TO: MSPM Distribution
FROM: D. R. Widrig
SUBJECT: BF.20.10
DATE: 12/01/67

The revisions in this document correlate latest GIM changes. Major changes have been made to the "define$channel" calling sequence and associated interface description. Minor changes have also been made to several GIM data bases, notably the LCT.

Published: 12/01/67
(Supersedes: BF.20.10, 07/19/67)

Identification

GIM - Setup and Housekeeping
D. R. Widrig and S. D. Dunten

Purpose

This section is part 2 of the complete description of the GIM: see BF.20.02.

Initial Device Setup - define$channel

In order to prepare the GIM to use a particular device, the DIM writer makes the following call:

```
call define$channel (device_name, device_index, event_id
     crtn);
```

where the arguments are defined as follows:

```
device_name char (*)          /* index of device in DCT (See
                                 BF.3.10) */
event_id bit (70)             /* event channel ID */
device_index fixed bin(17)    /* user device tag */
crtn bit (36)                 /* standard GIM error return
                                 word */
```

The intent of the define$channel call is to condition the GIM to accept further calls necessary to operate the indicated device in any manner the DIM writer sees fit. Upon receiving the define$channel call from a DIM, the GIM sets up the most important housekeeping mechanism for this device, the Logical Channel Table.

The Logical Channel Table (LCT) is a per-device structured segment containing all information relating to a DIM's lists, hardware channel status, Class Driving Tables allowed, GIOC to be used, etc. It has the following per-device declaration:

```
/* Declarations for the Logical Channel Table */

dcl 1 lct based(p),                  /* logical channel table */
    2 cdt ptr,                       /* ptr to Class Driving Table */
    2 cdtnl(3 /* n_cdt_names
      */ ) char(32)                  /* list of good CDT names */
```

```
    2 giocno fixed bin(17),      /* GIOC identification number */
    2 phychn fixed bin(12),      /*  physical channel number/2 */
    2 conno fixed bin(17),       /* connect channel number */
    2 statmap(4), bit(3),        /* physical stat channel map */
    2 stlpw bit(72),             /* saved LPW for status checks */
    2 dir_chan bit(1),           /* ON if direct channel */
    2 fststat bit(18),           /* offset of first status frame */
    2 laststat bit(18),          /* offset of last status frame */
    2 copid fixed bin(12),       /* Id of latest data move */
    2 copidx fixed bin(12),      /* index of latest data move */
    2 copbtc fixed bin(18),      /* number of bits moved */
    2 spare fixed bin(17),
    2 aux(2),                    /* array of auxiliary list info */
      3 auxid fixed bin(12),     /* id and index for which an */
      3 auxidx fixed bin(12),    /* ..auxiliary list is
                                    maintained */
      3 auxend fixed bin(12),    /* first index not covered by
                                    patch */
      3 auxcpd bit(1),           /* "1"b if auxiliary list is
                                    copied */
      3 late bit(1),             /* on if patch not taken */
    2 nlst fixed bin(12),        /* total number of operation
                                    lists */
    2 lst(10 /* max_lists */ )
      ptr,                       /* list status ptr array */
    2 free area((14336));        /* free storage area */

dcl max_lists fixed bin(17)
    ext static;                  /* total allowable lists per
                                    LCT */
```

The GIM's initial task upon receiving the define$channel call is to establish a Logical Channel Table (LCT) for the indicated device by appending a branch for the LCT in the GIM's directory, ">io". The name of the branch is formed by concatenating the device index, "device_index", (converted to a character string) with a secondary name of ".lct". The device index is found by inspection of the entry in the Device Configuration Table (DCT) corresponding to "device_name". A call to check$device_name will supply the above relationship.

EXAMPLE

Suppose that the first entry in the system-wide Device Configuration Table (DCT) indicated that:

```
dctp  dct.desc(1).device_index = 3308
                    and
dctp  dct.desc(1).dev_nam = "my_device"
```

Then a call to define$channel with device_name = "my_device" would result in the establishing of a segment named "3308.lct" in the directory "root>system_root>io.

The segment so established is pageable and has the READ, WRITE, and APPEND attributes. Errors in establishing the segment include illegal DCT index, "badcall", and segment has already been established, "badseg". Error bits are set in the standard manner as described in BF.20.05, Errors Detected by the GIOC Interface Module. Assuming no errors in establishing the LCT, the GIM continues by initializing device dependent material within the LCT.

Using entries found within the Device Configuration Table (DCT) for this device, the GIM copies the following data into appropriate areas within the LCT:

a) List of allowable CDT names for this device

b) GIOC number of this device

c) GIM channel number for this device. The GIM channel number of a particular device is always half the physical channel number of the list channel for that device. This is because the GIM treats the data and list channel for a device as a single "channel".

d) The symbolic connect channel number to be used when issuing connects to this device.

e) The status channel mapping for this device. Status channel mapping is explained in the later section entitled Generation of DCW's.

After copying in the above data, the GIM indicates that no Class Driving Table (CDT) has been selected yet. The entry "lct.cdt" is set to null indicating no valid CDT exists.

Upon completion of the above tasks, the GIM has initialized the major bookkeeping segment, the LCT, for a particular device attached to the Multics configuration.

Selecting a Class Driving Table - define$class

After a device has been established via the define$channel call, the DIM must further set up the logical channel table (LCT). The DIM makes the following call:

```
call define$class (device_index, class_id, rtnc);
```

where the arguments are declared as follows:

```
device_index fixed bin (17)  /* user device tag */
class_id char (*)            /* primary name of Class
                                Driving Table */
rtnc bit (36)                /* standard GIM error return
                                word */
```

In the same manner outlined in the define$channel call, define$class again seeks out the pointer to the LCT associated ith the device indicated by "device_index". The call to the file system entry "estblseg" should return with an error indicating the LCT segment is already established. (If this is not the case, then the DIM did not precede the call to define$class with the call to set up the LCT, define$channel. The GIM sets the error "lctnf" to indicate that the LCT was not found.)

Having verified the LCT, a check is made to insure that a Class Driving Table (CDT) has not already been assigned via a previous call to define$class. If the CDT pointer, "lct_cdt", is not null, a previous call was successfully made and error "badseq" is set indicating this call is out of sequence. Assuming that the CDT was not previously assigned, define$class proceeds by calling the traffic controller entry "dstm$get_route" (See MSPM BQ.6.07) to get the relationship between the user device tag, "device_index", and the logical channel number, a GIM bookkeeping number. Illegal device indices result in the error "baddev" being set. Both the LCT segment number and "device_index" are stored in the appropriate logical channel slot in the Channel Assignment Table (CAT) for future reference. The logical channel number is stored in the CAT slot for the proper GIOC and physical channel to facilitate the relating of interrupts on a particular channel and GIOC to a certain logical channel.

Having entered the necessary bookwork into the CAT, define$class proceeds by matching the offered CDT name, "class_id", against the list of allowable names contained within the

LCT. If "class_id" does not match any name, the error "cdtvol" is set indicating a CDT violation. Assuming a match, a small amount of LCT housekeeping is performed. Specifically, "lct.fststat", "lct.lststat", and "lct.stlpw" are zeroed. All the pointers to the list area are set to null. Finally, the list area is initialized for subsequent allocations.

A search in the GIM's directory for the indicated CDT is now made. The CDT is established by concatenating "class_id" with a secondary name of ".cdt" and establishing this segment in the GIM hierarchy branch via a call to the file system entry estblseg. A successful establishment of the CDT segment stores the pointer to the segment in "lct_cdt". Failure to establish the segment results in the error "cdtnf" being set indicating the CDT was not found. Note that several define$class calls may be done until a proper CDT is established.

Upon successful completion of the define$class call, the DIM has completed the mandatory sequence of calls necessary to use a particular device. From this point on, the user may make the remaining GIM calls in essentially any order or number.

Defining a List - define$list

Having previously set up a device for use via the define$channel and define$class call, the DIM is now free to create, edit, and delete any reasonable number of I/O requests to the GIM. To assist the DIM writer in his operations, it will be found convenient to group his requests into certain user-defined areas. For instance, the DIM writer might see fit to think of the I/O necessary to turn on a 1050 proceed light as a single logical operation composed of several GIOC instructions. To assist the DIM writer in organizing his concepts, the GIM requires that the writer define lists. These lists may be of any reasonable number and length. Most of the I/O requests to the GIM center around the notion of creation and editing of items within a list. Thus a DIM writer may see fit to define a list which is to contain the necessary items for turning a 1050 proceed light on. Whenever he wishes to turn the light on, he then instructs the GIM to perform the operations within that list. We shall discuss the editing and activation of lists in the following sections; at issue here is the list definition.

The DIM creates a list via the following call:

```
call define$list (id, device_index, lgth, lrtn);
```

where the arguments are defined as follows:

```
device_index fixed bin (17)  /* user device tag */
id bit (24)                  /* list ID returned to user */
lgth fixed bin (12)          /* maximum number of items
                                in list */
lrtn bit (36)                /* standard error return word */
```

Define$list proceeds by verifying the user device tag, "device_index", by calling check$device_index. If "device_index" is valid, check$device_index returns with the logical channel number and a pointer to the proper LCT. An illegal "device_index" will return with error "baddev" set. Similarly, if the LCT has not been completely established via a call to define$class, the error "lctnf" will be set. Assuming no problems, define$list tries to establish a new user list via a call to change$dfl, the main list allocator. Possible errors include only the case of no more list space available for definitions. This causes the setting of error "tmlst". Assuming success in the list definition, the list ID is fabricated. The list ID may be viewed

as a kind of "laundry ticket" which the DIM must keep and present whenever reference to that particular list is made. It is the only key to a given list and, if lost or destroyed, will have the effect of removing a list from the DIM's manipulation. The "ticket" may be restored via a call to "request$lists", BF.20.12.

Note that the list may still be used by the GIM, it simply cannot be altered by the DIM if the DIM destroys the ticket, "id".

The list identification ticket, "id", is formed by concatenating the 12-bit logical channel number with the 12-bit list number. The list number is nothing more than the index of the slot for the indicated list within the LCT.

After generating the list ticket, "id", define$list is finished. The DIM is now free to edit and use the list defined by "id" subject to the limits of the list length and restrictions contained within the Class Driving Table pointed at by the LCT entry, "lct.cdt".

Releasing a List - define$release

Occasionally, a DIM writer may discover that after certain setup stages have been passed, certain lists are no longer needed. When a list is no longer needed, it is to the DIM writer's advantage to be able to tell the GIM to release the list and free up space for other list manipulations. Also, one would like to have a general cleanup mechanism so that the releasing of a device (e.g. when a console user logs out) is accompanied by a general cleanup of the GIM file hierarchy, etc. To release a list or to de-assign a device, the DIM makes the following call to the GIM:

```
call define$release (id, terminate, rrtn);
```

where the arguments are declared as follows:

```
id bit (24)          /* list identification ticket */
terminate bit (1)    /* ON if de-assigning device */
rrtn bit (36)        /* standard error return word */
```

The GIM entry define$release validates the list laundry ticket, "id", by calling check$list. The checking routine verifies the list id and separates out the list number and logical channel number. The logical channel number is used as the index into the Channel Assignment Table (CAT) to get the pointer to the Logical Channel Table (LCT). Possible errors in the check$list call include an illegal ID, "badid", or LCT not found, "lctnf". Upon successful return from check$list, define$release continues.

If the termination switch, "terminate", is ON, the GIM requires all channel activity for this channel to be stopped. A call to lpw$safe will shut down the channel and condition it so that it cannot be accidentally restarted. The technique employed in stopping a channel was discussed in an earlier section, GIOC Channel Activity. Possible errors from lpw$safe will be ignored here although the error bits may be set. These error return bits include illegal GIOC number, "badcall", and GIOC not available, "giocnf". The motivation for ignoring the errors (if any) stems from the supposition that one terminates a device only on close-out or de-assignment and shutdown errors are of no consequence. If the DIM indicated termination, any list id from any previous define$list call is adequate.

If the DIM indicated termination, the bounds of a "release loop" are set to encompass all possible lists contained within the DIM's LCT. If only one list is indicated, the bounds are so adjusted.

The "release loop" is then entered. A list entry is selected. If the list is not defined, (that is, the pointer to the list is null), the "release loop" is skipped and a new list selected. For a defined list, the following action occurs.

For a defined list, the pointer to the DCW space, "lst.dcw", is checked. A null pointer, indicating no DCWs allocated, causes a skip over the DCW releasing mechanism. For non-null pointers, DCW space has been allocated and there exists a possibility that the DCWs are actually being used. A call to lpw$active will confirm or deny the list activity. If the list is active, chaos could result if the DCW area were to be released so the GIM returns an error bit, "lstact", indicating the list is active. If the list is not active, a call is made to mkdcw$free to release DCW areas and any associated data areas associated with transmission to/from the wired-down data area, "data_seg".

The address space existence is verified by checking the address space pointer, "lst.adrlst". If there exists an address space (i.e. "lst.adrlst" is non-null) the space is freed up.

Finally, the List Status Table itself is freed. One pass of the "release loop" is now completed and subsequent passes (if any) will select and release other lists. The released list may later be re-established by a define$list call, but interim editing or manipulation calls will be rejected with the error for a list not defined, "lndef".

After completing the "release loop" the "terminate" switch is again tested. If the switch is ON, the GIM now eliminates the entire Logical Channel Table by a call to the file system entries makeunknown and delentry. After the two file system calls, the DIM user may no longer reference the device until a subsequent call to define$channel is made.