an F-return is made. Otherwise, the AB's resources (Reserved space, CP time available, & MOT slots available) are ~~given~~ returned to its father AB & its CP time consumed ~~field~~ is added to that of its father.

6 0 AB gone

Display AB

IP1 C: AB
IP2 D: address of buffer area
IP3 D: buffer size

The charge meters in the AB are updated & min (buffer size, allocation block size) words of the AB are moved into the buffer.

6	0	AB gone
2	2	buffer address negative
2	0	buffer size negative
2	3	buffer ~~at~~ exceeds FL

D. <u>Move reserved space</u>

IP1	C:	donor AB	(OB.GIVE)
IP2	C:	donee AB	(OB.GET)
IP3	D:	donation, must be +	

Either IP1 must be the father of IP2 or vice-versa. The reserved space in the donor ~~is decremented by the donation, providing~~ must exceed the in use field by at least the amount of the donation. If so, the donor reserved space field is decremented & the donee reserved field is incremented by the donation

6	0	1 or 2	AB gone
6	5		donor can't afford donation
6	9		neither AB is the father of the other
2	0	3	donation is negative

E. Move CP time

IP1 C: donor AB (OB.GIVCP)
IP2 C: donee AB (OB.GETCP)
IP3 D: donation, must be +

Either IP1 must be the father of IP2 or vice-versa. The CP time available in the donor must be at least as large as the donation. If ~~it~~ so, the donor CP time available field ~~in the donor~~ is decremented & the donee CP time available field is incremented by the donation.

6	0	1 or 2	AB gone
6	6		donor can't afford donation
6	9		neither AB is the father of the other
2	0	3	donation is negative

F. Move MOT slots

IP1 C: donor AB (OB.GIVMT)

IP2 C: donee AB (OB.GETMT)

IP3 D: donation, must be +

Either IP1 must be the father of IP2 or vice-versa.

~~The number of MOT slots available in the donor must be as large as the donation. If so, the donation is removed from the donor MOT slots available & added to the donee MOT slots available.~~

6	0	1 or 2	AB gone
6	7		donor can't afford donation
6	9		neither AB is the father of the other
2	0	3	donation is negative

The MOT slots reserved in the donor must exceed the MOT slots in use by at least the amount of the donation. If so, the donor MOT slots reserved field is decremented & the donee MOT slots reserved field is incremented by the amount of the donation.

G. Increment Charge Rate

IP1 C: AB (OB.INCHR)

IP2 D: increment, + or -

The charge ~~rate~~ meter of the AB is updated. The charge rate is incremented. The resulting charge rate must be positive & less than 2^{30}.

6	0	AB gone
6	10	resulting charge rate illegal

H. Increment Charge Meter

IP1 C : AB (OB.INMTR)

IP2 D : increment, +or-

The increment is added to the ~~DTS field~~ discontinuous charge meter of the specified AB using an integer add instruction (that is, signs, large numbers becoming negative, etc., are all ignored).

6 0 ABgone

I. Return Capability for N^th Object in Allocation Block

IP1 C: Allocation Block (OB.GOD)

IP2 D: index in full C-list for returned capability

IP3 D: index of desired object

This action returns to the user the capability for any desired object which is a first generation descendant of an allocation block. The first parameter is the index of the capability for the allocation block to which the object is associated; the second

parameter specifies a C-list index where the system will return the capability, and the third parameter gives the position in the list of the desired object. If this index is zero, a value of one is assumed and the capability for the first object in the list is returned. If *n* exceeds the number of objects in the list for the specified allocation block, an F-return is made. If the capability is returned, all options bits are set.

Possible errors:

Class	#	Description
6	0	Allocation block does not exist
2	4	C-list index is negative
2	5	C-list index exceeds full C-list
2	0	Index for object is negative.

J. Display allocator

IP1 D: Pointer to a ~~11 word~~ buffer

IP2 D: Size of buffer

If the buffer is legal, returns

- EC.FLOOR - A(block) above which compaction occurs
- GARBCNT - number of times compiled maps have been invalidated + 1
- COMPCNT - number of compactions to date + 1
- CLASCNT - last class code issued
- AUTHCNT - last capability type issued
- First available MOT slot
- Unique name for next object to be created
- Free chain pointer
- Number of cells in free blocks
- Number of cells in slop space
- Number of cells in use
- Total of previous 3
- Number of blocks in the free chain
- Number of ~~objects in ECS~~ blocks in use

K. Secret Operation (Display Object)

See process section for operations to move time between an AB & a process.

See Change Unique Name action in C-list section for revoking access to an object.

Possible errors while creating an Allocation Block

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available in that block
~~6~~	~~2~~	~~No money available in that block~~
2	4	C-list index is negative
2	5	C-list index exceeds full C-list

B. Transfer funds (and/or space) from one Allocation Block to another

IP1 C: Allocation block (donor) (OB.GIVE)
IP2 C: Allocation block (donee) (OB.GET)
IP3 D: Space to be transferred
IP4 D: Money to be transferred

Money and/or ECS space may be transferred from one allocation block to another using four parameters. The indices for the capabilities of the donor and donee allocation blocks must be given as well as the amount of money and/or space to be transferred.

Possible errors:

Class	#	Modifier	Description
6	0		No such allocation block
6	1		Money specified for ECS not available
6	2		Money specified for CPU time not available
2	0	3	Money specified for ECS is negative
2	0	4	Money specified for CPU time is negative

C. Return capability for *n*-th object in Allocation Block*

IP1 C: Allocation block (OB.GOD)
IP2 D: Full C-list index for returned capability
IP3 D: Index of desired object (*n*)

D. Destroy Allocation Block C·DELAB

IP1 C: Allocation Block to be destroyed (OB.DSTRY)

When an allocation block is destroyed, there must be no objects associated with it. The ECS space and money owned by the allocation block as well as its expenditures are reflected back to the allocation block which is its father in the tree. If the allocation block to be destroyed still has objects in its chain, it cannot be destroyed, and an F-return is made.

Possible errors:

Class	#	Description
6	0	Allocation block does not exist.

II C-List Actions

User access to all objects within the ECS system is controlled by capabilities. A capability identifies the object it refers to, specifies the type of the object, and the set of allowed actions on that object (options). Capabilities for objects accessible by a given subprocess are grouped together in capability-lists (C-lists) which are themselves objects within the ECS system. Individual capabilities are referred to by their index within a C-list. Since the capability, residing in a C-list, authorizes access to an object, the user is never allowed to fabricate a capability. The system creates a capability with all options allowed when an object is created. System actions are provided to permit the user also to create a C-list, as well as to examine a capability, to copy capabilities between C-lists and within a C-list, and to downgrade the option mask. Thus, the user can transfer the right to access an object and can curtail that access, but he may never manufacture that right or increase the set of allowable actions on the object. He must ask the system to perform these actions for him.

A C-list is assigned to every subprocess within a process. For every process there exists a sequence of subprocesses called the full path. Corresponding to the full path, the full C-list is defined as the concatenation of the C-lists belonging to the subprocesses in the full path. When referring to capabilities in the full C-list, the capability index is interpreted as if the C-lists in the full C-list were joined to form one long C-list.

A. Create a C-list

IP1 C: Capability for allocation block (OB.CRECL)
IP2 D: Index in full C-list to return new capability
IP3 D: Length of new C-list

A capability list (C-list) is a sequence of capabilities and "empty" positions. Each C-list is filled with "empties" (zero words) upon creation. To create a capability list, the user must supply the index of the Allocation block which funds the space occupied by the C-list. In addition to the length of the new C-list, the user must supply an index in the full C-list for the capability for the new C-list.

Possible errors while creating a C-list:

Class	#	Modifier	Description
6	0		Allocation block does not exist
6	1		No ECS available
6	2		No money available
2	4		C-list index is negative
2	5		C-list index exceeds full C-list
2	0	3	Length of new C-list ≤ 0
2	1	3	Length of new C-list exceeds core buffer area

CL100 checks indirect Clist reference

return! CL200 B. Display a Capability from the Full C-list   C.OSPCAP

D: Index in full C-list

When referring to capabilities within the full C-list, the capability index used is interepreted as if the C-lists in the full C-list were joined to form one long C-list. Thus, the index of the desired capability is all that is required to display it. The two words of the capability are returned in X6 and X7.

X6 =

59 option mask	17 type 0

X7 =

59 unique name 21	(unused)	17 MOT index 0

are types explained somewhere? p55

Possible errors while displaying a capability:

Class	#	Modifier	Description
2	4	1	Capability index negative
2	5	1	Capability index exceeds full C-list length

(return in CL210,211; CL220,221)

~~C. Display a Capability from an arbitrary C-list~~   C.DSPARB

IP1 C: Capability for C-list
IP2 D: Index in the C-list

To display a capability from a C-list which is not in the full C-list, the user must specify both the index of the capability for the C-list and the index within that C-list of the desired capability. The capability is returned as in B- above.

Possible errors while displaying a capability from aribtrary C-list:

Class	#	Modifier	Description
2	4	1	Capability index negative
2	5	1	Capability index exceeds C-list size
8	0		C-list does not exist

OSP.1 D. Copy a Capability within full C-list and Decrease the Options   C.MVECAP

D: Index of desired capability
D: Index of destination C-list entry
D: Mask of options to preserve ~~(in top 42 bits - bottom 18 ignored)~~

The user can copy a capability from one location in the full C-list to another and in doing so may decrease the number of allowed options. Recall that when an object is created, a capability is returned which has all the option bits (the high order 42 bits of the first word) set. The user must indicate the C-list index of the capability he wishes to copy, the C-list index where the altered capability will be placed, and a bit-mask which will be logically "ANDed" with the option bits of the original capability to produce the option mask for the new version of the capability.

Possible errors while copying a C-list and decreasing the options:

Class	#	Modifier	Description
2	4	1	Index of desired capability is negative
2	4	2	Index of destination C-list entry is negative
2	5	1	Index of desired capability is too large
2	5	2	Index of desintation C-list entry too large

E. Copy capability from Full C-list to Arbitrary C-list (and vice-versa)

IP1 C: Destination (source) C-list (OP.CPYIN, (OB.CPYOT))
IP2 D: Index within destination (source) C-list of capability
IP3 D: Index in the full C-list of source (destination) capability

In order to simply transfer a capability between the full C-list and an arbitrary C-list two parameters are required to indicate the location of the capability in the arbitrary C-list, and a third to locate the capability in the full C-list.

Possible errors:

Class	#	Modifier	Description
8	0		C-list does not exist
2	4	2	IP2 is negative
2	4	3	IP3 is negative
2	5	2	IP2 is too large
2	5	3	IP3 is too large

F. Change Unique Name in Capability

IP1 D:C-list index of object (OB.CHNAM)

This action allows the user to change the unique name of an object. The system generates a new capability for the object with all option bits set, thereby invalidating all old capabilities for that object. The capability for the object whose name is to be changed must carry the option bit which allows such a change (OB.CHNAM). If the object is a file for which there are references in any map entries, all such maps will be recompiled.

Possible errors while changing unique name:

Class	#	Description
8	1	No such object

CL11 G. Destroy a C-list C.DELCL

IP1 C: Capability for C-list (OB.DSTRY)

The user may destroy a C-list when he no longer needs it; only the index of a capability for the C-list is required. If the C-list to be destroyed is in the full path of the user's process, an F-return is initiated and the C-list is not destroyed.

Possible errors while destroying a C-list:

Class	#	Description
DSTR.3 8	0	C-list does not exist

III File Actions

Files are organized in a tree structure (see Figure 3). The leaves of the tree are called data blocks and contain the addressable words of the file. The non-terminal nodes of the file tree are called pointer blocks and contain links to either data blocks or other pointer blocks. Empty or non-existent portions of a file are not allocated space in ECS until they are needed. The user can create a file, add and/or delete parts (data blocks) of a file; he can check for missing data blocks and read the shape (parameters of the tree structure) of a file; he can transfer data blocks of the same size within a file or from one file to another, [he can check whether or not a given data block is "dirty";] and finally, he can read (write) information from (into) a file.

(addressing starts at 0)

FILE TREE

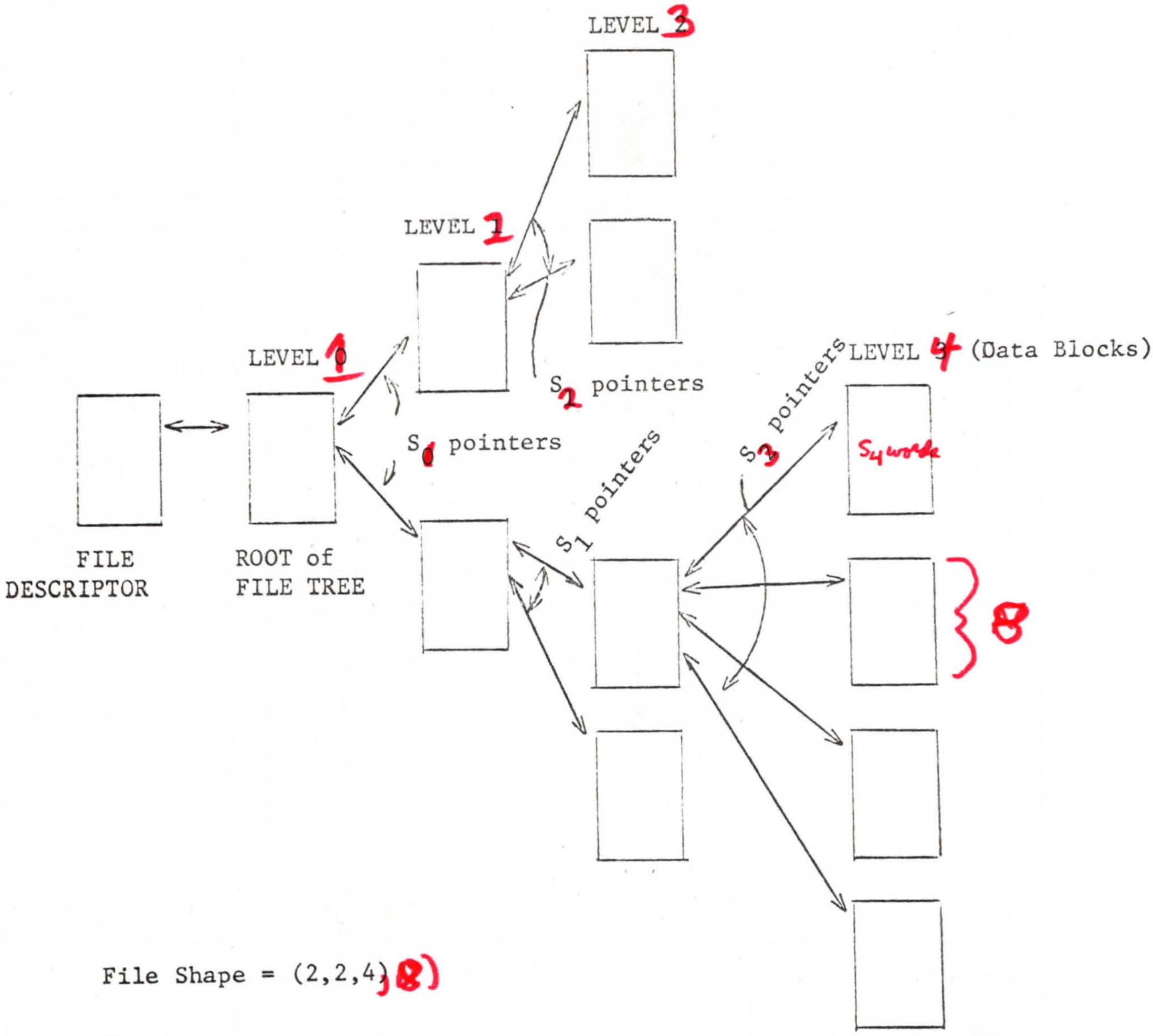


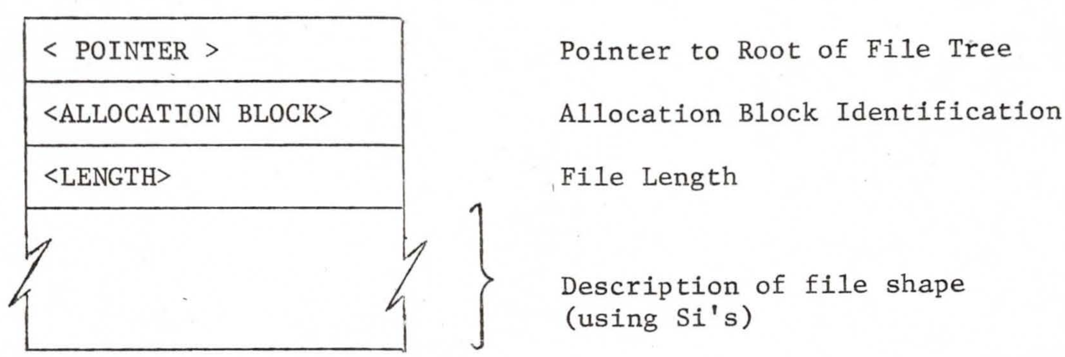
Figure 3

A. Create a File

IP1 C: Capability for allocation block (OB.CRFIL)
IP2 D: C-list index to return capability
IP3 D: Number of levels in the file
IP4 D: Pointer to a list of shape numbers

When a file is created, only the file descriptor is constructed (see Figure 4). The file descriptor contains a pointer to the root of the file tree (initially zero since no data or pointer blocks exist). The user supplies an index for the capability of the Allocation block which is to fund the ECS space occupied by the file. Identification of the funding allocation block is also kept in the file descriptor. The user must also supply a C-list index where the system will put the capability for the file being created (all option bits in the capability for the new file are turned on). The last two parameters indicate the number of levels (n) contained in the structure of the file tree, and a pointer to a list of n+1 shape numbers (S_0 through S_n), the first n of which indicate the number of branches from each block at each level; the last (Sn) gives the uniform size of all data blocks in the file. A "zero level file" (IP3 = 0) consists of a single data block of length Sn (n=0). Each shape number (S_0 excepted) must be an integral power of two. The last two parameters are used by the system to complete the file descriptor.

Figure 4 File Descriptor



$$\text{SHAPE} = (S_0, S_1, \ldots, S_n)$$

$$\langle\text{LENGTH}\rangle ::= (\text{maximum file address}) + 1 = \prod_{i=0}^{n} S_i$$

Possible errors while creating a file:

Class	#	Modifier	Description
6	0		Allocation block does not exist
6	1		No ECS Available
~~6~~	~~2~~		~~No money available~~
2	4		C-list index is negative
2	5		C-list index exceeds full C-list
2	2	4	Pointer to list of shape numbers is negative
2	0	3	Level number n ≤ 0
2	1	3	Level number is too large ($>37_{10}$)
2	1	4	Pointer to list of shape numbers plus list length exceeds user's FL
3	7		Negative shape number
3	8		Shape number exceeds 2^{17}
3	9		Shape number other than S1 not a power of 2
3	10		Total size of file is too large (how large?)

B. Create a Block

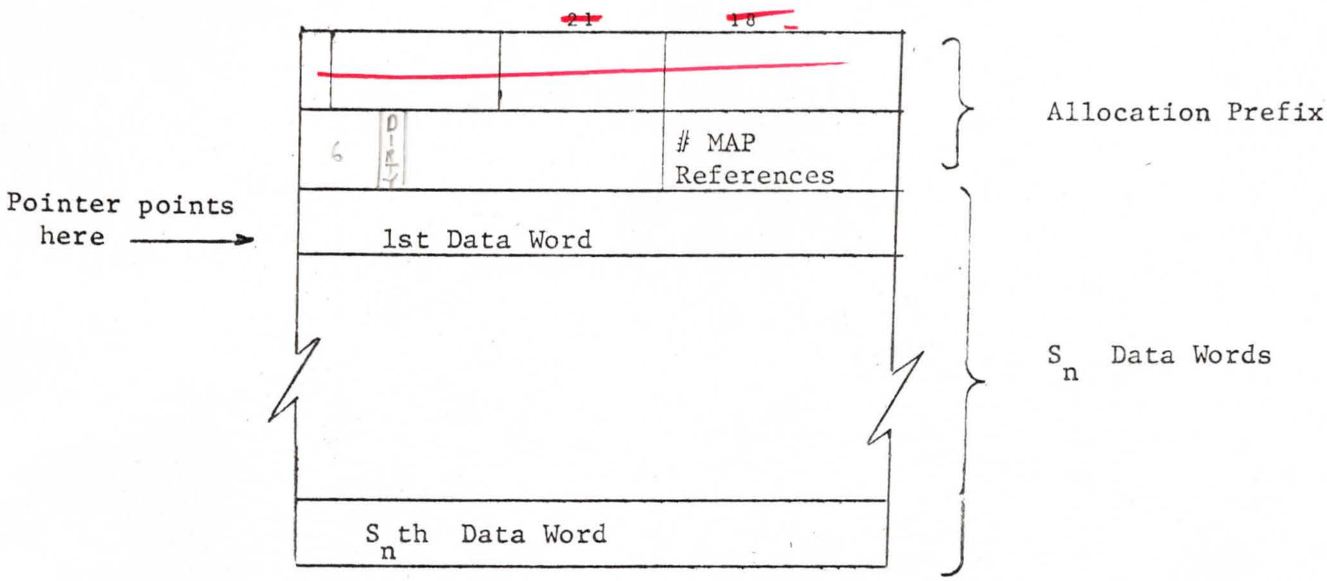
IP1 C: Capability for file (OB.CREBL)
IP2 D: Address of block in the file

Once a file has been created, data blocks of the declared length (Sn) may be added subsequently, one at a time, to hold data or code. (See Figure 5.) A count of the map entries which reference the data block is maintained with each data block. (This count is important when deleting a block - see below). To create a block, the user supplies the index of the capability for the file to which the block is being added, and the address in the file where the block is to be placed. (Any address in the block will do.)

When a data block is added to a file, it may also be necessary to create some or all of the pointer blocks between that data block and the file descriptor. Recall that pointer blocks are required to link the file descriptor to the data blocks in any file with more than one shape number (i.e., not a zero level file).

Figure 5 Data Blocks

Shape = $(S_0, S_1, \ldots, S_n)$



Possible errors while creating a block:

Class	#	Modifier	Description
6	0		Allocation block does not exist
6	1		No ECS available
~~6~~	~~2~~		~~No money available~~
3	0		The file does not exist
2	2	2	The address of the new block is negative
2	3	2	The address of the new block is greater than the file length
3	1		The address of the new block indicates an already existing block

C. <u>Check for missing blocks</u>

IP1 C: Capability for file
IP2 D: Address of block in file

Allows the user to check for the presence of a block: The parameters required are the index of the capability for the file to which the block belongs, and the address within the file where the block is supposed to be located. The number of missing levels in the path from the root of the file tree to that particular block is returned in X6. Thus, if the block is present, X6 $\leftarrow$ 0; if the n level file is empty, X6 $\leftarrow n$; and if only the data block is missing (its pointer block is present), X6 $\leftarrow$ 1.

Possible errors while checking for missing blocks:

Class	#	Modifier	Description
3	0		The file does not exist
2	2	2	The address of the block is negative
2	3	2	The address of the block is too large

D. Read the Shape of a File

IP1 C: Capability for file
IP2 D: Address of buffer for the shape numbers
IP3 D: Buffer size

The shape of a file is described by a sequence of positive integers (S0,S1, ...,Sn), each of which is the number of branches in the file tree at each node of level i ($0 \le i \le n$). Each Si ($i > 0$) must be an integral power of two. The user can obtain these shape numbers by specifying the index of the capability for the file whose shape he wants to read, and the address and size of a buffer for the shape numbers. The number of levels in the file is placed in the first word of the buffer and the shape numbers (S0,...,Sn) are placed in succeeding words until either the buffer is full or all the shape numbers have been passed.

Positive errors while reading shape:

Class	#	Modifier	Description
3	0		File whose shape is to be read does not exist
2	2	2	Buffer address is negative
2	0	2	Buffer size $\le$ 0
2	1	3	Buffer address + size exceeds user field length

E. Read (write) a File

IP1 C: Capability for file (OB.RDFIL,(OB.WFILE)
IP2 D: Address in file
IP3 D: Address in Central Memory
IP4 D: Count of words to be transferred

The action of reading (writing) a file transfers words between the address space of the running (current) subprocess and the data blocks of a file. In addition to the capability index for the file, the user specifies the address in the file of (for) the desired information, the address in Central Memory of the area to be read into (written from), and the number of words that are to be read (written). If a transfer is requested which involves a file address corresponding to a non-existent data block, the transfer proceeds until the non-existent file address is encountered, whereupon an F-return is initiated. The actions to read the shape of a file (D) and to check for missing blocks (C) can be used to check how far the transfer proceeded.

Possible errors while reading (writing a file):

Class	#	Modifier	Description
3	0		File does not exist
2	0	4	Word count negative
2	2	2	File address negative
2	2	3	CM address negative
2	1	3	CM address plus word count exceeds user's field length
2	1	4	File address plus word count exceeds ~~user's field~~ file length ?

F. Move a File Block (+ delete it!) ✓

IP1 C: Capability for source file (OB.RDFIL, OB.DELBL)
IP2 D: Address in source file of source block
IP3 C: Capability for destination file (OB.WFILE,OB.CKEBL)
IP4 D: Address in destination file of destination block

File blocks can be transferred between files whose data block sizes (Sn) are equal. In addition to the capability indices for the source and des-

The file is moved not copied.

tination files, the system expects to receive from the user the address of the source block within the source file and the address in the destination file to which the block is being moved. ✓ If the block to be moved is referenced by a map, moving it (which deletes it from the source file) would cause problems when swapping, therefore an F-return is made. Any address within each block will do

Possible errors while moving a block:

Class	#	Modifier	Description
3	2		Block to be moved is in map
3	3		Block to be moved does not exist
3	4		Files do not have equal data block sizes
2	2	2 or 4	File address negative
2	3	2 or 4	File address too large

G. File to file copy

IP1 C: Source file (OB.RDFIL)
IP2 D: Address in source file
IP3 C: Destination file (OB.WFILE)
IP4 D: Address in destination file
IP5 D: Count of words to be transferred

This action copies a specified number of words from one ECS file to another ECS file. In addition to the capability indices for the source and destination files, the system expects the user to specify the source and destination addresses and the number of words to be copied.

Possible errors during a file-to-file copy:

Class	#	Modifier	Description
2	2	2 or 4	File address is negative
2	3	2 or 4	File address is too large
2	0	5	Word count is negative
2	1	5	File address plus word count is too large

H. Delete a Block from a File

IP1 C:Capability for file (OB.DELBL)
IP2 D:Address of block to be deleted

A block can be deleted from a file as long as it is not referenced by an entry in some subprocess map (reference count = 0). The user must supply the capability index for the file and the address within the file of the block which is to be deleted. If the block is referenced by a map entry, an F-return is made.

Possible errors while deleting a block:

Class	#	Description
3	3	Block to be deleted does not exist

I. Delete a File

IP1 C: Capability for file (OB.DSTRY)

When a file is deleted, it must not contain any data blocks, i.e., it must consist only of the file descriptor. Only the capability index of the file is required as a parameter.

Possible errors while deleting a file:

Class	#	Description
3	0	File to be deleted does not exist
3	6	File to be deleted is not empty

IV Process and Subprocess Actions

Processes are the active elements of the ECS portion of the Time Sharing System. Only within the context of a process may code be executed and system actions initiated. A process consists of 1) a set of central registers (called the exchange jump package), 2) a set of subprocesses organized in a tree structure, 3) a call stack recording the flow of control among the subprocesses, and 4) a set of state flags describing the state of the process.

There are system actions to create, examine, destroy and manipulate the elements of a process. There are also actions which control the processing environment of a process by transferring control from one subprocess to another and by controlling the error processing and external interrupt status of the process.

D index in full C-list for returned class code ???

A. Create a Class Code (subprocess name) with new permament part

IP1 C: Capability for class code

A class code is a protected 60-bit datum which is used to identify a subprocess within a process and to identify classes of users to the directory system. The 60 bits are divided into two 30-bit parts; the upper 30 bits constitute the permanent part and the lower 30 bits, the temporary part. This action causes a new class code to be constructed by the system with a permanent part that is different from the permanent part of all other class codes (temporary part = 0). The Capability for the new class code is returned in the full C-list at the location specified by the parameter of the action.

Possible errors while creating a class code:

Only those detected during System entry/exit.

B. Set temporary part of class code C.NWTMP

IP1 C: Capability for class code (OB.TEMP)
IP2 D: C-list index for ~~modified class code~~ new class code
IP3 D: New temporary part (30 bits)

The temporary part specified by the user is inserted into the class code (lower 30 bits). This action may be used to create "classes" of class codes which have the same permanent part and different temporary parts. The class code with the new temporary part is returned in the full C-list at the specified location.

C. Create a Process

IP1 C: Capability for Allocation block (OB.CREPR)
IP2 D: C-list index for returned process capability
IP3 D: Number of event channel chaining words (max # ECs the process can hang on at one time)
IP4 D: Number of stack entries (subprocess call depth)
IP5 C: Capability for class code for initial subprocess (OB.SONSP)
IP6 D: Number of map entries in initial subprocess
IP7 D: Compiled map buffer size for initial subprocess
IP8 D: Subprocess field length
IP9 D: Subprocess entry point
IP10 C: Capability of C-list for subprocess (OB.LOCCL)
IP11 C: Capability of file for 1st map entry (Read/Write: OB.WFILE, OB.RDFIL, OBPLMAP) for initial subprocess
IP12 D: Address within file
IP13 D: Address in CM

OB.PLMAP

IP14 D: Count of words to be swapped
IP15 D: Capability of file for 2nd map entry (Read Only: OB.RDFIL, OB.PLMAP) for initial subprocess
IP16 D: Address within file
IP17 D: Address in CM
IP18 D: Count of words to be swapped

There are 18 parameters required for the system action which creates a process. The first four are used to construct the process descriptor while the remaining 14 are necessary to specify the initial subprocess which is created along with the process. As usual when creating any system object, the first two parameters required are the C-list index of the Allocation block which is to fund the area in ECS where the object is to be placed, and the C-list index where the system will return the capability for the object.

The data necessary to maintain and run a process are gathered together in the process descriptor, which is stored in two sections: the fixed length process descriptor and the variable length process descriptor. These two sections of the process descriptor are copied into CM when the process is being run on the CPU. While the process resides in ECS (Figure 8), the fixed length descriptor and variable descriptor are separated by the process queuing word buffer, used when a process is hung on one or more event channels. Parameter IP3, the number of E.C. chaining words, defines ~~giving~~ the size of the queuing word buffer, ~~is contained in the first word of the process descriptor.~~ + thus gives the max # of ECs that the process can hang on simultaneously.

The call stack, which records the flow of control among the subprocesses belonging to the process, is contained in the variable length process descriptor. Each entry in the call stack contains the information necessary to reinitiate processing where it was terminated due to a subprocess call. The total number of stack entries the process can accommodate is supplied by the user in IP4 when the process is created. (what about the funny top of stack?)

Among the parameters defining the initial subprocess, the first six (IP5-IP10) are used to fill in the subprocess descriptor and the last eight parameters specify the contents of the two initial map entries (Read/Write and Read Only) which control the swapping of the local address space. The data necessary to describe a subprocess are gathered into the subprocess descriptor. The user supplies 1) the class code (identifying name) of the

PROCESS DESCRIPTOR (IN ECS)

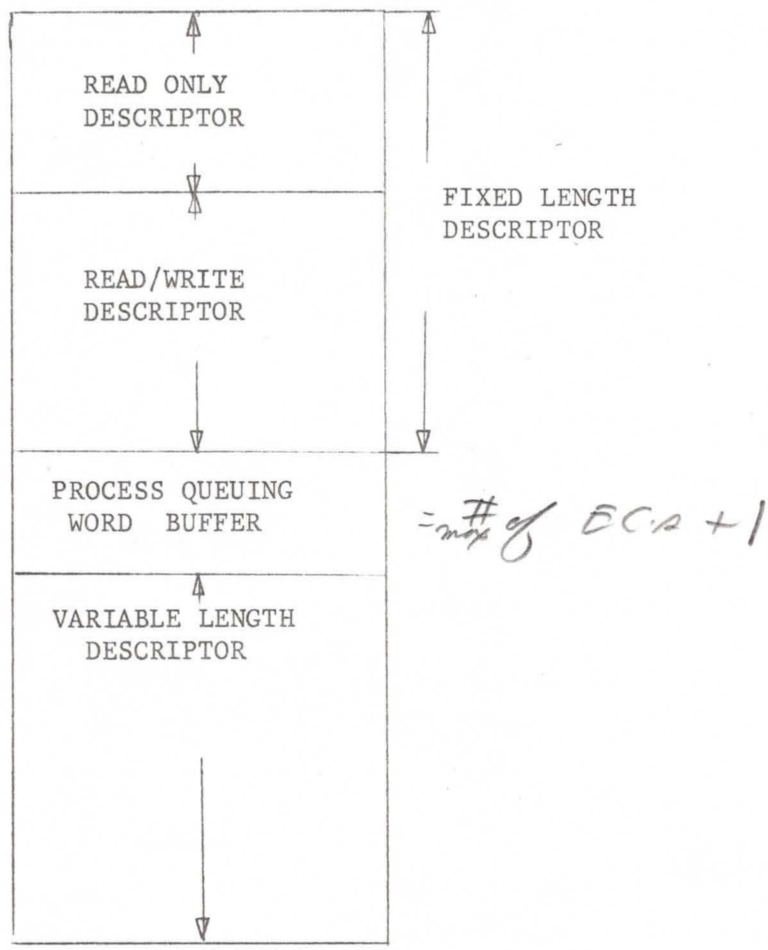


Figure 8

subprocess, 2) the number of entries which will be in the logical map, 3) the size of a buffer area which will be allocated to hold the compiled map, 4) the length of the subprocess local address space, 5) the entry point of the subprocess where execution begins when it is called, and 6) a C-list index designating the local C-list of the new subprocess. The logical map contains an entry for each contiguous portion of information which is to be copied between ECS files and the local address space in CM of a subprocess at the beginning and/or end of the processing within that subprocess. To expedite this procedure, the compiled map is generated from the logical map, using the absolute ECS addresses of the sections of ECS files referenced by the logical map entries. Since one map entry may span several data blocks in a file, the size of the compiled form of the map will increase accordingly. The length of the local address space (IP8) of a subprocess is the upper limit on the information copied into CM under the direction of the subprocesses map. The local C-list of a subprocess controls the objects which the subprocess can access.

User doesn't know how to compute size of compiled map. If he does, explain it

The eight remaining parameters specify the contents of the first two logical map entries, which describe the initial body of the subprocess. The first map entry (specified by parameters IP11-IP14) defines a portion of an ECS file which is copied into CM before processing under the control of the subprocess is initiated, and when this processing stops, is copied back into the ECS file from which it came, thereby (possibly) altering the content of the ECS file. The second map entry, however, defines a section of an ECS file which is read into CM only, and will never be copied back into ECS, thus protecting the ECS file from being altered. The parameters include the C-list index of the associated ECS File(s), the addresses in the file(s) and in CM between which the information is to be transferred (swapped) and the number of words to be swapped.

The new process, after being constructed, is scheduled to run and will begin execution at ~~the entry point~~ of the initial process.

3 cells befor the entry point

Possible errors while creating a process:

Class	#	Modifier	Description
6	0		Allocation block does not exit
6	1		No ECS available
6	2		No money available
2	4		C-list index is negative
2	5		C-list index is too large
2	0	3	Number of chaining words ≤ 0
2	1	3	Number of chaining words too large
2	0	4	Number of stack entries < 1
2	0	6	Number of map entries < 2
2	0	7	Compiled map buffer size is negative
2	0	8	Length of local address space is negative
2	1	8	Length of local address space is too large
2	0	9 or 15	Subprocess entry point < 2
2	1	9 or 15	Subprocess entry point exceeds field length
2	2	12 or 16	File address is negative
2	3	12 or 16	File address is too large
2	2	13 or 17	CM address is negative
2	3	13 or 17	CM address exceeds field length
2	0	14 or 18	Word count for map entry < 0
2	1	14 or 18	Word count for map entry too large

D. <u>Display Fixed Length Descriptor of a Process</u>

IP1 C: Capability for the process
IP2 D: Address of buffer area
IP3 D: Size of buffer area

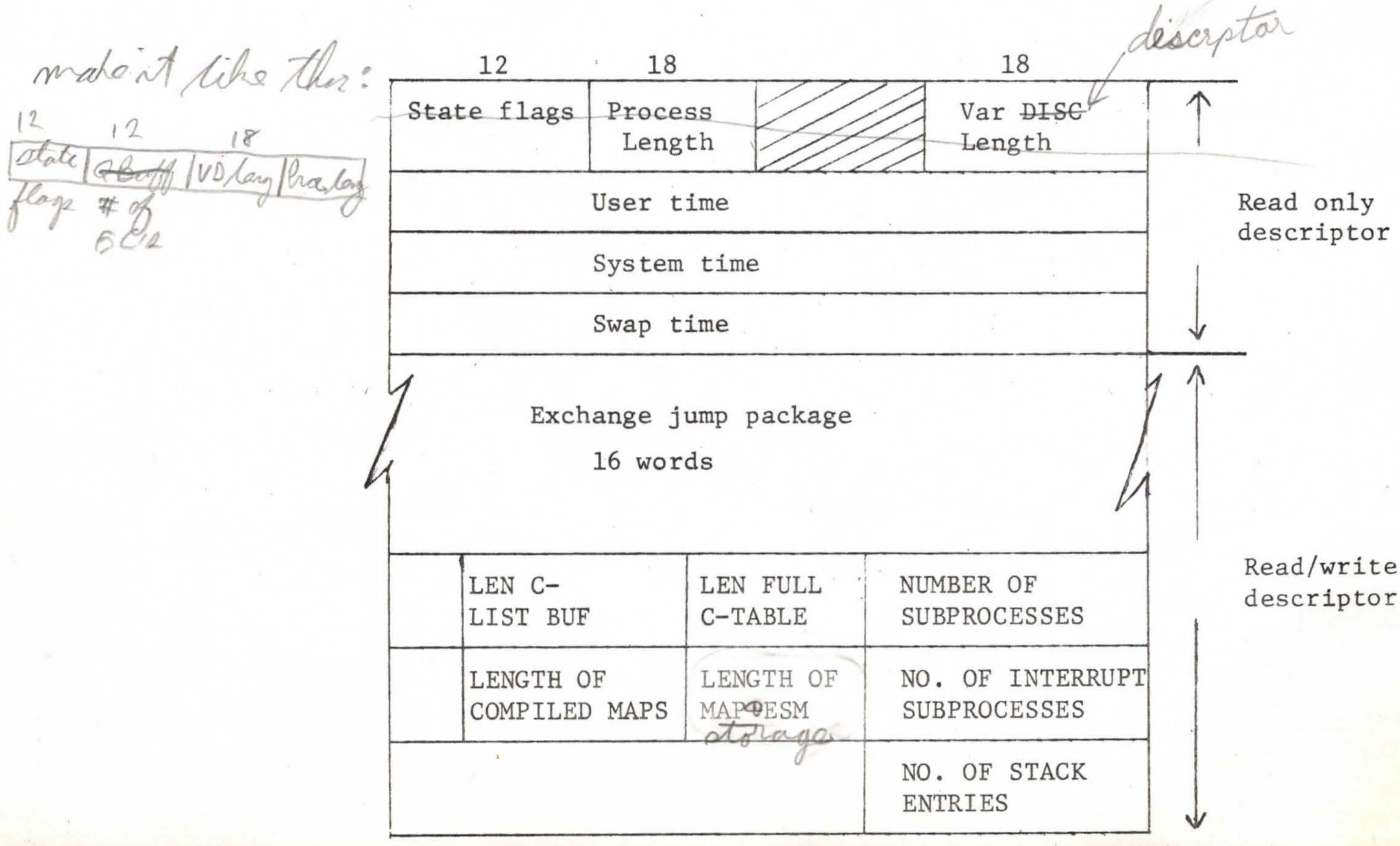
The fixed length process descriptor contains much of the information necessary to maintain and run a process (see Figure 9). It is divided into two sections: the read only descriptor and the read/write descriptor. The read only descriptor shows the state flags of the process, the length of the process, the length of the variable length descriptor and the clock times consumed by the user, the system and in swapping, respectively. The read/write portion of the fixed length descriptor contains the process exchange jump package as well as data and pointers used to maintain portions of the variable length process descriptor: the full C-list table, call stack, the subprocess descriptor table, logical map and error selection mask (ESM) storage, and compiled map storage.

In order to display the fixed length process descriptor, the user supplies the index for the capability for the process whose fixed length descriptor is desired, an address within the user's FL where the information will be displayed and the length of this area which, to hold all the information, should be 23 words long. The system will copy as much of the fixed length process descriptor into this user area as there is room for. The information has the format given below in Figure 9.

Possible errors:

Class	#	Description
2	2	Address is negative
2	3	Address exceeds user's FL
2	0	Length of area ≤ 0
2	1	Address plus length exceeds user's FL

Figure 9 Display of Fixed Length Process Descriptor



Process State Flags

Eight flags describe the state of the process. These state flags are used primarily to control the swapper, but are set and checked by other routines (event channel, process interrupt, and destroy process). The eight flags function as follows:

flags given from left?? no

The E flag indicates that the process is actually a pseudo-process and is used by the event channel routines to distinguish between genuine and pseudo-processes.

The "in core" flag, C, is set whenever the process is actually running on the CPU. This flag is checked by the process interrupt routine.

The "pending action" flag, P, directs the swapper to interrogate the "W", "I", "D" amd "V" flags. These four flags cause the swapper to:

- W - (the wakeup waiting flag) unchain the process flow from the event channels;
- I - check the "ancestors" of the current subprocess for an interrupt process;
- D - destroy the process; and
- V - modify the swapper return because of the arrival of an event for the process.

The "running flag", R, indicates that the process is scheduled to run or is running on the CPU. The running flag (R) and the wake-up waiting flag (W) interact in the event channel routines as well as in the process interrupt routines. They are used to permit the process to "hang" on several event channels and still be able to accept an incoming event.

tell where the flags are!

E. Display user clock times

IP1 D: Address of buffer area in user's FL

The current times on the following five clocks: real clock, user clock, system clock, swapping clock, and quantum clock, are displayed in consecutive words beginning at the address supplied by the user. The buffer area should be at least five words long since this action causes 5 words to be passed.

Possible errors while displaying clock times:

Class	#	Modifier	Description
2	2	1	Buffer address is negative
2	3	1	Buffer address plus 5 exceeds user's FL

F. Creating a Subprocess

IP1 C: New subprocess class code (OB.SONSP)
IP2 C: Class code of the "father" of the subprocess (OB.FATHR)
IP3 D: Number of map entries
IP4 D: Compiled map buffer size
IP5 D: Subprocess FL
IP6 D: Subprocess entry point
IP7 C: Subprocess local C-list index (OB.LOCCL)

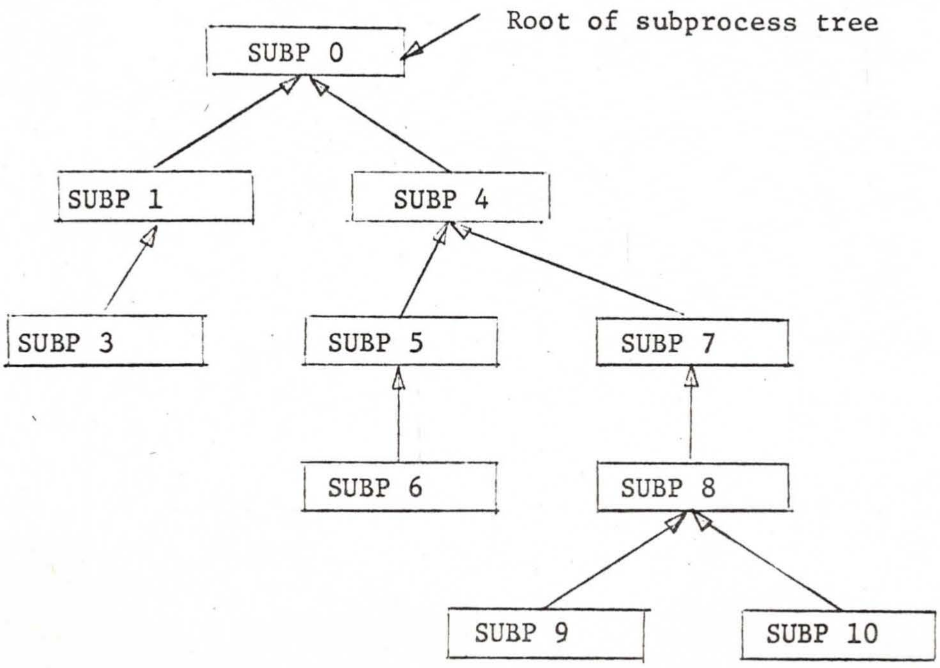
The action of creating a subprocess involves constructing the 8 word subprocess descriptor. The parameters are similar to those required to create the initial subprocess except for IP2 and the absence of logical map entry parameters. The subprocesses in a process are organized in a tree structure in which each subprocess "points" only to its predecessor ("father") (see Figure 10). For each subprocess, the term "ancestors" refers to the sequence of subprocesses which starts with the subprocess and terminates with the root of the subprocess tree. Note that a subprocess is always an "ancestor" of itself. The term "son" of a subprocess refers to any of the subprocesses for which that subprocess is the "father".

Each newly created subprocess is linked into the subprocess tree at the subprocess referenced by IP2. Note that since no map entries are made for the subprocess at the time of its creation, they must be constructed via the appropriate system actions in order to provide executable code and a data area for the subprocess, before the subprocess can be used. Note also that since the first few cells of the subprocess address space are used for storing the parameters of subprocess calls, they should be given a read/write map entry. expand this + see pg 32

Possible errors while creating a subprocess:

Class	#	Modifier	Description
6	0		Allocation block does not exist
6	1		No ECS available
6	2		No money available
4	0		Duplicate subprocess name (same as some other subprocess in the process)
4	1		"Father" does not exist
2	0	3	Number of map entries ≤ 0
2	1	3	Number of map entries exceeds field length
2	0	4	Compiled map buffer size is negative
2	0	5	Subprocess field length < 0
2	1	5	Subprocess field length is too large how large
2	0	6	Entry point < ~~2~~ 8?
2	1	6	Entry point > FL
4	3		No space for compiled map
8	0		C-list does not exist
4	4		Process becomes too big for CM size of machine

Figure 10 Subprocess Tree



G. Display Subprocess descriptor

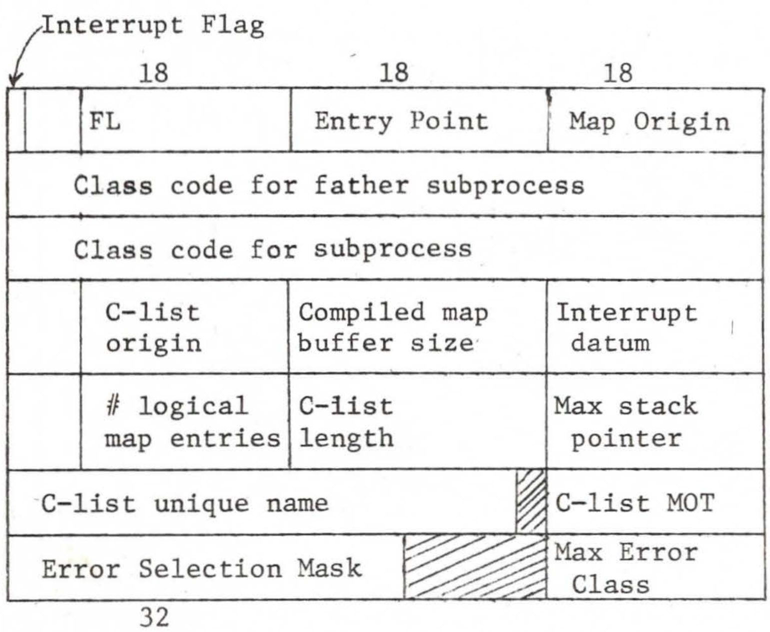
IP1 C: Capability for class code (subprocess name)
IP2 D: Address of buffer area
IP3 D: Size of buffer area

This action allows the user to display a subprocess descriptor in a designated area within his own FL (see Figure 11). The system copies the subprocess descriptor into the user's area starting at the address specified by the second parameter and ending either with the last word of the displayed subprocess descriptor (7 words) or the last word of the buffer area, whichever comes first. The contents of the subprocess descriptor are described above (p. 24).

Possible errors:

Class	#	Modifier	Descriptor
4	5		Subprocess does not exist
2	2		Address is negative
2	3		Address exceeds user's FL
2	0		Length of buffer area ≤ 0
2	1		Length of buffer area too large

Figure 11 Display of Subprocess Descriptor



how do you specify the subprocess you're calling? i.e., what are the IPs?? see VII A

H. Subprocess Call

A normal subprocess call is initiated by calling on the system in the usual manner, using an operation whose action is "subprocess call". A normal subprocess call may also be initiated as the result of F-return action under the control of a multi-ordered operation (see p. 4 above). A new processing environment is established (described below) as a result of the transfer of control to a different subprocess. At any given time, there are two distinguished subprocesses within a subprocess. They are the current subprocess and the end-of-path subprocess. (Note that the current subprocess is always an "ancestor" of the end-of-path subprocess.) The sequence of subprocesses from the end-of-path to the current subprocess (inclusive) is called the full path. The end-of-path is defined dynamically by the flow of control among the subprocesses. The current subprocess may be considered to be the subprocess currently in control. The end-of-path and current subprocesses are reassigned whenever a new subprocess is called. The subprocess being called (the callee) becomes the new current subprocess. If the callee is an "ancestor" of the old end-of-path, the end-of-path remains unchanged. If the callee is not an "ancestor" of the end-of-path, the new end-of-path becomes the same as the callee (i.e., the full path consists of a single subprocess - the callee). See Figure 12.

The full path determines the sphere of protection invoked by the current subprocess by defining the full C-list, full map, and full address space. The access afforded the current process to other objects within the system is controlled by the full C-list. The full map determines the configuration of the address space available to the current subprocess and the full address space is the size of the address space available to the current subprocess. The configuration of the subprocess tree defines the static relationship between the subprocesses (subprocesses closer to the root may be given the privileges of their descendents) while the full path dynamically controls the boundaries of access applied to the current subprocess. This system of controlling the bounds of protection allows the construction of processes which may exercise varying degrees of protection while maintaining synchronization between the subprocesses involved.

Figure 12 Full Path Example using Tree in Figure 10

CALLING SEQUENCE			CURRENT SUBP	END-OF-PATH SUBP	FULL PATH
		SUBP0	SUBP0	SUBP0	SUBP0
SUBP0	calls	SUBP9	SUBP9	SUBP9	SUBP9
SUBP9	calls	SUBP6	SUBP6	SUBP6	SUBP6
SUBP6	calls	SUBP4	SUBP4	SUBP6	SUBP6,5,4
SUBP4	calls	SUBP0	SUBP0	SUBP6	SUBP6,5,4,0
SUBP0	calls	SUBP5	SUBP5	SUBP6	SUBP6,5
SUBP5	calls	SUBP3	SUBP3	SUBP3	SUBP3

A subprocess call also causes a new stack entry to be constructed and placed on the call stack. Stack entries are used to re-establish the correct processing environment during subprocess returns. Cells 0 and 1 of the full address space are zeroed (these cells are used by the hardware Arith Error mechanisms and to simulate SCOPE system calls). In addition, if the calling subprocess is a member of the new full path, the origins (relative to the new environment) of the address space, C-list, and map of the calling subprocess are computed and stored in cells 2, 3, and 4 of the new address space. If the calling subprocess is not a member of the new full path, then these cells are zeroed. The parameters of the subprocess call are copied to the new address space starting in cell 5.

For a normal call the parameters of the call are first formatted in the actual parameter area of the process descriptor by the system entry mechanism. These parameters are drawn from the calling subprocess input parameter list (IP list) under the direction of the operation being used for the subprocess call (IPO). In addition to formatting the actual parameter list, the system entry routine places the name (class code) of the called subprocess, the number of parameters, and a bit string denoting the types (capability or datum) of the parameters at the end of the actual parameter area. After establishing the correct processing environment for the called subprocess, the parameters are transferred to the local address space and local C-list of the called subprocess. Datum parameters are simply copied to the next parameter cell in the local address space. Capability parameters are copied to successive positions in the local C-list and the index of the parameter in the local C-list is stored in the next parameter cell in the local address space. On the completion of the parameter passing, execution is initiated at the entry point of the called subprocess.

Possible errors during subprocess call:

Class	#	Description
4	5	Named subprocess does not exist
4	6	No room on stack for subprocess
4	7	No room for parameters
4	8	Too many capability parameters
8	0	Local C-list does not exist

I Subprocess Return

Like the subprocess call, the subprocess return must construct a new processing environment before returning control to the user. The return routines reactivate a subprocess using information left in a stack entry. The full path recorded in the stack entry is sufficient to reconstruct the processing environment. The P-counter from the stack entry controls where in the subprocess execution is re-initiated. The normal return causes the P-counter to be modified by adding the low order 18 bits of the CEJ instruction which originally caused control to pass to another subprocess. (See p. 1 above.) ?

will change with new para type

Possible errors during subprocess return:

Class	#	Description
4	9	Stack empty
2	2	P-counter < 0
2	3	P-counter exceeds field length

J Subprocess F-return

A subprocess (or the system) may initiate an F-return whenever F-return processing is appropriate. F- return processing causes the operation which called the subprocess (system) to be re-examined for additional actions (see Requesting a System Action). The operation is located (after re-establishing the processing environment of the previous subprocess) by using the "last IP list pointer" stored in the stack entry for the previous subprocess. If the F-return count (also saved in the stack) is not equal to the number of orders in the original operation, the F-return count is incremented and the next order of the operation is processed. (Note that the action of all orders other than the first is "subprocess call" or "subprocess jump".) Otherwise, control returns to the subprocess which originally called the operation, but the P-counter of that subprocess is not incremented as it is for the normal return.

Possible errors during a subprocess F-return:

Class	#	Description
4	10	Stack empty
7	0	IP0 is not a capability for an operation
7	1	Operation does not exist
2	1	IP list is too big

I. Subprocess Jump Return

IP1 C: Capability for class code for subprocess to return to (OB.SPRET)
IP2 D: Number of stack occurrences of IP1 to skip (0=1, -1=down to last)

The subprocess jump return provides a method for getting calls off of the process call stack. The user specifies the class code for the subprocess to which the return is to be made. In addition, he indicates the number of occurrences of that subprocess in the call stack which should be skipped in looking for the call which is to become the new top of the stack. Zero indicates the first (most recent) call whereas -1 indicates the last (earliest) call. Upon finding the proper stack entry, the stack is reduced to make that entry the top of stack and normal subprocess return action is initiated.

J. Return with Error

IP1 D: Error class
IP2 D: Error number

The subprocess which requests this action will be removed from the top of the call stack and error processing for the error designated by the two parameters will be initiated.

Possible error

Class	#	Description
10	0	No subprocess to handle error

K. Modify P-counter of subprocess

IP1 C: Capability for class code for subprocess (OB.PCNT)
IP2 D: Number of stack occurrences of IP1 to skip
IP3 D: New P-counter

The user can modify the P-counter in a subprocess which has already been called by identifying the subprocess, the number of stack occurrences of the subprocess to skip (see H above) and the new P-counter. The P-counter is modified in the stack and the new P-counter will be used the next time that entry becomes the top of the stack. If the caller attempts to modify his own P-counter, an F-return is made.

Possible errors while modifying the P-counter:

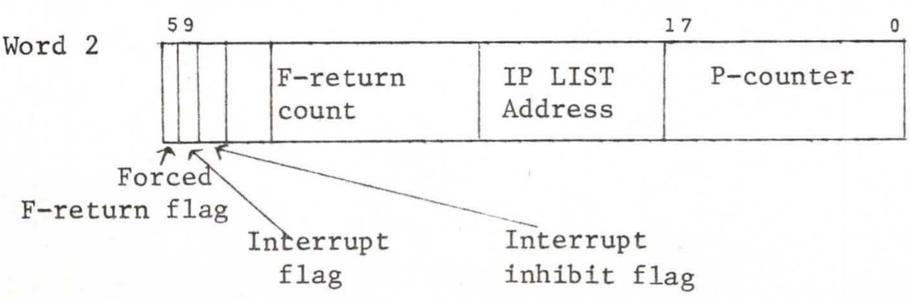
Class	#	Description
2	2	P-counter is negative
2	3	P-counter exceeds user's FL

L. Display Stack

IP1 D: CM address of a buffer area
IP2 D: Size of buffer area ($\geq$ 4)

The user may examine the call stack of a process. He must supply the address of a buffer area and its length so that the system can copy the stack into the specified area. The number of entries in the stack is stored in the first word of the buffer. As many entries as possible starting with the current top of stack are then copied into succeeding 3 word sections of the buffer. The stack entries are reformatted.

Word 0 Capability for class code for current subprocess
Word 1 Capability for class code for end-of-path subprocess
Word 2



Possible errors while displaying stack:

Class	#	Modifier	Description
2	2	1	CM address negative
2	3	1	CM address exceeds user's FL
2	0	2	Size of buffer area < 4
2	1	2	CM address + size of buffer area exceeds user's FL

M. Display Stack Entry

IP1 D: CM address of buffer
IP2 D: Desired stack entry

A particular entry in the call stack of a process can be examined if the system is supplied with the CM address of a buffer area (each entry is 3 words long) and the index (relative to the top of the stack) of the desired stack entry. Format same as in L above.

Possible errors while displaying a stack entry:

Class	#	Modifier	Description
2	2	1	CM address is negative
2	3	1	CM address exceeds user's FL
2	2	2	Stack entry pointer negative
2	3	2	Stack entry pointer exceeds stack
2	1	2	CM address plus 3 exceeds user's FL

N. Send Process Interrupt

IP1 C: A process (OB.SDINT)
IP2 C: Capability for class code for a subprocess (OB.INTSP)
IP3 D: An 18 bit interrupt datum

The process interrupt is one of the two ways in which a running process may effect the execution of another process (the other is via an event channel). The process interrupt enables one process to force the calling of a specified subprocess (IP2) (called the interrupt subprocess) within another process (IP1) (called the interrupted process); i.e., the first process forces the interrupted process to call the interrupt subprocess. However, the interrupt is given a "priority" in that the interrupt subprocess will not be called unless (or until) it is an "ancestor" of the "current subprocess", that is, of the subprocess which is actually executing in the interrupted process at the time of the call (or thereafter). Therefore, how soon the interrupt subprocess gets entered depends upon its position in the subprocess tree and the flow of control in the interrupted process. An 18-bit interrupt datum (IP3) is passed in cell 2 as the parameter of the call of the interrupt subprocess. Once a subprocess becomes an interrupt subprocess, and until that subprocess is called as an interrupt subprocess, all subsequent interrupts to that subprocess are disabled (have no effect).

The disposition of the interrupt is returned to the user in X7.

X7 = 0	Interrupt sent and interrupted process is running
X7 = 1	Interrupt process currently "in core" of ~~another~~ some CPU (Best to try again)
X7 = 2	Interrupt subprocess ~~is~~ already ~~an interrupt subprocess~~ has interrupt pending
X7 = 3	Interrupt sent but interrupted process is not running

(interrupt not sent ?? correct)

Since each subprocess is technically its own ancestor, it is necessary when an interrupt subprocess is called to automatically inhibit interrupts for the current (= interrupt) subprocess. When interrupts are inhibited for a subprocess, an interrupt to the subprocess will be remembered but cannot cause the interrupt subprocess call as long as the interrupt inhibit is set and the subprocess in question is the current subprocess. who resets the bit? explicit call by ??

At every normal subprocess call and return, a check is made for waiting interrupt subprocesses (subprocesses for which a process interrupt has been issued but which have not yet happened to be the ancestor of any current subprocess). If any interrupt subprocesses are waiting, the ancestors of the new current subprocess are checked to see if any of them is an interrupt subprocess. If so, the interrupt subprocess is called. Execution in the interrupt subprocess begins two words before its normal entry point.

The interrupt subprocess is responsible for saving & restoring registers for his descendants (i.e., a SAVREG + RESTREG at the beginning & end of his execution may be appropriate).

Possible errors while sending a process interrupt:

Class	#	Modifier	Description
5	3		Process does not exist
4	5		Subprocess does not exist in designated process
2	1	3	Interrupt datum exceeds 18 bits

O. Set/Clear Interrupt Inhibit of Current Subprocess

These parameterless action(s) allow the user to clear the interrupt inhibit flag which is normally in effect for the current subprocess if it was called as an interrupt subprocess. The interrupt inhibit flag can also be reset once it has been cleared.

Possible errors:

None.

P. Reduce/Restore Path of Current Subprocess

No parameters.

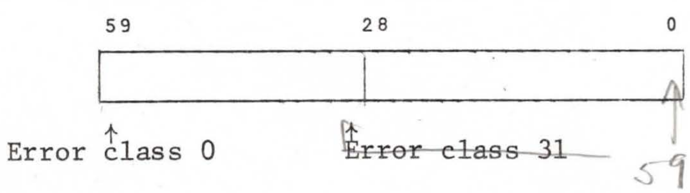
The user may reduce the path of the current subprocess, i.e., the chain of subprocesses from the root of the path to the current subprocess, so that it consists of just one subprocess, the current subprocess itself. Once the path has been reduced, it may be restored again using this action.

There are no possible errors.

Q. Set Local ESM (Error Selection Mask)

IP1 D: Pointer to new ESM

The error selection mask, which determines which classes of errors a subprocess can handle, may be set in the current subprocess by specifying a pointer to the new ESM. The ESM is a bit string (32 bits per word) in which a 1 indicates acceptance of the corresponding error class; i.e.,



Possible errors while setting local ESM:

Class	#	Modifier	Description
2	2	1	Pointer to ESM < 0
2	3	1	Pointer to ESM > FL

R. Set ESM in any subprocess

IP1 D: Pointer to new ESM
IP2 C: Capability for class code (subprocess name) (OB.STESM)

By specifying the name (class code) of a subprocess in addition to a pointer to a new ESM, the Error Selection Mask for any given subprocess may be reset.

Possible errors while setting ESM in any subprocess:

Class	#	Modifier	Description
4	5		Subprocess does not exist
2	2	1	Pointer to ESM < 0
2	3	1	Pointer to ESM > FL

S. Destroy Process

IP1 C: Capability for process to be destroyed (OB.DSTRY)

The system action of destroying a process requires only a parameter giving the C-list index of the process which is to be deleted. The process will be removed from any event channels on which it is waiting and its address space in ECS released.

Possible error while destroying a process:

Class	#	Description
5	3	Process does not exist

T. Destroy a Subprocess

IP1 C: Capability for class code of subprocess to be destroyed (OB.DSTRY)

A subprocess can be destroyed if it is currently a leaf of the subprocess tree; otherwise an F-return will be made. If the subprocess is in the call stack, an error is generated.

Possible errors while destroying a subprocess:

Class	#	Description
4	5	Subprocess does not exist
4	11	Attempt to delete subprocess in stack
4	11	Attempt to delete root of a subprocess tree
4	11	Subprocess is pointed to by another subprocess

X U. Save (Restore) Registers

IP1 D: Pointer to 16 word buffer for registers

The exchange jump package for a process can be saved in (restored from) the user's area if a pointer to a 16 word buffer is specified. When the registers are restored, only the programmable registers (A,B and X) are restored.

Possible errors while saving (restoring) registers:

Class	#	Description
2	2	Pointer to buffer is negative
2	3	Pointer to buffer is too large (within 16 words of user's FL)

V Map Actions

Associated with each subprocess is a map which directs the swapping of the subprocess address space between Central Memory and ECS files. A map consists of a fixed length sequence of map entries each of which is either zero or contains a swapping directive. The user may zero or change a map entry, and may display an entry from the full map or from the map associated with any given subprocess. A swapping directive consists of 1) an ECS file, 2) a file address, 3) a central memory address, 4) a word count, and 5) a read-only flag. Thus the map indicates what portions of which files are copied to/from specified portions of the subprocess space at the beginning/ end of processing.

When swapping a subprocess, the entries in the logical map (see Figure 6) are processed in the order of their appearance. To speed up the swapping process, the entries of the logical map are "compiled" to absolute ECS and CM addresses. Each file data block carries a count of all logical map entries which reference it. This "reference count" is important since the absolute ECS addresses associated with the "compiled" map (see Figure 7) are sensitive to 1) garbage collections and 2) deletion of data blocks. Before any of the swapping directives in a map are executed, the "local garbage collection count" is compared to the "global garbage collection count". If they do not match, the map must be recompiled since some file block may have been moved in ECS.

A. Zero a Map Entry

IP1 C: Capability for class code (subprocess name) (OB.CHAMP)
IP2 D: Index in logical map of the subprocess
IP3 C: cap for file in current entry (OB.PLMAP)

When zeroing a map entry, the user specifies the name of the subprocess (class code) whose map entry is to be zeroed, and the index of the entry in the subprocess logical map. If 0, IP3 not used; otherwise IP3 must match entry. If the map is part of full map and if it is a read/write entry, then that area is swapped out before the entry is zeroed. The result is that when the subprocess address space is swapped between ECS and Central Memory, the portion of the address space formerly referenced by the zeroed entry will not be swapped.

Possible errors while zeroing a map entry:

Class	#	Modifier	Description
4	5		Subprocess does not exist
2	2	2	Negative map index
2	3	2	Map index exceeds map length
11	0		Attempt to change or zero DAE (Direct Access Entry)
11	5		IP3 doesn't match existing entry

B. ~~Change~~ {create} a map entry (read/write or read only)

IP1 C: Class code of subprocess whose entry is to be changed (OB.CHMAP)
IP2 D: Index of entry in logical map of subprocess
IP3 C: Associated file (read only: OB.PLMAP, OB.RDFIL;
read/write: { OB.PLMAP, OB.RDFIL, OB.WFILE }
IP4 D: Address in file

IP5 D: Address in CM
IP6 D: Word count of new entry

When a map entry is changed, care must be taken if the map involved is part of the full map. In this case, the same procedure must be followed as in zeroing a map entry. The new entry is then constructed and swapped in. Note that overlapping map entries will behave oddly since the portions swapped under one map entry may be partially or completely overwritten by the information swapped under a subsequent map entry.

Possible errors while changing a map entry:

Class	#	Modifier	Description
4	5		Subprocess does not exist
2	2	2	Negative map index
2	3	2	Map index exceeds map length
			Missing block encountered
4	3		Buffer full
2	2	4	Negative file address
2	2	5	Negative word count
3	11		File address + word count exceeds file size
3	11		CM address + word count exceeds field length

C. Display a Map Entry from the Map of a Named Subprocess

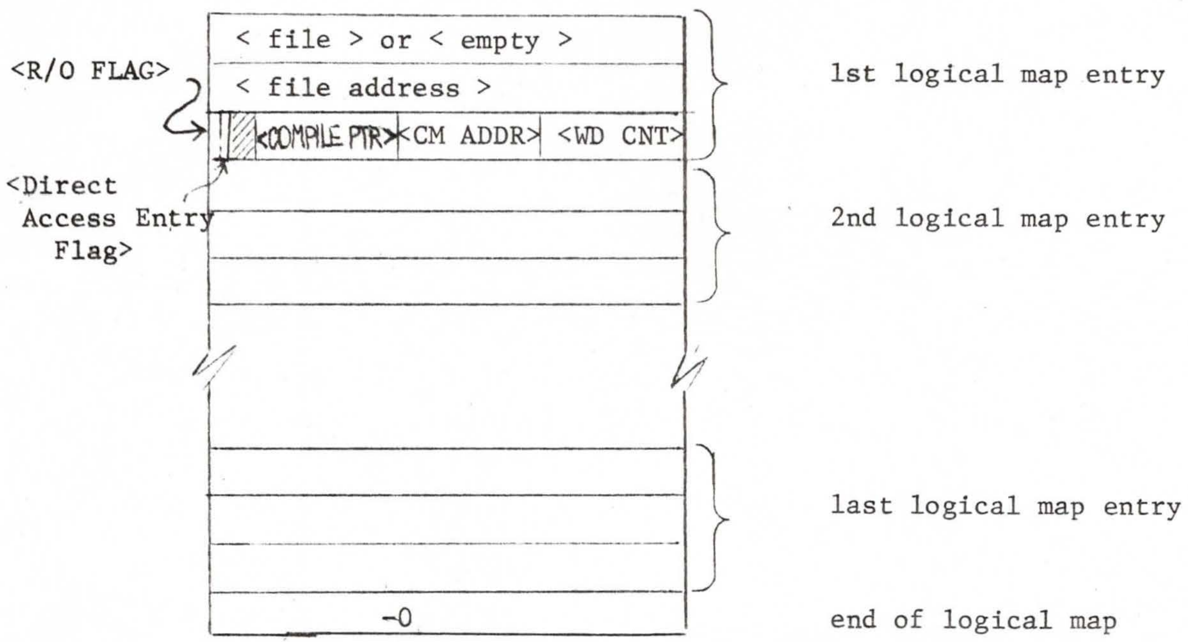
IP1 C: Class code of subprocess whose entry is to be displayed
IP2 D: Index of entry in logical map of subprocess
IP3 D: Address of a 3 word buffer

This action will insert into the 3 word buffer area (IP3) the current contents of the indicated map entry of the subprocess specified. Note that the length of the map (maximum for IP2) can be obtained by using the Display Subprocess Descriptor action. The three words of the designated map (see Figure 6) are copied to the specified buffer.

Possible errors while displaying a map entry:

Class	#	Modifier	Description
4	5		Subprocess does not exist
2	2	3	Negative address for buffer
2	2	3	Buffer address + 3 exceeds user's FL
2	2	2	Negative map index
2	1	2	Map index too large

Figure 6 Logical Map



< empty > ::= +0 Denotes an "empty" map entry

< file > ::= | UNIQUE NAME (39) | MOT INDEX (18) | file identification

< file address > ::= $0 \rightarrow 2^{60} - 1$

< R/O FLAG > ::= 1 ⇒ read only; 0 ⇒ read/write

< compile ptr > ::= index in compiled map buffer of first compiled map entry for this swapping directive

< CM ADDR > ::= CM address within subprocess local address space

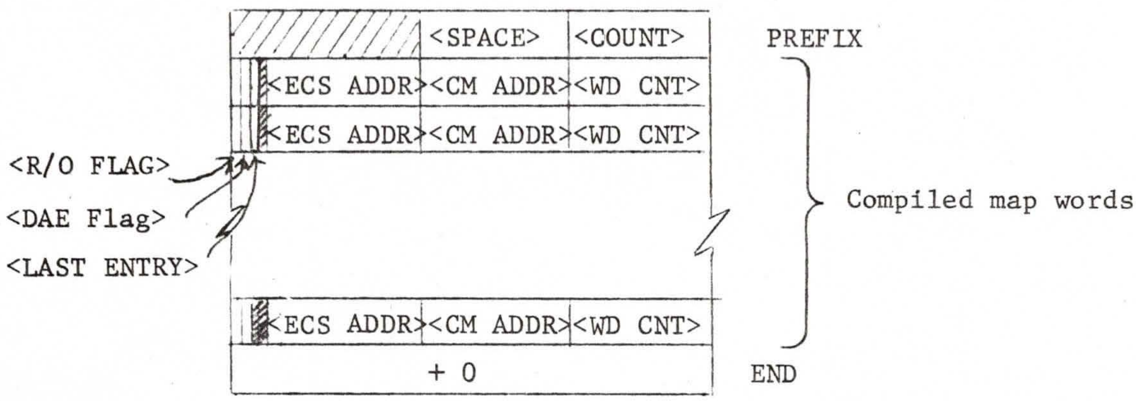
< WD CNT > ::= word count

Note: < CM ADDR > + < WD CNT > ≤ length of subprocess local address space

< DAE Flag > ::= 1 -- this is a direct ECS access entry (Legal only for first entry) in the logical map?

0 -- regular map entry

Figure 7 Compiled Map



< COUNT > ::= { 0 ⇒ must recompile
  >0 ⇒ map is good if same as GARBCNT

< SPACE > ::= number of un-used words in the compiled map buffer

< WD CNT > ::= number of words to transfer

< CM ADDR > ::= CM address relative to CM process origin (B1)

< ECS ADDR > ::= absolute ECS address to start transfer

< R/O flag > ::= read only flag { 0 ⇒ read/write
  1 ⇒ read only

< DAE flag > ::= 1 -- DAE (legal only on 1st entry in compiled map)

< last entry > ::= 1 -- last compiled map word corresponding to a particular swapping directive

D. Display Entry in Full Map

IP1 D: Index of entry in full map
IP2 D: Address of a 3 word buffer

The maps of the subprocesses in the full path are concatenated to form the full map in much the same way as the full C-list is formed. An entry in the full map can be displayed if the index of the entry in the full map is given along with the address of a buffer where the entry should be "displayed". The format of the display entry is the same as for named subprocess version of Display Map Entry.

Possible errors in display entry in full map:

Class	#	Modifier	Description
2	3	1	Index of entry too large (exceeds length of full map)
2	2	1	Index of entry negative
2	2	2	Pointer to buffer negative
2	3	2	Pointer to buffer + 3 greater than user's FL

Direct User Access to ECS

To afford the user an ECS RA and FL so that he may access an often used segment of ECS directly, the system permits the current subprocess to have a single direct access entry (DAE). This DAE must be the first entry in the logical map; it may reference only one file block (due to obvious physical limitations), and is set and cleared via two special actions described below. A DAE map entry has only two features which distinguish it from other map entries: 1) the CM address portion is always zero, and 2) the DAE flag (in first entry only) is set.

E. Set Direct Access Map Entry

IP1 C: Capability for class code (OB.DAE)
IP2 C: File (OB.FDAE)
IP3 D: File address ~~of beginning of block~~
~~IP4 D: Word count (mod 100_8)~~

This action sets the direct access ECS entry in the map of the subprocess named by the first parameter. [The map gives access to the block containing the specified addr.] ~~The file address must be the beginning of a block and the word count~~ [The block] must be a multiple of 64 words since storage handling in ECS is in 64 word blocks.

[what if the block is < 64 words?]

Possible errors while setting a DAE:

Class	#	Modifier	Description
2	2		File address negative
2	3		File is too large
~~11~~	~~1~~		~~Word count extends to more than 1 block~~
~~2~~	~~0~~		~~Word count negative~~
?	?		Block doesn't exist

F. Clear the Direct Access Map Entry

IP1 C: Capability for class code (OB.CHMAP)

This action clears the direct access entry in the map of a named subprocess. The only parameter required is a class code (IP1). The action is equivalent to A. above except that it may only be used on DAE's.

Possible errors while clearing DAE:

Class	#	Modifier	Description
4	5		Subprocess does not exist
11	2		Attempt to zero swapping directive

VI Event Channel Actions

Event channels are ECS objects which are used to synchronize the behavior of running processes as well as to implement "block" and "wake-up" mechanisms. Events consist of two 60 bit words: the first identifies the sending process; the second is a 60-bit datum. Each event channel should handle a particular kind of event. The user can create an event channel, send an event, get an event from an event channel, get an event from any one of a list of event channels, and destroy an event channel. If the user attempts to get an event from a channel which has no events, the user's process is either blocked (stops running) until some other process sends an event to the event channel, or F-return action is initiated.

EV10 — A. Create an Event Channel

EV150

IP1 C: Capability for allocation block (OB.CREEC)
IP2 D: C-list index for new event channel capability
IP3 D: Number of events that queue can hold

When an event channel is created it consists of a three word header and an event queue which is initially empty. The header words are used to maintain the queue of events and a queue of waiting processes, which develops if the queue of events becomes empty and processes request events from that channel. When creating an event channel, the user specifies the name of the Allocation block which funds the ECS space occupied by the event channel, a C-list index where the system can put the capability (with all options allowed) for the event channel when it creates it, and the length (number of possible events) of the event queue.

Possible errors while creating an event channel:

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available
6	2	No money available
2	4	C-list index is negative
2	5	C-list index exceeds full C-list
9	0	Length of event queue ≤ 0
9	1	Event queue too large

B. Send an Event ~~(with/without duplicate event checking)~~

IP1 C: Capability for the event channel (OB.SNDEV)
IP2 D: Datum part of event

These actions allow the user to send an event to an event channel. He specifies the index of the capability for the event channel and specifies a 60-bit datum to be passed with the event. The system responds by indicating the disposition of the event to the user in X6. The following responses are possible:

Condition	Response
Event put in event queue	1
Event passed to a process	2
"YOU LOSE" event put in event queue	3
Event queue full	4
~~Duplicate event found~~	~~5~~

The first response indicates that all went well, and there was no process a waiting an event in the process queue. The second response indicates that there was a process waiting in the queue and that it was passed the event. The third response indicates that there was only one free slot in the event queue (an event occupies two words); the intended datum has been replaced by a "you lose" datum (-0) so that the process which ultimately gets the datum will be aware that the event queue was full and that information was lost.

The fourth response indicates that no action was taken because the queue was full. The fifth response is returned only if the action called for a search for duplicate events and a duplicate was found, in which case no further action is taken.

Possible errors resulting from sending an event:

Class	#	Description
9	2	Event channel does not exist

C. Get an event or hang

IP1 C: Capability for event channel (OB.GETEV)

A user requests an event from a channel using the C-list index of the capability of the channel in question. If the event queue is empty, the process must wait ("hang" or "block") until an event arrives before resuming execution. If more than one process is awaiting an event, the first event sent to that channel is passed to the first process while the other process(es) continues to wait. The event is returned to the calling process in X6 and X7. X6 contains the unique name of the process which sent the event while X7 contains the event datum.

Possible error while getting an event:

Class	#	Description
9	2	Event channel does not exist

D. Get an Event or F-return

IP1 C: Capability for event channel (OB.GTEVF)

The user requests an event from a channel using the C-list index of the event channel's capability. If the event queue is empty, an F-return will be initiated in order to permit the process to take alternative action. The event is returned in X6 and X7 as in C. above.

Possible error while getting an event:

	Class	#	Description
430	9	2	Event channel does not exist

500 E. Get an event from one of a list of event channels or hang (F-return) ??

IP1 D: Pointer to list of C-list indices for event channels (OB.SNDEV...)
IP2 D: Number of event channels involved (OB.GETEV or OB.GTEVF...)

The procedure for getting an event from one of a list of event channels is similar to that for getting a single event (see C. above). The channels are interrogated one at a time and if their respective event queue is empty, the user's process will be queued on the process queue of the event channel. If an event subsequently arrives or is discovered on one of the event channels in the list, the process is removed from all the process queues on which it has already been chained and it is passed the event. If no event arrives or is discovered before the last event channel is interrogated, the process must wait ("hang" or "block") until an event arrives on one of the event channel (F-return).

When an event is finally passed in X6 and X7, the index in the user's list of the event channel producing the event is ~~masked into bits 0-18~~ packed as the scale of X6.

Possible errors while getting an event from a list of channels:

	Class	#	Modifier	Description
590,91	9	2		Event channel does not exist
592,93	2	0		Number of channels is ~~negative~~ ≤ 0
594,95	2	1	2	Number of channels exceeds the number of chaining words in the process (only in hang case)
596	2	2		Pointer to list of event channel indices is negative
597,98	2	1	1	Pointer to list + number of channels exceeds FL
599, 599.1	2	1	3	Number of channels exceeds P.PARML-2
599.2 ↓ 599.9	7	2	index	cap type or options bad "too large"

F. Destroy an Event Channel

IP1 C:Capability for event channel (OB.DSTRY)

An event channel can be destroyed. The only parameter required is the C-list index of the event channel which is to be destroyed. If there are any processes waiting on the event channel's process queue, an F-return is initiated leaving the event channel intact.

Possible errors while destroying an event channel:

Class	#	Description
9	2	Event channel does not exist

VII Operations

Operations are ECS objects which direct the transfer of control from the user to the system when the user calls upon the system. They describe the actions to be taken by the system and direct the passing of parameters to the system or between user subprocesses. (See Subprocesses above.) Each operation is composed of an initial order specifying a desired action, some parameter checking information, and when the action is "subprocess call", a class code naming the subprocess to be called. The initial order is followed optionally by a sequence of orders (containing similar information) indicating alternative actions should all the preceding orders result in F-returns. The user may invoke any of the ECS system actions described in this document, or can create his own operations of which the orders may either specify subprocess call or jump actions or actions which are modifications of ECS system actions.

The checking information in each order consists of 1) a parameter specification type for each parameter required in the actual parameter list for the indicated action; 2) words containing the required option bits and type for capability parameters to be supplied by the user; and 3) all fixed parameters, whether capabilities or data. This checking information is used by the system entry/exit routines when constructing the actual parameter list.

The parameter specification types are:

Type	Description
none	when an operation is created, all parameter specifications are initialized to "none", and must be fixed-up using the various actions supplied before the operation may be used.
any capability	a capability is expected from the user, but no type or option checking is to be performed on it. will check option
user-supplied capability	the user must supply a capability whose type and option bits include those set in the operation.
user-supplied datum	the user must supply a 60-bit datum but no checking is performed on it.
fixed capability	both words of a capability are stored in the operation and no corresponding information is taken from the user's input parameter list.
fixed datum	a 60-bit datum word is stored in the operation; it is passed to the actual parameter list unchanged.

There are two actions for creating new operations; the first creates an operation with one order to "call" or "jump-call" a designated subprocess and the second creates an operation of order N by adding one order to "call" or "jump-call" a subprocess to an already existing operation of N-1 orders. All operations constructed by the user specify "subprocess call" actions or are modified versions of already existing actions. Actions are also available for copying an operation and for changing the parameter specifications in an operation.

A. Make a subprocess call or subprocess jump operation

IP1 C: Capability for Allocation block (OB.ALORD)
IP2 D: C-list index for new operation
IP3 D: Type (0=call; nonzero=jump)
IP4 C: Class code of subprocess to be called by the new operation (OB.CALOP)
IP5 D: Number of parameters to be used by the subprocess call

To create a new operation to be used for subprocess call or jump call, the user supplies the index of a capability for the Allocation block which is to fund space in ECS for the new operation.

In addition the user gives 1) the C-list index where the system will place the capability for the new operation, 2) the type subprocess call action (call or jump call) of the new operation, 3) the name (class code) of the subprocess to be called by the operation, and 4) the number of parameter specifications needed for the subprocess call. Upon creation, all of the parameter specifications of the new operation are initialized to "none" and therefore the operation may not yet be invoked (unless it is parameterless).

Possible errors while creating a new operation:

Class	#	Description	
6	0	Allocation block does not exist	
6	1	No ECS available	
~~6~~	~~2~~	~~No money available~~	
2	0	Number of parameter specifications is negative	
7	7	Too many parameters	how many?
2	4	Negative C-list index	
7	6	Order too large	how so?

B. Add an Order to an Operation

```
IP1  C: Capability for Allocation block  (OB.ALORD)
IP2  D: C-list index for new operation (order)
IP3  C: Capability for existing operation  (OB.ADDOR)
~~IP4  D: Type of order  (0=call; nonzero=jump).~~
IP4  C: Class code of subprocess called by the new order  (OB.CALOP)
IP5  D: Number of new parameters being added
```

This action creates a new operation of order N out of an operation of order N-1. The first parameter is the C-list index for the Allocation block which is to fund space in ECS for the new operation; the second parameter is the C-list index where the system will put the capability for the new operation. In the third parameter the user specifies an already existing operation of order N-1, which is copied with the new order appended. The last three parameters describe the new order by indicating whether it is a "call" or "jump call" to a subprocess, the name (class code) of the subprocess to be called, and the number of additional parameters. The parameters of the new order will be initialized to type "none" and must be fixed-up before the new order of the operation is used.

Possible errors while adding an order:

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available
6	2	No money available
2	4	C-list index is negative
2	5	C-list index is too large
7	1	Operation has been deleted (doesn't exist)
4	5	Subprocess does not exist
2	0	Number of parameter specifications is negative
2	1	Number of parameter specifications is too large
7	6	Order too large

C. Copy an operation of order n

IP1 C: Capability for Allocation block (OB.ALORD)
IP2 D: Full C-list index for new operation
IP3 C: Operation to copy (OB.ADDOR)

The user can copy an already existing operation of order n $(n \geq 0)$ by specifying the C-list index of the funding Allocation block, the full C-list index for the desired operation, and the full C-list index of a slot for the capability for the new copy of the operation. This action is used prior to fixing parameter specifications of an operation to avoid changing the original version of the operation.

Possible errors while copying an operation:

Class	#	Description
6	0	Allocation block does not exist
6	1	No ECS available
6	2	No money available
2	4	C-list index is negative
2	5	C-list index exceeds full C-list
7	1	Operation does not exist

D. Change a parameter specification type

In order to specify the parameter specification types in an order of an operation created by either A or B above, a set of actions is provided. Each takes as parameters a C-list index for an operation and a parameter specification index (considering the parameter specification for the first parameter of the first order as having an index of 0). Some require additional information depending on the type of parameter specification being changed.

1. Change parameter specification from "none" to "user-supplied datum"

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification to change

Possible errors:

Class	#	Description
7	1	Operation does not exist
2	2	Index is negative
2	3	Index is too large
7	4	Parameter specification type is not currently "none"

2. Change parameter specification from "none" to "any capability"

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification to change
IP3 D: reqd option bits
Possible errors. See 1 above.

3. Change parameter specification type from "none" to "user-supplied capability"

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification ~~type~~ to change
IP3 D: Capability type, in low 18 bits
IP4 D: Capability option bit mask, in low 42 bits

The type of a capability occupies the lower 18 bits of the Option bit/Type field of which exactly 9 of the 18 bits must be set.* Table 1

---
* This arrangement allows the validity of the entire 60-bit field to be checked in one instruction (using the implication function).

below gives the types for ECS objects currently available.

Table 1. Capability types

Object	Type
Process	777_8
C-list	1377_8
File	1577_8
Operation	1677_8
Class Code	1737_8
Event Channel	1757_8
Allocation Block	1767_8

The option bit mask stored in a capability occupies the upper 42-bits of the Option bit/Type field and the meanings of the various option bits is determined by the type of object the capability identifies. See Appendix B for the name, description and relative position of all option bits. The option bit mask is checked for all required option bits. The positions of the bits are given reading from right to left; thus bit position 0 is the low order bit of the field.

Possible errors while changing parameter specification type from "none" to "user-supplied capability":

Class	#	Modifier	Description
~~6~~	~~0~~		~~No Allocation block~~
6	1		No ECS available
~~6~~	~~2~~		~~No money available~~
2	2	2	Index is negative
2	3	2	Index is too large
7	2		Capability type does not have exactly 9 bits set
7	1		Operation does not exist
7	2		Option bits bad
7	4		Parameter specification is not currently "none"

4. Change a parameter specification type from "user-supplied datum" to "fixed datum"

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification type
IP3 D: 60-bit datum word

Possible errors while changing parameter specification from "user-supplied datum" to "fixed datum":

Class	#	Modifier	Description
~~6~~	~~0~~		~~No Allocation block~~
6	1		No ECS available
~~6~~	~~2~~		~~No money available~~
7	1		Operation does not exist
2	2	2	Index is negative
2	2	3	Index is too large
7	5		Parameter specification is not currently "user-supplied datum"

5. Change a parameter specification type from "user-supplied capability" to "fixed capability"

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification type in operation
IP3 C: A capability

The capability supplied must agree in type and option bits with what is already in the operation.

Possible errors while changing a parameter specification type from "user-supplied capability" to "fixed capability"

Class	#	Modifier	Description
~~6~~	~~0~~		~~No Allocation block~~
6	1		No ECS available
~~6~~	~~2~~		~~No money available~~
7	1		Operation does not exist
2	2	2	Index is negative
2	3	2	Index is too large
7	5		Parameter specification is not currently "user-supplied capability"

Note in the last two cases (4 and 5) that "fixing" a parameter specification type requires two steps, changing the specification first to "user-supplied" type and then to the corresponding "fixed" type.

Actions 3, 4, and 5 involve reallocating the operation in ECS, since each requires inserting one additional word to the order.

E. Change parameter specification option bits for type "user-supplied capability"

IP1 C: Capability for operation (OB.CHOPT)
IP2 D: Index of parameter specification
IP3 D: Option bit mask

After the parameter specification option bit mask has been specified when a parameter specification type is changed from "none" to "user-supplied capability", this action may be used to alter the mask.

Possible errors while changing option mask:

Class	#	Modifier	Description
2	2	2	Index is negative
2	3	2	Index is too large
7	1		Operation does not exist
7	5		Parameter specification type is not currently "user-supplied capability"

F. Destroy an Operation

IP1 C: Capability for the operation to be destroyed (OB.DSTRY)

This action may be used to destroy an operation created by the user. The only parameter required is the C-list index of the capability for the operation to be destroyed.

Possible error when destroying an operation:

Class	#	Description
7	1	Operation does not exist

Appendix A

User supplied parameters (with option bits) for ECS system and subprocess call actions

I Allocation Blocks

A. Create an Allocation Block C.CRALBK

IP1 C: Allocation Block (OB.CREAB)
IP1 D: C-list index for returned capability

B. Transfer funds (and/or space) from one Allocation Block to another C.DONATE

IP1 C: Allocation Block (donor) (OB.GIVE)
IP2 C: Allocation Block (donee) (OB.GET)
IP3 D: Space to be transferred
IP4 D: Money to be transferred

C. Return capability for n-th object in Allocation Block

IP1 C: Allocation Block (OB.GOD)
IP2 D: Full C-list index for returned capability
IP3 D: Index of desired object (n)

D. Destroy Allocation Block C.DELAB

IP1 C: Allocation Block to be destroyed (OB.DSTRY)

II C-List Actions

A. Create a C-list C.CCLIST

IP1 C: Capability for Allocation block (OB.CRECL)
IP2 D: Index in full C-list to return new capability
IP3 D: Length of new C-list

B. Display a capability from full C-list C.DSPCAP

IP1 D: Index in full C-list

C. Display a capability from an arbitrary C-list C.DSPARB

IP1 C: Capability for C-list
IP2 D: Index in the C-list

D. Copy a capability within full C-list and decrease options C.MVECAP

IP1 D: Index of desired capability
IP2 D: Index of destination C-list entry
IP3 D: Mask if options to preserve of

E. Copy capability from full C-list to arbitrary C-list (vice-versa) C.CAPOUT C.CAPIN

IP1 C: Index of destination (source) C-list (OB.CPYIN, OB.CPYOT)
IP2 D: Index within destination (source) C-list of capability
IP3 D: Index in full C-list of source (destination) capability

F. Change unique name in capability C.NEWUN

IP1 D: C-list index of object (OB.CHNAM)

G. Destroy a C-list C.DELCL

IP1 C: Capability for C-list (OB.DSTRY)

III File Actions

A. Create a File C.CFILE

IP1 C: Capability for an Allocation block (OB.CRFIL)
IP2 D: C-list index to return capability
IP3 D: Number of levels in file
IP4 D: Pointer to list of shape numbers

B. Create a Block C.CBLK

IP1 C: Capability for file (OB.CREBL)
IP2 D: Address of block in file

C. Check for missing blocks C.CHKBLK

IP1 C: Capability for file
IP2 D: Address of block in file

D. Read shape of a file C.REDSHP

IP1 C: Capability for file
IP2 D: Address of buffer for shape numbers
IP3 D: Buffer size

E. Read (Write) a File C.RFILE C.WFILE

IP1 C: Capability for file (OB.RDFIL, OB.WFILE)
IP2 D: Address in file
IP3 D: Address in CM
IP4 D: Word count

F. Move a block of a file C.MOVBLK

IP1 C: Capability of a source file (OB.RDFIL, OB.DELBL)
IP2 D: Address of source block
IP3 D: Capability for destination file (OB.WFILE, OB.CREBL)
IP4 D: Address of destination block

G. File to file copy C.DELBL

IP1 C: Source file (OB.RDFIL)
IP2 D: Address in source file
IP3 C: Destination file (OB.WFILE)
IP4 D: Address in destination file
IP5 D: Count of words to be transferred

G Test + reset dirty bit
IP1 C: file
IP2

H. Delete a Block from a File C.DELBLK

IP1 C: Capability for file (OB.DELBL)
IP2 D: Address of block to be deleted

I. Delete a File C.DELFIL

IP1 C: Capability for file (OB.DSTRY)

IV Process and Subprocess

A. Create a class code C.CCC

IP1 C: Capability for class code D

B. Set temporary part of class code C.NWTMP

IP1 C: Capability for class code (OB.TEMP)
IP2 D: C-list index for modified class code
IP3 D: New temporary part (30 bits)

C. Create a Process C.CPROC

IP1 C: Capability for Allocation block (OB.CREPR)
IP2 D: C-list index for returned process capability
IP3 D: Number of event channel chaining words
IP4 D: Number of stack entries
IP5 C: Class code for initial subprocess (OB.SONSP)
IP6 D: Number of map entries in initial subprocess
IP7 D: Compiled map buffer size for initial subprocess
IP8 D: Subprocess field length
IP9 D: Subprocess entry point
IP10 C: Capability of C-list for subprocess (OB.LOCCL)
IP11 C: Capability of file for 1st map entry (Read/Write: OB.WFILE, OB.RDFIL, OB.PLMAP) for initial subprocess
IP12 D: Address within file
IP13 D: Address in CM
IP14 D: Count of words to be swapped
IP15 D: Capability of file for 2nd map entry (Read Only: OB.RDFIL, OB.PLMAP) for initial subprocess

IP16 D; Address within file
IP17 D: Address in CM
IP18 D: Count of words to be swapped

D. Display Fixed Length Process Descriptor

IP1 C: Capability for the process
IP2 D: Address of buffer area
IP3 D: Size of buffer area

E. Display Clock Times

IP1 D: Address of buffer area

F. Create a Subprocess C.CSPROC

IP1 C: Capability for new subprocess class code (OB.SONSP)
IP2 C: Capability for class code of the "father" of subprocess (OB.FATHR)
IP3 D: Number of map entries
IP4 D: Compiled map buffer size
IP5 D: Subprocess field length
IP6 D: Subprocess entry point
IP7 C: Capability for subprocess local C-list index (OB.LOCCL)

G. Display Subprocess Descriptor

IP1 C: Class code (subprocess name)
IP2 D: Address of buffer area
IP3 D: Size of buffer area

H. Subprocess call

See Operations

I. Subprocess return C.RETURN

See Operations this isn't correct, yes?

J. Subprocess F-return C.FRETUR

See Operations

K. Subprocess Jump Return C.JUMP?

IP1 C: Capability for class code of subprocess to return to (OB.SPRET)
IP2 D: Number of stack occurrences of AP1 to skip

L. Return with Error

IP1 D: Error Class
IP2 D: Error Number

M. Return with parameters
IP1 D:
IP2 D:

N M. Modify P-counter of subprocess C.MODPC

IP1 C: Capability for class code of subprocess (OB.PCNT)
IP2 D: Number of stack occurrences of AP1 to skip
IP3 D: New P-counter

O N. Display stack C.DISPST

IP1 D: CM address of a buffer area
IP2 D: Size of buffer area ($\geq$ 4)

P O. Display stack entry C.DISSEN

IP1 D: CM address of buffer
IP2 D: Desired stack entry

P. Send process interrupt C.PINT

IP1 C: Capability for a process (OB.SDINT)
IP2 C: Capability for a class code of a subprocess (OB.INTSP)
IP3 D: An 18 bit interrupt datum

Q. Set/Clear Interrupt Inhibit of Current Subprocess C.SETIIB C.CLRIIB

No parameters

R. Reduce/Restore Path of Current Subprocess

No parameters

S. Set local ESM (Error Selection Mask) C.ESMLOC

IP1 D: Pointer to new ESM

T. Set ESM in any subprocess C.ESMGEN

IP1 D: Pointer to new ESM
IP2 C: Capability for class code (OB.STESM)

U. Destroy a process C.DLPROC

IP1 C: Capability for process to be destroyed (OB.DSTRY)

V. Destroy a subprocess C.DELSUB

IP1 C: Capability for the class code of subprocess to be destroyed (OB.DSTRY)

W. Save (restore) registers C.SAVE C.RESTOR

IP1 D: Pointer to 16 word buffer for registers

V Map Actions

A. Zero a map entry — C.MAPZRO

IP1 C: Class code (subprocess name) (OB.CHMAP)
IP2 D: Index in logical map of subprocess
IP3 C: file currently in map, if any (OB.PLMAP)

B. Change (create) a map entry (read/write or read only) — create {C.MPCHRW = C.MKMPRW, C.MPCHRO = C.MKMPRO}; change {C.CHMPRW, C.CHMPRO}

IP1 C: Class code of subprocess (OB.CHMAP)
IP2 D: Index of map entry in AP1
IP3 C: Associated file (read only or read/write) (OB.PLMAP,OB.RDFIL,OB.WFILE)
IP4 D: Address in file
IP5 D: Address in CM
IP6 D: Word count of new entry
reqd for change – IP7 C: file currently in map, if any (OB.PLMAP)

C. Display map entry from map of named subprocess — C.DISMAP

IP1 C: Class code for subprocess
IP2 D: Index of entry in logical map of AP1
IP3 D: Address of 3 word buffer

D. Display entry in full map — C.DSFMAP

IP1 D: Index of entry in full map
IP2 D: Address of 3 word buffer

E. Set Direct Access Map Entry — C.SETDAE

IP1 C: Class code (OB.DAE)
IP2 C: File (OB.FDAE)
IP3 D: File address of beginning of block
IP4 D: Word count (mod 100_8)

F. Clear Direct Access Map Entry — C.CLRDAE

IP1 C: Class code (OB.CHMAP)

VI Event Channel Actions

A. Create an event channel — C.CEVCH

IP1 C: Capability for allocation block (OB.CREEC)
IP2 D: C-list index for new event channel capability
IP3 D: Length of event queue

B. Send an event ~~(with/without duplicate checking)~~ — C.SENDE

IP1 C: Capability for event channel (OB.SNDEV)
IP2 D: Datum part of event

C. Get an event or hang C.GETE

IP1 C: Capability for event channel (OB.GETEV)

D. Get an event or F-return C.GETEVF

IP1 C: Capability for event channel (OB.GTEVF)

E. Get an event from one of a list of event channels or hang (F-return) C.MGETH C.MGETF

IP1 D: Pointer to list of event channel C-list indices (OB.SNDEV...) (OB.GETEV or GTEVF...)

IP2 D: Number of channels in list

F. Destroy an event channel

IP1 C: Capability for event channel (OB.DSTRY)

VII Operations

A. Make a subprocess call or subprocess jump operation. C.MKOPR

IP1 C: Capability for Allocation block (OB.ALORD)
IP2 D: C-list index to return new operation
IP3 D: Type (0=call, nonzero=jump)
IP4 C: Class code for subprocess called by new operation (OB.CALOP)
IP5 D: Number of parameters used by the subprocess call

B. Add an order to an operation C.ADDORD

IP1 C: Capability for Allocation block (OB.ALORD)
IP2 D: C-list index to return new operation
IP3 C: Capability for existing operation (OB.ADDOR)
IP4 D: Type of order (0=call, nonzero=jump)
IP5 C: Class code of subprocess called by new order (OB.CALOP)
IP6 D: Number of new parameters being added

C. Copy an operation of order n $(n \geq 0)$ C.COPYOP

IP1 C: Capability for Allocation block (OB.ALORD)
IP2 D: Full C-list index for new operation
IP3 C: Operation to copy (OB.ADDOR)

C.UDAT

D1. Change a parameter specification type from "none" to "user-supplied datum"

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification

D2. Change a parameter specification type from "none" to "any capability" C.ACAP

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification type to change
IP3 D: option bits

D3. Change a parameter specification type from "none" to "user-supplied capability" C.UCAP

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification type
IP3 D: Capability type
IP4 D: Capability option bit mask

D4. Change a parameter specification type from "user-supplied datum" to "fixed-datum" C.FIXD

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification type
IP3 D: 60-bit datum word

D5. Change a parameter specification type from "user-supplied capability" to "fixed capability" C.FIXC

IP1 C: Capability for operation (OB.CHTYP)
IP2 D: Index of parameter specification type in operation
IP3 C: A capability

E. Change Parameter Specification Option Bits for "user-supplied capability"

C.ADDOPT ?

IP1 C: Capability of operation (OB.CHOPT)
IP2 D: Index of parameter specification
IP3 D: Option bit mask

F. Destroy an Operation

IP1 C:Capability for operation to be destroyed (OB.DSTRY)

Appendix B

Options

Object	Mnemonic	Description	Relative Bit Position
Allocation Block	OB.DSTRY	Destroy Allocation Block	0
	OB.CHNAM	Change Unique name	1
	OB.CREAB	Create Allocation Block	2
	OB.CRECL	Create a C-list	3
	OB.CRFIL	Create a file	4
	OB.CREPR	Create a process	5
	OB.CRESP	Create a subprocess	6
	OB.CREEC	Create an event channel	7
	OB.ALORD	Create an operation	8
	OB.GIVE	Donor Allocation block	9
	OB.GET	Donee Allocation block	10
	OB.GOD	Return capability of n-th object	11
C-list	OB.DSTRY	Destroy C-list	0
	OB.CHNAM	Change unique name	1
	OB.CPYIN	Copy capability into C-list	2
	OB.CPYOT	Copy capability out of C-list	3
	OB.LOCCL	Local C-list for initial subprocess	4
File	OB.DSTRY	Destroy a file	0
	OB.CHNAM	Change unique name	1
	OB.CREBL	Create a block	2
	OB.DELBL	Delete a block	3
	OB.RDFIL	Read a file	4
	OB.WFILE	Write on the file	5
	OB.PLMPP	Place portion of file in map	6
	OB.FDAE	Direct ECS Access	7
Process	OB.DSTRY	Destroy a process	0
	OB.CHNAM	Change unique name	1
	OB.SDINT	Interrupted process	
Subprocess	OB.DSTRY	Destroy subprocess	0
	OB.TEMP	Set temporary part of class code	1
	OB.FATHR	Father subprocess	2
	OB.SPRET	Subprocess may be jump returned to	3
	OB.PCNT	P-counter of subprocess may be modified	4
	OB.INTSP	Interrupt subprocess	5
	OB.CALOP	Subprocess called by operator	6
	OB.SONSP	Son subprocess	7
	OB.CHMAP	Create, zero, or change map entry	8
	OB.DAE	Direct ECS Access map entry	9
	OB.STESM	Set Error Selection Mask	10

Object	Mnemonic	Description	Relative Bit Position	
Event Channel	OB.DSTRY	Destroy event channel	0	1
	OB.CHNAM	Change unique name	1	2
	OB.SNDEV	Send an event	2	4
	OB.GETEV	Get an event (or hang)	3	10_8
	OB.GTEVF	Get an event (or F-return)	4	20_8
Operation	OB.DSTRY	Destroy an operation	0	
	OB.CHNAM	Change unique name	1	
	OB.ADDOR	Order may be added to operation	2	
	OB.CHTYP	Change parameter specification type in an operation	3	
	OB.CHOPT	Change option bits for "user-supplied capability"	4	

```
OPTIONS     1000          10
ECSACT      1400          1045
JIMGREY     1000          1051
GRAYCDE     7400          1055
TYPES        400          1061
PROCSYM      400          1065
INTSYS      1000          1071
ECSMAC       400          1075
ASCII       1000          1101
OBBITS      1000          1305
BLKBOX      1000          1421
GETEM       1000          1701
MASTR       1000          2051
IOMAC       1000          2665
ALOCSYM      400          3205
LIST COMPLETE
GET,ERRNUMS,,ERRNUMS,XTEXT
OBTAINED
FIN
BEAD HERE
C,EDITOR,S,ERNU←←RNUMS
EDIT
M/E.SBLOCK;P10
* NOT FOUND
M/E.ABLOCK;P10
E.ABLOCK   EQU     6          ALLOCATION BLOCK ERROR CLASS
*
E.NOABLK   EQU     0          NO ALLOCATION BLOCK
E.NOECS    EQU     1          NOT ENOUGH RES. SPACE TO CREATE OBJ.
E.NOFUND   EQU     2 E.NOSLOT NO MORE MONEY
E.NOSWP    EQU      3            NO SWAPPED ECS SPACE
E.NODSK    EQU      4            NO DISK SPACE
E.ABLIM    EQU      5         ATTEMP TO INCREASE CHARGED AB SPACE
*                             BEYOND LIMIT
E.ABPOOR   EQU      6         DONOR AB TOO POOR TO MAKE DONATION
M;P4
*
*
E.OPER     EQU     7          ERROR IN INTERPRETING OPERATION
*
Q
```

E.NORES 5
E.NOCP 6
E.NOMOT 7
E.NORLC 8
E.NOPATH 9
E.CRGER 10

```
OK
PSPACE,20000
OK
C,DISK,S
DISK HERE
GET,OBBITS,,OBBITS,XTEXT
OBTAINED
FIN
BEAD HERE
C,EDITOR,S,OBITS
EDIT
Q
BEAD HERE
C,EDITOR,S,OBBITS
EDIT
P10

          TITLE DEFINE OPTION POSITIONS
          MACRO     OPT,NAME
OB.NAME   EQU       :0:
:0:       SET       2*:0:
          ENDM

OPTORG    MACRO     N
          IFC       NE,/N//
:0:       SET       N

:0:       SET       N
          ELSE
:0:       SET       2
          ENDIF
          ENDM

*
*  ALLOCATION BLOCK
*
OB.DSTRY  SET       1

OB.DSTRY  SET       1
OB.CHNAM  SET       2
OB.CREAB  SET       4
OB.CRECL  SET       10B
OB.CRFIL  SET       20B
OB.CREPR  SET       40B
OB.CRESP  SET       100B
OB.CREEC  SET       200B
OB.ALORD  SET       400B
OB.GIVE   SET       1000B
M;P10
OB.GET    SET       2000B
OB.GOD    SET       4000B
OB.SLIM   SET       10000B    SET LIMIT FIELD
*
*  C-LIST
*
OB.CPYIN  SET       4
OB.CPYOT  SET       10B
OB.LOCCL  SET       20B
*
Q
BEAD HERE
LOGOUT
EMPTY
LOGGED OUT
TRIM
OK
```

Allocation Block Option Bits

1	OB.DSTRY		1
1	OB.CHNAM		2
	OB.CREAB		4
	OB.CRECL		10
	.		20
	.		40
	.		100
	.		200
	OB.ALORD		400
*	OB.GIVE	Reserved space Donor	1000
	OB.GET	" " Donee	2000
1	OB.GOD		4000
*	OB.INCHR	Increment charge field	10000
↓	OB.GIVCP	CP time Donor	20000
	OB.GETCP	" " Donee	40000
	OB.GIVMT	MOT slot Donor	100000
	OB.GETMT	" " Donee	200000
*	OB.INATR (MTR)	Increment DTS integral	400000

OB.SETLM disappears

Appendix C

Error Classes and Numbers

Class	Numbers	Description
0		SCOPE call error class
1		Arith error class
2		Parameter or pointer error class
	0	Parameter too small
	1	Parameter too large
		Param number is masked into errnum
	2	Pointer is negative
	3	Pointer is too large
		Pointer is masked into errnum
	4	C-list index is negative
	5	C-list index is too large
		Index is masked into errnum
3		File-processing error class
	0	File does not exist
	1	Block to be created exists
	2	Block is in map
	3	Block to be moved does not exist
	4	Block sizes not equal for move
	5	Block to be destroyed does not exist
	6	File to be destroyed is nonempty
	7	Negative shape number
	8	Shape number is too large
	9	Shape number is not power of two
	10	File size is too great
4		Error class for subprocess creation, call, and return
	0	Duplicate subp name
	1	Named father does not exist
	2	Block in swapping directive missing
	3	Not enough room for map
	4	Process becomes too big
	5	Named subp does not exist
	6	No room for subp in stack
	7	No room for parameters
	8	Too many capability params
	9	Empty stack (on return)
	10	Empty stack (on F-return)
	11	Attempt to delete subp in stack
	12	attempt to modify own p-counter

Class	Numbers	Description
5		Error class for process creation
	0	Block missing in swapping directive
	1	Not enough room for map
	3	Process gone from MOT
6		Allocation block error class
	0	No Allocation block
	1	Not enough space
	2	No more money
7		Error in interpreting operation
	0	IPO not capability for operation
	1	Operation not in MOT
	2	Capability type or options bad
	3	Param spec (any) encountered
	4	Param spec (any) not encountered
	5	Should be user supplied parameter
	6	Order too big for scratch area
	7	Too many parameters
8		Miscellaneous error class
	0	Capability list not in MOT
	1	Misc object not in MOT
9		Event channel error class
	0	Event queue too short
	1	Event queue too long
	2	Event channel not in MOT
10	0	No subp to take error class
11		Error class for maps
	0	Attempt to change or zero DAE
	1	DAE attempts to bridge blocks
	2	DAE action applied to swapping dir.
	3	Bad word count or missing file

5 Charged space would exceed limit
6 Donor can't cover donation
7 New limit would be less than charged space