To: MSPM Distribution
From: J. F. Ossanna
Subj: BF.2.27
Date: 1/10/68

In addition to minor corrections, the attached revision of BF.2.27 contains the following changes.

1. If an erroneous mode is found, the mode handler now returns without altering bmode.

2. A new call, mode$backup is described.

3. A revised declararion for the mode control structure is given.

4. The assignment of modes to positions in the mode bit string, bmode, has been altered.

5. The status bits returned by the mode handler are specified.

Published: 1/10/68
(Supersedes: BF.2.27, 8/14/67)

Identification

I/O System Mode Handling Discipline for Outer Modules.
J. F. Ossanna and K. L. Thompson.

Purpose

This section describes the standard mode handling discipline for I/O System outer modules. The modes whose handling is described are those which reach an outer module as a character string mode argument in attach, detach, changemode, divert, revert, readsync, writesync, and worksync calls. See Sections BF.1.01-4 for actual definitions of the various modes.

A table-driven mode handler is described which accepts a character string mode argument and produces an updated version of a mode bit string. The driving table, known as the mode control structure, is tailored to the requirements of the particular outer module, and at the same time enforces a standard mode handling behavior.

General

Whenever an outer module receives an outer call with a mode character string, the outer module calls the mode handler to interpret the incoming modes in the proper context. The mode handler returns an updated mode bit string, which can subsequently be interrogated by the outer module to determine the state of any particular mode. Two mode bit strings are defined. The first, "bmode", is pertinent during the lifetime of an attachment, is stored in the outer module's Per-Ioname Base (PIB), and contains the states of all the defined standard I/O System (IOS) modes (except detachment and disposal modes) and any special modes peculiar to the module. The second, "bdisp", is pertinent only during detachment, is not saved, and contains the states of the standard detachment and disposal modes. The same driving table is used by the mode handler for both sets of modes.

The mode handler also returns to the outer module a "pass-on" mode character string suitable for use as the mode argument in the call being passed-on to the next outer module in the iopath. Whether or not an incoming mode is included in the pass-on string by the handler is controlled by the driving table.

The mode arguments of attach, detach, divert, revert, and changemode calls are fed directly to the mode handler. The synchronization modes specified by readsync, writesync, and worksync calls are not; instead the handler is passed a mode formed by concatenation of a call-dependent character ("R", "W", and "K" respectively) with the synchronization mode. For example, the read synchronization mode is passed in the form ("R") concat (rsmode), where rsmode is obtained from the incoming

readsync call.

The mode control structure contains all the information necessary for determining default modes, for ascertaining the applicability of any modes to the particular outer module or to the current mode context, and for enforcing the interdependence between related modes. The interaction between the asynchrony prevention modes and the synchronization modes is automatically handled.

A mode character string consists of mode mnemonics separated by a delimiter. See Section BF.1.02 for a more complete discussion.

Mode Handler Calls

When an outer module has a mode character string to feed to the mode handler, it issues one of the following calls.

```
call mode$bset(mcsp,mode,bmode,passmode,cstatus);

call mode$dset(mcsp,mode,bdisp,passmode,cstatus);

     dcl mcsp ptr,         /*pointer to mode control structure*/
      mode char (*),       /*incoming mode char string*/
      bmode bit (72),      /*mode bit string*/
      bdisp bit (72),      /*detach/disposal bit string*/
      passmode char (*),   /*pass-on modes*/
      cstatus bit (18);    /*call status, see Table 3*/
```

The mode$bset call is used to process a mode that came from an attach, changemode, readsync, writesync, worksync or divert call; the mode$dset call is used to process a mode (disposal) from a detach or revert call. mcsp is a pointer to the base of the driving table containing the mode control structure (MCS). This driving table is one of those managed by the switching complex (see Section BF.2.20); a pointer is automatically provided in the PIB upon each outer call. mcsp is obtained as (pibp->pib.dtabp1). mode is the incoming mode character string being fed to the handler; in the case of the synchronization modes, it is formed by concatenation as explained earlier. bmode is kept in the PIB and is always referenced as (pibp->pib.bmode). bdisp is kept in automatic storage, since it is used only at detachment or reversion time. The internal structure of bmode and bdisp is detailed later below. passmode is a mode character string containing those incoming modes which, according to the MCS, are to be passed on to the next module.

The first call to the mode handler with a particular MCS must be a mode$bset call. This first call results in the setting of all relevant modes; those not mentioned in mode are set to their default values. Similarly, the eventual call to mode$dset returns a bdisp with all modes set.

Subsequent calls to mode$bset can result in mode states being changed provided the modes concerned are intrinsically

changeable. At each such subsequent call a fresh bmode is produced. If an error is detected by the mode handler while handling a particular mode, the call is abandoned, the error is reported in cstatus, and bmode remains unaltered.

Two treatments are provided for mode mnemonics in mode which cannot be found in the MCS. If the not_found flag in the MCS is OFF, the presence of such modes is considered an error. If the not_found flag is ON, such modes are merely added to the right end of the passmode string. The value of the not_found flag is compiled into the MCS at the time of its creation.

The handler scans mode from left to right. The presence of incompatible modes is not detected; the most recent (in terms of the scan) of any conflicting modes determines the final mode states. This behavior can be utilized when concatenating mode strings to control which component can override the other.

For modes which are changeable only at certain times -- e.g. the access mode is unchangeable once actual input/output has begun -- the following call is provided to alter the changeability switch in the MCS.

```
call mode$change_sw(mcsp,mode,change,cstatus);

     dcl mode char (8),  /*mode to be affected*/
      change bit (1);    /*value to be set into change_sw*/
```

The changeability switch in the MCS is described later below.

Propagation of Modes

The propagation of calls (to the next module in an iopath) containing mode arguments is covered in detail in Section BF.1.01-4. The discussion herein is intended only as a supplement. When an attach or detach call is to be passed on, the mode (or disposal) argument is formed by concatenating the passmode string returned by the handler with any modes the module itself wishes to send on. The concatenation order will determine which component can override the other (see earlier discussion). A changemode call is usually passed on only if passmode is not null, or if the module itself determines that some other mode is to be sent on. If both conditions exist, the two components are appropriately concatenated.

Inasmuch as the read and write synchronization modes are of interest only to Device Strategy Modules (DSMs), other modules pass on readsync and writesync calls without calling the mode handler. All modules call the handler after receiving a worksync call, and the call is always passed on. divert and revert calls are normally seen only by DSMs; in those exceptional cases where other modules receive these calls, these calls are passed directly on without calling the handler.

It is possible that a passed-on mode may be found unacceptable to the next outer module. If such an error occurs, the original mode must ordinarily be regarded as in error. The following call is provided to undo the effect of the corresponding call to the mode handler.

```
call mode$backup(mcsp,cstatus);
```

The Mode Bit Strings

The mode bit strings, bmode and bdisp, are each 72 bits long and contain the states of 36 modes. Each mode is represented by a pair of bits; the first bit indicates whether or not the mode is applicable (applicable = "0"b) and the second indicates its value if applicable. The location (index) of the pair of bits representing a particular mode is standardized. This index is contained in the MCS entry for each mode so that special (outer module dependent) modes may be handled.

The assigned locations for various modes in bmode and bdisp are given in Tables 1 and 2 respectively. These tables also give the standard mnemonics corresponding to both values of each mode; the columns labeled "m0" and "m1" give the mnemonics for the "0"b and "1"b mode values respectively. For example, if the value of the sequential mode is "backspaceable", the second pair of bits in bmode would be "01"b; the "0"b indicates the mode is applicable and "1"b indicates that the value if backspaceable".

The bmode string is only interrogated and is never altered by the outer module. Interrogation consists of comparing pairs of bits in bmode with test pairs, and can be accomplished several ways. First, the desired bit pair can be extracted by the "substr" built-in function in PL/I. A better method is to use an auxiliary based structure designed to permit mnemonic interrogation. The declaration for such a structure follows.

```
dcl 1 test based (p),
 2 access bit (2),
 2 seq bit (2),
 2 data bit (2),
 2 logical bit (2),
 2 read bit (2),
 2 rewrite bit (2),
    ...     ;
```

The pointer p is computed from p = addr(pibp->pib.bmode). A test for the backspaceable submode consists of comparing (p->test.seq) with "01"b. A similar structure can be used for testing the bdisp string.

Mode locations in bmode for which no MCS entries exist are filled with "10" by the handler.

The getmode Outer Call

The user of an outer module (which may be another outer module) can make the following outer call to obtain the callee's mode states.

```
call getmode(ioname,bmode,status);

     dcl ioname char (32),  /*ioname associated with callee*/
      bmode bit (72),     /*returned mode states*/
      status bit (144);   /*getmode call status*/
```

The interpretation of the returned mode bit string, the callee's current bmode, is the responsibility of the caller.

The Mode Control Structure

The Mode Control Structure (MCS) is the driving table for the Mode Handler. A specially-tailored version must be produced for each outer module produced. Inasmuch as most of the mode handling is constrained by IOS standardization of mode behavior, MCS production is expected to take the form of editing a standard MCS representation and "compiling" the edited version. The tools for MCS production will be described in an appendix to be added to a later version of this section.

The MCS has the following declaration.

```
dcl 1 mcs based (mcsp),          /*mode control structure*/
 2 size fixed bin,               /*control array size*/
 2 first_disp fixed bin,         /*index of first disposal mode*/
 2 flag,
  3 first_sw bit (1),            /*set to 1 by handler on first call*/
  3 not_found bit (1),           /*0=error, 1=pass on*/
  3 hist bit (1),                /*1=one level history valid*/
  3 dset bit(1),                 /*mode$dset has been called*/
 2 mode (N),                     /*mcsp->mcs.size=N*/
  3 m0 (2) char (8),             /*mnemonics for value=0*/
  3 m1 (2) char (8),             /*mnemonics for value=1*/
  3 bits,
   4 nextx1 bit (18),            /*index in MCS of first related mode*/
   4 nextx2 bit (18),            /*index in MCS of second related mode*/
   4 bindex bit (6),             /*index of mode in bmode or bdisp*/
   4 ref_val bit (1),            /*reference value*/
   4 ref_type bit (1),           /*0=default, 1=required*/
   4 applic bit (1),             /*applicability switch, 0=applicable*/
   4 cur_val bit (1),            /*current value of mode*/
   4 overridden bit (1),         /*mode-overridden switch, 1=overridden*/
   4 old_val bit (1),            /*overridden cur_val*/
   4 change_sw bit (1),          /*changeability sw, 0=changeable*/
   4 hist_applic bit (1),        /*one level history*/
   4 hist_cur_val bit (1),       /*of writeable bits*/
   4 hist_overridden bit (1),    /* " */
```

```
    4 hist_old_val bit (1),     /* " */
    4 hist_change_sw bit (1),   /* " */
    4 pass bit (1),             /* pass-on switch, 1=pass on*/
    4 reset_nx1_appl_0 bit (1), /*reset-next1-applic switch, when val=0*/
    4 reset_nx1_appl_1 bit (1), /*reset-next1-applic switch, when val=1*/
    4 reset_nx2_appl_0 bit (1), /*reset-next2-applic switch, when val=0*/
    4 reset_nx2_appl_1 bit (1), /*reset-next2-applic switch, when val=1*/
    4 set_nx1_val bit (1),      /*if 1, set next1 value*/
    4 set_nx2_val bit (1),      /*if 1, set next2 value*/
    4 next1_val bit (1),        /*next1 value*/
    4 next2_val bit (1),        /*next2 value*/
    4 next1_over_0 bit (1),     /*set-next1-override switch, when val=0*/
    4 next1_over_1 bit (1),     /*set-next1-override switch, when val=1*/
    4 next1_over_val bit (1);   /*next1 override value*/
```

The MCS contains modes for both bmode and bdisp; the latter follow the former and begin at MCS index first_disp. Each "mode" represented in the MCS is binary and can be related to two, one, or no other modes. Thus the MCS represents all modes and mode relationships in the form of a binary mode tree. The mode trees correspond to those given in Section BF.1.02, except in those cases where more than two submodes occur at the same level. Such extrabinary submode levels are expanded into a series of binary levels, with the default submode at the last level. For example, the output code mode has three submodes, straight (STR), edited (EDIT), and normal (N). In the MCS the output code mode is represented by two modes: (1) output, having the values not-straight and straight; and output1, having the values normal and edited. If output is straight, output1 is inapplicable. Thus the two pairs of bits representing the output code mode are "0110"b, "0001"b, "0000"b, and "1010"b for straight, edited, normal, and totally-inapplicable, respectively.

The first_sw is used by the handler to determine whether or not modes not present in mode should be set to their default value. The not_found switch determines the treatment of modes found in mode but not in the MCS in the manner described earlier. Standard IOS modes not relevant to a particular outer module can be omitted from it's MCS, provided the not-found alternatives are appropriate. It should be noted that the not_found switch applies to all modes handled. The inclusion of inapplicable mode (with the applicable switch set to inapplicable) permits individual treatment of such modes. The hist and dset flags together with the five history bits associated with each mode are used in the implementation of the mode$backup call.

Space is provided for two eight-character mnemonics for each mode value. nextx1 and nextx2 are the MCS indices for the first and second related mode respectively; nextx1 is intended to indicate an upper (superior) mode, and nextx2 is intended to indicate a lower (inferior) mode. When only one related mode exists, either index may be used, except that an override can be propagated only to a mode indicated by nextx1. An absence of a related mode is indicated by zero indices. bindex indicates the location of the

mode within bmode or bdisp. The applicability switch, applic, indicates whether or not a mode is currently applicable; applic = 0 indicates the mode is applicable. If ref_type is zero, ref_val contains the default mode value; if ref_type is one, ref_val contains a required value. The current value of the mode is kept in cur_val. The pair of bits returned in bmode or bdisp consists of applic and cur_val. The changeability switch, change_sw, indicates the intrinsic changeability of the mode. If the pass-on switch, pass, is one, the mode is concatenated on the right end of the pass-on string, passmode.

The next set of bits generally indicate the effect of this mode upon any related modes. For example, if cur_val is zero and reset_nxl_appl_0 is one, the mode indicated by nextxl, if any, is to have its applicability switch reset (set to one). If set_nxl_val is one, the mode indicated by nextxl is to have it cur_val set to nextl_val, with all attendant ramifications. If cur_val is zero and nextl_over_0 is one, the mode indicated by nextl is to have its cur_val replaced by nextl_over_val; the old cur_val is saved in the next mode's old_val. Also, if cur_val is zero and nextl_over_0 is zero, any override on the mode indicated by nextl is removed. The overridden switch, if one, indicates that this mode has been overridden by some other mode. While overridden, mode changes are accepted but are kept in old_val rather than cur_val.

Table 1.

Mode bit string assignments and associated mnemonics for bmode.

Index	Mode or submode	m0	m1
1	access	Q	D
		SEQ	RAND
2	seq	F	B
		FOR	BACK
3	data	G	P
		LOG	PHY
4	logical	L	S
		LIN	SECT
5	read	R	¬R
		READ	¬READ
6	rewrite	W	¬W
		WRITE	¬WRITE
7	append	A	¬A
		APP	¬APP
8	input	-	RAW
		-	-
9	input1	-	-
		-	-
10	input2	-	-
		-	-
11	input3	C	-
		CANON	-
12	output	-	STR
		-	-
13	output1	N	EDIT
		NORM	-
14	attach	-	PVT
		-	-
15	attach1	-	NODMP
		-	-
16	rdsync	RA	RS
		-	-
17	wrsync	WA	WS
		-	-
18	wksync	KS	KA
		-	-
19	syncrd	¬YR	YR
		-	-
20	syncwr	¬YW	YW
		-	-
21-24	Unassigned.		
25-36	Outer module special modes.		

"-" = blanks

Table 2.

Mode bit string assignments and associated mnemonics for bdisp.

Index	Mode or submode	m0	m1
1	detach	- -	DEV -
2	detach1	- -	DEV1 -
3	detach2	MIN -	MAX -
4	reserve	R REL	H HOLD
5	load	U UNLOAD	M MOUNT
6	save	S SAVE	A AVAIL
7	restart	- -	RESET -
8-36	unassigned.		

"-" = blanks.

Table 3.

Status bits returned by the Mode Handler.

Bit	Meaning when equal to one
1	invalid mode (bset, dset, and change sw).
2	invalid history (backup).
3	first call not bset.
4-18	unassigned.