MIDAS-3

MONITOR AND SCOPE
PROGRAM DESCRIPTIONS

R.E. Brody
September 1969
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE</td>
<td>1</td>
</tr>
<tr>
<td>MONITOR PROGRAM DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>Layout</td>
<td>6</td>
</tr>
<tr>
<td>Introduction</td>
<td>8</td>
</tr>
<tr>
<td>Concepts</td>
<td>9</td>
</tr>
<tr>
<td>Structure</td>
<td>12</td>
</tr>
<tr>
<td>Details and Flowcharts</td>
<td>20</td>
</tr>
<tr>
<td>SCOPE PROGRAM DESCRIPTION</td>
<td></td>
</tr>
<tr>
<td>Layout</td>
<td>59</td>
</tr>
<tr>
<td>Introduction</td>
<td>70</td>
</tr>
<tr>
<td>Concepts</td>
<td>71</td>
</tr>
<tr>
<td>Structure</td>
<td>75</td>
</tr>
<tr>
<td>Details and Flowcharts</td>
<td>81</td>
</tr>
</tbody>
</table>
This is the first attempt at SLAC to monitor the progress of an experiment using the IBM/360 M91 computer. Previous experiments have used the 1800 computer to perform the dual function of collecting and partially analyzing the data. However, in this and future cases the data rate will be too high for small computers, such as the IBM 1800, to perform any analysis. Their full capacity will be absorbed in collecting the data, with little core space or time available for checking it. One solution to this problem is to use the M91-1800 link to transfer data periodically to the M91 where a complete and rapid analysis of the sample can be performed.

In order to satisfy this requirement a program was designed for the IBM/360 M91 which would fulfill the following functions: obtain a significant sample of the data being collected on the IBM 1800 computer on-line to a wire spark chamber experiment; perform a complete and immediate analysis of that sample on the M91; give the experimenter an instant display of the results of that analysis and provide him with a means of fast interaction through the use of a 2550 scope display situated in the experimental area.

The program which was finally designed for the M91 was called the Midas-3 system, as its functions were similar to those performed by the Midas program for the IBM 1800 computer for an earlier experiment.

Midas-3 consists of four separate tasks operating independently but able to communicate between each other and the 1800 computer in a 150K region of core of the M91. These four tasks were named the Monitor, the Scope, the Analysis, and the Command task.
Monitor is the control program; it has the highest priority, and it controls the data being passed across the 1800 link; it initiates and terminates the other three programs; it transfers information between the other programs and is in fact a sub-system program. The Scope program will display whatever is requested by the other three programs which may be, in turn, a response to a request typed in at the scope console by the user. The Scope task will have a high priority, in the M/1 so that response delay will be small, and will probably appear immediate to the user. The Analysis task will be the standard M/1 program used to analyse the data tapes collected by the 1800-computer. The analysis of the sample data will be complete and results will be put on disc packs before being printed. The Analysis task will operate at a low priority so as not to disrupt the normal batch running of the M/1. All the left-over CPU cycles available in the M/1 will be used by the Analysis task which will then perform as a background job. The Command task is the receiver of all commands entered into the system, whether from cards (when running off-line) from the scope or from the 1800, and is responsible for interpreting and responding to them.

Midas-3 is primarily designed to sample data being collected at the end station by an 1800 computer, perform a complete analysis of it in the M/1, and display histograms and other information, on request, on the 2250 scope display situated in the end station. The Midas-3 system will be permanently in core during the experimental run. It will control the M/1-1800 link and will request buffers of data from the 1800 periodically. This data will be fed to the M/1 analysis program, which will do a complete analysis on that data and when finished will request more data from the 1800. The amount of data transferred and
analyzed in Midas-3 will depend on the work load of the M91 and will probably vary hourly. In addition to the Analysis program, the Scope package will be available to control the 2250 scope in the end station. This scope will be used in roughly the same manner as it was for the 1800-Midas program. The keyboard will be unlocked and histograms and other information may be displayed on request.

The following points were important design considerations.

1. All programs are written in assembler language. Due to the fact that this program was to be in core permanently while the experiment was running, and also some time prior to that, only 150K was allowed for all four programs. It was thought to be almost impossible to do this if either Fortran or PL1 was used. Further, at least part of the Monitor had to be written in assembler language as some system macros for attaching and detaching tasks were required in any case.

2. The Monitor, Scope, and Command tasks are at a high priority. This will ensure a rapid response to the scope user.

3. The Analysis task is at a fairly low priority so that it can soak up the remaining M91 cycles without disturbing the batch programs too much. The amount of analysis being done will probably depend on the work load of the M91 and will vary hourly. However it is hoped that a certain minimum will be obtained at all times of the day.

4. The M91 controls the link completely. Since data is only being sampled there is no necessity for the 1800 to interrupt the M91 unless requested to do so. As a result of this, unsolicited
interrupts from the 1800 are currently not accepted. At some future
date these will be accepted, since the code has been written to
allow for them.

5. Independent programs were needed as far as possible so that
several people could work on them simultaneously without too much
confusion.

6. A very elastic design was necessary as no very clear idea of how
the program should work was available when writing began.

This document contains descriptions of the Monitor and Scope tasks
only. Each of these tasks is described separately as their operation
is quite independent. The description is divided into four parts:
the Introduction, Concepts, Structure, and Details. Each section is
increasingly more detailed so that the final section is only for the use
of those people or person who may wish to make modifications.

The code itself is fairly well documented and this paper is
intended not only to give a general idea of the program but also to
supplement the comments in the code.
MONITOR PROGRAM DESCRIPTION
LAYOUT FOR MONITOR PROGRAM DESCRIPTION

INTRODUCTION

CONCEPTS - General

Background for Design 9
Design Considerations 9
Functions of the Program 10
Future Expansion 11

STRUCTURE

General Layout of Subroutines 12
1800 Link Control 13
Timing 14
Communication 15
Abnormal Termination 18

DETAILED DESCRIPTIONS AND FLOWCHARTS

General Routines

Initialisation 20
Monitor Main Loop 21
Check Command Request 23
Check Scope Request 23
Check Analysis Request 23
Check Link Data Buffers 25
Get Commands Routine 27
Scope Recovery Program 29
Send Link Message 31
Attention Handler 34
ECP Appendage 36
Exit for Main Loop Timer 38
Analysis Exit Routine 39
Scope Exit Routine 40
Command Exit Routine 41

Scope Request Routines

Scope ECP and Termination Parameters 42
Send Message to Command Task 44

Analysis Request Routines

Analysis ECP 46
Request Data from Link 47
Send Message to Scope 49
Command Request Routines

Command ECB
Attach Scope without Interrupts 52
Attach Scope with Interrupts 53
Enable the Interrupts on the Scope 57
Open 1800 Link 59
Send Message from Command Task to Scope 61
Attach Analysis Task 64
Abend Request 66
Detach Scope Task 67
Close Link 67
Disable Interrupts on Scope 67
Detach Analysis Task 67
This Monitor was written as the main system task in the Midas-3 system. It is designed to function with three subtasks, currently called the Scope, the Analysis and the Command tasks, and to operate the 1800 Link. It was designed so that if any of the subtasks were altered or removed, the main body of the Monitor program could still function. This would allow the program to be easily modified for use in other environments.

The original design of the IBM 1800 Link included the possibility of receiving and servicing unsolicited interrupts from the IBM 1800 computer. This facility was later removed in order to simplify the testing of the program and the link. However, when the link is known to be working reliably, this facility should be restored. The EXCPAM access method was used in the NOL code for communication across the link.

This write-up describes the functions of the Monitor in the Midas-3 system. For description of the Scope program see the second part of this document. The Analysis and Command programs will be documented separately but a very brief description of their functions can be found in the preface.

The last part of the Monitor write-up containing the detailed descriptions and flow charts is primarily for the use of anyone who may have to alter or even (heaven forbid!) debug it. It is not intended to be light reading but is included here for the sake of completeness.
Background for Design

Since this is the first attempt at SLAC to write a set of programs as Midas-3 to link a small and large computer together, several unknown factors had to be taken into consideration. Whether or not the program would be required to work in another environment, i.e. with different subtasks, or even a different number of subtasks, how much M3L core would be available for the final run, whether or not the link would be reliable, whether or not this system would be used off-line to analyze the final experimental results, were all factors which were unknown. Consequently, some kind of a balance had to be maintained between making the program very general, which has the disadvantage of increasing both the size of the program and the time required to write it, and keeping the program small enough to perform certain specific functions. The latter has the disadvantage of making it difficult if not sometimes impossible to alter.

Design Considerations

The Monitor was not designed by itself but as an integral part of the Midas-3 system. It is therefore impossible to consider the design of the Monitor without examining the whole Midas-3 system. Several separate functions had to be put together so that they could operate effectively. For example a scope package was needed to operate the 2250 Scope, a program to control the 1800 Link and the Analysis program were primary requirements. To put these functions in one program would have had several disadvantages, mainly that they would all be operating at the same priority, but also that more than one person would have been working on the program making it more inconvenient. Making each function a separate job would satisfy the priority requirements but would have necessitated some system changes to allow communication between jobs. It was decided that no system changes should be made unless there was no alternative. Finally it was decided that a set of subtasks in one region with communication tables set up between the tasks satisfied most of the primary requirements. These tasks were then outlined as the Monitor, the Scope, the Analysis and the Command tasks. The Monitor, Scope, and Command tasks were to have high priority, while the Analysis, which was to use most of this program's cycles, was to have a low priority. This would enable the Scope user and the 1800 Link a fast
response while not disturbing the regular operation of the MK1. Fig. (1) shows the physical layout of the system with respect to the experimental area while Fig. (2) shows the distribution of core in the MK1, 150K region.

To summarise then, the basic design considerations of the Midas-3 system are listed below.

1. All programs must be written in assembler language, for the sake of speed and space as only 150K of MK1 core was available.
2. Each function should be as separate as possible with clear interfaces to allow several people to work on them with a minimum of confusion.
3. Monitor, Command, and Scope tasks must have a high priority for rapid response to the user and the 1800 compiler.
4. Analysis module must have a low priority so as to be permanently in core soaking up the remaining CPU cycles without disturbing the normal running of the MK1. This was to be for the duration of the experimental run.
5. Separate functions (i.e. tasks) had to be able to operate independently but still be able to communicate.
6. If possible no system changes were to be made.

Functions of the Monitor

As noted earlier the Monitor is the master task in the region, and as such performs certain major functions with respect to the other tasks in the region. The following is a list of these functions.

1. The Monitor performs all communication between its subtasks. This is necessary if the Monitor was to perform any error recovery at all; for example, the Analysis program might wish to send a message to be displayed on the scope while in the meantime the Scope program has attended. Only the Monitor is aware of the termination of the Scope program and any attempts to re-initiate it.
2. The Monitor performs all I/O to the 1800 link, and totally controls the link.
3. The Monitor does the data buffer handling for the Analysis program, i.e. when the link is open the Monitor will attempt to get two buffers for the data from the 1800; it will then on attempt to keep those buffers full so that there is a minimum wait time for the Analysis program. It will also free one of those buffers at a later date if the space is needed for something else.
MONITOR: Controls the partition.
ANALYSIS: Performs the analysis of the experimental data.
SCOPE: Used to display histograms and other conformation.
COMMAND: Receiver of all commands entering the system.

FIG 2.
4. Since unsolicited interrupts from the 1800 are not accepted at present, the Monitor has the task of making periodic requests for commands and comments etc. to the 1800. When obtained it passes these on to the respective programs to deal with them.

5. The Monitor also attaches and detaches subtasks at the request of the Command module. This is necessary when changing from on-line to off-line and vice versa.

6. The Monitor does all error handling and attempts to recover if one of the subtasks terminates.

Future Expansion

At the present time (beginning Sept. 1969) the Monitor is working satisfactorily in an off-line environment. That is, the Monitor, Scope, Analysis, and Command tasks have been operating together with all the functions performing except that data for the analysis program has been read from tapes only. During the experimental run, data for the analysis program will be obtained through the Link from the 1800 computer. Since the link is not yet operating, although the Monitor code which controls the link has been tested with the use of a 2260 display, it is to be expected that most of the modifications which will take place will be in this area of the code. These modifications are really unpredictable as they will depend on the operation of the link and the procedures decided when either programs or hardware in each of the computers fail.
General Layout

The design of the Monitor is basically very simple (see fig. (3)). The entry point for the program is called MONIT, and the first subroutine entered is the initialization routine. On entry the initialization routine saves the registers, links the save areas and sets some flags. It then issues two IDENTIFY macros, one for the Scope task entry point and one for the Command task entry point. This then means that both the Scope and Command programs have to be part of the same load module as the Monitor. If at any time it is desired to make the Scope and Command tasks separate load modules which are loaded from a Joblib library when they are attached, then these two IDENTIFY macros must be removed. The initialization routine then attaches the Command task and finally exits into the Monitor Main Loop.

The Main Loop is the backbone of the Monitor task. It remains in this loop until the Command task, and later the other subtasks signal that they have a request. Initially only the Command task is attached. Other tasks are attached and the 1800 Link opened by the Monitor at the request of the Command task. Starting at the beginning of the loop the Monitor searches through three tables to see if any of its subtasks have any requests. If any subtask has a request it branches to a subroutine to check whether it is a valid request, and if it is, control is passed to a lower level routine. This lower level routine corresponds to the code number passed in the request and services that request only. When the request is serviced, control is returned to the Main Loop along the identical path and the request flag is turned off. If for any reason the request is unable to be serviced at that time, then a flag is set to indicate the fact, control is returned to the Main Loop along the same path as if the request had been completely serviced, but the request flag is left on. On a later pass round the Main Loop another attempt will be made to finish that request. After servicing a request, control is returned to the Main Loop at exactly the same point it left. After checking and servicing any requests which the subtasks might have, the Monitor tests whether the 1800 Link is open. If it is, it will then check if there is an unsolicited request from the 1800. Currently this is a dummy routine as no unsolicited requests from the 1800 computer are accepted. Again if the link is open the Monitor will check
FIG 3.
the Analysis programs 1800 data buffers. If either or both of these buffers are empty, the Monitor sends a message to the 1800 computer to request data for these buffers. The Monitor will now attempt to send a message to the 1800 to request commands for the Command task. It will be able to do this if the Analysis data buffers are both full and the Command task has serviced its previous command. Finally the Monitor will attempt to send a message to the 1800 computer to request comments, INVT's, etc. for the Analysis program. This will be done only if the link is not being used and the Analysis program has serviced the earlier request. The Monitor will now set a timer and attempt to go into the wait state. This timer is only a precautionary measure so that the Monitor never accidentally enters a permanent wait state.

The Monitor will come out of the wait state when one of the following things happens; either the timer terminates, one of the subtasks has a request and therefore posts the Monitor, or there has been some activity on the 1800 link and the Appendage or Attention Handler posts the Monitor. The first thing the Monitor does when it becomes active again is to clear the ECB it has been waiting on. This ensures that if a request becomes active after it has been checked and found inactive, the Monitor will not at the end of this cycle enter the wait state, but will find its ECB already posted and after clearing its own ECB it will return to the beginning of the Main Loop to start its cycle once again.

All tables and constants used by the Monitor are in the Monitor Work Area control section called WORKAREA; see the Communication section for further details.

1800 Link Control

The 1800 computer link is currently controlled entirely by the Monitor in the N91. No communication takes place across the link which has not been initiated by the Monitor. This means that the 1800 computer cannot send commands to the Command task in the N91 but has to wait until they are requested by the Monitor. This was done to simplify the communication across this link during the initial period of running. Tables and some code are available to handle unsolicited messages from the 1800 but will have to be examined and probably debugged before being used.

The Monitor manages the operation of the 1800 link by use of a table called the Link Activity Table which is in the Monitor's work area. See fig. (b) for details of this table. This table will indicate the state
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBUF</td>
<td>SUFFIX of SEGMENT</td>
</tr>
<tr>
<td>SLEN</td>
<td>LENGTH</td>
</tr>
<tr>
<td>SBUFADDR</td>
<td>SUFFIX of SEGMENT</td>
</tr>
<tr>
<td>SOUTLMAD</td>
<td></td>
</tr>
<tr>
<td>INVLIG</td>
<td>INVLIG</td>
</tr>
<tr>
<td>ILEN</td>
<td>LENGTH</td>
</tr>
<tr>
<td>IADDR</td>
<td>LENGTH</td>
</tr>
<tr>
<td>OUT</td>
<td>ACCESS where actual data length passed is to be stored (outside of SEG)</td>
</tr>
<tr>
<td>OUTLMAD</td>
<td>ACCESS where actual data length passed is to be stored (outside of SEG)</td>
</tr>
<tr>
<td>OUTLF5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>INDICATES a READ has been issued</td>
</tr>
<tr>
<td></td>
<td>1 INDICATES an attention is expected, and data</td>
</tr>
<tr>
<td></td>
<td>is to be read</td>
</tr>
<tr>
<td></td>
<td>2 INDICATES a flag to be set, the last bit of the byte given by the address</td>
</tr>
<tr>
<td></td>
<td>in SEG is set, set to 1 when the read terminates</td>
</tr>
<tr>
<td>OUTLF6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>INDICATES a READ has been issued</td>
</tr>
<tr>
<td>2</td>
<td>INDICATES a flag to be set, the last bit of the byte given by the address</td>
</tr>
<tr>
<td></td>
<td>in SEG is set, set to 1 when the read terminates</td>
</tr>
<tr>
<td>OUTFL5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>INDICATES a WRITE has been issued</td>
</tr>
<tr>
<td>2</td>
<td>INDICATES a flag to be set, the last bit of the byte given by the address</td>
</tr>
<tr>
<td></td>
<td>in SEG is set, set to 1 when the write terminates</td>
</tr>
<tr>
<td>OUTFL6</td>
<td></td>
</tr>
<tr>
<td>1, 3, 7</td>
<td>UNUSED</td>
</tr>
</tbody>
</table>

**Note:**

- The OUTLMAD is used to store the address where the actual data length passed is to be stored.
- The OUTLF5 and OUTLF6 fields are used to indicate the status of the read and write operations.
- The OUTFL5 and OUTFL6 fields are used similarly but for the write operation.

**Figure 4:** Activity Table

Link activity table and methods for data transmission.
of the link at any instant in time. Only two types of activity can take place across the link, i.e., either Reads or Writes. A read can be issued when a message is expected or when an attention occurs unexpectedly; these are solicited and unsolicited Reads respectively. Then there are three types of activity which may take place across the link, Unsolicited Reads, Solicited Reads, and Writes. The table is therefore divided into three sections; the solicited read buffer, the unsolicited read buffer, and the write buffer, although only one buffer may be active at any one time.

Currently the unsolicited read buffer must always be empty. The write buffer indicates whether a write has been issued from the Monitor to the 1800 computer. No other writes may be issued until this write has been terminated and this buffer cleared; in fact, writes may not be issued unless all the link activity buffers are empty. The write buffer will indicate the status of that write command. When a write is issued and a reply is expected, a flag in the solicited read buffer will indicate that a reply from the 1800 is expected. When an attention is obtained from the 1800 Link the Attention handler can check from the solicited read buffer that a message is requested, where it is to be placed, and its maximum length. With this information the attention handler can issue the Read to obtain the reply. The Appendage routine which gains control at the end of any transfer of data uses the Link activity table to determine whether it is the end of a read or write and whether an END of Flag needs to be posted to indicate that the information has been transferred.

There is a general routine (SENDRLMD, Send Link Message) which is used to set up the Link Activity Table and issue the write whenever messages are to be sent to the 1800 computer. Please see the subroutine write up for details of the parameters.

**Timing**

The Monitor has one timer which is used to cause it to become active at periodic intervals. Currently that timer is set at ten seconds but might have to be made smaller if the commands and comments, EMU's, etc., have to be requested more frequently.
The Monitor has been designed so that it does not wait for any event; it only goes into the wait state if there is nothing further to be done. For example, if there is a request to send a message down the link, the Monitor issues the write and, without waiting for the end of data transfer, will continue checking and servicing requests until there are none left. The monitor should never be in a position where it cannot service one request because it is waiting for another. Bottlenecks, such as in the 1800 link, may occur but probably the data and messages being transferred across it will not be so great as to cause any slowdown either in the Monitor or any of its subtasks.

Any requests to the Monitor when accompanied by posting the Monitor RCB gets serviced immediately, if the Monitor is in the wait state, or immediately after completing its current requests if it is already busy. Any request to the Monitor by one of its subtasks which is not accompanied by posting the Monitor RCB may have to wait until the Monitor becomes active when the timer expires. This may be anything up to ten seconds which is the current Monitor timer interval.

Communication between Tasks

One of the main functions of the Monitor is to perform the communication between its subtasks. It does this by setting up two tables for each task in its own work area and passing the addresses of these tables to each task as it is attached. Communication between the Monitor and each task is then effected through these tables.

The communication between the Monitor and the Analysis task will be described in detail here, but communication between the Monitor and all other of its subtasks takes place in this manner.
In addition to the two tables mentioned above, two ECB's are necessary to maintain the flow of information between the two tasks. These are: the ECB on which the Monitor waits when it is inactive and there are no requests to service, and the ECB on which the Analysis program waits when it also is inactive. The address of the former is passed to the Analysis program when it is attached, and the address of the latter is communicated to the Monitor during the Analysis program's initialization step. In this manner each program can command the attention of the other by posting these ECB's respectively.

The first of the two tables whose address is passed to the Analysis program is the Monitor Analysis Request Table MNAKREQ (this would be the Monitor Scope Request Table MSDKREQ, in the case of the Monitor/Scope Communication tables). This table is used by the Analysis program to signal a request to the Monitor. Only one request can be made to the Monitor by the Analysis task at a time. No queueing of requests can take place; the Analysis may not issue another request until the Monitor has finished servicing the previous one. See fig (6) for details of this table. The Monitor will post an ECB or set a flag to signal the request has been serviced if the Analysis indicates that is needed when it sets up the request.

The second table is the Analysis Request Table, ANSKREQ (this would be the Scope Request Table, SOKREQ, in the case of the Scope task). This table is used by the Monitor to signal requests from other subtasks or itself to the Analysis task. Each other subtask may send one request at a time to the Analysis task. No second request may be sent from a particular task until the Analysis has accepted the first one, i.e., no queueing of requests takes place. See fig (7). Requests on this table can be set up in two forms: The first is as a message with the first byte of flags and the last three bytes containing the message address, the second form would have a code number in the last three bytes and the first byte of flags would have a bit set to indicate this fact. See fig 8, for the Scope request table.
The order of events if the Analysis task wanted to send a message to the Scope program to be displayed on the 2250 screen would be as follows:

The Analysis task would set up its MHANLREQ table with AMEGCODE equal to 3, this would indicate to the Monitor that this was a request for a message to be sent to the Scope. AMEGADER would be set up to contain the address of the message to be displayed. APUSHBAD and BREADTAD would be set to zero assuming that the Analysis program did not want an ECB posted or a flag set when the request was complete. Finally the Analysis program would set the first bit of AFLAGS to 1 to indicate the request is active and post the Monitor ECB to obtain a fast response. If the Monitor was in the wait state it would now become active and on its cycle in the main loop would discover the Analysis programs request. It would then examine the code number in AMEGCODE and would branch to a subroutine (SCANLMSG) corresponding to that code number. This routine would then look at the Scope Request Table SCRQFTL to determine if the Scope had serviced any earlier Analysis request. If it had not done so, i.e. that buffer was full, the routine SCANLMSG would return to the Main Loop without doing anything. At some later time the Monitor would again notice the Analysis request and would again enter SCANLMSG to service it. This would continue until the Scope had cleared the earlier Analysis request. Once the Scope is ready to accept a new Analysis message, the Monitor places the message address AMEGADER in the last three bytes of the SCANLMSG buffer and sets the first bit of that buffer to 1 to indicate to the Scope program that there is an active request from the Analysis program (The SCANLREQ buffer is the equivalent to the Scope program as the MEOVRQVR buffer is to the Analysis program, see fig 8). The Monitor finally posts the Scope task’s ECB. If the Scope task were at this point in the wait state it would immediately become active and eventually discover the Analysis request. The message would be displayed on the screen and the Scope would clear the SCANLMSG buffer to indicate that it had serviced the request from the Analysis program. The Scope program then posts the Monitor ECB. The Monitor notes that the Scope has displayed the Analysis program’s message and in turn clears the MHANLREQ table to indicate that the Analysis request has been terminated and the Analysis may now send other requests to the Monitor.

This is a typical example of communication between tasks. All the details of the request have not been explained; however the general order of events is as described.
A list of code numbers used for communication between the Monitor and its subtasks are listed below. The table in which each of these codes could be set up is also indicated.

a. Requests to the Monitor from the Scope task. (set up in MMONFREQ)

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scope ECB to be posted when a request for the Scope is active</td>
</tr>
<tr>
<td>2</td>
<td>Send message to the Command task.</td>
</tr>
</tbody>
</table>

b. Requests to the Monitor from the Analysis task. (set up in MMONFREQ)

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Analysis ECB to be posted when a request for the Analysis is active</td>
</tr>
<tr>
<td>2</td>
<td>Request data from the 1800 computer</td>
</tr>
<tr>
<td>3</td>
<td>Send a message to be displayed on the scope</td>
</tr>
</tbody>
</table>

c. Requests to the Monitor from the Command task. (set up in MMONFREQ)

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Command ECB to be posted when a request for the Command task is active.</td>
</tr>
<tr>
<td>2</td>
<td>Attach Scope without interrupts.</td>
</tr>
<tr>
<td>3</td>
<td>Attach Scope with interrupts enabled.</td>
</tr>
<tr>
<td>4</td>
<td>Enable the interrupts on the scope.</td>
</tr>
<tr>
<td>5</td>
<td>Open 1800 Link</td>
</tr>
<tr>
<td>6</td>
<td>Send message from the Command tasks to the Scope</td>
</tr>
<tr>
<td>7</td>
<td>Detach the Scope task.</td>
</tr>
<tr>
<td>8</td>
<td>Close the 1800 Link.</td>
</tr>
<tr>
<td>9</td>
<td>Disable the interrupts on the Scope.</td>
</tr>
<tr>
<td>10</td>
<td>Attach the Analysis task.</td>
</tr>
<tr>
<td>11</td>
<td>Abend the job.</td>
</tr>
<tr>
<td>12</td>
<td>Detach the Analysis task.</td>
</tr>
<tr>
<td>13</td>
<td>Change the priority of the Analysis task.</td>
</tr>
</tbody>
</table>

d. Requests to the Analysis from the Monitor (set up in ADFREQ)

These requests can take two forms, either an address can be set up in the last three bytes, or a code number alone may replace the address. The following is the only request from the Monitor to the Analysis task and is set up as a code number in the request buffer.

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scope task has abended, last request, if any, was lost.</td>
</tr>
</tbody>
</table>
e. Requests to the Command task from the Monitor (set up in COMMONREQ)

These requests can take two forms, either an address can be set up in the last three bytes, or a code number alone may replace the address. The following is the only request from the Monitor to the Command task and is set up as a code number in the request buffer.

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scope task has abended, last request, if any, was lost.</td>
</tr>
</tbody>
</table>

Messages sent to the Scope task are of the format shown in fig. 12. The code number in the message indicates to the Scope task what is requested. A list of these code numbers which are recognisable to the Scope from the other tasks in the region are shown below. Note that these code number can also be set up directly into the Scope request buffer when no additional information needs to be passed. For example code number 3 which is used to request the interrupts to be enabled can be set up in the message as shown in fig 12. or alternatively it can be placed directly in the Scope request table provided the appropriate flag in the first byte of that request is on.

a. Message codes to the Scope from the Command task.

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Display the message on the scope.</td>
</tr>
<tr>
<td>2</td>
<td>Display a message, allow a message to be returned to the Command task.</td>
</tr>
<tr>
<td>3</td>
<td>Allow interrupts to be enabled.</td>
</tr>
<tr>
<td>4</td>
<td>Command module now accepting new commands.</td>
</tr>
<tr>
<td>5</td>
<td>Disable the interrupts and do not allow them to be enabled.</td>
</tr>
<tr>
<td>6</td>
<td>New page for area of screen.</td>
</tr>
<tr>
<td>7</td>
<td>Display histogram.</td>
</tr>
</tbody>
</table>

b. Message codes to the Scope from the Analysis task.

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Display the message on the Scope</td>
</tr>
</tbody>
</table>

Requests from the 1800 computer to the Analysis and the Command tasks (in ANALREQ and CMNREQ respectively) are always set up in the form of an address which points to the message from the 1800 computer. A 'no-data' message consisting of four bytes of zeros is recognised by the Monitor when the 1800 has no information to send.
Requests to the 1800 from the Monitor are in the form shown below. This is similar to the form of messages sent to the Scope, see fig. 12, except that the location byte contains flags.

2 bytes

<table>
<thead>
<tr>
<th>LENGTH</th>
<th>CODE</th>
<th>FLAGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE (if any)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LENGTH Length of message plus header in bytes
CODE Code number of message, (see below).
FLAGS Not yet used.

Currently there are only four messages, each of four bytes, sent to the 1800 computer. These messages have the code numbers shown below.

CODE REQUEST
1 Send data for the Analysis task
2 Send commands for the command task, if any are available.
3 Send comments etc for the Analysis tasks, if any are available.
4 Link is now open. No reply is expected to this message.
Monitor Request Table

All requests to the Monitor from the subtasks or the 1800 Link are placed in this table.

**MNCFREQ**  
* SFLAGS  
  * MSGCODE  
  ******************************************  
  * SPSWALC  
  AWC SRTNCODE  
  ******************************************  
  * SPSTRTAC  
  (POST ECB ADDRESS)  
  ******************************************  
  * SETRTAC  
  (SET BIT ADDRESS)  
  ******************************************  

**MNNAKREQ**  
* AFLAGS  
  * MSGCODE  
  ******************************************  
  * SPSGAODK  
  AWC AWTNCODE  
  ******************************************  
  * SPSTRTAC  
  (POST ECB ADDRESS)  
  ******************************************  
  * SETRTAC  
  (SET BIT ADDRESS)  
  ******************************************  

**MNLAKREQ**  
* LFLAGS  
  * LSMGCODE  
  ******************************************  
  * LSGAOCK  
  QR LMTNCODE  
  ******************************************  
  * LPSRTAC  
  (POST ECB ADDRESS)  
  ******************************************  
  * LSETRTAC  
  (SET BIT ADDRESS)  
  ******************************************  

**MNCRFEK**  
* CFLAGS  
  * CMSGCODE  
  ******************************************  
  * CMSGAODK  
  QR CTNICODE  
  ******************************************  
  * CSPANSTAO  
  (POST ECB ADDRESS)  
  ******************************************  
  * CSETRTAC  
  (SET BIT ADDRESS)  
  ******************************************

**FIG 3a.**
**MONITOR REQUEST TABLE**

Explanations of the flags and names:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFLAGS</td>
<td></td>
</tr>
<tr>
<td>1-7</td>
<td>1 signifies request is active, *I means no request</td>
</tr>
<tr>
<td></td>
<td>used by the program whose code number is in moncode, this must be zero when initialized</td>
</tr>
<tr>
<td>8</td>
<td>1 signifies an ecr to be posted on return, if psrtag is zero when this bit is on, error</td>
</tr>
<tr>
<td></td>
<td>condition results</td>
</tr>
<tr>
<td>9</td>
<td>1 signifies a bit to be flagged when request completed, if psrtag is zero when this flag is</td>
</tr>
<tr>
<td></td>
<td>in an error condition will result</td>
</tr>
<tr>
<td>1-15</td>
<td>not used</td>
</tr>
<tr>
<td>AFLAGS</td>
<td></td>
</tr>
<tr>
<td>1-7</td>
<td>1 indicates a request is active, *I means no request</td>
</tr>
<tr>
<td></td>
<td>used by the program whose code number is in moncode, this must be zero when initialized</td>
</tr>
<tr>
<td>8</td>
<td>1 means an ecr to be posted when request satisfied</td>
</tr>
<tr>
<td>9</td>
<td>1 means bit to be flagged on return, first bit of psrtag is set to 1 when this flag is on</td>
</tr>
<tr>
<td></td>
<td>error condition results</td>
</tr>
<tr>
<td>1-15</td>
<td>not used</td>
</tr>
<tr>
<td>LFLAGS</td>
<td></td>
</tr>
<tr>
<td>1-7</td>
<td>1 indicates a request is active, zero means no request</td>
</tr>
<tr>
<td></td>
<td>used by the program whose code number is in moncode, this must be zero when initialized</td>
</tr>
<tr>
<td>8-15</td>
<td>not used</td>
</tr>
<tr>
<td>CFLAGS</td>
<td></td>
</tr>
<tr>
<td>1-7</td>
<td>1 signifies request is active, *I means no request</td>
</tr>
<tr>
<td></td>
<td>used by the program whose code number is in moncode, this must be zero when initialized</td>
</tr>
<tr>
<td>8</td>
<td>1 signifies an ecr to be posted on return, psrtag must not be zero when this bit is on</td>
</tr>
<tr>
<td>9</td>
<td>1 means bit to be flagged on return, first bit of this address is set to 1</td>
</tr>
<tr>
<td>1-15</td>
<td>not used</td>
</tr>
</tbody>
</table>

**FIG 28.**
Requests from the Analysis program to the Monitor are placed in this table.

<table>
<thead>
<tr>
<th>ANAL_REQ</th>
<th>AFLAGS</th>
<th>AMSGCODE</th>
<th>AMSGOAD</th>
<th>ARINGCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### AFLAGS BIT

1-7 1 Indicates a request is active, 0 means no request

1-7 Used by the program whose code number is in

MSGCORE. This must be zero when initialzed.

9 1 means an ECR to be posted when request satisfied

If PSTRTAD zero when this bit ON, error results

This bit to be flagged on return. First bit

of ABITRTAD is set to 1 if ABITRTAD zero when

this bit ON then ERRPR condition results

10-13 Not used

### MSGCORE

A code is placed here to indicate which subroutines is

To be branched to, in order to satisfy this request

### AMSGOAD

The address of any message or data connected with this

Request, otherwise zero

### ARINGCODE

Return code after the request has been satisfied

### PRTRTAD

Address of an ECR to be posted when this request satisfied

### ABITRTAD

Address of a byte the first bit of which is set to 1,

When this request finished (or zero)

**FIG 6**
Requests for the analysis program from any of the other programs in the region or from the Monitor are placed in this table.

**ANLREFQ** #ANLNKFLG #ADDRESS OF LINK MESSAGE

**ANSCPFREQ** #ANSCPFILG #ADDRESS OF SCOPE MESSAGE

**ANMENREQ** #ANMENFLG #ADDRESS OF MONITOR MESSAGE

**ANSCMREQ** #ANSCMFLG #ADDRESS OF SCUP COMMAND MSG

BYTES 0 1 2 3

**ANLNKFLG BIT**

0 A 1 indicates a request active, zero indicates no active request.
1 l indicates a code number in the last half of the word
0 indicates there is a message address in the last three bytes of the word
2-7 Not used

**SCAMPFLG BIT**

0 A 1 indicates a request active, zero indicates no request active
1 l indicates a code number in the last half of this word
0 indicates a message address in last three bytes.
2-7 Not used

**SCMONFLG BIT**

0 A 1 indicates a request active, zero indicates no request
1 l indicates a code number in the last half of the word
0 indicates that the last three bytes contain a message address.
2-7 Not used

**SCSMFLG** Same as above.

FIG 7
SCORPE REQUEST TABLE

All requests for the Scope task are placed here by the Monitor.

SCORETBL 4F  SCOPE REQUESTS ARE PLACED HERE

SCORNEG  **********

SCANNEG  **************

SCANFLAG  **************

SCMNEG  **************

SCCMFLAG  **************

SCSEPREG  **************

BYTES  1  2  3

SCLANKFLG BIT

11 INDICATES A REQUEST ACTIVE
10 INDICATES NO REQUEST
1  1 INDICATES A CODE IN THE LAST HALF OF WORD
0  1 INDICATES AN ADDRESS OF MESSAGE IN THIS WORD

2-7 ACT USED

SCANFLG BIT

11 INDICATES A REQUEST ACTIVE
10 INDICATES NO REQUEST
1  1 INDICATES A CODE IN THE LAST HALF OF WORD
0  1 INDICATES AN ADDRESS OF MESSAGE IN THIS WORD

2-7 ACT USED

SCMCLKFLG BIT

11 INDICATES A REQUEST ACTIVE
10 INDICATES NO REQUEST
1  1 INDICATES A CODE IN THE LAST HALF OF WORD
0  1 INDICATES AN ADDRESS OF MESSAGE IN THIS WORD

2-7 ACT USED

SCCMFLAG BIT

11 INDICATES A REQUEST ACTIVE
10 INDICATES NO REQUEST
1  1 INDICATES A CODE IN THE LAST HALF OF WORD
0  1 INDICATES AN ADDRESS OF MESSAGE IN THIS WORD

2-7 ACT USED

FIG 8.
Abnormal Termination

If the O.S. system 360 discovers an error in the Monitor task or the Monitor discovers an error in itself, the Monitor task will be abnormally terminated. When this happens all subtasks are also abnormally terminated and both the 1800 Link and the 2250 Scope are closed. In effect a disaster has occurred, at least as far as Midas.3 is concerned. It is therefore an essential part of the Monitor design that it should be extremely reliable. However, errors do occur in even the most carefully debugged programs.

For this reason this section is included as an attempt to aid the unfortunate programmer in his or her search for the offending piece of code.

If the system discovers that the Monitor has an error then the termination code will be a system number and the dump has to be solved with the aid of "Messages and Completion Codes" and faith, [see your friendly Customer Engineer if the error persists].

If the Monitor discovers the problem itself then there is more hope. The following table contains a list of Monitor Termination Codes and the control sections in which they occur.

<table>
<thead>
<tr>
<th>CODE</th>
<th>CBRT</th>
<th>Description of CBRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3333</td>
<td>MEIT</td>
<td>Main Monitor Program</td>
</tr>
<tr>
<td>1111</td>
<td>ATTHAND</td>
<td>1800 Link Attention Handler</td>
</tr>
<tr>
<td>2222</td>
<td>ABENDAGE</td>
<td>1800 Link BDF Appendage</td>
</tr>
<tr>
<td>4444</td>
<td>ANLEEXIT</td>
<td>Analysis Task Exit Routine</td>
</tr>
<tr>
<td>5555</td>
<td>SOMEXIT</td>
<td>Command Task Exit Routine</td>
</tr>
</tbody>
</table>

User code 3333 will occur if an error is detected anywhere in the main Monitor code. Code 1111 occurs only in the attention handler for the 1800 Link. Code 2222 occurs only in the 1800 Link BDF Appendage exit. Code 4444 occurs in the Analysis exit routine. This routine obtains control when the Analysis task terminates abnormally. Currently there is no attempt to re-attach the task; the Monitor just abends. Code 5555 occurs in the Command task exit routine. This routine obtains control when the Command task terminates abnormally; the Monitor makes no attempt to recover but merely abends with the code shown.

For all of the above termination codes the exact address where the error was discovered is saved in the Monitor Work Area location called
ERORCODE before the ABEND was issued. This can easily be discovered in the dump as it is preceded by the characters "MONITOR ABEND ERROR ADDRESS". The address given in ERORCODE is the address of the code immediately following the point where the error was discovered. By using the linkage editor map to determine what part of the code this is, the source of the problem will become clear by reference to the comments in the code at that point.

In case of either the Analysis or Command tasks abnormally terminating, the Monitor makes no attempt to recover but terminates with a dump. The problem must then be solved by examining the subtask itself and solving the problem which caused it to terminate.

In case of the Scope task terminating, the Scope Exit Routine sets a flag to indicate the Scope task has abended and returns without abending the Monitor. When the Monitor enters the Main Loop it will test that flag and discover that the Scope has abended. Control will then pass to the Scope Recovery Routine which will attempt to clear all the necessary buffers, re-attach the Scope task and inform the other tasks in the region.
INITIALISATION

ENTRY POINT MONIT
CHECK MONIT
INPUT PARAMETERS none
OUTPUT PARAMETERS none
RETURN CODE none
LOWER LEVEL ROUTINES none
EXIT MAINLOOP

FUNCTIONS

This routine will initialise some flags, issue the IDENTIFY instructions, if necessary for the subtasks, and attach the Command task. Control then passes to the Main Loop.
**MAIN LOOP**

**ENTRY POINT**  
MOMAINLP

**CHECK**  
MONIT

**INPUT PARAMETERS**  
none

**OUTPUT PARAMETERS**  
none

**RETURN CODE**  
none

**LOWER LEVEL ROUTINES**  
SFPARED  
COMIERST  
SCOPEAET  
ANALYRST  
LINKRST  
DATACHECK

**EXIT**  
none

**FUNCTIONS**

This routine cycles a loop checking for requests and if there are none it becomes inactive after setting a timer. The first thing this routine does on becoming active is to clear its own ECB before entering the loop again. The order of events inside the loop is as follows: first check if the Scope task has abended; if yes, branch to the Scope recovery program; next check if there is a Command request, Scope request, Analysis request or unsolicited link request in turn. If there is, then it branches to the respective routine to check that request; then to a lower level routine to service it, finally returning by the same path to the Main Loop. Before setting the timer, a test is made to determine if the 1800 Link is open, and if it is, whether the data buffers are full, and whether Command, comments, ENV's, etc. should be requested from the 1800 computer. Finally a timer (ten seconds current value) is set and the Monitor enters a wait state.
Monitor Main Loop

- **Scope** opened?
  - Yes: Branch to SPECTRUM for recovery
  - No: Clear ECB

- If there a request for Command?
  - Yes: Branch to COMMDIST to check it
  - No: WAIT

- If there a request for Scope?
  - Yes: Branch to SPECMDIST to check it
  - No: Branch to SET_COMMAND to check it

- If there a request for Analysis?
  - Yes: Branch to ANALYSIS to check it
  - No: Set Timer

- Was there a request from Link?
  - Yes: Branch to LINKDIST to check it
  - No: If the link open?
    - Yes: Branch to DATACHECK to see if buffers are full
    - No: Branch to GETCOMMAND to get commands etc. (if any)

- Branch to GETCOMMAND to get commands (if any)
ENTRY POINT
COMMREQST, SCOPOPRST, ANLRST

CSEGCT
MONIT

INPUT PARAMETERS
none

OUTPUT PARAMETERS
none

RETURN CODE
none

LOWER LEVEL ROUTINES
Command Scope Analysis
CMECBAD SCHKBAD ANLRBAD
SCATCHEN SCATCHEN STOOCSF LIKADAR
SCATCHEN
HEALINK
OPENLINK
SCFCOMSG
SCFCHCH (dummy)
CLOSELINK (dummy)
DISABLED (dummy)
ATTACHNL
COMASEND
DIACHNL

EXIT
To Main Loop

FUNCTIONS

The function of these routines is to check that a request has a valid code number and branch to the routine corresponding to that code number. On return this routine checks whether the request was completed, and if it was not (i.e. return code was hex FF=THYLAKE) this routine simply returns to the Main Loop. If the request was completed (i.e. any return code except hex FF) then the Request buffer is cleared, and if any flag is to be set or ECB posted to signal the request terminated, it is done at this stage. The routine finally returns to the Main Loop.
Check Subtask Requests

- Is the task attached?
  - Yes: Abend code 3333
  - No: Is the code number valid?
    - Yes: Branch to routine corresponding to the code number
    - No: Abend code 3333

- In return code, 'Y'?
  - Yes: Return to Main Loop
  - No: Save return code in Monitor task request buffer

- Zero the request flags buffer after saving flags

- Is a bit set?
  - Yes: Get the address and set the bit
  - No: Get the ECB address and post the ECB

- Is the ECB to be posted at regeneration?
  - Yes: Return to Main Loop
  - No:
DATA CHECK ROUTINE

ENTRY POINT DATACHEK
CHECK MONIT
INPUT PARAMETERS none
OUTPUT PARAMETERS none
RETURN CODE none
LOWER LEVEL ROUTINES SMILERS
EXIT To Main Loop

FUNCTIONS

This routine checks the first data buffer to see if it is empty. Provided that a message has not already been sent, a request is made to the 1500 for more data and control is returned to the Main Loop. If the message has already been sent, then a check is made to see whether it has arrived. If it has not, then control is again returned to the Main Loop, but if it has arrived then the buffer flag is switched on to show the buffer is full.

When the first buffer is full, either if it is found full originally or the data has arrived and the buffer is just marked as full, a check is made on the second buffer if there is one. The same procedure for requesting data for the second buffer, if necessary, as for the first is then followed.
DATACHEK
Check Data Buffers Routine

Get first data buffer address

Is the buffer empty?

Branch to SNDMSG to send msg to 1800 for data

No

Yes

Has data been requested?

Get the second buffer address

No

Yes

Has data arrived?

Set flags to show new status of link

No

Return to Main Loop

Set flag to show that data was requested

Return to Main Loop

No

Yes

Was data length transferred?

Abend code 3333

No

This is first or second buffer

Return to Main Loop

Yes

Are there two data buffers?

Return to Main Loop

Yes

Check return code

Zero
GET COMMAND ROUTINE

ENTRY POINT
GETCOMMAND

ERROR
MONIT

INPUT PARAMETERS
none

OUTPUT PARAMETERS
none

RETURN CODE
none

LOWER LEVEL ROUTINES
SLOURCS

EXIT
To Main Loop

FUNCTIONS
The purpose of this routine is to request and ensure the smooth flow of commands from the 1800 computer to the Command task in the M91. Commands from the 1800 computer when they arrive are placed in a buffer called LINKCOMBG. This is a 256 byte buffer consisting of a four byte header and 252 bytes for the Command itself. When commands arrive from the 1800, an attempt is made to inform the Command task that there is a command in that buffer. If the Command task has not serviced the earlier command then the routine returns to the Main Loop and attempts to send the new command at a later time.
Get Command from 1800

Is LNMCMOS not empty?

Yes: Set up parms to send msg to 1800 requesting new command.

No: Branch to SMNLSMS to send message to 1800.

Has a command already been requested?

Yes: Command arrived?

No: Set flags off and return to Main Loop.

Yes: Set flag to show command requested.

Return to Main Loop.

Is this a no data msg?

Yes: Set up CMLNMDQ buffer to show there is a msg from 1800.

No: Return to Main Loop.

Is command req. biff. free?

Yes: Post Command task's EC5 if there is one.

No: Check return code.

0: Set flag to show command requested.

Return to Main Loop.

other: Return to Main Loop.
FUNCTIONS

This program makes an attempt to re-attach the Scope task after it has abnormally terminated. The first thing it does is to close the scope, freeing those areas which have been 'getmained' by the Scope task, and detach the Scope task. The address of the DCB and the areas to freemain were passed back to the Monitor by the Scope when it was last attached. Both the Analysis task and the Command task must then be informed that the Scope has abnormally terminated before the Monitor can do anything else. Finally the Scope's request buffers are cleared, the scope is attached again, and the interrupts enabled, if that was their state before the Scope task abended. This routine then exits to the Monitor Main Loop.
**ENTRY POINT**

SNILKIND

**OBJECT**

MONIT

**INPUT PARAMETERS**

R1 points to a parameter address
R1# contains the return address

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Bit(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARAMADD</td>
<td>0, 1</td>
<td>Address of the parameters passed</td>
</tr>
<tr>
<td>PRAMFLAG</td>
<td>2</td>
<td>1 means rpstad the address of a flag to be set. The last bit of the byte given by rpstad will be set to one when reply is obtained</td>
</tr>
<tr>
<td>WTMMSGLEN</td>
<td>0, 1</td>
<td>length of msg to be sent</td>
</tr>
<tr>
<td>WTMMSGAD</td>
<td>0, 1</td>
<td>address of msg to be sent</td>
</tr>
<tr>
<td>FTPRFLAG</td>
<td>3, 7</td>
<td>not used</td>
</tr>
</tbody>
</table>

R1

- PRAMFLAG 1
- PARAMADD 3

bytes 1 3

WTMMSGLEN

WTMMSGAD

FPSTRIP

bytes 1 3

EXPLIMET

DATADDR

DATLENAD
address to be flagged or posted when reply is obtained

length of data expected

address of data buffer

address where actual length of data received to be saved or zero if it is not required

OUTPUT PARAMETERS

none

RETURN CODE

R15 will contain one of the following return codes:

0 message was sent and reply (if any) is expected

4 link busy

LOWER LEVEL ROUTINES

none

EXIT

On register 14

FUNCTIONS

This routine checks whether the link is busy; if it is, it returns with code 4 in register 15. If the link is not busy, then a check is made on the input parameters to determine if a reply is expected. If it is, the Solicited Read buffer is prepared to indicate that fact. The Write buffer is then set up and the Write macro is issued. The routine then exits with a return code of zero.
ENDLINES
Send Link Message

Save Registers

Has a Read been issued? yes

Is a message expected? yes

Was unsolicited read issued? no

Was a Write issued? yes

Is reply expected for this msg? yes

Set up Solicited Read buf. to show reply expected.

Set up Write buf. of Link Activity Table.

Restore the regs. and return. RC=4

Restore the regs and return. RC=0

Issue Write
ATTENTION HANDLER

ENTRY POINT       ATTINHAND

CHECK            ATTINHAND

INPUT PARAMETERS  none

OUTPUT PARAMETERS none

RETURN CODE       none

LOWER LEVEL ROUTINES none

EXIT              On Register 14

FUNCTIONS
This routine checks whether the Link Activity Table indicates any READ or WRITE macros to have been issued without having terminated. It also checks that this attention occurred when a message from the 1800 is expected. If the table is in order and the message is expected, a READ is issued and this routine exits on Register 14.
ATTEND

Link Attention Handler

- Save regs and link
- Save areas

- Has solicited read been issued? yes no
  - Has a write been issued? yes no
  - Has an unsolicited read been issued? yes no
  - Is a msg. expected? yes no

- Issue Read
- Restore the registers and return

- Abend code
The E.O.F. Appendage Routine checks the completion of passing data across the link. The flags in the Link Activity Table indicate the type of activity in the link and inform the E.O.F. Appendage what to do about it.

A test is first made to determine whether a read or a write was issued. If a read was issued, i.e., this is the end of data transfer to the MOL from the 1900, the completion code is checked; if it is not 7F the Monitor abends. The residual byte count is checked to see that the data transferred is not more than the maximum expected. This routine then checks the Link Activity Table to determine if a flag is to be set or an ECB posted to signal the completion of the data transfer and perform whichever is necessary. Finally the Monitor Main ECB is posted and this routine exits.

The procedure is exactly the same at termination of a "Write" except that the length of data transferred must be exactly equal to that length issued with the write.
<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>MTIMEOUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>MTIMEOUT</td>
</tr>
<tr>
<td>INPUT PARAMETERS</td>
<td>none</td>
</tr>
<tr>
<td>OUTPUT PARAMETERS</td>
<td>none</td>
</tr>
<tr>
<td>RETURN CODE</td>
<td>none</td>
</tr>
<tr>
<td>LOWER LEVEL Routines</td>
<td>none</td>
</tr>
<tr>
<td>EXIT</td>
<td>On Register 14</td>
</tr>
</tbody>
</table>

**FUNCTION**
Post Monitor main ECB.
**ANALYSIS EXIT ROUTINE**

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>ANLSEXIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>ANLSEXIT</td>
</tr>
<tr>
<td>INPUT PARAMETERS</td>
<td>none</td>
</tr>
<tr>
<td>OUTPUT PARAMETERS</td>
<td>none</td>
</tr>
<tr>
<td>RETURN CODE</td>
<td>none</td>
</tr>
<tr>
<td>LOWER LEVEL ROUTINES</td>
<td>none</td>
</tr>
<tr>
<td>EXIT</td>
<td>On Register 14.</td>
</tr>
</tbody>
</table>

**FUNCTION**

Abend Job with User Code 4444. This routine gets executed when the Analysis task terminates.
SCOPE EXIT ROUTINE

ENTRY POINT
SCOPEEXIT

EXIT
On Register 14

FUNCTIONS
This routine gets executed when the Scope task terminates. It sets a flag in the Scope Program Table to indicate that the Scope has terminated. (That flag gets checked in the Monitor Main Loop). The program then exits.
<table>
<thead>
<tr>
<th><strong>FUNCTIONS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Abend job with user code 5555. This routine gets executed when the Command task terminates.</td>
</tr>
</tbody>
</table>
NOTE: Scope Request ECB

ENTRY POINT
SCEBCBAD

OBJECT
MONIT

INPUT PARAMETERS
The message address in the Monitor Scope request table points to a parameter list.

MNSCPREQ

SYFLAGS  SCOCE  SMOSGAD
↑            SCEBCBAD
            ICBCAD
            SUCBAD
            SUCHLY
            SMBCBAD
            SMCBLN
            SMCBLN

SYFLAGS  SCOCE  SMOSGAD  SCEBCBAD  ICBCAD  SUCBAD  SUCHLY  SMBCBAD  SMCBLN

Request flags see fig (5k)
Must be value 1
Address of parameter list
Header of parameter list
bytes
length of list including header in bytes
not used.
The address of the Scope ECB
The address of the Scope DCB
Address of SUCB table
The address of the SUCB table
Address of the SMCB table
Length of the SMCB table

OUTPUT PARAMETERS
none

RETURN CODE
A return code is placed in Register 1
0  The parameters were saved.
FUNCTIONS

This routine saves the ECB address (second parameter) in the Scope Program Flags table. If the header indicates that the length of the message is greater than 8, then the remaining five parameters are saved in the Scope Program Flags table. The routine then exits. This ECB address which is passed to the Monitor is the ECB on which the Scope task waits when it has nothing to do. When the Monitor has a request for the Scope task this ECB gets posted causing the Scope task to become active immediately and service that request.
PASS SCOPE MESSAGE TO COMMAND TASK

ENTRY POINT
STCOOMEG

CSECT
MONIT

INPUT PARAMETERS
The message address in the Monitor Scope request table points to the message to be passed to the command task.

<table>
<thead>
<tr>
<th>MYSCFREQ</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>yte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SFLAGS Request flags see fig (9b)
SCORE Must be value 2
SMEGAIDR Message address
SYSTAD bytes 0-1 length of message including header (value 78)
SHRTAD not used
MESSAGE User message

OUTPUT PARAMETERS
The Command Request buffer (CMSCFREQ) is set up to show this request from the scope.

<table>
<thead>
<tr>
<th>CMSCFREQ</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>yte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FLAGS flags see fig (7) for Analysis request
Bit 0 set to 1 to indicate a request.
1-7 not used
A(MEGADR) Address of MEGRDR
RETURN CODE

In Register 1
0 Message was sent
PF Message could not yet be sent

EXIT

On Register 14

FUNCTIONS

The function of this routine is to inform the Command task that there is a message from the Scope task. It does this by setting up the Command Scope request buffer as shown above. It then posts the Command modules ECB to inform it that there is a request, and exits.
NOTE ANALYSIS REQUEST ECB

ENTRY POINT
ANLSCBAD

CSECT
MONIT

INPUT PARAMETERS
The message address in the Monitor Scope request table points to the Analysis task's ECB

<table>
<thead>
<tr>
<th>AFLAGS</th>
<th>ACODE</th>
<th>ECB</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMSGADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AHOSTADR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARHOSTADR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AFLAGS Request flags see Fig (7)
ACODE Must contain a value of 1
AMSGADR Contains address of Analysis Task's ECB

OUTPUT PARAMETERS
none

RETURN CODE
In Register 1
0 The request was completed.

EXIT
On Register 1

FUNCTIONS
This routine stores the address of the Analysis program's ECB in
AMHOSTADR which is in the Program Flags Buffer for the Analysis program.
It then exits with a return code of zero.

This ECB is the ECB on which the analysis program will wait when
it has nothing to do. When the Monitor has a request for the Analysis
program, this ECB will be posted causing the Analysis program to
become active.
ANALYSIS DATA REQUEST

ENTRY POINT

ENTRY

INPUT PARAMETERS

This routine is entered as a result of the code number in the MNASLSEQ table and uses some information from this table.

MNASLSEQ

OUTPUT PARAMETERS

The address of the full buffer is placed in BUFFERAD as shown above.

RETURN CODE

In Register $1
0 The request was completed
FF There are not yet any full buffers

EXIT

On Register $14
This routine searches the data buffers to see if there are any full ones. If there is a full one it places the address in BUFFBAD and returns with a RC of zero. If there are no full buffers, the routine returns with a code of FF to try later.
SEND ANALYSIS MESSAGE TO SCOPE

ENTRY POINT
SCANLMESG

OBJECT
MONIT

INPUT PARAMETERS
This routine uses the information in the Monitor Analysis request buffer (MNANLMESG).

<table>
<thead>
<tr>
<th>MNANLMESG</th>
<th>MESSGR</th>
<th>LENGTH</th>
<th>CODE</th>
<th>LOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFLGESS</td>
<td>ACOSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASEGGER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AFFTNTAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AERTKAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MESSAGE</td>
<td>(74 bytes)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AFLGESS: BIT (2 bytes)

0 1 indicates an active request,
0 means no request used by SCANLMESG
and valid only when bit 0 is on
Bit 1 1 indicates a msg sent to
scope but not yet displayed.
When scope has cleared
request buffer message has
been displayed.

2-7 not used

8 1 signifies an ECB to be posted on
return. If AFFTNTAD is zero when
this bit is on then an error
condition results

9 1 signifies a bit to be flagged
when request completed. If
AERTKAD is zero when this flag is
on an error condition will result.

10-15 not used
ACODE: Must have a value of 3
AMESSAGER: Points to the message
MESSAGE: Message header
LENGTH: Length of message plus header in bytes, currently must have a value of 78
CODE: Value of l recognized by the Scope program as a request to display the message
LOCN: Values of l through 6 indicate to the Scope where the message is to be displayed. Currently a value of 1 only accepted, in the Analysis message request
MESSAGE: The message to be displayed

OUTPUT PARAMETERS
The Scope Analysis request buffer (SCANLSEG) is set up to show the Scope task that there is a request from the Analysis task

SCANLSEG

FLAGS A (MESSAGE) LENGTH CODE LOCN

MESSAGE

FLAGS
Bit 0 set to 1 to indicate an active request
1-7 not used

RETURN CODE
In Register 1
0 Indicates that the message was displayed and request finished
FF Indicates that either SCANLSEG was already full or that the request was set up as shown and has not been displayed
FUNCTIONS

This routine attempts to set up the SCANLSEQ buffer as shown above.
If the buffer is already busy it returns with code FP to try later.
If the buffer is not busy it sets up the SCANLSEQ buffer, and sets a
flag in AFLAGS to indicate that the message was sent. The routine then
exits with a return code of FP. At a later time this routine will again
be entered and if SCANLSEQ has been cleared, i.e. the message has been
displayed, this routine exits with return code of zero indicating that
the request has been finished and MOANLSEQ can be cleared.
ENTRY POINT

CMCSBACD

CSECT

MONIT

INPUT PARAMETERS

The Monitor Command Request table contains the information this routine uses.

MISCMRQ

<table>
<thead>
<tr>
<th>CF/LS</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMCSAVER</td>
<td></td>
</tr>
<tr>
<td>CSTRRAD</td>
<td></td>
</tr>
<tr>
<td>CRTRRAD</td>
<td></td>
</tr>
</tbody>
</table>

CF/LS

See fig (5b)

CODE

Must have value 1 for this routine to be executed

CMCSAVER

Must contain the address of the Command task's ECB.

OUTPUT PARAMETERS

none

RETURN CODE

In Register 1
0 The ECB address has been saved.

EXIT

On Register 14

FUNCTION

This routine saves the address of the Command task’s ECB in the Command task’s program flags buffer. If there is a value there already it is overwritten. The routine then exits with a return code of zero.
ATTACH SCOPE WITHOUT INTERRUPTS

ENTRY POINT

SCATCHR

OSERR

MONIT

INPUT PARAMETERS

The code number in MNSCOPE causes this routine to be executed; not other parameters in that request buffer are needed but it is described here for the sake of completeness.

MNSCOPE

<table>
<thead>
<tr>
<th>CFLAGS</th>
<th>OCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSGAER</td>
<td>CMTRTAD</td>
</tr>
</tbody>
</table>

CFLAGS | See fig (5b) |
OCODE | Must have a value of 2 |
CMSGAER | Not used. |
CMTRTAD

OUTPUT PARAMETERS

none

RETURN CODE

On return R1 will contain one of the following codes:

0 - scope was attached
1 - scope is already attached with interrupts not enabled
2 - scope is already attached with interrupts enabled.

EXIT

On Register 14
FUNCTIONS

This routine checks whether the Scope task is already attached. If it isn't, it attaches it and sets up the flags in the Scope program flags to indicate this fact. The routine then returns with a return code of zero. If the Scope task is already attached then a return code of either 1 or 2 is placed in Register 1 and the routine exits.
ATTACH SCOPE WITH INTERRUPTS ENABLED

ENTRY POINT

SCATCHEN

GORT

MONIT

INPUT PARAMETERS

The code number in MSBOM causes this routine to be executed; no other parameters in that request buffer are needed by that routine but it is described here for the sake of completeness.

<table>
<thead>
<tr>
<th>MSBOMENQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>CYLASS</td>
</tr>
<tr>
<td>OMESDAER</td>
</tr>
<tr>
<td>CPSTREAD</td>
</tr>
<tr>
<td>CPRTREAD</td>
</tr>
</tbody>
</table>

CYLASS See fig 73
CODE Must have a value of 3
OMESDAER Not used.

OUTPUT PARAMETERS

The Scope request buffer is set up with a request for the interrupts to be enabled and the cursor inserted after the Scope task has been attached. The SCBOMENQ buffer is set up as shown.

<table>
<thead>
<tr>
<th>SCBOMENQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLAGS</td>
</tr>
<tr>
<td>= 1</td>
</tr>
</tbody>
</table>
FLAGs

bit 0 Set to 1 to indicate an active request
1 Set to 1 to indicate last three bytes contain a code number, not a message address
2-7 Not used
CODE Set to value 3 to indicate to the Scope task that the interrupts are to be enabled

RETURN CODE

On return RI contains one of the following codes
0 - Scope attached and interrupts enabled
1 - Scope is already attached, interrupts already enabled
2 - Scope is already attached, interrupts were enabled
All F's - Scope already attached not yet able to enable interrupts

EOT

On Register 14

FUNCTIONS

The purpose of this routine is to ensure that the Scope is attached and the interrupts are enabled. It checks the Scope program flags to determine the status of the Scope. If the Scope is not attached, this routine attaches it and sets up the request buffer for the Scope as shown above. This means that as soon as the Scope task becomes active it will not the request to enable the interrupts. When the request has been set up for the Scope task, the Scope program flags are adjusted and this routine exits with a return code of zero.

If the Scope is already attached then, if necessary, the Scope is requested to enable the interrupts and the routine returns with one of the return codes shown above.
ENTRY POINT

ENELINTS

ENTRY POINT

MONIT

INPUT PARAMETERS

The code number in XMSOMSRQ (value 4) causes this routine to be entered; no other parameters in that request buffer are needed and CMSWADER is not used.

OUTPUT PARAMETERS

The Scope request buffer is set up with a request for the interrupts to be enabled and the cursor inserted. The Scope request buffer SCSONSRQ is set up as shown.

SCSONSRQ

<table>
<thead>
<tr>
<th>FLAGS</th>
<th>CODE</th>
</tr>
</thead>
</table>

FLAGS

bit 0 Set to 1 to indicate an active request
1 Set to 1 to indicate that the last three bytes contain a code number, not a message address
2-7 not used.
CODE Set to value 3 to request the Scope task to enable the interrupts.

RETURN CODE

On return RI will contain one of the following values:
0 - Interrupts were enabled
1 - Interrupts were already enabled
All F's not yet able to enable as Scope is processing earlier request.
FUNCTION

This routine has to set up a request to the Scope task to enable the interrupts, if the interrupts were not already enabled. If the Scope request buffer is still full from an earlier request then a return code of FP is returned so that an attempt is made at a later time.
ENTRY POINT
OPENLINK

CONTROL
MONIT

INPUT PARAMETERS
The Monitor Command request buffer which causes this routine to be executed is not used by this routine but is described below for the sake of completeness.

MISCOMERG

<table>
<thead>
<tr>
<th>CYLAGE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMLOGABER</td>
<td></td>
</tr>
<tr>
<td>CMSTRTAB</td>
<td></td>
</tr>
<tr>
<td>CMFINTAB</td>
<td></td>
</tr>
</tbody>
</table>

CYLAGE See fig (3b)
CODE Must have value 5 for this routine to be executed.
CMLOGABER Not used.

OUTPUT PARAMETERS
None

RETURN CODE
In Register 1
0 The Link was opened.

EXIT
On Register 14

FUNCTIONS
If the link is already open, this routine causes an abend.
A conditional getmain is issued to obtain space for the data buffer for the Analysis program. If this is successful a second Getmain is issued for a second data buffer. The word DATARUN in the Monitor work area
is set up to contain the number of data buffers while PGETBUFAD and SONRBUFAD will contain the addresses. Finally the link is opened and the routine exits.
SEND MESSAGE FROM COMMAND TASK TO SCOPE

ENTRY POINT

SCPCOMSG

CHECT

MONIT

INPUT PARAMETERS

The Monitor Command request buffer contains information which this routine needs.

MSGCOMSG

<table>
<thead>
<tr>
<th>CYFLAG</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMTREAD</td>
<td>CMTREAD</td>
</tr>
<tr>
<td>CMTREAD</td>
<td>CMTREAD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CYFLAG</th>
<th>BIT (2 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 indicates an active request, 0 means no request</td>
</tr>
<tr>
<td>1-7</td>
<td>Used by SCPCOMSG and valid only when bit 0 is on</td>
</tr>
<tr>
<td></td>
<td>Bit 1 indicates a msg sent to scope but not yet displayed. When scope has cleared request buffer, message has been displayed</td>
</tr>
<tr>
<td>2-7</td>
<td>not used</td>
</tr>
<tr>
<td>8</td>
<td>1 signifies an ECB to be posted on return. If ETRHD is zero when this bit is on then an error condition results</td>
</tr>
<tr>
<td>9</td>
<td>1 signifies a bit to be flagged when request completed. If ETRHD is zero when this flag is on an error condition will result</td>
</tr>
<tr>
<td>10-15</td>
<td>not used</td>
</tr>
</tbody>
</table>

LENGTH | CODE | LOCK
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE</td>
<td>(74 bytes)</td>
<td></td>
</tr>
</tbody>
</table>
CODE Must have a value of 6 for this routine to be executed

MESSAGE Contains the address of the message to be passed to the Scope

MESSAGE Message header for the message

LENGTH Length of the message plus header in bytes; currently this must be 73

CODE The code number which determines what the Scope does with this message. See Scope code for details

LOCN Value 1 through 4 determines which area of the Scope the message is to be displayed, if any

MESSAGE The message to be displayed, if any

OUTPUT PARAMETERS

The Scope request buffer is set up to indicate that there is a request from the Command task

FLAGS

<table>
<thead>
<tr>
<th>Bit</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Must be 1 to indicate a request active</td>
</tr>
<tr>
<td>1</td>
<td>Must be 0 to indicate last three bytes of this word contain an address not a code number</td>
</tr>
<tr>
<td>2-7</td>
<td>Not used</td>
</tr>
</tbody>
</table>

A(MESSAGE) The address of the Message header is set up in these three bytes.
RETURN CODE

On return Register 1 contains one of the following codes:

0 - Message was passed to the Scope
1 - Scope program is not attached
All F's - Scope had not processed earlier request

EXIT
On Register 1

FUNCTION

This routine checks if SCOMREE buffer is empty. If it is not, it returns with a code of FF to try again later. If the buffer is empty then it sets up the request for the Scope task as shown above and returns with completion code FF. At a later time when the SCOMREE buffer is again empty, which signifies the message has been accepted, this routine returns with a completion code of zero signifying that the Command tasks request has been terminated.
ATTACH ANALYSIS TASK

ENTRY POINT
ATTACHANL

INPUT PARAMETERS
The Monitor Command request buffer is set up
as shown below although it is not used by this
routine

MISOMREQ
CFLAGS CODE
CMSEGAIER
CINITTHAD
CINITTHAD

CFLAGS See fig (5b)
CODE Must have a value 10 for this routine
to be executed
CMSEGAIER Not used

OUTPUT PARAMETERS
none

RETURN CODE
On return R1 will contain one of the following
codes
0 - The Analysis program was attached
1 - The Analysis program was already attached

EXIT
On Register 14

FUNCTION
This routine first checks if the Analysis is already attached.
If it is it returns with return code of 1 in register 1. If not, then
an attach macro for that Analysis task is issued and the program flags for the Analysis task altered accordingly. This routine then checks if there are any data buffers. If there are, then these are flagged to show that they are empty and the routine exits with a return code of zero.
ENTRY POINT
COMABEND

CONTROL
AUDIT

INPUT PARAMETERS
The Monitor Command request buffer is set up as shown before although not used by this routine.

MONITOR

<table>
<thead>
<tr>
<th>CYLAGE</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSAGDR</td>
<td></td>
</tr>
<tr>
<td>CREATHEAD</td>
<td></td>
</tr>
<tr>
<td>CRTREAD</td>
<td></td>
</tr>
</tbody>
</table>

CYLAGE See fig (5b)
CODE Must have value 11 for this routine to be executed
CMSAGDR Not used.

OUTPUT PARAMETERS
none

RETURN CODE
none

EXIT
none

FUNCTION
This routine issues an abend with a user termination code of 5555 causing the job to terminate.
DUMMY ROUTINES

SCRATCH    Detach the Scope Task
CLOSELINK  Close 1300 Link
DISABLE    Disable the Interrupts on the Scope
DETACH     Detach the Analysis Task

The routines named above are available and included in the code; however they have not been debugged so no description of them will be given.
INTRODUCTION

CONCEPTS- General

Background for Design
Design Considerations
Functions of the Program
Future Expansion

STRUCTURE

General Layout of Subroutines
User Response
Screen Organisation
Timing
Communication
Abnormal Termination

DETAILED DESCRIPTIONS AND FLOWCHARTS

Main Loop Routines
  Initialisation
  Main Loop
  Check Command Request
  Check Analysis Request
  Service Scope Users Request

Analysis requests to Scope
  Display message from the Analysis

Command requests to Scope
  Display message from Command
  Clear Display Area
  Display Command Message with reply expected
  Allow Interrupts to be Enabled
  Accept Commands for the Command task
  Display Histogram
  Disable the Interrupts

General Routines
  General Message Display Routine
  Integer to Decimal Conversion
  Floating point to Integer Conversion
  Scope Attention Handler
  Histogram Scaling Routine
  Timer Routines
  SAAASI Error Handler

page
70
72
72
73
73
75
76
77
77
77
80
81
83
85
86
88
90
92
93
95
97
99
101
104
106
110
112
114
117
120
124
INTRODUCTION

This Scope program was designed to operate as a subtask of the Monitor in the Midas 3 environment. Midas 3 is a set of programs designed to be resident in a 128K region of core in the IBM 3031 computer throughout the duration of a spark chamber experiment. The purpose of Midas 3 is briefly defined as follows:

To obtain a significant sample of the data being collected on-line from a wire spark chamber experiment on the IBM 3031 computer; perform a complete and immediate analysis of that sample on the 3031; to give the experimenter an instant display of the results of that analysis and provide him with a means of fast interaction through the use of a 2250 scope display situated in the experimental area.

This document will assume that the Scope program is part of the Midas 3 system, and will attempt to explain its functions in that light, even though given the same interface that is established with the Monitor it could function equally well as a subtask of any other program. For details concerning the Midas 3 system and the relationship of the Monitor to its subtasks please refer to the documentation of the Monitor.

When the plans were made to write the Scope routine it appeared that it would be required to be working in a very short space of time. A decision was therefore made to use the Fortran version of the 2250 Scope Package written at SLAC by R. Beach. These are a set of assembler language subroutines originally designed to allow the Fortran programmer to write programs for the 2250 Scope display. It now appears that this was an excellent decision as these routines which have greatly facilitated the writing of the Scope program have turned out to be extremely reliable.
Background for Design

This task was designed to operate in Midas 3 as a subtask in a region containing at least four tasks. These tasks are the Monitor, the Analysis, the Command module, and the Scope, plus any subtasks which each of these may add.

The Monitor is the mother task of the region and it attaches the other three tasks. It serves as the communication center for the subtasks, controls the link to the 1800 computer and attempts to recover if any subtask aborts. The Analysis program is a version of the standard program used for analyzing data from the spark chamber experiments, and can function offline or online to the 1800. The Command module is the receiver of all commands entering the system whether from cards, the 1800, or the scope. Its function is to interpret these commands and respond to them.

Design Considerations

Most of the factors which have to be taken into consideration when writing any scope program were also applicable in this case. Several others were added due to the special design of Midas 3 and the other subtasks in the region.

A fast response to the user was a primary requirement and was obtained by having the Scope routine at a high priority, while it performs minimum functions, i.e., no calculations, so as not to degrade the computer performance in general.

Reliability was also essential as the Midas 3 system would be resident in the M91 for several hours at a time. Reloading the program because of failure of the Scope task would be inconvenient not only because of the disruption of the analysis program but also because the 1800 has to switch mode when the M91 changes from online to offline operation.

The basic design of the scope program was made relatively simple. This and the ability of the monitor to clean up and restart the scope task if it does terminate abnormally should ensure the Scope program has the required degree of stability.
A 150K partition was all that was available for the Midas I system. As a result all the programs were written in assembler language. One subroutine (SAARA) of the RC Beach's scope package was also re-written in assembler language.

Each subroutine of the Scope was written as separate control sections. It was hoped that this would make expansion and changes to the program easier to effect.

Communication between the Scope and other tasks was made possible through the Monitor. All requests to the scope are in a standard form as shown in the section on communication, and can only take place through the Monitor.

Functions of the Scope Program

Since the Scope is only one task in the region, its functions have been limited to the manipulation of messages and displays being recorded on the screen, and the management of the interrupts and cursor which allow the user to enter his requests into the system. These functions can be listed as shown below:

1. Display all messages from the other tasks in the region in the appropriate area of the screen.
2. Respond to users requests by sending them to the correct task to deal with them.
3. Control the scope interrupts and set up the cursor so that the user can enter his requests.
4. Scale histograms and other displays so that they will fit on the screen.

Future Modifications

No program that is being used is ever terminated and plans for future modifications, even if long-range, are never final. However, a list of obvious changes is available for this program; some of them are mere efficiency improvements while others are valid program modifications.

A) Currently there are no requests to the Scope from the Monitor directly so that a Monitor request checking routine has not been written. This routine would be almost identical to the command request checking routine and therefore a simple addition.
B) Due to the fact that unsolicited requests from the 1800 to the Monitor are regarded as an error, there is no request checking routine for link requests. This will have to be added when unsolicited requests from the 1800 are programmed.

C) Currently two subtasks have been added to enable three timers to operate for the scope. These must be eliminated and a general coding routine written so that M91 system is not degraded.

D) A scaling routine to enable straight line graphs to be displayed must be added.

E) A scaling routine for 2-dimensional histograms has also been requested.

F) If a user types in a request for the command module while the command module is still processing an earlier command, a message to that effect appears on the screen. When that command has been finished, a message "Commands will now be accepted" appears and the user may type in a message; however, the earlier message is now lost. Some buffering should be provided.

G) The last five user messages are displayed on the screen. This should be altered to display only four.

H) The message HISTOGRAM TOO LARGE FOR SCALING should be replaced by THE HISTOGRAM CANNOT BE SCALING.
General Layout of Subroutines

Fig. 9 gives a general flowchart of the scope task. Detailed descriptions and flowcharts of each of the subroutines are given in the last section of this document.

When the Scope is first attached it passes through an initialization routine. This routine saves the input parameters, which are mainly addresses of communication tables, and zeroes out pointers in core to enable the scope to be opened. This is necessary so that the same copy of the Scope program in core may be re-attached in case of abnormal termination of the Scope routine.

The Scope then enters a main loop. This loop is the backbone of the entire scope program. All other subroutines (except for the attention handler and timing routines) are called directly or indirectly from this loop.

On entering the loop, the program checks to see if there is a request from one of the other subtasks in the region; if not, it checks to see if the interrupts are enabled, and if they are, whether or not a user has typed in a request. If there is no user request, the Scope routine enters the wait state. If there is any active request either from one of the other subtasks in the region or from a user, control goes from the main loop into the respective subroutine to check the validity of that request and eventually into a lower level routine to service that request. Control is returned along the same path back to the main loop at the point where it was left, and processing in the main loop continues.

When the main loop enters the wait state, there are three ways it can become active again. The first way is through the Monitor; the Monitor has the address of the EIB on which the Scope is waiting so that when the Monitor has a request for the Scope it will post that EIB and thus wake the Scope active again. The second way the Scope can become active is when a user types in a request and enters an attention at the scope console. This attention causes the asynchronous execution of the Attention Handling Routine. This means that this routine gets control whether or not the main scope program is in the wait state. The Attention Handling routine will read the user's message, disable the interrupts, set a flag to show that there is
a user request and finally post the main Scope ECB. This means that if
the Scope was in the wait state, it immediately becomes active. The last way
of forcing the Scope to become active is through the use of a timer. Before
the scope goes into the wait state it sets a timer, so that at the end of
the interval the Scope becomes active again under any circumstances. This
was an extra precaution to ensure that the Scope never forgets a request
and does not for any reason go into a permanent wait state. When the Scope
becomes active its first function is to clear its own ECB. This ensures that
if a request has been made after it has been checked as inactive and before
the scope goes into the wait state, it will get serviced and not be forgotten.
In this case when the Scope attempts to go into the wait state, it will find
it has already been posted and will then execute the main loop once more.

User Response

The Scope program is attached by the Monitor at the request of the
Command task. It is attached at a high priority to enable fast response.
Interrupts are enabled and the cursor inserted also at the request of the
Command task. Once the cursor appears on the screen the user may type in
requests.

When a user has typed in his request, the end key interrupt is disabled,
and the cursor removed. The Scope scans the request and decides where to
send it. If it is for the Command module and the Command module is not
already servicing an earlier command, the request will be sent. If the
Command module is already servicing a request the new command will be lost.
When the command module has finished servicing a request a message appears
on the screen to the effect that new commands will be accepted. If it is not
for the Command module the Scope program will recognize it and perform
the necessary function. Currently only one request of this type will be
serviced; this is a Z which causes the Scope to terminate. Others may be
added.

After the decision has been made on whether or not a command can be
sent and any necessary action taken, the end key interrupt is then enabled
and the cursor inserted. The user may then start typing in requests again.
The delay between hitting the end key and the cursor re-appearing is not
noticeable to the user.
System messages

Run information (1 or more lines)

MAIN DISPLAY AREA

Reply to user message

Users last five messages displayed here

Cursor - User message (74 characters)
Screen Organisation

See fig. (10) for a diagram of the organisation of the screen. The screen is divided into six areas. The four upper areas are for program displays. The lower two areas are used for user messages.

Starting at the top of the screen the first area is for system messages from either the Monitor or the other tasks. These messages, from either the Monitor or the other tasks usually indicate status information.

The second area is for run information which may or may not be used for displaying periodically updated information on the run.

The third area is the main display area. Displays are made here as a result of users' requests.

The fourth area is a message response to user commands. It usually contains a duplicate or perhaps modified version of what the user has typed in.

The fifth area is a memory bank of the user's last five requests, and the last area contains the cursor and is where the user types in his request. Messages can be displayed on the four upper areas. These areas are internally referred to by location code. The system message area at the top of the screen has location code 1, the run information area has location code 2, the message response area has location code 3, while the main display area has location code 4. (These are referred to as LOCx in the diagram)

Timing Considerations

As far as the user is concerned, the current version of the scope program, running at priority 10 in the MQL, has no appreciable delay time between his request and the response. However, when the analysis is running at a low priority and obtaining data from the link, noticeable delays will be occasioned when the response has to wait for parts of the analysis to complete execution. During this time the keyboard will be enabled and the user may type in requests; but as explained in the User Response section, no commands will be sent to the Command module.

Internally there are three separate timers operating, one belonging to the scope task and two separate subtasks. Although the latter two will eventually disappear, their functions will remain.
The function of the main Scope timer is mainly precautionary. Care has been taken to enable the keyboard at all times so the user is never locked out; the timer performs the same function internally. Although whenever there is a request from the user or the Monitor, the Scope is posted and the scope always looks at a request; it may not be able to service it immediately (e.g., It cannot erase a display if user is holding the previous page); consequently it will return to that request when an interrupt is obtained from the lightpen or one of the other timers. The main scope timer is to ensure that under any circumstances no request is "forgotten". The two other timers reflect the appearance of the HOLD REPL (hold, replace) signs which appear for the paging of data and display of system messages.

Requests are serviced from each subtask in the order that they are given. Since only one at a time may be passed to the Scope, for example, if the Command module is waiting to display a new page of data because the user is still examining an earlier page, he cannot in the meantime send any other message to be displayed.

Communication and Parameter Passing

Communication between the various subtasks in Midas 3 was one of the primary requirements of the system. This communication takes place through the Monitor whose main function is to ensure the smooth flow of information between these tasks and the 1500.

When the scope is initially attached it is passed three parameters: -- see fig. (12). These are: 1. The Monitor Scope Request table address; 2. The Scope Request table address, and 3. The Monitor main ECB address.

It is through these tables that inter-task communication takes place.

The first table MSCOREQ, see fig. (11), is used to pass requests from the Scope to the Monitor. With the aid of an identifying code number the Monitor will know whether the Scope requires the message passed to one of the other subtasks or whether it is a direct request to the Monitor.

The second table is used to pass requests from the other tasks in the region to the Scope task. The Scope scans this table for requests whenever it becomes active. Currently there is only one buffer per task so that messages can only be sent one at a time from each task.
The last address passed is the address of the Monitors ECB. Whenever the Scope has a request for the Monitor it posts this ECB which causes the Monitor immediately to become active and to service the request.

Messages passed to the Scope for display on the screen must be of the format shown in fig. (12).

Each of the communication tables used by the Scope and the manner in which they can be set up are described in detail in the Monitor write-up, as this is one of the main functions of the Monitor.

Scope Requests to the Monitor
There are currently only two requests that the Monitor will recognize and these are identified by code number. They are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of Scope main ECB. The Monitor is requested to post this ECB each time it flags a request active for the Scope.</td>
</tr>
<tr>
<td>2</td>
<td>Pass the address of the user message to the Command module.</td>
</tr>
</tbody>
</table>

The Scope program will currently recognize no requests originating from the Monitor itself, none from the 1800 link, one from the analysis program and seven from the Command module. These again are identified by code number. They are:

Requests from the Analysis:

<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Display message on screen from the Command module;</td>
</tr>
</tbody>
</table>

Requests from the Command module:

<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Display a message</td>
</tr>
<tr>
<td>2</td>
<td>Clear main display area</td>
</tr>
<tr>
<td>3</td>
<td>Display message with reply</td>
</tr>
<tr>
<td>4</td>
<td>Command module now accepting commands</td>
</tr>
<tr>
<td>5</td>
<td>Disable interrupts</td>
</tr>
<tr>
<td>6</td>
<td>Clear main display area</td>
</tr>
<tr>
<td>7</td>
<td>Display Histogram</td>
</tr>
</tbody>
</table>
Initialising Parameters passed to the Scope when Attached

Parameter address list:
- A(MSCFREQ)
- A(SCREQTEL)
- A(MAINCB)

Monitor Scope Request Buffer
- SREQTEL
- SCNEQREQ
- SCANJNERQ
- SOMONREQ
- SOSOMREQ

Scope Request Table

Monitor Work Area

MAINCB

Monitor KCB

RD ll.
**MESSAGE FORMAT**

<table>
<thead>
<tr>
<th>Byte</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LENGTH (including hdr.)</th>
<th>CODE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **LENGTH**: Length of the message including the four byte header.
- **CODE**: The code number which determines which subroutine in the Scope program will service the request.
- **LOCATION**: The code number which will determine where on the screen the message is to be displayed.
- **MESSAGE**: The message to be displayed (if any).

**NOTE**: All messages to the Scope program from the other subtasks in the region must be of this format. This includes messages which are requests for action such as enabling the interrupts which do not require any text to be displayed.

FIG 12.
**Abnormal Termination**

The Scope program is currently set up to abend if it encounters errors in its own functions which may or may not be a result of incorrect parameters passed. User abend termination codes are numbered from 1000 upwards and the locations where each of these error codes occur are listed below.

<table>
<thead>
<tr>
<th>CODE</th>
<th>CHBT</th>
<th>CHBT FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1212</td>
<td>SCOPE</td>
<td>Initialisation</td>
</tr>
<tr>
<td>1515</td>
<td>MERIDFLY</td>
<td>Display message</td>
</tr>
<tr>
<td>3131</td>
<td>MERULLETQ</td>
<td>Display message with reply</td>
</tr>
<tr>
<td>8888</td>
<td>SCOPATTN</td>
<td>Scope Attention Handler</td>
</tr>
<tr>
<td>8828</td>
<td>MERBSCAL</td>
<td>Scale Histogram</td>
</tr>
<tr>
<td>9038</td>
<td>SCOPE</td>
<td>Error in recognising user message</td>
</tr>
<tr>
<td>3030</td>
<td>SAAAB1</td>
<td>Graphics package error handler</td>
</tr>
</tbody>
</table>

All except the last user code listed above store the exact location where the error was detected in SCENORCD in the main Scope work area (control section name SCOPWOK).

By examining the address in SCENORCD the exact location where the error was detected can be found.

SAAAB1 is the error exit program for the graphics package routines. In this routine there is a double word called SUBRNAME which contains the name of the graphics subroutine call which caused the error to be found. Immediately following this there is a one word location called SCRVNOCD which contains the error code number. Investigation of the contents of these storage locations together with the explanation of the code number (found in P.C. Beach's User Note 80) will readily explain the problem.

All subroutines of the Scope program have standard OS save area linkages so that in case the task terminates with a system error code then the error can be traced in the usual manner.
INITIALISATION

INPUT PARAMETERS
RI points to an address list containing these values
ADMNSCP Address of MNSCPHD table
ADSCPRQ Address of SCRTBL
ADMNCB Address of monitor ECB

OUTPUT PARAMETERS
Not applicable as control is never returned from this routine

RETURN CODE
Not applicable.

LOWER LEVEL ROUTINES
SIPA1
SISV1
SIGHT1
STEXT1
BSATH1

EXIT
Main loop

FUNCTIONS
This routine initialises some graphics package core locations to zero, in case this is a restart of the program. It initialises the scope, sets up the initial display on the screen, enables the lightpen interrupts, and attaches the timer subtasks. It finally sends the address of the Scope main ECB and some termination addresses to the Monitor for restart procedures and finally exits to the Main Loop.
MAIN LOOP

CONFIG

SCOPE

INPUT PARAMETERS
none

OUTPUT PARAMETERS
none

RETURN CODE
none

LOWER LEVEL ROUTINES
LINKERR (dummy)
ANLSEER
MONITERR (dummy)
COMMANDER
SCOPEEXIT

EXIT
none

FUNCTIONS:
This routine checks if there is an active request from the link, analysis program, the Monitor, the Command routine, or the Scope user. If there is, it branches to one of the routines listed above to check the request servicing each in turn. If no request is active, a timer is set and the Main Loop then enters the wait state. Immediately on becoming active the EOB on which this routine waits is cleared and control is returned to the beginning of the loop.
MFC MAIN LOOP

Main Scope Loop

1. Does the link have a req? Yes → dummy
   No

2. Does Analysis have a req? Yes → Branch to ANALSEQ to check it
   No

3. Does Monitor have a req? Yes → dummy
   No

4. Does Command have a req? Yes → Branch to COMMSEQ to check it
   No

5. Is there a scope user? Yes → Branch to SCOPESEQ to service it
   No

6. WAIT

7. Set Timer
CHECK COMMAND REQUEST

CSECT SCOPE - ENTRY AT COMMAND

INPUT PARAMETERS none

OUTPUT PARAMETERS none

RETURN CODE none

LOWER LEVEL Routines MEQDISIL
MPERLSEQ
INTERIAL
NEWCOMMD
SCPCOMMD
NEWPAGES
HSTORMES

EXIT Main Loop

FUNCTIONS

This routine checks whether the request in SCSCOMQP of the Scope Request Table (SCSREQTBL) is given in the form of a code number, or whether it points to a message which contains the code number in the header (see rgs (B) and (L) for format of these requests). Once the code number is obtained, a check is made to ensure that it is a valid code number. This routine then branches to a routine listed in the COMCOHES table (in the work area) which corresponds to code number given. On return from that routine the return code is checked. If the return code (in RL) is hex FF, control is immediately returned to the Main Loop. This means that the routine has been unable to terminate the request and will try again later. If the return code is zero, the SCSCOMQP is zeroed, indicating that the request has been serviced and control is returned to the Main Loop.
CHECK ANALYSIS REQUEST

INPUT PARAMETERS none

OUTPUT PARAMETERS none

RETURN CODE none

LOWER LEVEL ROUTINES ANRREQ

EXIT Main Loop

FUNCTIONS

This routine checks whether the request in SCANLRF of the Scope Request Table (SCREQTEL) is given in the form of a code number or whether it points to a message which contains the code number (see figs. (8) and (12) for format of these requests). Once the code number is obtained a check is made to ensure that it is a valid code. This routine then branches to a routine listed in SANCHORS tables (in the scope work area) which corresponds to the code number given. On return from that routine the return code is checked. If the return code (in R15) is hex FF, control is immediately returned to the Main Loop. This means that the routine has been unable to terminate the request and will try again later. If the return code is zero then SCANLRF is zeroed, indicating that the request has been serviced and control is returned to the Main Loop.
Check analysis request and branch to service it.

Check Command request and branch to service it.

1. Get the code value in the message hdr.
2. Branch to corresponding subroutine.
3. Test return code:
   - Other: Return to Main Loop
   - 0: Return to Main Loop
4. Abend with code 1212
SCOPE USER REQUEST

ENTRY AT SCOPEREQ

INPUT PARAMETERS
none

OUTPUT PARAMETERS
none

RETURN CODE
none

LOWER LEVEL ROUTINES
SCBKTI
Sитет
SICEI
SEATK
SHEAL

EXIT
Main Loop

FUNCTIONS

The length of the user message is found in RCHHEAD and is checked
for a zero length. If zero, the interrupts are re-enabled and an
exit is made to the main loop. If the message length is not zero, then
SCOPESTAT is checked for the address of the message. A check is then
made on the message itself and if it is not recognizable to the scope,
it is sent to the command module. In order to send the message to the
Command module, checks have to be made that the Command module is accept-
ing commands and that the buffer to send the requests to the Monitor
is empty. If one or neither of these conditions is fulfilled, a return
is made to the Main Loop; otherwise the message is sent to the Command
module by setting up a request to the Monitor. Finally SCOPESTAT, which
contains the user address and flag, is set to zero, the scope screen is
re-adjusted to add the last user message to the buffer of five most
recent messages, the interrupts are enabled and the Monitor ECB is posted.
A return is then made to the Main Loop.
Service Scope Users Request

1. Is user msg. len<>0?
   - Yes: Return to Main Loop
   - No: Go to the next step.

2. Is addr. of message = zero?
   - Yes: Abend with code 1212
   - No: Go to the next step.

3. If no send message to other, it a one char msg.
   - Yes: Command module
     - Yes: Move up user msg. Insert cursor, and enable ins
       - Yes: Return to Main Loop
     - No: Go to the next step.
   - No: Abend with code 1212

4. Is a reply expected?
   - Yes: Send msg. to user that called not accepted
     - Yes: Move up user msg. Insert cursor, and enable ins
       - Yes: Return to Main Loop
     - No: Go to the next step.
   - No: Last msg. to Monitor serviced
     - Yes: Set up msg. for Monitor to send to Command rtm.
       - Yes: Post Monitor S.C.B.
       - No: Return to Main Loop
     - No: Go to the next step.
(DISPLAY) ANALYSIS MESSAGE

CODE

AMGIDISP

INPUT PARAMETERS

R14 contains the return address.
R1 points to the message header.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RL                  Length   Code   Location
Message

LENGTH
Length of message plus header.
(currently = 78)

CODE
Should be value 1

LOCATION
Should be value 1 to display message on system message display

MESSAGE
Message to be displayed

OUTPUT PARAMETERS

One of the following codes is returned in Register 15:

0     The message was displayed successfully, or the message could not be displayed at all.
FF    The message could not yet be displayed.

LOWER LEVEL ROUTINES

MEGIDISP

EXIT
On register 14

FUNCTIONS

The analysis program can currently only send messages to the system message area so this routine first checks the location code. If it is not correct, no message is displayed and control is returned with return code of zero. If the location code is one, then control is passed to MEGIDISP, the main message display routine. On return, register 15 is untouched and this routine exits on register 14.
DISPLAY MESSAGE FROM COMMAND TASK

**INPUT PARAMETERS**

- **RL** must point to the message header.

```
<table>
<thead>
<tr>
<th>Length</th>
<th>Code</th>
<th>Locn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- **LENGTH** Value 76, must be header plus message length
- **CODE** Value 1 indicates message to be displayed
- **LOCATION** Code number 1 through 5 indicates area of screen to contain message
- **MESSAGE** 74 byte message to be displayed.

**OUTPUT PARAMETERS**

- None

**RETURN CODE**

RL5 will contain one of the following return codes:

- 0  The message was displayed
- FF  The message could not yet be displayed

**LOWER LEVEL ROUTINES**

- MSGDISPL

**EXIT**

On register 13

**FUNCTIONS**

Control is immediately passed to MSGDISPL which is the central message handler for the scope. On return, RL5 contains one of the return codes; this is left undisturbed and control is returned to the address in register 14.
Display message from Command task

- Save regs. and link save areas

- Branch to display msg to scope user

- Return Rc*[S15]
CLEAR AREA OF SCOPE

INPUT PARAMETERS
- RL must point to the message header
- LENGTH: Length of header (must be value = \( \frac{1}{2} \))
- CODE: To indicate the new pages, this must be 6
- LOCATION: Area to be cleared containing the request
- RL must contain the return address

OUTPUT PARAMETERS
- none

RETURN CODE
- RL5 contains one of the following return codes:
  - 0: The area was deleted
  - FF: The request could not yet be performed

LOWER LEVEL ROUTINES
- SISRT1
- SISERT1

EXIT
- On register 14

FUNCTIONS
- The location code is checked to see which area of the screen is to be cleared. Currently only the central area with location code 4 may be cleared. Other codes are treated as NCP's. When the code number is determined as 4, the screen is cleared and the set numbers re-initialized. The linecount for displaying is set to minus one and the routine exits on register 14.
Clear Main Display Area

1. Save regs. and link save areas

2. Check location code
   - Check other location code

3. Delete Main Display area

4. Re-initialise the sets

Set the line count to minus 1

Return RC=0
DISPLAY MESSAGE WITH REPLY

INPUT PARAMETERS
RI must contain the address of the message header
Register 1 points to a message header exactly the same as for Command Message Display except that the code should now have a value of 3. RI4 must contain the return address.

OUTPUT PARAMETERS
none

RETURN CODE
On return, R15 will contain one of the following codes:
0 The message was displayed
FF The message could not be displayed

LOWER LEVEL ROUTINES
MEDISPLY

EXIT
Main Loop

FUNCTIONS
The functions of this routine are identical with the Command Message Display routine (MEDISPLY) except that a flag is set to allow the user to send a reply to the command module.
MURPREG

Display message with Reply Requested

Save regs. and link save areas

Branch to display msg. to scope user

Apend with code 3131

May the int err be enabled?

Set flag to show reply requested

Is the return code zero?

Return R0=0

Return RC=815
ALLOW INTERRUPTS TO BE ENABLED

CSECT

INTERMABL

INPUT PARAMETERS

RI4 must contain the return address
Although this routine does not check it,
Register 1 points to the message which
requests this service

\[
\begin{array}{ccc}
\text{RI} & \text{LENGTH} & \text{CODE} \\
\text{LOCATION} & \text{Value must equal 8} & \text{Value must equal 3} \\
\text{Not checked} & & \\
\end{array}
\]

OUTPUT PARAMETERS

none

RETURN CODE

0  Interrupts were enabled (in RI5)

LOWER LEVEL ROUTINES

SPECTI
SICURI
MESSAGE

EXIT

On Register 14

FUNCTION

A blank line of text is inserted at the bottom of the screen and
the cursor is inserted at the beginning of it. The ENIX interrupt is
enabled and the flags set up to indicate that the interrupts may be
enabled and also that they are currently enabled. A message is sent
to the system display area to this effect. The routine then exits
through register 14.
INTERRUPT

Interrupts may be Enabled

1. Save regs. and link save areas
2. Set flag to show ints may be enabled
3. Put blank line in user message area
4. Enable ENDX Insert cursor
5. Show ints are enabled
6. Branch to send message to scope user
7. Return
FUNCTION

This routine first sets a flag to indicate that the command module will accept new commands. It then branches to MESSAGE to put a message on the screen indicating this fact. The control then returns to the address in Register 14. This now allows the user to send new commands to the Command module.
DISPLAY HISTOGRAM

INPUT PARAMETERS

RI points to the message requesting the histogram to be displayed. The second word of the message contains the first histogram parameter.

<table>
<thead>
<tr>
<th></th>
<th>LENGTH</th>
<th>CODE</th>
<th>LOCN.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>bytes</td>
</tr>
</tbody>
</table>

- LENGTH: Length in bytes of the message header plus the parameters.
- CODE: Value of 7 indicates a histogram to be displayed.
- LOCATION: Currently ignored, histograms are always displayed in the main display area.
- NUMH: Number of histogram bins.
- HISTORMAD: The address of the histogram.
- HISTOBPER: Number of bytes per histogram bin.
- STARTX: Starting value of the X coordinate in floating point.
- ENDX: Ending value of the X coordinate in floating point.
- YMSEGAD: The address of the Y coordinate message.
- XMSEGAD: The address of the X coordinate message.

OUTPUT PARAMETERS:

None
FUNCTIONS

The function of this routine is to scale the histogram and display it. On entry the address of the histogram parameters is saved in NETFRIOMAD. The routine then branches to RHESCAL, a subroutine which will scale the histogram bins to a size which will fit on the screen. If the histogram cannot be scaled, a message is put on the screen, and the routine exits to the address in Register 14. If the histogram has been successfully scaled, then the lines are displayed together with the points on the Y axis (which were calculated by RHESCAL). Values are calculated for each of the points on the Y axis and displayed.

The displacement of the points on the X axis are now calculated allowing a maximum of ten points between the first and the last histogram bin, and a round number for the value of the displacement. The actual values of the points are then calculated and displayed. A total of seven digits, a maximum of four before the point and a maximum of three after the point, can be displayed for each point on the X axis. If the number corresponding to a certain point reaches too close or beyond where the number for the next point on the axis should appear, then the number for the next point is omitted. Finally, the messages for the X coordinates and Y coordinates are displayed and this routine returns control to the address in register 14.
Save regs. and link save areas
Save the addr. of input data
Branch to REMCAL to scale the histogram bins
Check the return code
Display X,Y axis. Display histogram lines. Display points on Y axis.
Calculate values on the Y axis and display them
Branch to REMPNT to convert to integer the value STARY minus EMBX
Find the value of the increment if a maximum of 10 pts. displayed
Round increment to a multiple of 2, 5, or 10.
Find number of raster units on the X axis corresponding to distance between points
Calculate value for each pt. on the X axis and print it
Return RC=0
Send msg. to user that histogram cannot be scaled.
Return RC=0
DISABLE INTERRUPTS

CSECT

INPUT PARAMETERS

R1$ must contain the return address
Although this routine does not check it,
Register 1 points to the message which requests
this service

<table>
<thead>
<tr>
<th>Length</th>
<th>Code</th>
<th>Loc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH</td>
<td>4 bytes only</td>
<td></td>
</tr>
<tr>
<td>CODE</td>
<td>must have value of 5</td>
<td></td>
</tr>
<tr>
<td>LOCATION</td>
<td>Not checked.</td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT PARAMETERS

none

RETURN CODE

0 The interrupts were disabled

LOWER LEVEL ROUTINES

SIMM1
SHELT1
SRCUR1
MEGISPLY

EXIT

On Register 1$

FUNCTIONS

The interrupts are disabled, the line currently being typed in
by the user is deleted, the cursor is removed, and a message is dis-
played on the screen that the interrupts have been disabled.
INTERRUPT

Disable the Interrupts

1. Save regs. and link
2. Save areas
3. Set flag to show ints. disabled
4. Disable INTK Interrupts
5. Delete any line typed in
6. Remove cursor
7. Branch to send ints. disabled message
8. Return
   RC=0
GENERAL MESSAGE DISPLAY

INPUT PARAMETERS

RI must contain the address of the message header.
RI4 must contain the return address.

OUTPUT PARAMETERS

RETURN CODE in RI5

0 The message was displayed
FF The message could not yet be displayed

LOWER LEVEL ROUTINES

SCHOOL
SALAME
STREET
NEWPAQES

EXIT

On Register 13.

FUNCTION

This display routine gets called whenever any messages are to be displayed on the screen. It does not get called when lines of text are displayed as part of a histogram.

The first function of this routine is to check the length of the message. Only 74 byte messages are currently accepted for display. If a message of another length is encountered, the message is not displayed, but the return code is set to zero and the message forgotten.

After checking the length, this routine then checks the location code. If the location code is 1, 2, 3, or 4, the routine branches to following points called SYSTEMS, RUNINFOR, RESPONSE, and IMPULSITI respectively. These correspond to the four areas of the screen.

RUNINFOR and RESPONSE cause the message to be shown immediately, a return code of zero obtained, and control returns to the calling program.
SYSTRESDG causes the following series of events to take place.

A "HOLD" message appears on the righthand side of the system message area. This informs the scope user that unless the "HOLD" is interrupted by the lightpen, the current message will be replaced by the new one within five seconds (this is the first timer request as shown in the flow chart). If the user interrupts "HOLD" with the lightpen within those five seconds, then REFL appears adjacent to HOLD. This informs the user that he may replace the current message by interrupting REFL any time within the next five minutes (this is the second timer as shown in the flow chart). If he interrupts HOLD at any time during this second period then he gets a second timer of five minutes started again. If the five minute timer expires without either HOLD or REFL being interrupted then REFL disappears from the screen and the user has again a five second period to interrupt HOLD before the message is replaced. Please see the flow chart for details.

When the message is replaced, the return code is set to zero and control returns to the calling program.

SYSTRESDG causes the same series of actions to take place as for SYSTRESDG except that the order is slightly changed. Both HOLD and REFL appear on the upper righthand side of the main display area. The current display remains on the screen for five minutes unless either HOLD or REFL is interrupted by the lightpen. If REFL is interrupted, then the current display is immediately replaced by the new one and this routine exits with return code of zero. If HOLD is interrupted, then the timer is again set to last for five minutes. If the timer expires after five minutes with neither HOLD or REFL interrupted, then REFL disappears from the screen and the user has five seconds to interrupt HOLD for another five minute period. If the five second timer expires without this happening, then the current display is deleted, the new display appears on the screen, and this routine then exits.
MODIFIED
Central Message Display Routine

Save regs. and link save areas

Is the msg. 74 bytes long?

No

Branch to SYSTEMMSG

Yes

Check the location code to determine where the message is to be displayed.

Branch to DEPLMTL

Run info-

System mag.

User msg. Response

Buffer

Display the message

Return RC=0

Display the message

Return RC=0
INTEGER TO DECIMAL CONVERSION

EXECUT

RECVNT

INPUT PARAMETERS

RL contains the number to be converted
RL1 contains the return address

OUTPUT PARAMETERS

ALDRCHS has the address of the first character (sign) of the unpacked signed decimal number
NUMCHARS contains the number of characters in that decimal number.

RETURN CODE in RL5

0 The conversion was performed
1 The conversion could not be performed.

LOWER LEVEL ROUTINES

none

EXIT

On Register 14

FURTHER

A check is first made to see if the number to be converted lies between + and - 9999999. If it does not, then the conversion cannot be performed. The conversion is now made to a signed unpacked number with the decimal point inserted three places from the right (i.e., equivalent to dividing by $10^3$).
INTEGER to Decimal Conversion

1. Save registers and link.
2. Save areas.
3. Convert to decimal with sign and decimal point.
4. Calculate number of chars and save in MEMCHARS.
5. Return RC=0.
FLOATING POINT TO INTEGER CONVERSION

CSEG

RETN fistpoint

INPUT PARAMETERS

R1 points to a parameter list.

K, NUMBER
RESULT

K ... 4 bytes

NUMBER Signed floating point number to be converted
RESULT Integer value of number is placed here after conversion.

R14 contains the return address

OUTPUT PARAMETERS

See input parameters

RETURN CODE (in R15)

0 The number was converted
1 The number given could not be converted

LOWER LEVEL ROUTINES

none

EXIT

On Register 14

FUNCTION

This routine checks that the absolute value of the short floating point number lies between 0.001 and 9999.999. If it lies outside this range, control is returned with a return code of one. This routine then multiplies the number by 10^3 to remove the fraction and finally converts it to integer.
Floating Point to Integer Conversion

- Save regs. and link save area;
- Check that the abs. value of the number lies between 9999.999 and 0.001;
- Take signed value and mult. by 10^3;
- Convert to integer;
- Correct the sign;
- Return RC=1;
- Return RC=0;
posting the main scope ECB. If the lightpen caused the interrupt
then a test is made to determine which element was interrupted and
a flag is set accordingly. The routine then exits after posting the
main scope ECB.
HISTOGRAM SCALING ROUTINE

RERICAL

INPUT PARAMETERS

Rl points to a parameter list.

Rl

NUMBINS

HISTORMAD

BYTESPERIN

NUMBINS The number of bins to be scaled
HISTORMAD Address of the first bin,
BYTESPERIN Number of bytes per bin (either 2 or four)

Rl+ contains the return address.

OUTPUT PARAMETERS

The output parameters are placed in the work area control section and are as follows:

XENAXIS THE ARRAY OF BEGINNING X COORDS FOR XAXIS AND YAXIS
YENAXIS THE ARRAY OF BEGINNING Y COORDS FOR XAXIS AND YAXIS
XENFAKIS THE ARRAY OF ENDING X COORDS FOR XAXIS AND YAXIS
YENFAKIS THE ARRAY OF ENDING Y COORDS FOR XAXIS AND YAXIS
XOXYAXIS THE ARRAY OF X COORDS FOR POINTS ON X AXIS
YOXYAXIS THE ARRAY OF Y COORDS FOR POINTS ON X AXIS
XEROXOD ARRAY OF BEGINNING X COORDS FOR HISTOGRAM LINES
YEROXOD ARRAY OF BEGINNING Y COORDS FOR HISTOGRAM LINES
XEROXOD ARRAY OF ENDING X COORDS FOR HISTOGRAM LINES
YEROXOD ARRAY OF ENDING Y COORDS FOR HISTOGRAM LINES
NUMXYOD NUMBER OF POINTS ON TRAXIS
NUMHORIN NUMBER OF HISTORM LINES

RETURN CODE

0 HISTOGRAM SCALLED CORRECTLY
1 HISTOGRAM COULD NOT BE SCALLED
This routine scales the histogram bins so that they will fit onto the 2250 screen. A maximum of fifty vertical histogram lines can currently be displayed on the screen. This number can be changed by increasing the assembled value of MAXHIST (maximum lines per histogram) and the space allocated to XENDOODE, XENDOODE, YENDOODE, and YENDOODE, all of which are in the Scope work area. The histogram is scaled by lumping the bins together in boxes, each box corresponding to one line of the histogram and containing the sum of one or more bins. There is a maximum raster separation of the lines on the screen so that if the histogram consists of only two lines, they will not be spread over the screen but will be close to the Y axis but having only the maximum raster separation between them.

It is calculated that the histogram cannot be scaled if there is only one bin. Failure also would occur if there were x number of bins per box and the integer value of the largest bin multiplied by x could not fit into one four-byte word.

This routine also calculates the displacement of the scaling points on the Y axis, and the X and Y axis themselves.
Histogram Scaling Routine

Find the accumulative value in each box.

Calculate the scaling factor.

Multiply scaling factor by boxsize to obtain Y coords of lines.

Set up remaining coords for histogram lines.

Set up coords for the X and Y axis.

Calculate points on Y axis.

Save regs. and link save areas.

Calculate the minimum box size using max. ins/hist.

Is numb. of bins >= 0?

Find the largest bin in the histogram.

Largest bin zero?

Calculate the number of bins per line.

Check that the largest boxsize will be small enough to fit in one word.

Calculate the numb. of full boxes and the numb. of bins in the last box.

Return ROC0.

Abort with code 8883.
SCOPE TIMER EXIT

COMMIT S.TIMEXIT

INPUT PARAMETERS none

OUTPUT PARAMETERS none

RETURN CODE none

LOWER LEVEL ROUTINE none

EXIT On Register 14.

FUNCTION

Post Main Scope BCB so that the routine does not go into a permanent wait state.
TIMER TASKS: TIM and TIM6.

INPUT PARAMETERS: R1 points to a parameter address list.

OUTPUT PARAMETERS: none

RETURN CODE: none

LOWER LEVEL ROUTERS: none

EXIT: none

**TFLAGS**

<table>
<thead>
<tr>
<th>BIT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Indicates a Timer was requested</td>
</tr>
<tr>
<td>1</td>
<td>Indicates a Timer macro issued</td>
</tr>
<tr>
<td>2</td>
<td>the Timer terminated</td>
</tr>
<tr>
<td>3</td>
<td>an interrupt on HOLD</td>
</tr>
<tr>
<td>4</td>
<td>&quot;        &quot; REFL</td>
</tr>
<tr>
<td>5-15</td>
<td>not used</td>
</tr>
</tbody>
</table>

**TINYVL**

The interval for which the timer was issued.

**ECB**

This is the ECB this task must wait on.
FUNCTION

The function of these tasks is simply to issue a timer for the main scope task. When the timer terminates, the flag in THAGS is set and the main Scope ECB is posted. This task then goes into the wait state on the ECB whose address was passed to it when the task was attached.
Graphics Package
Error Exit

1. Save regs. and link save a.
2. Save name of subroutine in SUBRNAME
3. Save error code in SCNAME
4. Abend with code 3030
GRAPHIC ROUTINES ERROR EXIT

**INPUT PARAMETERS**

t points to a parameter address list

**OUTPUT PARAMETERS**

**RETURN CODE**

**LOWER LEVEL ROUTINE**

**EXIT**

**FUNCTION**

The name and code in the input list are moved to SUBRNAME and SPCGOODE which are in this control section. The task is then made to abend 3030, with a dump.
When the dump is examined SUBSTNAM and SIERGOCOD will contain the name of the routine where the error was found. Bob Beach's User Note 20 must then be referenced to obtain the cause for the error.