To: MSPM Distribution
From: J. F. Ossanna
Subj: BF.2.24
Date: 9/28/67

In addition to minor corrections, the attached revision of BF.2.24 contains the following changes.

1. nelemt is described as a delayed use argument.

2. localattach replaces attach as a queuable call.

3. restart is added as a queuable call.

4. An error in the definition of status-mask match is corrected.

5. The driver's handling of the status change flag (primary status bit 10) is described.

6. Certain details of driver operation have been added.

7. Arguments have been added to the rq$get chain and the driver$detach calls.

8. The status returned by the request queuer and driver is detailed.

Identification

Published: 9/28/67
Supersedes 08/14/67

The Working-Process/Device-Manager-Process Interface.
J. F. Ossanna.

Purpose

This section describes the Request Queuer and Device-Manager-Process Driver. The Queuer is called within a Device Strategy Module (DSM) to queue requests (outer calls) being sent to a Device Manager Process (DMP). The Driver is called by the DMP's Dispatcher to fetch a queued request and issue the corresponding outer call to the first outer module in the DMP's iopath. The Driver also plays various supporting roles in general I/O System operation; a complete understanding of these roles requires an understanding of Sections BF.2.23 and BF.2.25.

General

Typically, an iopath includes a Device Strategy Module (DSM) which calls a Device Control Module (DCM) which calls the GIOC Interface Module (GIM). For reasons detailed in Section BF.1.04, independent and asynchronous operation of a lower portion of this iopath is desirable. Such operation is accomplished by including the independent lower portion in a separate process, known as the Device Manager Process (DMP). Inasmuch as program-device synchronization (i.e. read-ahead and write-behind) is implemented by the DSM, the process boundary must occur effectively inside the DSM at what might be called the synchronization point. The two functions which must straddle the boundary are queueing calls to the DCM and forwarding queued calls to the DCM. The specific implementation consists of incorporating all DSM functions except the call forwarding function into a DSM in the user's working process, and of incorporating the call forwarding function in a module known as the DMP Driver in the DMP.

The DSM's per-ioname segment (IS) is the common data base between the working process and the DMP. Thus all data communication between these two processes can only involve data in the IS.

The queueing function is implemented in a procedure, known as the Request Queuer, which is called by the main part of the DSM whenever a call is to be queued. The queuer calls resemble outer calls; the call names correspond to outer call names and the call arguments include the necessary outer call arguments. Additional arguments are included to control the response signaling from the driver. The queuer returns to the main part of the DSM after queueing a request and signaling the DMP. Any waiting necessary for synchronization purposes is done by the main part of the DSM.

As a result of the signal set by the queuer, the DMP's Dispatcher calls the driver. The driver fetches the next request, reconstitutes the corresponding outer call, and issues the outer call to the first module (usually the DCM) in the DMP portion of the iopath.

The queuer communicates requests to the driver using the auxiliary transaction block chain based in the DSM's per-ioname base (PIB) (see Section BF.2.20). The per-request data is kept in transaction block extensions (TBEs). There is a one-to-one correspondence between these blocks and the call transaction blocks in the DMP corresponding to the forwarded outer calls. The driver updates the request block status using the call block status at every opportunity. Certain status conditions (controllable by the DSM) cause the driver to signal response events to the DSM.

The driver calls described in this section are the following.

driver$init
driver$iocall
driver$quit
driver$restart
driver$hardware
driver$detach

The driver$iocall call is the call used to cause the driver to fetch queued requests. The functions corresponding to the other calls are detailed later in this section.

Request Queuer Calls

The Request Queuer is a subroutine called by the main part of the DSM whenever an outer call is to be passed to the DMP. Calls to the queuer to queue requests have the following general form.

```
call rq$name(pibp, status_mask, comp_event, error_event, tbx,
          ..., cstatus);

        dcl pibp ptr,          /*PIB pointer*/
         status_mask bit (18), /*response control status mask*/
         comp_event bit (70), /*completion event*/
         error_event bit (70), /*error event*/
         tbx bit (18),         /*transaction block index*/
         cstatus bit (18);     /*rq call status*/
```

"name" in rq$name represents the outer call name of a queuable outer call. Not all outer calls can be queued; see discussion later below. The arguments between tbx and cstatus are the corresponding outer call arguments in the regular order, except that the first and last outer call arguments, ioname and status are omitted. For example, the call to queue a read call is as follows.

```
call rq$read(pibp, status_mask, comp_event, error_event, tbx,
             workspace, offset, nelem, nelemt, cstatus);
```

pibp is the pointer to the DSM's PIB. status mask is an 18-bit comparison mask used ultimately by the driver to decide when comp event is to be signaled; the exact mechanism is explained later below. error event is an event signaled by the driver under conditions explained later below. tbx is returned and is the transaction block index of the block allocated by the queuer for this request. cstatus is the status returned for this call itself; these bits are listed in Table 3. All the arguments except tbx and cstatus represent information provided to the request queuer. The outer call arguments are defined and declared in other sections of Section BF.

The call to queue an order call is an exception to the preceding rule. Instead the following call is used.

```
call rq$order(pibp, status_mask, comp_event, error_event, tbx,
              request, argptr1, argptr2, size1, size2, cstatus);

          dcl size1 fixed bin,  /*arg@ment structure sizes in bits*/
           size2 fixed bin;
```

request, argptr1, and argptr2 are the order call arguments. size1 and size2 are the sizes in bits of the argument structures pointed to by argptr1 and argptr2 respectively. The DSM is expected to verify all order calls received by it; either the driving table or code used for this purpose must contain the size values.

The status bit strings returned by the DMP are not stored in the usual locations in the auxiliary transaction blocks (for a reason discussed later below); instead, the status for each request is kept in each block's first transaction block extension. The following call is provided to be used instead of tbm$get chain when chasing down chains which include auxiliary blocks.

```
call   rq$get_chain(pibp, tbindex, type, orig, cnt, listptr,
                    cstatus);
```

Except for pibp, the arguments are identical to those described for tbm$get chain in Section BF.2.20. The queuer calls tbm$get chain and then copies the status bit strings from the corresponding transaction block extensions into the status locations in the array pointed to by listptr before returning to the main part of the DSM.

Other queuer calls are provided to assist the operation of the DSM. The receipt of an iowait call to the DSM may imply alteration of the status mask and completion event for a corresponding queued request. The following call is provided for this purpose.

```
call rq$new_event(pibp, tbx, status_mask, comp_event, cstatus);
```

tbx is the index of the block corresponding to the request whose mask and event are to be replaced. The DSM obtains this index by chasing the down chain based in the call block whose index is provided by the oldstatus argument of the iowait call (see Section BF.2.20).

During error handling and restart operations it may be desirable to reissue a previously queued request. The following call is provided for that purpose.

```
call rq$reissue(pibp, tbx, cstatus);
```

tbx is the index of the auxiliary block corresponding to the request to be reissued. The queuer reuses the same block and TBE and appropriately reinitializes certain data before signaling the DMP. The position of the block in the auxiliary chain is not altered.

Request Queuer Operation

When the queuer receives a call to queue a request for a DMP, it calls the TBM to allocate a new transaction block in the DSM's auxiliary chain. That is, the chain base pointer used is computed from addr(pibp->pib.chain_base.alindex). The queuer does not set any hold bits at this or any other time. It is the DSM's responsibility to set hold bit hold2 and/or arrange for appropriate down-chain-inclusion. Then the queuer allocates one or more transaction block extensions (TBEs), all chained together in the normal manner and based in the new block. The TBE chain based in any block holds the arguments for that request; in certain cases only relative pointers are kept pointing to arguments elsewhere in the DSM's per-ioname segment (IS). In addition, the TBEs contain the status bit string, the status mask, various events, and other data needed by the driver. Once the TBE chain for a request is fully prepared, the TBM is called to store a relative pointer to the first TBE in the block, and the iocall event is signaled.

Since the auxiliary transaction block chain is used by both the queuer and the driver, which are in different processes, certain uses of this chain require it to be locked to prevent simultaneous access. The queuer locks the chain only when accessing TBEs. The TBM lock on the Transaction Block Segment (TBS) suffices during block allocation and modification and during chain chasing. The locking is accomplished by calling the Locker (see Section BY) with the auxiliary chain's lock list, which is located in the ICB.

The manner in which the queuer handles request arguments is influenced by whether an argument is forward-only information or is information to be returned by the callee in the DMP. In the following discussion these two classes of arguments will be referred to as "forward" and "return" arguments respectively. Further, in the case of return arguments the behavior of the queuer depends on the location (segment) of the return argument.

The following discussion treats the handling of the following classes of arguments.

1. Fixed- and variable-length forward arguments.

2. Fixed- and variable-length return arguments located in the DSM's IS.

3. Fixed- and variable-length return arguments not located in the DSM's IS.

4. Two-way arguments (forward and return).

5. Delayed-use arguments (e. g. read/write workspaces).

6. Order call argument structures.

The treatment of forward arguments is as follows.

1. Fixed-length, forward arguments are copied into the first TBE, which is designed to hold same for all queuable requests.

2. Variable-length, forward arguments are copied into an additional TBE allocated expressly for the argument. A relative pointer to this TBE is stored in the first TBE, which contains specific places for these relative pointers. The actual size of the argument is also stored in the first TBE.

The treatment of return arguments which are determined by the queuer to reside in the DSM's IS is as follows.

1. Fixed-length, return arguments in the IS have only a relative pointer to them stored in the first TBE.

2. Variable-length, return arguments in the IS have both a relative pointer to them and their size stored in the first TBE.

This treatment allows the DMP driver to store the returned value into its final IS location immediately upon return from first module in the DMP's iopath. Two constraints should be noted. First, the returned argument cannot be updated subsequent to the original return to the driver. Second, it is the DSM's responsibility to see that the storage in the IS for the argument is not prematurely deallocated.

The treatment of return arguments which are determined by the queuer not to reside in the DSM's IS is as follows.

1. Fixed-length, return arguments not in the DSM's IS are allocated storage in an additional TBE, and the relative pointer

stored in the first TBE is set to point to the freshly-allocated space.

2. Variable-length, return arguments not in the IS are handled as above except that the size is also stored in the first TBE.

Inasmuch as the driver, does not know that the returned value is being stored into a temporary location rather than into the final location, it is the responsibility of the queuer to copy the value from the temporary location into the final location. Under these circumstances, the queuer does not return to the main part of the DSM, but waits for a return response event from the driver.

Two-way arguments are treated exactly like return arguments except for the following: (1) after temporary space (an additional TBE) is allocated by the queuer, the queuer must copy the original value into the temporary space; and (2) the driver must provide this copy in the reconstituted call. At the time of this writing, no two-way arguments occur among any of the queuable calls.

Delayed-use arguments are ones which may be used (read or written) by a callee subsequent to the return of the original call. The only arguments admitted to this class are _workspace_ and _nelemt_ in the _read_, _write_, _readrec_, and _writerec_ calls. In the workspace case, the workspace pointed to is loosely regarded as the argument. When the queuer is called to queue a read/write call, _workspace_ and _nelemt_ _must_ be located in DSM's IS. The queuer will store relative pointers to the workspace and to _nelemt_ in the first TBE. The driver will reconstitute the workspace pointer when passing the outer call to the callee in the DMP. It is the responsibility of the DSM not to cause premature deallocation of _nelemt_ and the workspace.

It should be noted that _nelemt_ is treated as a delayed-use argument only by the DSM, queuer, driver, and modules in the DMP; the normal definition of _nelemt_ is that of an ordinary returned argument.

The _order_ call contains two pointers, _argptr1_ and _argptr2_ which point to a forward data structure and a return data structure respectively. The forward structure is treated like a variable-length forward argument and the return structure like a variable-length return argument. The DSM must provide the structure sizes in bits to the queuer. The queuer and driver internally treat these structures as bit strings. It is presumed that the DSM screens _order_ calls and accepts only those which are relevant; the table or procedure which implements this screening can contain the sizes.

Queuable Outer Calls

Not all outer calls are appropriate calls to send to a DMP via the queuer. For example, calls relating to read and write synchronization are intended for the DSM and need not be queuable. The upstate call need not be queuable because the status available in the DSM's auxiliary (queuer) transaction blocks is as up-to-date as is possible. A complete list of queuable outer calls is given in Table 1, attached to this Section.

Completion Response Control

When the DSM calls the queuer to queue a request, the DSM supplies a status mask, a completion event, and an error event. The mask is used by the driver to determine when the completion event should be signaled. The driver signals the error event instead, if the driver determines that the completion condition can never occur. If the completion event is zero (event not supplied), the mask is used to control triggering of the error event.

When the return to the driver occurs following the forwarding of the outer call, the driver copies the status returned by the callee into the corresponding DSM auxiliary transaction block. Every time the driver receives a return on subsequent calls the driver updates the status of all outstanding calls by copying the status from the blocks in the callee's call transaction block chain into the corresponding blocks in the DSM's auxiliary block chain.

The status mask is an 18-bit string whose bits correspond to the 18 primary bits (1-18) of the returned status bit string. Every time the driver stores a new status bit string, the driver compares the new primary bits against the status mask. The following describes the signaling algorithm. Once either event is signaled, no further events will ordinarily be signaled for that request. The status mask is said to match the status if all the bits equal to one in the mask are also equal to one in the primary status bits (i.e. (mask) or (primary) = (primary)).

1. If the mask does not match the status and status bit 5 is zero and the completion event is nonzero, no signal is set.

2. If the mask does not match and status bit 5 is one (no more status change) and both the completion and error events are nonzero, the error event is signaled.

3. If the mask matches the status and the completion event is nonzero, the completion event is signaled.

4. If the mask matches the status and the completion event is zero and the error event is nonzero, the error event is signaled. The mask can be used to control error signaling when no completion signal is required.

5. Under any other conditions no events are signaled.

The driver keeps an event-signaled flag in the first TBE. This flag is reset by the queuer upon receipt of an rq$new_event or an rq$reissue for a previously-queued request. Table 2 attached to the end of this Section summarizes the primary status bits for convenience. It should be noted that bits 11-13, and 18 will never be set to one by the driver.

The DSM normally specifies a nonzero completion and error event when it intends to subsequently call the wait coordinator to wait on the event(s). See Section BF.2.21 for a general discussion of the behavior of a generic DSM.

Normally the queuer returns to the DSM promptly after queuing the request and signaling the DMP. Upon said return the DSM may or may not choose to wait on events. In an earlier discussion of request queuer operation, a situation was revealed under which the queuer did not promptly return. When confronted with a request having a return argument not located in the DSM's IS, the queuer must itself wait for the returned value. A third event, called the return event, is defined for this case; this event is created only by the queuer. If this event is nonzero at the time the driver gets the original return from the callee, the driver signals the return event. If conditions are also met for signaling either the completion or error event, that event is also signaled.

Request Queuer Data Bases

The request queuer utilizes the name of the "iocall" event, the DMP's process identification, and the auxiliary chain's lock list from the Interprocess Communication Base (ICB); it also uses the auxiliary chain base indices and the allocation area in the DSM's PIB. Except for the foregoing, the queuer uses only per-request data bases. The pointer to the ICB is computed from:

```
icbp = ptr(pibp,ptr(pibp,0)->hdr.relp.icb);
```

Declarations for the ioname segment header (HDR) and the PIB may be found in Section BF.2.20; the declaration for the ICB may be found in Section BF.2.23.

The primary per-request data base is the first TBE allocated for the request. This TBE contains all the items needed for general queuer and driver operation and has items corresponding to all fixed-length, forward request arguments. This TBE also contains the sizes of variable-length arguments and relative pointers, when necessary, pointing to variable-length arguments or to return arguments not located in the DSM's IS, all of which are located in additional TBEs. These additional TBEs are allocated expressly for each such argument.

The declaration for the first (primary) TBE follows.

```
dcl 1 rqtbe based (p),          /*request queuer main TBE*/
 2 chain,                       /*standard TBE chaining*/
  3 next_tbe bit (18),
  3 last_relp bit (18),
 2 call_type fixed bin,         /*call type index*/
 2 status bit (144),            /*queued-call status*/
 2 bits,
  3 status_mask bit (18),       /*response control status mask*/
  3 dmp_tbx bit (18),           /*callee call block index*/
  3 event_sig bit (1),          /*event signaled flag, 1=signaled*/
 2 comp_event bit (70),         /*completion event*/
 2 error_event bit (70),        /*error event*/
 2 return_event bit (70),       /*return event*/
 2 proc_id bit (36),            /*calling process id*/
          /*request argument data*/
 2 (a1,a2) char (32),           /*fixed-length forward items*/
 2 b1 bit (144),                /* " */
 2 (c1,c2,c3,c4,c5,c6) fixed bin (35),  /* ", also variable item sizes*/
 2 relp,
  3 (r1,r2,r3,r4) bit (18);     /*relative pointers to variable items*/
```

Any items in the preceding declaration which have not yet been discussed are discussed later below. The number of each kind of item is determined by the needs of the queuable calls; the addition of new calls may require extension of this primary TBE.

The declaration for the additional TBEs required for variable or return arguments depends on the specific call being queued. For example, the <u>localattach</u> call has a variable-length <u>mode</u> argument; it requires the following extra TBE.

```
dcl 1 rqtbe1 based (p1),        /*localattach TBE*/
 2 chain,
  3 next_tbe bit (18),
  3 last_relp bit (18),
 2 mode1 char (N);              /*N=length (mode)*/
```

The length of <u>mode</u> is stored in (p->rqtbe.c1). <u>type</u> and <u>ioname2</u> are stored in (p->rqtbe.a1) and (p->rqtbe.a2) respectively; ptr$rel(addr(p1->rqtbe1.mode1)) is stored in (p->rqtbe.r1).

As another example, consider the <u>readrec</u> call. N = <u>reccount</u> is stored in (p->rqtbe.c1). The following extra TBE is allocated.

```
dcl 1 rqtbe2 based (p2),        /*readrec TBE*/
 2 chain,
  3 next_tbe bit (18),
  3 last_relp bit (18),
 2 nelem1 (N) fixed bin (35),   /*N=reccount*/
 2 offset (N) fixed bin (35),
 2 relp,
```

```
3 workspacel (N) bit (18); /*workspace relative pointers*/
```

Relative pointers to all four of these variable items are stored in the primary TBE.

The call type indices, mapping details, and extra TBE declarations for all the queueable calls are given in Appendix 1 (to be attached to a later version of this Section).

The Device-Manager-Process Driver

The DMP Driver is called only by the DMP Dispatcher (see Sections BF.2.23 and BF.2.25). Much of the operation of the driver when forwarding calls has already been mentioned in earlier discussion. The following discussion describes the driver operation upon receipt of various calls.

The driver's primary data bases are the per-request TBEs allocated by the queuer. The driver also needs the auxiliary chain base indices from the DSM's PIB. In addition, the driver stores return arguments into the DSM's IS. No other data bases are directly accessed.

The following is a declaration for some arguments used in most calls to the driver.

```
dcl ioname char (32),   /* callee's ioname */
 pibp ptr,             /* ptr to DSM's PIB */
 cstatus bit (18);     /* driver call status */
```

The _ioname_ of the module to be called is created by the dispatcher at attachment time and is not supplied by the user's DSM. _pibp_ is a pointer to the DSM's PIB. _cstatus_ is used primarily to report the status of the iopath to the dispatcher; these status bits are listed in Table 3. The path conditions reported include: (1) internal quit detected; (2) device absent from channel; and (3) iopath detached.

Because the driver is concerned with the DSM's auxiliary blocks in the user's TBS, the TBM must be called to switch to using the user's TBS. This call is made by the Dispatcher prior to calling the driver; the driver will call the TBM to reset the TBS to normal prior to calling the first outer module in the DMP iopath. Upon return, the driver makes no calls to the TBM requiring the user's TBS. Another function performed for the driver by the Dispatcher is the locking and unlocking of the DSM's auxiliary chain.

The following steps summarize the Dispatcher functions performed for the driver.

1. An attempt is made to lock the DSM's auxiliary chain; the Locker is called with the auxiliary chain's lock list (in the ICB) and with an event. If the attempt fails, the Locker returns

having stacked the event in the lock list, and the Dispatcher does not call the driver. Following a subsequent wake-up due to the event, the Dispatcher repeats the lock attempt.

2. After the auxiliary chain is locked, the Dispatcher calls the TBM to switch to using the user's TBS (see Section BF.2.20). An event is provided for the TBM to use in its calls to the Locker.

3. The driver is called.

4. If the driver returns indicating that it could not perform its function because the user's TBS was in use, the Dispatcher arranges to call the driver again later, when the unlocking of the user's TBS causes a wake-up associated with the event provided the TBM.

5. Upon return from the driver, the auxiliary chain is unlocked.

When the request queuer signals the iocall event, the DMP Dispatcher wakes up and calls the driver with the following call.

```
call driver$iocall(ioname,pibp,cstatus);
```

The driver then performs the following functions.

1. Using <u>pibp</u>, the DSM's auxiliary chain base indices are obtained. <u>tbm$get chain</u> is called to chase the auxiliary chain. The oldest queued request which has not yet been forwarded is located and becomes the current request. If the TBM returns indicating that the TBS was locked, the driver returns to the Dispatcher indicating that the call could not be performed for that reason.

2. An outer call corresponding to the current request is reconstituted. In the case of an <u>abort</u> call, bits 127-144 of <u>oldstatus</u> are set equal to (p->rqtbe.bits.dmp_tbx). The TBM is called to switch the TBS back to normal, and the outer call is issued using <u>ioname</u>.

3. Upon return a <u>hold</u> call is issued to the TBM to hold the callee's call transaction block. The corresponding call block index is stored in the first TBE for future correlation. <u>tbm$get chain</u> is called to chase the entire callee call chain and obtain all the block indices and available status.

4. By matching these call block indices against those stored in each first TBE, the auxiliary and corresponding call blocks can be correlated. Any unmatched call block is released by calling <u>release</u>. Any call block yielding a status bit string with bit 5 equal to one is released.

5. The status bit strings of the current and all older matched (to call blocks) requests are updated. Each request's status bit string is stored in the request's primary TBE rather than in the

auxiliary block, to avoid using a possibly locked TBS. In each case the status mask is compared with the status to determine whether any events need to be signaled. During this status updating, the driver compares the old and new status bit strings (ignoring bit 10 and bits 127-144); if the new status differs from the old, status bit 10 is set equal to one, otherwise bit 10 is set to zero. In the case of the current request where status is being stored for the first time, status bit 10 is set to zero. In addition, bits 127-144 of the stored status are set to the transaction block index of the corresponding auxiliary block.

6. If the current return event is nonzero, the return event is signaled.

7. The driver returns to the Dispatcher. Only one request is forwarded at a time; existing additional requests result in subsequent calls to the driver.

When an iopath is to be created in the DMP, the Dispatcher issues the following call.

call driver$init(ioname,pibp,cstatus);

This call is handled identically like <u>driver$iocall</u>, except that only a <u>localattach</u> call can be forwarded. If the current request is not a <u>localattach</u> call, the driver returns to the dispatcher with <u>cstatus</u> indicating that the path is not attached.

When a real or simulated hardware interrupt event occurs for the iopath, the Dispatcher issues the following call.

call driver$hardware(ioname,pibp,cstatus);

This call is handled similarly to <u>driver$iocall</u> except that any current request is ignored and an internally generated <u>upstate</u> call is issued instead. All status updating and event signaling occurs normally.

When an quit event occurs, the Dispatcher issues the following call.

call driver$quit(ioname,pibp,int_quit,cstatus);

The argument <u>int quit</u> is a one-bit string; if one, the iopath is already in internal quit condition. If <u>int quit</u> is zero, the driver handles the call similarly to <u>driver$hardware</u> except that an internally generated <u>abort</u> call is forwarded with its <u>oldstatus</u> argument equal to zero. Status updating and event signaling occur normally except that the driver adds the abort-due-to-quit status bit (bit 15) to the status of freshly-aborted requests. If any unprocessed requests exist in the auxiliary chain, the driver sets status bits 5, 6, 14, and 15 equal to one to simulate an abort due to a quit. If <u>int quit</u> is one, only the latter simulated aborts need be performed; event signaling occurs normally. The driver then returns to the Dispatcher.

When an iopath is to be restarted, the Dispatcher makes the following call.

```
call driver$restart(pibp,reset,cstatus);
```

The argument _reset_ is a one bit string indicating whether or not the iopath is to be reset before restarting. The restart mechanism involves the DSM's observing that some requests have been aborted due to a quit. The DSM usually reissues such requests unless the reset status bit (bit 16) is one. Using _tbm$get chain_, the driver obtains the status bit strings for all the requests in the DSM's auxiliary chain. Any request having status bits 14 and 15 equal to one (aborted due to quit) now have bit 16 (the reset bit) set equal to _reset_. Any nonzero error events for these blocks (with bits 14 and 15 one) are signaled. The driver returns to the Dispatcher.

If the Dispatcher needs to eliminate an iopath, it makes the following call.

```
call driver$detach(ioname,pibp,cstatus);
```

The driver issues a _detach_ call with _ioname1_ = _ioname_ and _ioname2_ equal to a null character string. The _disposal_ argument contains the "MAX" detach propagation mode and other modes necessary to prevent outer modules in the iopath from disturbing the attached device. A successful detachment is reported in _cstatus_.

Table 1.

List of outer calls queuable by Request Queuer.
Return or delayed-use arguments are underlined.

Queuable Outer Calls

localattach(ioname1,type,ioname2,mode,status)

detach(ioname1,ioname2,disposal,status)

restart(ioname,status)

changemode(ioname,mode,status)

getmode(ioname,bmode,status)

worksync(ioname,wkmode,status)

abort(ioname,oldstatus,status)

format(ioname,epl,epw,tsl,tsw,down,indent,status)

tabs(ioname,tmode,hv,ntabs,tablist,status)

order(ioname,request,argptr1,argptr2,status)

getsize(ioname,elsize,status)

setsize(ioname,elsize,status)

read(ioname,workspace,offset,nelem,nelemt,status)

write(ioname,workspace,offset,nelem,nelemt,status)

setdelim(ioname,nbreaks,breaklist,nreads,readlist,status)

getdelim(ioname,nbreaks,breaklist,nreads,readlist,status)

seek(ioname,ptrname1,ptrname2,offset,status)

tell(ioname,ptrname1,ptrname2,offset,status)

readrec(ioname,reccount,workspace,offset,nelem,nelemt,status)

writerec(ioname,reccount,workspace,offset,nelem,nelemt,status)

Table 2.

Summary of primary status bits.

Bit	Meaning when set to value = 1
1	successful logical initiation (see Section BF.1.04).
2	successful logical completion (see Section BF.1.04).
3	successful physical initiation (see Section BF.1.04).
4	successful physical completion (see Section BF.1.04).
5	transaction terminated (no more status change).
6	serious or fatal error (nonzero bits in 19-54).
7	advisory status or nonfatal error (nonzero bits in 55-90).
8	call-oriented status (nonzero bits in 91-108).
9	hardware status (nonzero bits in 109-126).
10	new status bits set (used during status exchange).
11	unassigned.
12	unassigned.
13	unassigned.
14	transaction aborted.
15	abort was due to quit condition.
16	reset condition (transaction not to be restarted).
17	device absent from channel.
18	sync control; return condition (see Section BF.2.02).

Table 3.

Status returned by the Request Queuer for all calls except rq$get chain. The latter call returns the same status as the tbm$get chain call (see Section BF.2.20, Table 1).

Bit	Meaning when set to one
1	fatal TBM error.
2	fatal ICF (Interprocess Communication Facility) error.
3	delayed-use argument not in per-ioname segment.
4	invalid transaction block index (new event and reissue).
5-18	unassigned.

Status returned by the Driver.

Bit	Meaning when set to one
1	fatal TBM error.
2	fatal ICF error.
3	user TBS locked.
4	missing or invalid current request.
5-15	unassigned.
16	path detached condition.
17	device absent from channel.
18	internal quit condition.