TO: MSPM Distribution
FROM: P. G. Neumann
SUBJECT: BF.1.00
DATE: 08/08/68

This document represents a minor rewrite reflecting the recent I/O system redesign. Externally little has changed.

Published: 08/08/68
(Supersedes: BF.1.00, 02/08/68
BF.1.00, 08/14/67)

Identification

Summary of I/O System User Calls
P. G. Neumann

Purpose

Section BF.1 represents the collection of essentially all information which an average user of the I/O system normally needs to know, assuming he has already read Section BF.0 for background. The purpose of Section BF.1.00 is to provide for each I/O-system user call an introductory description of the call and a reference to the subsection(s) of BF.1 in which the call is discussed.

Introduction

The list of user calls described herein is as follows. Each call is followed by the section number in which it appears.

attach	BF.1.01
detach	1.01
changemode	1.01
getmode	1.01
readsync	1.04
writesync	1.04
resetread	1.04
resetwrite	1.04
worksync	1.04
iowait	1.04
abort	1.04
format	
tabs	
order	
getsize	1.05
setsize	1.05
read	1.06
write	1.06
setdelim	1.06
getdelim	1.06
seek	1.06
tell	1.06
readrec	1.06
writerec	1.06
upstate	1.07
divert	
revert	
restartio	

Glossary of Terms

The I/O system (IOS) outer calls summarized in this document provide the interface for all modules normally called by an IOS user. These outer calls are presented in detail in subsequent subsections of Section BF.1. Certain terms used in the overview of IOS outer calls given below are summarized here for the convenience of the reader. The section in which each term is first thoroughly defined is indicated.

ioname (Section BF.1.01)

An ioname is a name used by the I/O switching complex to route calls within the IOS. An ioname is either a device identifier (e.g., tape reel number or typewriter description) or a framename (see frames, below). An ioname is generally the symbolic name of data known to the IOS and accessible to the user by that name.

attachment (Section BF.1.01)

An attachment is the association of one ioname with another ioname; this association is established by an attach call (see below). Each attachment is remembered by the I/O system until detached by a detach call. An attachment may be the association of a framename (see frames, below) with a device or with another frame. Subsequent to an attachment, data may be read or written by issuing a read or write call (see below) with the appropriate ioname.

iopath, attachment graph (Section BF.1.01)

For any ioname, an associated ioname is specified by each attachment. An iopath is a chain of ionames iteratively implied by a given chain of attachments (e.g., 'a' to 'b', 'b' to 'c', etc.) and followed by the GIOC interface module or terminating with the file system interface module. The graph defined by the totality of all given associations is the attachment graph.

element (Section BF.1.05)

An element is a linear array of bits. It is the smallest data entity normally referred to by an I/O outer call. The most frequent element sizes are expected to be 1, 9 and 36 bits.

frame, item (Section BF.1.05)

A data frame (hereinafter called a frame) is an entity of data which is accessible through the I/O system outer calls, and which in particular may be read from or written into as if it were a device. An ioname referring to a frame is a framename. Each frame is a linear array of items. A linear frame is a linear array of items, where each item is an element. A sectional frame is a linear array of items, where each item consists of two

subframes (e.g., a linear frame and a sectional frame). Note that, since a sectional frame may have a sectional subframe which behaves similarly to the original frame, this definition of a sectional frame is recursive, and that any sectional frame may thus have recursively defined substructure. Data is contained in each linear subframe, while each sectional subframe implicitly defines substructure. The linear subframe may thus, for example, serve as label information for the associated sectional subframe (perhaps for use by a data management system), or as independent data.

For the purpose of descriptive simplicity, a distinction is made between direct frames (associated directly with devices or pseudodevices) and indirect frames (associated only indirectly). In general, the effects of attachments and detachments propagate down the iopath for direct frames, and do not propagate for indirect frames. This distinction is discussed in BF.1.01.

reference pointers (BF.1.05)

Associated with various outer calls (read, write, seek and tell discussed below) are several reference pointers. These include the "read", "write", "first", "last" and "bound" pointers. These pointers indicate specific items within the frame in question. The current item with respect to a read or write call on a given framename is that pointed to by the "read" or "write" pointer, respectively. The "read" pointer indicates the next item to be read. The "write" pointer indicates the next item to be written. The "first" pointer always indicates the first item in the frame. The "last" pointer indicates the last item in the frame. The "bound" pointer indicates the item beyond which the given frame may not grow. In a sectional frame, the "read" and "write" pointers are not meaningful. They are functionally replaced by the "current" pointer, which serves essentially as a combined "read" and "write" pointer, and which points to the next sectional subframe to be read or written. There are separate relevant pointers for each level of sectional subframe. In the case of physical input/output, the "currentrec" pointer replaces the "read" and "write" pointers, and points to the next record to be read or written.

delimiters (Section BF.1.06)

There are two kinds of I/O delimiters meaningful to an I/O user on input. These are the break characters and the read delimiters which are established by means of the setdelim call (see BF.1.06). A break character is meaningful only to a character-oriented device, and serves three functions: it delimits physical interrupts, canonicalization and erase-and-kill processing. A break character is an interrupt delimiter in that it is recognized by the GIOC and causes an immediate interrupt. A break character is an erase-and-kill delimiter in that its presence permits erase and kill processing (see BC.2.03) to take place over all characters received since the preceding break

character. A break character is a canonicalization delimiter in that its presence permits canonicalization (see BC.2.02) to take place over all elements received since the preceding canonicalization delimiter. For certain devices (e.g., typewriters), the new-line character is the default break character. In addition, whether established as a break character or not, the new-line character always delimits canonicalization and erase-and-kill processing.

A read delimiter is an element whose presence terminates the data transmission of a read call. Read delimiters are applicable to all read calls, irrespective of the corresponding device(s) and the element size. There is no default read delimiter. That is, in the absence of read delimiters given in the setdelim call, there are none. On output, there are no delimiters meaningful to the IOS.

status (Section BF.1.07)

The status of a given call consists of 72 bits of information which are passed to successive calls further along the iopath and which are then returned back up the iopath. This status information is modified by each module in a given iopath as required. The status includes an 18-bit transaction identifier which is unique among outstanding transactions for a given ioname.

The Outer Calls

A brief description of each call and its arguments follows. The status argument is contained in each call, and is discussed above. All calls are generally applicable to every outer module representing a device or pseudodevice (see BF.1.03), with exceptions due to peculiarities of each module. For example, the tabs and format calls are applicable only where printing is involved. Similarly, calls dealing with reading or writing are not meaningful where devices are write-only or read-only, respectively. It is assumed that the reader is familiar with the glossary of terms given above.

```
call attach(ioname1,type,ioname2,mode,status);
```

The attach call associates the given ioname (ioname1) with a previously defined name or otherwise known device specified by ioname2. This association is meaningful within the framework of the user's process group. The resulting attachment remains in force until removed by a detach call (see below). A type and a mode (see the changemode call below) are associated with the attachment. See BF.1.01.

```
call detach(ioname1,ioname2,disposal,status);
```

The detach call removes for the given ioname(s) an association established by an attach call. The disposal argument indicates how dedicated resources (e.g., tapes and tape drives) are to be treated. See BF.1.01.

```
call changemode(ioname,mode,status);
```

The mode (specified by mode) of an attachment describes certain characteristics related to the attachment (e.g., readable; writable; appendable; random or sequential; if sequential, forward only or backspaceable; physical or logical; if logical, linear or sectional). The changemode call permits mode changes to be invoked for the given ioname(s) which modify the mode of the attachment. See BF.1.01.

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

The getmode call returns a terse encoding (bmode) of the mode of the attachment specified by the given ioname. This call is intended primarily for use by IOS modules. See BF.1.01. (Design is tentative.)

```
call readsync(ioname,rsmode,limit,status);
```

For a given valid ioname (i.e., a name which has previously been properly attached by means of an attach call), the readsync call sets the read synchronization mode (rsmode) of subsequent read calls (see below). This mode is either synchronous or asynchronous. Synchrony implies that control is not returned to

the caller until the read request is either physically initiated or physically completed, depending upon whether the workspace synchronization mode (see the worksync call below) is asynchronous or synchronous, respectively. Asynchrony implies that read-ahead is possible to the extent permitted by the limit argument, which points to the desired maximum number of elements which may be read ahead. The default mode is asynchronous. See BF.1.04.

```
call writesync(ioname,wsmode,limit,status);
```

For a given (valid) ioname, the writesync call sets the write synchronization mode (wsmode) of subsequent write calls (see below). The mode is either synchronous or asynchronous. Synchrony implies that control is not returned to the caller until the write request is either physically initiated or physically completed, depending upon whether the workspace synchronization mode (see worksync) is asynchronous or synchronous, respectively. Asynchrony implies that write-behind is possible to the extent permitted by the limit argument, which points to the desired maximum number of elements which may be written behind. The default mode is asynchronous. See BF.1.04.

```
call resetread(ioname,status);
```

The resetread call is used to delete unused read-ahead data collected by the I/O system as a result of read-ahead associated with the given ioname. See BF.1.04.

```
call resetwrite(ioname,status);
```

The resetwrite call is used to delete unused write-behind data collected by the I/O system as a result of write-behind associated with the given ioname. See BF.1.04.

```
call worksync(ioname,wkmode,status);
```

For a given ioname, the worksync call sets the workspace synchronization mode. The mode (wkmode) is either synchronous or asynchronous. Synchrony implies that control is not returned to the caller until the I/O system no longer requires the user's workspace (see read and write calls below). Asynchrony implies that some kind of initiation of the call has taken place, although the workspace may still be in use. The default mode is synchronous. See BF.1.04.

```
call iowait(ioname,oldstatus,status);
```

For a given ioname whose workspace synchronization mode is asynchronous, the iowait call defers the return of control as if the workspace synchronization mode were synchronous for the most recent read or write call or for a specified previous call. The argument oldstatus is the original status argument returned for the particular previous transaction, and is used to identify that

transaction uniquely. If oldstatus is missing, the most recent transaction is implied. See BF.1.04.

```
call abort(ioname,oldstatus,status);
```

When the workspace synchronization mode is synchronous, the abort call causes all outstanding transactions to be aborted (oldstatus is ignored). When the workspace synchronization mode is asynchronous, transactions are aborted beginning with the one corresponding to oldstatus, which contains the identification of an earlier call. See BF.1.04.

```
call format(ioname,...,,status);
```

This call is included here as a placeholder.

```
call tabs(ioname,...,status);
```

This call is included here as a placeholder.

```
call order(ioname,request,argptr,status);
```

The order call is used to issue a request (request) to outer modules. argptr points to a data structure containing arguments relevant to the particular request. The call is used for communication among I/O system modules. It may also be used to set hardware device modes.

```
call getsize(ioname,elsize,status);
```

The getsize call returns the current element size (elsize) associated with read and write calls for the given ioname. See BF.1.05.

```
call setsize(ioname,elsize,status);
```

The setsize call sets the element size (elsize) for subsequent read and write calls with the given ioname. See BF.1.05.

```
call read(ioname,workspace,offset,nelem,nelemt,status);
```

The read call attempts to read into the specified workspace (starting offset items from the beginning of the workspace) the requested number (nelem) of elements from the frame specified by the given ioname. Reading begins with current item of frame. Thus for a linear frame, reading begins with the element pointed to by the "read" pointer. Reading is normally terminated by the occurrence of a read delimiter or by the reading of nelem elements, whichever comes first. The "read" pointer is moved to correspond to the element after the one last read. For a sectional frame Y, reading begins with the first element (pointed to by the "read" pointer for X) of the current subframe X, where the current subframe is that pointed to by the "current" pointer for the frame Y of which X is a subframe. Reading is normally

terminated by the occurrence of the end of the subframe, by the occurrence of a read delimiter, or by the reading of nelem elements, whichever comes first. The "current" pointer for Y and the "read" pointer for X are moved to correspond to the first element of the next frame X. See BF.1.06.

```
call write(ioname,workspace,offset,nelem,nelemt,status);
```

The write call attempts to write from the specified workspace (starting offset items from the beginning of the workspace) the requested number (nelem) of elements onto the frame specified by the given ioname. The number of elements actually written is returned (nelemt). The behavior of the write call with respect to the "write" pointer is similar to that described above for the read call with respect to the "read" pointer, except that there is no write delimiter. Writing begins with the current item of the frame. Thus for a linear frame, writing begins with the element pointed to by the "write" pointer. Writing is normally terminated by the writing of nelem elements. The "write" pointer is moved to correspond to the element after the last one written. For a sectional frame Y, writing begins with the first element (pointed to by the "write" pointer for X) of the current subframe X, where the current subframe is that pointed to by the "current" pointer for the frame Y of which X is a subframe. Writing is normally terminated by the writing of nelem elements. The "current" pointer for Y and the "write" pointer for X are moved to correspond to the first element of the next frame X. See BF.1.06.

```
call setdelim(ioname,nbreaks,breaklist,nreads,readlist,status);
```

The setdelim call establishes elements which delimit data read by subsequent linear read calls with the given ioname. The argument breaklist points to a list of break characters (containing nbreaks elements), each serving simultaneously as an interrupt, canonicalization and erase-kill delimiters. Break characters are meaningful only on character-oriented devices. The argument readlist points to a list of read delimiters (containing nreads elements). The new delimiters established by this call are in effect until superseded by a subsequent setdelim call. See BF.1.06.

```
call getdelim(ioname,nbreaks,breaklist,nreads,readlist,status);
```

The getdelim call returns to the caller the delimiters established by the most recent setdelim call, with the arguments having precisely the same meaning for both calls. See BF.1.06.

```
call seek(ioname,ptrname1,ptrname2,offset,status);
```

The seek call sets the reference pointer specified by ptrname1 to the value of the pointer specified by ptrname2 plus the value of a signed offset (if offset is present). ptrname1 may be "read", "write", "last" or "bound", or in the case of a sectional

frame, "current", "last" or "bound". ptrname2 may be "read", "write", "first", "last" or "bound", or in the case of a sectional frame, "current", "first", "last" or "bound". For physical I/O (using the readrec and writerec calls), ptrname1 may be "currentrec", "last" or "bound", while ptrname2 may be "currentrec", "first", "last" or "bound". The seek call is used to truncate, e.g., seek(ioname,"last","last",-40), or to set the bound of the frame, e.g., seek(ioname,"bound","last",27), in addition to its more traditional usage involving the "read" and "write" pointers, e.g., seek (ioname,"read","write",-2). The "read" and "write" pointers are also set as a result of read and write calls, respectively (see above). Each reference pointer refers to an item number. Which frame is referred to depends upon the type argument of the attach call. See BF.1.06.

```
call tell(ioname,ptrname1,ptrname2,offset,status);
```

The tell call returns the value of the pointer specified by ptrname1 as an offset (offset) with respect to the given ptrname2. The arguments ptrname1, ptrname2 and offset have the same meaning as in the seek call. As an example, the tell call may be used to obtain the bound of a frame by call tell(ioname,"bound","first",offset). See BF.1.06.

```
call readrec(ioname,reccount,rworkspace,roffset,rnelem,rnelemt,status);
```

The readrec call is intended solely for reading devices concerned with physical records, such as card readers, printers and magnetic tapes. It is accepted by the tape DCM and by the unit record DCMs. It is also accepted by DSMs calling these DCMs when the device attachment mode contains "P" (physical). The argument reccount indicates the number of records which the readrec call represents. The call is similar to the read call, except that rnelem, roffset and rnelemt are arrays of element counts, and rworkspace is an array of pointers to the corresponding workspaces. The significance of each of these arguments corresponds to the counterpart in the read call without the initial 'r'. An item in each array (of size reccount) corresponds to a physical record. See BF.1.06.

```
call writerec(ioname,reccount,rworkspace,roffset,rnelem,rnelemt,status);
```

The writerec call is to the write call as the readrec call is to the read call. The arguments are as in the writerec call. See BF.1.06.

```
call upstate(ioname,status);
```

The upstate call is used for two purposes. First, it provides a way to give control to the I/O system, so that the I/O system can determine up-to-date status and perhaps physically initiate additional work. In the case of an asynchronous workspace mode, the user's status for outstanding transactions will be updated.

Second, the upstate call is used to obtain status for an earlier transaction that suffered a fatal error, when certain Device Interface Modules are in an "error" state. See BF.1.07.

```
call divert(ioname,mode,status);
```

The divert call suspends any current I/O on the attached device specified by ioname and allows immediate initiation of new I/O. The design of this call is tentative. If ioname and newioname are identical, ioname is renamed. The design of this call is tentative.

```
call revert(ioname,mode,status);
```

The revert call reinstates the original attachment suspended by the previous divert call. The design of this call is tentative.

```
call restartio(ioname,status);
```

The restart call is used to restart input-output for the given ioname subsequent to a quit. This call is primarily for the use of the overseer. The design of this call is tentative.

Argument Declarations

The declarations for the arguments of the above calls are tabulated below in the order of their appearance. The arguments for the format and tabs calls are not included, since these calls are only placeholders for ultimately required calls.

```
declare
          ioname1 char(*),     /* maximum=32 */
          type char(*),        /* maximum=32 */
          ioname2 char(*),     /* maximum=32 */
          mode char(*),
          status bit(72),
          disposal char(*),
          ioname char(*),      /* maximum=32 */
          bmode bit(72),
          rsmode char(*),
          limit ptr,
          wsmode char(*),
          wkmode char(*),
          oldstatus bit(72),
          request char(*),
          argptr ptr,
          elsize fixed bin(35),
          workspace ptr,
          offset fixed bin(35),
          nelem fixed bin(35),
          nelemt fixed bin(35),
          nbreaks fixed bin,
          breaklist bit(*),
          nreads fixed bin,
          readlist bit(*),
          ptrname1 char(*),
          ptrname2 char(*),
          reccount fixed bin,
          rworkspace(*) ptr,
          roffset(*) fixed bin(35),
          rnelem(*) fixed bin(35),
          rnelemt(*) fixed bin(35);
```