FROM: R. B. Snyder

TO: MTB Distribution

DATE: 03/15/74

SUBJECT: Replacement for the Current TTYDIM/355 Software

A replacement for the current ring-0 ttydim is needed in order to provide more capabilities, to get away from GIOC dependent coding, and to enable a move of the ttydim software out of ring-0. In addition, a replacement for the current DataNet-355 software is needed to move terminal control from the 6180 to the 355 and to get away from GIOC dependencies. The accomplishment of these two goals will also produce better terminal response.

We recognize the importance of the Network Processing Supervisor (NPS) software as a front-end processor not only for GCOS but for Multics as well. To this end, the 6180/355 interface specified in this MTB is exactly the same as the 6000/355 interface used by NPS. The 355 software is included in this specification as possibly interim support of the 6180 capabilites. Final support of these capabilities may be supplied by NPS.

This document is intended to be the basis for discussion at a design review (the date of which will be announced when appropriate) and to provide something which can be expanded into a full design specification. Comments on the material herein are welcome prior to the design review.

Why the Replacement is Being Considered

The current Multics ring-0 ttydim and associated 355 software simulate the GIOC. It is desired to get rid of the simulation of an obsolete piece of hardware since the original reason for the simulation (software compatibility) no longer exists. In addition, this simulation necessitated that terminal control be done in the 6180 with the 355 merely being an unintelligent sender and receiver of characters from terminals. This proved to be too costly in terms of time with the end result being a noticeable slowdown in terminal response when compared with a real GIOC based ttydim on the 645.

Functional Splitup

There are several explicit functions performed currently by the ttydim/355 software. While these functions will still exist in the proposed new software, they will be performed differently. The 355 will still manage control of its communications adaptors, the Low Speed Line Adaptor (LSLA), and High Speed Line Adaptor (HSLA), but in addition, it will also contain the software necessary to drive each of the types of terminals that can be connected to Multics. This is a function which is currently performed in the 6180 by tty_inter and its associated set of terminal control tables, tty_ctl.

Another functional entity is buffer control within the 6180. This includes input buffering of type-ahead data and output buffering of write-behind data. Currently, a fixed size (at bootload time) buffer pool exists out of which are allocated small buffers. These buffers are usually filled with data and the necessary GIOC dcws to transmit the data or receive it into the buffer. While the buffering function will stay in the 6180 and in ring 0, it will be changed to the extent that input and output buffering will be performed using two different strategies. In addition, the buffers will only contain data and not DCWs. Furthermore, the current ttydim strategy of allocating a new small input buffer while the current small input buffer is being filled by a transmitting terminal will be changed as this strategy precludes the use of high speed terminals on Multics since these terminals transmit faster than the ttydim can answer interrupts and allocate new buffers. This type of device buffering will be moved to the 355. Buffering between the 355 and the 6180 will consist of buffers of data with appropriate provisions for either machine being unable to accept a buffer of data at any given time.

A further function to be altered is output data formatting. This is currently handled in ring 0 by tty_write. This will be moved out of ring 0 with only a few minor changes. In a similar vein, the formatting and canonicalization of input data currently

performed by tty_read and tty_con will be moved out of ring 0 and again with only a few minor changes. The nature of these small changes is that of getting rid of DCW's from buffers and addition of non-ring-0 device code translation tables which can be easily changed by the individual user process.

Finally, the large block of code in both tty_read and tty_write that is concerned with the ARDS graphics capabilities will be totally removed from the main user ring ttydim and will be replaced by a front end module which can be spliced into the i/o streams to provide graphics capabilities not just for the ARDS but a variety of similar graphics devices.

Modifications to the 6180 ttydim

As indicated above, the 6180 ttydim will no longer concern itself with terminal control, but will instead have responsibility only for the conversion of data from a user's process into device specific code and the inverse conversion of data received from the various terminals. Users of the ttydim will use it in the same general way as they do now. The exceptions to this statement are that the ttydim will no longer be in the supervisor ring, graphics will no longer be incorporated into the main ttydim flow, and various new features will be added.

Graphics support as it currently exists really is a large block of code in the ring 0 ttydim which concerns itself with mapping the Multics Standard Graphics Code into the appropriate ARDS graphics characters on output and the reverse mapping on input. Since it can be expected that more and more graphics devices will be used on Multics, this approach is clearly unacceptable. Furthermore, since cursor control on a graphics device involves the transmission of data to that device (as opposed to data set control functions), the proper place to put graphics support is in the 6180 user rings as a "front end" to the standard ttydim rather than in the 355. Therefore, a module will be constructed to do the graphics mappings and this module will use the raw mode facilities of the ttydim to ship its data to the individual graphics devices. It is anticipated that this graphics module will be a temporary solution and that a graphics environment currently being implemented using the present ttydim will be a more permanent solution. This environment provides a partially table driven facility but more importantly it provides a means of extending graphics support to new terminal types in a defined rational way without requiring hardcore/355 changes.

As was mentioned above, the ttydim will contain some new features not currently provided by the present software. These features are briefly described below:

1) An order call will be provided to allow a user to change his current erase, escape, and kill characters.

2) An order call will be provided to allow a user to change his break character (currently the newline - 012). Note that a break character is defined to be a character that will send a wakeup to a process that is blocked on input, will limit the effect of kill and canonicalization, and will limit a read. The user will be allowed to specify a new break character or a list of break characters. Users may also specify that the input line should be terminated one character after the specified break character (this is for devices which send an end of data character like EOT or ETX followed immediately by a longitudinal parity check character). Finally, users will also be allowed to specify breaking on every character (for full duplex echoing from his process) or no break at all (for devices which are not really typewriters but perhaps small computers which plan to send data in 8 bit bytes).

3) The current "set_table" order call will be expanded to allow users to really supply their own translation tables in the user ring. These translation tables will have approximately the same format as they do now but special attention will be paid to try to make them such that any output from Multics which is played back as input will translate correctly. This means that line control characters (EOA, STX, etc.) and tab or carriage return delay characters will be properly stripped off on input.

4) Users will be allowed to change their terminal type with an order call (or in the login line as described later). Before going into the meaning of this order call, it must be pointed out that there are two notions of "terminal type". From the point of view of thf 355, terminals are divided not into types but into "device classes" based upon the fact that special control functions are required for that device class. Hence, all slow speed ASCII terminals (Terminet 300, TTY37, TTY38, TTY35) currently supported by Multics are in one device class. IBM 2741 like devices are another device class. From the Multics view however, terminals are divided into types based upon different criteria. Trendatas, Dura 1021's, IBM 2741s can all be considered to be different types. The various terminal types will be installation definable. How the installation defines its types is described later but it is this terminal type rather than the device class recognized by the 355 that the user will specify in an order call.

5) An order call will be provided to allow a user to switch his terminal into what might be called "no-control mode".

This mode is provided for the user who wants the 355 to do no line or device control whatsoever. For example, assume that a user has dialed into a 133 baud port on a minicomputer whose line control procedures are different enough from a 2741 that the standard line control procedures employed by the 355 cause the device to behave incorrectly. The user could then enter this no-control mode to cause the 355 to stop any control of his terminal. The 355 will act only as a transmitter and receiver of characters. It will be up to the user's process to do device control by sending special commands to the 355 in the normal 9-bit byte data stream. The data stream will be interpreted in the 355 as control groups. The first byte of the control group will be a command indicating whether a dataset or line control function is to be performed or a character string is to be sent. If a character string is to be sent, then following this control byte will be a byte containing the length of the string to be sent followed by the string itself. If a control function (send break, drop carrier, raise supervisory transmit, etc.) is to be performed, it will be specified in the byte following the control byte.

6) The user will be allowed to change his carriage return and tab delay formulae (the coefficients for the formulae are currently kept on a device type/baud rate basis).

7) Another order call will allow users to entfr full duplex mode wherein characters typed on the user's terminal will be echoed by the 355 rather than by local echoing at the terminal. Of course, this mode will only be honored for those devices possessing full duplex capabilities. The following full duplex conventions will be followed in this mode:

a. We will echo spaces for received tabs if the terminal does not have hardware tabs and has the tab echo mode set.

b. If we reach a limit stop on the number of characters in the output echo buffer, we will stop accepting input characters.

c. When the 6180 tells the 355 to start outputting, it will also send a flag indicating that there is/isnot more output waiting. This will allow the 355 to figure out when to echo type ahead if output is currently going on. That is, we will echo input only when not busy and when the flag said there would be no more output coming from the 6180. If more output does come, we will deplete the output echo buffer first before transmitting the new output.

8) Users will be allowed to place their terminals into "polite mode". This means that output to be sent to a terminal will not be transmitted by the 355 until the carriage is at the left margin. If a user in polite mode desires to send a message to his terminal regardless of carriage position, he must bracket the write call by order calls to reset and then set polite mode. If a user is in polite mode, the 355 will implement a timeout feature to guard against the case where a user types 3 characters and then goes to lunch thereby preventing any output from being printed on the terminal.

9) An order call will allow the user to "sense carriage position and lock keyboard". This order call will request the 355 to return to the 6180 any input from the terminal even if the break character has not been typed and lock the keyboard if possible. This order call allows the user to gather up any input, lock the keyboard, interrupt the input with an output message, and then "replay" the input on the console so the typist whose inputting activity was interrupted will be able to continue typing his input line. (Note that this order call is not necessary in full duplex.) Sense carriage position may also be use to compute how much room is left on the carriage for the first line of output (i.e., length of first line of output = line length minus number of characters input).

10) In addition to the current rather non-standard mode "crecho", a new mode "lfecho" will be supplied to cause the 355 to echo a line feed character upon receipt of a carriage return.

11) A mode call will be supplied (referenced in 7 above) to allow users with terminals that have no hardware tab capability to cause the 355 to echo spaces upon receipt of an input tab character when in full duplex mode.

12) A mode call will be supplied to allow users to specify that keyboard locking is to take place prior to sending output. The default for this mode will be to not do the locking for ASCII type terminals and to do the locking for IBM type terminals. This will allow users of ASCII type terminals which do have a locking feature to turn on this facility.

The Ring-0 ttydim

The above section title is really a misnomer in that ring-0 will not contain any code having to do with managing typewriters. Ring-0 support for the user ring ttydim consists mainly of a buffer manager and an interface routine for communication with

the 355. These two facets will be discussed below.

Buffer Management

The proposed buffering strategy is being offered as a replacement for the present scheme primarily to solve the current input buffering restrictions on high speed terminals. The current strategy of course is to only allocate one small (16 word) buffer for input and to try to allocate another when the current buffer is almost full. Since high speed terminals can transmit faster than a new buffer can be allocated, this strategy fails. Output buffering will remain essentially the same as it is right now. There will be one system wide wired buffer area which will be used for output buffering only. This buffer area will be broken up into small buffers containing data, the address of the next small buffer, and a length in characters of the data. The 6180 ttydim will accept a string of output from a user program and copy a portion of it into a buffer chain. A dcw will be constructed for each buffer in the chain. Each dcw will have the address of the buffer and the length of the buffer in characters. This dcw list will be created in a buffer which will be sent immediately to the 355. As much as possible of the remainder of the output string will be copied into another inactive chain whose starting address will be saved. The 355 will read the entire short chain so that the 6180 may free this chain as soon as the 355 has read it. The 355 will proceed to transmit this small chain until there remain just a few characters left to send. At this point, it will request more output from the 6180. The 6180 will strip off another small chain off its inactive chain and using the chain words at the end of each buffer, another dcw list will be constructed as described above. This dcw list will then be passed to the 355.

Input buffering will use a totally different approach. The 355 will want to write input data into the 6180 as soon as possible since its buffering capabilities are limited to its physical memory size of 32K 18 bit words. Consequently, a fairly small wired buffer will exist in the 6180 for use by the 355 as an area in which it may deposit data. The 355 will write strings of data into this system wide buffer in a circular fashion (i.e., wrapping back to the start of the buffer when the end is encountered). The use of this buffer will be as follows. The 355 will inform the 6180 when it has some input data to write into the 6180. The 6180 will select the next free word in the circular buffer and will pass the address of this word to the 355. In the case where the 355 message cannot fit in the area between the free word and the end of the buffer, the 6180 will pass two addresses to the 355. The first will be the address of the free word as described above, and the second will be the address of the start of the buffer so that the message may "wrap around". After the 355 gets the address(es), it will write its message

into the circular buffer and interrupt the 6180.

The handler for this interrupt will be specified as being able to take page faults. This is important since its job will be to remove messages from the small wired buffer and copy them into a paged buffer where they will wait until users call in for them. The interrupt handler will be masked against further 355 interrupts. It will copy the message into the paged buffer. The offset of the message in the paged buffer will be kept in a per tty channel database so that it may be found when the process associated with the tty channel calls into ring-0 for its data. (This database will also contain things necessary to ring 0 such as the event channel over which to signal hangups and dialups to the answering service). If there is already a message address in the per tty channel database, the new message will be threaded onto the back of the previous message(s).

Interface to the 355

As was mentioned above, ring 0 support for the user ring ttydim consists of buffer management and an interface to the 355. The interface will operate in a fashion quite similar to the current program dn355. Each 355 attacied to Multics will have an area in the 6180 memory called a mailbox with which it communicates to the 6180. This mailbox is divided into a header area of software defined length followed by 16 submailboxes of software defined length. Each submailbox is used to communicate information between the two machines. When the 6180 desires to send some type of command to the 355, it selects a submailbox, places the command and any associated data in the submailbox, and then tells the 355 to read that submailbox by interrupting the 355 on one of 16 interrupt levels (the one corresponding to the submailbox number). Similarly, when the 355 wants to send some information to the 6180, it requests the 6180 to select one of the 16 submailboxes. The 6180 will do so and then place a command in that submailbox which will indicate to the 355 that it may use this submailbox. Then the 6180 sends an interrupt whose level number corresponds to the number of the selected submailbox as described above. The 355 may then fill that submailbox with the information it wanted to send to the 6180 and write the submailbox back into the 6180. A detailed interface specification is provided in Appendix A for the interested reader.

The interface module is called from two sources. The user ring ttydim calls it (through a gate) to pass several types of order and mode calls on to the 355 for implementation. The buffer management programs call it to cause output data to be sent to the 355 for transmission. Of course, the 355 calls it indirectly by generating interrupts to cause it to look at a submailbox for the data passed with the "call".

355 Software Specifications

This section of the proposal describes the functions to be performed by the 355 software. No detail is provided here for the moment since most changes in design decisions will be made to the 6180 side of the ttydim. This information is really supplied here for completeness and will not be expanded until this document is expanded into detailed design information.

The software is functionally divided into four groups. There are those modules which deal with the 355 hardware (HSLA, LSLA, etc), the modules responsible for supervisory control of the 355 itself, the modules which perform actual terminal control, and those modules which perform utility functions.

There will be three hardware managers, one for the HSLAs, one for the LSLAs, and one for the Direct Interface Adaptor (DIA). The HSLA and LSLA managers will be responsible for assembling messages to be passed to the terminal control modules and for transmitting messages from the terminal control modules. The DIA module will be responsible for supporting the 355 side of the 355/6180 interface as described above.

The terminal control modules will consist of a macro-built table for device control state/transition data quite similar in nature to the tty_ctl module in the present ttydim, and a program to "execute" (interpret) these tables. This is the function performed by tty_inter in the present ttydim. The 355 device control tables will contain control information for seven device classes. These classes are:

ASCII	TTY37, TTY38, Terminet 300, CDI
1050	IBM 1050
2741	IBM 2741, Trendata, Novar
NO-CONTROL	as discussed previously
ARDS	ARDS, Tektronix
SYNC	Mohawk 2400
BISYNC	Various IBM devices

The table interpreting program will also contain the algorithms for determining the device speed of terminals dialing up on asynchronous HSLA lines at up to 600 baud and for the dynamic configuring of these HSLA subchannels to match the baud rate of the incoming terminal.

The supervisory control modules are responsible for scheduling program executions, dispatching interrupts, buffer management, and the scheduling of delayed (timer initiated) program executions. At this point, it is currently undecided whether or not to use a software stack (there are only 3 index registers and one would have to be used for the stack leaving only two), but if

a stack is used, these modules will also include the stack management routines for call/save/return and stack allocation.

The utility modules will take care of post-bootload initialization, use of the attached 355 control console, dumping the 355, printer tracing of system activity, etc.

355 Operational Aspects

The 355 is currently bootloaded by a BOS program prior to every Multics bootload. This creates several problems. Bootloading the 355 with its subsequent initialization causes lines to be hung up. Everytime the 355 software changes, a new BOS tape must be created. These changes really should go on a Multics tape. Consequently, the 355 will be bootloaded from Multics during a regular bootload. However, before doing this, it will be interrogated to see if it is up, and if so, it will not be rebootloaded but will instead just purge any software queues and stand ready to run. (Of course it will always be possible to force a 355 bootload even though the 355 claims it is up.) The 355 will also be informed when Multics crashes. This will be done to prevent it from trying to talk to Multics after it has crashed. It will also be possible for the Multics operator to load into the 355 a precanned message, in all the popular device codes, which the 355 will send to any terminal dialing up. The message would be enough to inform the user that Multics is currently down. The operator could change this message at any time to keep users informed on the progress of the crash recovery (e.g., "Multics down as of ttt", "Salvaging started", "Salvaging done", etc).

Another operational issue is 355 crashes. During an initial debugging period of course, it will be desirable to crash Multics when the 355 goes down to obtain the necessary dumps. After this period, all users dialed in through a 355 when it crashes will be logged out, the 355 will be rebooted, and dialups will be allowed. This is especially desirable when there are multiple 355s. If the initializer happens to be one of the consoles dialed up through that 355, the message coordinator will leave all messages in a file until the initializer reconnects. It will also be arranged to read the contents of the 355 before rebooting it so that a dump can be printed for analysis without crashing the entire system.

General Issues

A general issue not discussed yet is the foreign terminal problem. Since Multics will only be able to recognize a limited number of terminal types, it is desirable for the user to be able to specify as soon as possible after dialing up that he has a

particular terminal type. Now if the terminal is so different that he cannot even go through the standard handshaking procedure with Multics, he is out of luck. But if he can manage to get the answering service to issue the login prompting lines, a new login line argument will be provided to allow him to specify his terminal type (e.g., login Snyder -tt S2741).

Another issue is the problem of the initializer wanting to type messages on a user's console. The user may have switched translation tables, perhaps after changing his typing element. When the initializer desires to type a message on the user's console, the message will be garbage since the initializer cannot know about the translation table change. To solve this problem, we will use the proposed new send_message facility with a special entry to allow the initializer to send an ips wakeup to cause the message to be printed on the user's console even if he is not blocked. If the user did not have the send_message segment, the initializer would be forced to send a message to the terminal using a default translation table type.

Another issue alluded to above was the specification of terminal types by the answering service. Currently there is code in the answering service to declare a terminal to be a particular type based upon its answerback coding (which will be read by the answering service rather than by ring-0 as with the current ttydim). This makes it difficult to add new terminal types for the local installation at MIT and it will be impossible for most foreign sites. To solve this problem and to allow easy addition of new terminal types, the terminal identification procedure and specification of the various characteristics for each terminal type will be tabularized within the answering service. There will be two tables. The first table will specify the way in which the initial terminal type is to be set up. The other table will specify the device characteristics for each terminal type. These tables can be replaced while the system is running. Shown below are examples of the two tables as specified in a pl1-like format. These examples are only intended to give the flavor of the concept rather than specify any final form.

Device Identification Table

```
type: 2741; speed: 134;
   code: EBCDIC;
   idcode: if id1 = digit then subtype = IBM;
           else subtype = DATEL;

type: TN300; speed: 110,150,300,1200;
   code: ASCII;
```

```
idcode:  if id1 = E then ok;
         else go to try_execuport;
set_tabs:  yes;
```

Device Characteristics Table

```
type:  IBM;
   code:  EBCDIC;
   missing:  poff,pon,rrs,brs;
   delay:  tab,5n/2+1;
   delay:  cr,10n/12+3;
   break:  EOT,CR;
   erase:  #;
   kill:  @;
   modes:  erkl,can,^red,^edited,ll132;
```

A final issue is the switchover to the new ttydim. Rather than have a joint hardcore/user ring installation with a flag day for those users calling the hcs_ entries directly, it is proposed to have users start calling a new user ring program which will merely be a transfer vector to the hcs_ entries. This program will eventually become the user ring ttydim. This will alleviate any problems in installing the new ttydim and then possibly having to uninstall it due to bugs of some type.

Implementation Plan

Since there is quite a bit of work implied in the preceeding description, the implementation of the new software will be split up into two phases. The first phase will consist of those changes to the 355 software necessary to move terminal control to the 355 as described above. It will not however, consist of the implementation of any of the new features (e.g., variable break characters) mentioned earlier. The only changes to be made to the 6180 ttydim in this first phase will be those necessary to implement the 6180/355 interface as described in Appendix A. The ttydim will still remain in ring-0 and no new features will be implemented. It is expected that this first phase will require about 7 months effort. The second phase of the implementation will complete the work as described in the MTB. A Multics Task Report will be generated following the design review in which the work to be done in each of the two phases will be specified in detail along with initiation and completion dates.

Appendix A - Detailed 355/6180 Interface Specifications

All communication between the 355 and 6180 software will be done by means of the mailbox region in the 6180 memory. This mailbox region shall be located in the bootload memory and will begin on a modulo 64 address. The starting address is specified in the 355 by switches. The mailbox region consists of an 8 word mailbox header followed by 16 8 word submailboxes. These submailboxes shall be used on an as-needed basis as areas in which to transfer control information. Refer to Figure 1 for a pll description of the mailbox region.

```
dcl 1 datanet_mbx aligned based (mbxp),          /* declaration of 355 mailbox */
    2 dia_pcw aligned,                           /* Peripheral Control Word for DIA */
      3 zero bit (18) unaligned,
      3 error bit (1) unaligned,                 /* set to "1"b if error on connect */
      3 unused1 bit (5) unaligned,
      3 mbx_no bit (6) unaligned,                /* number of submbx being sent to 355 */
      3 command bit (6) unaligned,               /* always 71 (octal) */
    2 mailbox_requests fixed bin,                /* 0 mod 256K cnt of mbx requests by 355 */
    2 term_inpt_mpx_wd bit (36) aligned,         /* terminate interrupt multiplex word */
    2 last_mbx_req_count fixed bin,              /* previous value of mailbox_requests */
    2 lock bit (36) aligned,                     /* mailbox lock */
    2 unused2 (3) fixed bin,                     /* unused area in header */
    2 sub_mbxes (0:15) aligned;                  /* 16 submailboxes */

dcl 1 sub_mbx aligned based (subp),              /* declaration of a submailbox */
    2 dn355_no bit (3) unaligned,                /* 355 number */
    2 line_number bit (15) unaligned,            /* line number assigned by 355 */
    2 terminal_id bit (18) unaligned,            /* not used */
    2 terminal_type bit (9) unaligned,           /* unused */
    2 cmd_data_len fixed bin (8) unaligned,      /* no. of 6 bit chars in command data */
    2 op_code fixed bin (8) unaligned,           /* op code */
    2 io_cmd fixed bin (8) unaligned,            /* i/o cmd */
    2 command_data (3) bit (36) unaligned,       /* data associated with op code */
    2 rel_data_addr fixed bin (17) unaligned,    /* data address */
    2 unused1 bit (9) unaligned,                 /* unused */
    2 word_cnt fixed bin (8) unaligned,          /* data length */
    2 unused2 bit (54) unaligned,                /* unused */
    2 checksum bit (18) unaligned;               /* checksum of 30 previous 9 bit chars */

dcl  subp pointer;
dcl  mbxp pointer;
```

Figure 1
Page no. 14

Mailbox Header

Word 0 contains a PCW for the Direct Interface Adaptor. It is used to interrupt the 355 on one of 16 interrupt levels (to correspond to one of 16 submailboxes that the 6180 wants the 355 to read).

Word 1 contains a modulo 256K counter of the number of 355 "mailbox requests" made since the system came up. A mailbox request is when the 355 wants to communicate with the 6180. The 6180 responds to a mailbox request by selecting a submailbox for the 355 to write into and interrupts the 355 on the interrupt level corresponding to the submailbox number (0-15). The mailbox requests counter is maintained in the 355. It is used in the following way. Assume the current value of the counter is N. If the 355 has X requests to make to the 6180, it will write the value N+X into word 1 of the mailbox header and then update its own counter to N+X. The 6180 saves each new value of word 1 in word 3. So by subtracting the value in word 3 from word 1 the number of outstanding mailbox requests can be determined.

Word 2 is called the "terminate interrupt multiplex word". When the 355 is told to read a particular submailbox, the 6180 has to wait until the 355 is finished with the submailbox before it can be reused. When the 355 is finished, it OR's a bit into the terminate interrupt multiplex word and interrupts the 6180. The position of this bit corresponds to the number of the submailbox being freed (bits are numbered from the left starting with 0).

Word 3 holds the previous value of word 1 as described earlier.

Word 4 is a lock. When a process is executing in the 355 interface program, it locks the mailbox area with its process id in this word to prevent interference from other CPUs.

Words 5-7 are not currently used.

Submailbox Format

Word 1 contains a 3 bit 355 number which goes from 0 to 3, a 15 bit line number which is assigned to a terminal when it connects to the 355 and which is referenced in all further references to the line, and an 18 bit terminal id which is used by NPS but not by Multics.

Word 2 contains a 9 bit terminal type which is not used, a 9 bit count of the number of 6 bit characters of command data (if any) in the command data field of the submailbox, a 9 bit op code (described later), and a 9 bit I/O command (described later).

Word 3 contains command data (different for each command).

Word 4 contains command data.

Word 5 contains command data.

Word 6 contains an 18 bit data address, an unused 9 bit field, and a 9 bit data length (in words).

Word 7 is unused.

Word 8 contains an 18 bit field not used by Multics but which is used by NPS. It must have the value 000777(8). The second 18 bit field holds a checksum of the previous 30 9 bit characters (7 1/2 words).

Operation Codes

From 6180 to 355

Op Code	Meaning
Terminal Accepted (00)	Sent to 355 in response to a dialup.
Disconnect This Line (01)	Sent to 355 to hangup a line.
Disconnect All Lines (02)	Sent to 355 when 6180 going down. This doesn't really mean for the 355 to hangup all lines, but is a way to perform the function of notifying the 355 that the 6180 is going down while still staying within the GCOS/NPS standard interface.
Accept Calls (04)	Sent to 355 when Multics ready to accept dialups.
Input Accepted (05)	Sent to 355 to tell it where to write input data in circular buffer. First dcw goes in word 5 of the submailbox and second dcw (if required) goes in last word of command data (word 4).
Accept Direct Output (12)	Sent to 355 to pass address of a buffer which contains a "dcw list". This buffer contains a series of dcws, one for each buffer in an output chain. The left 18 bits of the dcw is a buffer address, the next 9 bits are not used, and the next nine bits contain a character tally. The address of the buffer full of dcws goes in the address portion of word 5 of the submailbox and the count of the number of dcws goes in the word count in word 5.
Accept Last Output (13)	(Known in 6000/355 interface as Accept Direct Output and Wait For Input.) Sent to 355 to pass address of output chain when this is the last chain to be output.

Op Code	Meaning
Reject Request - Temp (16)	Sent to the 355 when the 6180 is unable to immediately take input from the 355 into its circular buffer.
Terminal Rejected (20)	Sent to the 355 if the 6180 does not want to accept a dialup (e.g., the line is not in the lines file).
Disconnect Accepted (21)	Sent to the 355 after it has reported a hangup to the 6180.
6000 Init Complete (22)	Sent to the 355 in response to the 355 being bootloaded (see example below).
Break Acknowledged (35)	Sent to the 355 in response to a break.
Alter Parameters (42)	This is an escape mechanism in which ttydim extensions mentioned previously in this MTB can be implemented. See below for a definition of the command data passed with this command.
Checksum Error (43)	Sent to the 355 when the 6180 discovers a checksum error on a mailbox which the 355 sent to it.

Operation Codes

From 355 to 6180

Op Code	Meaning
Accept New Terminal (100)	Sent to 6180 when a dialup occurs. The first word of the command data contains the physical connection information as follows: bits 0 - 15 are zero bits 16 - 17 are the 355 number. bits 18 - 23 hold the 355 iom channel number. bits 24 - 28 hold the hsla subchannel (0 - 31) if an hsla line. bits 24 - 29 hold the time slot (0 - 51) if an lsla line. bit 35 is 0 if an hsla line, 1 if lsla
Disconnected Line (101)	Sent to the 6180 when a hangup occurs.
Send Output (105)	Sent to the 6180 when the 355 is ready to send the next output chain.
Connect To Slave (111)	The last command in a dialup sequence (ignored by 6180).
Accept Direct Input (112)	Sent by the 355 when it has input to transmit to Multics. In the left half of first word of the command data is passed the number of characters of input that the 355 has for the 6180. In the right half of the first word is a number. If 0, the input contains no break character. If 1, the input does contain a break character. When the 355 writes its data into the circular buffer, the message is preceded by a 36 bit word of which the left half is the character count and the right half is unused.

Op Code	Meaning
Break Condition (113)	Sent to the 6180 when a line break is received from a terminal.
Checksum Error (126)	Sent by the 355 to the 6180 when it discovers a checksum error in a mailbox received from the 6180.
Alter Parameters (127)	This is an escape mechanism in which ttydim extensions mentioned previously in this MTB can be implemented. See below for a definition of the command data passed with this command.

I/O Commands

Within the submailbox is an I/O command field and an op-code field. The I/O command field is defined to contain one of the following:

1	RCD	Read Control Data
2	RTX	Read Text
3	WCD	Write Control Data
4	WTX	Write Text
5	ACU	Automatic Call Unit
6	RJT	Reject Mailbox
7	STI	Send Terminate Interrupt
8	DWT	Delayed Write Text

The first four of these (RCD, RTX, WCD, WTX) are the commands used most extensively by the 355 and are defined more fully below.

The op-code field is used to further define the I/O command. For some I/O commands, within certain contexts, an op-code field may not be required. The dialog between the 355 and the 6180 is really a handshaking where the I/O commands define in general what the submailbox contains and the op-code defines specifically the function being performed.

In general the following statements may be made about the RTX, WTX, RCD, and WCD I/O commands.

1. The 6180 places all 4 of the I/O commands into the submailboxes. The 355 merely uses the I/O command which is in the submailbox.
2. The I/O commands which deal with transfer of data other than submailbox information always are the RTX or WTX I/O commands.
3. The I/O commands which deal with transfer of only submailbox information are the RCD and the WCD I/O commands.

4. The P I/O commands designate flow from the 355 to the 6180 while the M I/O commands designate flow from the 6180 to the 355.

PCD

a. The 6180 sends this I/O command in response to a mailbox request from the 355. It means "here is a free submailbox".
Specifically, when the 6180 receives a mailbox request from the 355 it ...

   1. finds a free submailbox
   2. places a PCD I/O command code into the submailbox
   3. interrupts the 355 on the correct interrupt level

b. When the 6180 receives a terminate interrupt and the I/O command code in that submailbox is PCD it means that the 355 has just put an op-code into that mailbox and it is an op-code from the 355 to the 6180.

RTX

a. The 6180 sends this I/O command (i.e. places it into a submailbox and interrupts the 355) to tell the 355 that it has placed an op-code into this submailbox which requires a data transfer from the 355 to the 6180. At this time the DCW is sent so that the 355 may use it.

b. When the 6180 receives a terminate interrupt and the I/O command code in that submailbox is RTX it means that the 355 has just completed processing the RTX I/O command (same one) that the 6180 had sent (e.g. input accepted) and the 6180 may now free the submailbox.

MCD

a. The 6180 sends this I/O command to tell the 355 that it has placed an op-code into this submailbox for the 355 to process. i.e. it is an op-code sent from the 6180 to the 355 via a connect.

b. When the 6180 receives a terminate interrupt and the I/O command code in that channel mailbox is MCD it means that the 355 has just completed processing the MCD op code the

6180 had just sent (same one) and the submailbox may be freed now.

WTX

a. The 6180 uses this I/O command to tell the 355 that it has placed an op-code into this submailbox which requires a data transfer from the 6180 to the 355. At this time the DCW is sent so that the 355 may use it.

b. When the 6180 receives a terminate interrupt and the I/O command code in that channel is WTX it means that the 355 has just completed processing the WTX op-code the 6180 had just sent (same one) and the submailbox may now be freed. The data may be released now as the 355 has moved the data to its memory.

Interface Conventions

The following material is presented to define the standard protocol used by the 355 and the 6180 in communicating with one another. These conventions are assumed in the material which follows which defines the use of the op codes and I/O commands defined previously.

A. Whenever the 355 wants to initiate a communication with the 6180, it adds one to its counter of mailbox requests. (This counter is 0 mod 256K.) It writes this counter into word 1 of the 355 mailbox header in the 6180 (datanet_mbx.mailbox_requests) and interrupts the 6180.

B. The 355 ends a communication with the 6180 by writing the submailbox it was using back to the 6180, or'ing a bit into the terminate interrupt multiplex word (timw) whose position corresponds to the number of the submailbox (starting with 0), and interrupting the 6180.

C. When the 6180 is interrupted, it first scans the timw to see if any terminates have occured. If so, it frees up these mailboxes. Then it checks the mailbox_requests word against the value stored in datanet_mbx.last_mbx_req_count. If these values are different, it computes the difference which is the number of submailbox requests made by the 355. It then updates last_mbx_req_count by copying mailbox_requests into it. Then it tries to satisfy as many of these requests as possible by selecting a free submailbox, placing a Read Control Data (RCD) I/O command in it, and interrupting the 355 on a special interrupt level (0-15) whose number is equal to the number of the submailbox.

D. The 6180 writes all the I/O commands in the submailboxes. Both the 355 and the 6180 start their processing of a submailbox by reading this command. If the 355 finds a 6180 command (see previous definition of I/O commands), it realizes this was in response to its mailbox request. If the 6180 finds a 355 command, it simply frees the submailbox.

Transaction Pageant

Listed below are a series of possible interactions between the 6180 and the 355 using the interface commands listed previously. These interactions are intended to give the flavor of the 6180/355 interface and are not intended to be an exhaustive list of all possible interactions between the two machines. The sequence starts with the bootloading of the 355.

1. The 355 is bootloaded and after initializing itself, it will write a word with -1 in the left half into the mailbox_requests word of the 355 mailbox header. It will then interrupt the 6180.

2. The 6180 will respond to this interrupt and when it sees the -1 it will select one of its 16 submailboxes. It will place a Write Control Data (WCD) I/O command and a 6000 Initialize Complete op code in the mailbox. It will then cause the 355 to read this submailbox by interrupting it as described in C of the Interface Conventions.

3. The 355 will read this submailbox and will wait until the 6180 tells it it is ready to accept calls.

4. When the 6180 is up and the answering service is ready to accept dialups, the 6180 selects a submailbox and places a WCD I/O command and an Accept Calls op code in the submailbox. It then causes an interrupt to the 355 as described in 2.

5. The 355 reads the submailbox and is now ready to accept dialups. It will set a bit in the timw to end this transaction as described in B of the Interface Conventions.

6. When the 355 receives a dialup, it will request a submailbox from the 6180. When it reads the submailbox, it will add the op code Accept New Terminal, the line number, and the command data which tells the physical connection into the 355 of the new terminal. It then writes the submailbox back into the 6180 using the terminate conventions as described in B of the Interface Conventions

7. The 6180 will find this bit in the timw and will look at the corresponding submailbox. It will see the RCD and the Accept New Terminal op code which will cause it to process the dialup. If it is ok for a dialup on this line, a submailbox will be selected in which will be placed a WCD I/O command and a Terminal Accepted opcode. If not, it will place a WCD and a Terminal Rejected op code in the submailbox. Then it will interrupt the 355 to cause it to read the submailbox.

8. The 355 will process the submailbox and then terminate to cause the 6180 to free the submailbox.

9. Finally, the 355 will request a new submailbox as described earlier. The 6180 will respond with one with an RCD in it. The 355 will add the Connect To Slave op code and write it back to the 6180 using the normal terminate procedure. This completes the dialup sequence.

10. At this point, the 6180 will want to write a greeting message to the new terminal, so it selects a submailbox, places a Write Text (WTX) I/O command and an Accept Direct Output op code in it. It also adds the address of the list of dcws which point to the buffers in the chain and the count of dcws in the list in word 5 of the submailbox. Then an interrupt is sent to the 355.

11. The 355 will read the submailbox and extract the dcw list address. It will then read the dcw list and using that, the entire buffer chain into internal buffers and terminate. When the 6180 processes this terminate, it will free up the submailbox, the buffer holding the dcw list, and the buffer chain which has now been read by the 355.

12. When the 355 is nearly done sending the buffer chain that it just read from the 6180, it will request a new submailbox from the 6180 as described above. The 6180 will reply with a submailbox with I/O command RCD. The 355 will set an op code of Send Output into the submailbox. Then it will terminate.

13. When the 6180 sees this op code, it will send the next output chain as described previously. If there is no more output after this chain, it will use the op code Accept Last Output. Otherwise, it will use the Accept Direct Output op code.

14. If this was the last output chain, the 355 will go into input mode. When it receives input, it will request a submailbox as described above. When the 6180 responds with a submailbox with I/O command RCD, it will enter the op code Accept Direct Input and a count of the number of characters in the input message into the submailbox and terminate.

15. The 6180 will find the next free word in its circular input buffer and will select a submailbox. In this submailbox will go the I/O command Read Text (RTX) and the op code Input Accepted. Word 5 will be filled in with the address in the circular queue and the number of words to write there. If the entire 355 message will not fit in the space between the free word and the end of the circular buffer, the 6180 will place a second address and count in the last word of the

command data (word 4) which will point to the beginning of the circular buffer.

16. The 355 will read this mailbox, write its data into the 6180 circular input queue and will then terminate. This will cause the 6180 to go read the message out of the circular input buffer and copy it into the paged input buffer.

Alter Parameters

As was mentioned in the description of the command Alter Parameters, this command exists as a way to extend the capabilities of the 355 software while still adhering to the standard 355/6180 interface. Listed below are the way in which extensions mentioned in the main body of the MTB will be communicated across the interface. In each case, the extension data will go in the 3 words of command data in the submailbox. The first 6 bit character will be a sub-command which defines how the rest of the command data is to be interpreted.

Sub-command	Meaning
Breakchar (01)	The first 9 bit byte of the command data following the sub-command is a coded number. If it has the value 777 (octal), it means that the 355 should break on every character. If it has the value 776 it means that the 355 should break on all ASCII control characters. If it has the value 775, it means that the 355 should break on the next character following recognition of the break character. This break character is specified in the next 9 bit byte of the command data. For example, one could specify the ETX character so that the 355 would break on the next character received after an ETX (such as a block check character). If all 0's, it means that the 355 should not break on any character. If it is between 1 and 8, it is a count of the number of following break characters (each occupying a 9 bit byte) that the 355 should accept as break characters.
Nocontrol (02)	The only datum is a 1 bit switch right justified in the 9 bit byte following the sub-command byte. If this switch is "1"b, it means that the 355 should enter "no-control mode" for this terminal (as described above). If "0"b, the 355 should leave this mode for this terminal.
Fullduplex (03)	The 355 should perform echoing of characters received. This mode is turned on/off as described for 'Nocontrol'.
Polite (04)	The 355 will not send any output to a terminal that is currently inputting. This mode is turned on/off as described for

Sub-command	Meaning
	'Nocontrol'.
Errormsg (05)	This command comes from the 355 to the 6180. It is used when the 355 wants to report an error on the system console like a DIA error. The data passed with the command is a 27 bit number which is used by the 6180 to index into a table of error messages. Following the number may be up to 72 bits of additional information. The additional information (if any) depends on the type of error. It might be a 36 bit status word for example.
Meter (06)	The 6180 wants to read the 355 meters. The command data is the 24 bit absolute address of a page in 6180 memory where the meters in the 355 are to be written. This address must be on a 1024 word boundary to insure that all the meters will fit in the one page (it is presumed that there will never be more than 1024 36 bit words of metering data).
Sensepos (07)	This sub-command will cause the 355 to write any input it has for a given terminal into the 6180 using the Accept Input op code as described previously. If there is none, it will still write the standard message header in which the character count will be zero.
Crecho (10)	The 355 will echo carriage return upon receipt of line feed. This mode is turned on/off as described for 'Nocontrol'.
Lfecho (11)	The 355 will echo line feed upon receipt of carriage return. This mode is turned on/off as described for 'Nocontrol'.
Lock (12)	The 355 will attempt to lock the keyboard before sending output to this terminal. This mode is turned on/off as described for 'Nocontrol'.
Msg (13)	The 6180 is passing a precanned message in ASCII or IBM BCD for transmission to a terminal which dials in (while the 6180 is down). This type of command is the same format as a normal Accept Direct Output op

Sub-command	Meaning
	code except that the 9 bit byte of command data following the sub-command indicates whether this message is ASCII (1) or IBM BCD (2). The 6180 will make the msg call twice to provide the 355 with both types of precanned messages.
Upstate (14)	The 6180 sends this command to the 355 to see if it is up. If the 355 sends a terminate for this message, it is up. If the 6180 times out with no terminate, it will assume that the 355 is down.
Dumpoutput (15)	The 355 will throw away any output. This command will usually be issued after a quit. The 6180 will send a newline, issue a Sensepos command to get any input so it can be thrown away (or replayed if the user so wishes), and then issue a Dumpoutput to cause any output being sent to a terminal to be stopped.
Tabecho (16)	The 355 will echo spaces when it receives input tabs if this mode is on. Only taks effect if in full duplex mode. This mode is turned on/off as described for 'Nocontrol'.
Setbusy (17)	This command allows the 6180 to direct the 355 to force some types of datasets to appear busy. This mode (the dataset busy lead) is turned on/off as described for 'Nocontrol'.