

AQUATROL CORP.

Software Document  
Version 1.0

By

Mohamed Fayad

REVIEWED

BY

Jerry Knoth

1000000000000000

0000000000000000

14

15

ALL :

ALL

0 : INITIAL POSITION

0.1 EMISSIONS TUNING  
0.2 STABILISATION

1 : GATE 1

0.1 INTRODUCING  
0.2 AMPLIFICATION

2 : EMISSION

2.0 INT  
2.1 Cxx  
CxxA  
2.2 CxxA  
2.3 CxxA  
2.4 CxxA  
2.5 CxxA  
2.6 CxxA  
2.7 CxxA  
2.8 CxxA  
2.9 CxxA  
3.0 CxxA  
3.1 CxxA  
3.2 CxxA  
3.3 CxxA  
3.4 CxxA

0.0 2 : HIGH GATE

2.0 INT  
2.1 Cxx  
2.2 CxxA  
2.3 CxxA  
2.4 CxxA  
2.5 CxxA  
2.6 CxxA  
2.7 CxxA  
2.8 CxxA  
2.9 CxxA  
3.0 CxxA  
3.1 CxxA  
3.2 CxxA  
3.3 CxxA  
3.4 CxxA

0.0 3 : INTERNAL POSITION

0.1 INTRODUCING  
0.2 AMPLIFICATION  
0.3 STABILISATION

4 : KEYING

0.0 INT  
0.1 GATE

4 : GATING

0.0 INT  
0.1 GATE

## TABLE OF CONTENTS

### CONTENTS :

#### SECTION 0 : INTRODUCTION

- 0.1 DOCUMENTATION DESIGN.
- 0.2 STANDARD FUNCTIONS & TOP\_DOWN DESIGN.

#### SECTION 1 : LITERALS

- 1.0 INTRODUCTION
- 1.1 Cxxxx.LIT

#### SECTION 2 : GLOBAL AND EXTERNAL VARIABLES & STRUCTURES.

- 2.0 INTRODUCTION
- 2.1 CxxxxGLOB.Pyy  
CxxxxGLOB.EXT
- 2.2 CxxxxTPUB.Pyy  
CxxxxTPUB.EXT
- 2.3 CxxxxGEXR.Pyy  
CxxxxGEXR.EXT
- 2.4 CxxxxTABL.PUB  
CxxxxTABL.EXT

#### SECTION 3 : HISTORICAL GLOBAL AND EXTERNAL VARIABLES & STRUCTURES.

- 3.0 INTRODUCTION
- 3.1 CxxxxHIST.Pyy  
CxxxxHIST.EXT
- 3.2 CxxxxHPUB.Pyy  
CxxxxHPUB.EXT
- 3.3 CxxxxHEXR.Pyy  
CxxxxHEXR.EXT

#### SECTION 4 : INITIALIZATION.

- 4.0 INTRODUCTION
- 4.1 CxxxxINIT.Pyy
- 4.2 CxxxxMODS.EXT

#### SECTION 5 : KEYBOARD.

- 5.0 INTRODUCTION
- 5.1 CxxxxKYBD.Pyy

#### SECTION 6 : OPTION.

- 6.0 INTRODUCTION
- 6.1 CxxxxOPTP.Pyy

9999 9999

TIN : A : 7

0.5  
1.5  
2.5  
3.2  
4.2  
5.2  
6.2  
7.2  
8.2  
9.2  
10.2

AJA : 8

0.8  
1.8  
2.8  
3.8  
4.8

TOT : 9

0.8  
1.8

10 : F417

10.0  
10.1

11 : D13

11.0 INMD  
11.1 Cxx-DI  
11.2 Cxx-MDI

12 : DATA B

12.0 INTMD  
12.1 Gxx-A  
12.2 Gxx-B

13 : D47A

13.0	13.0
13.1	13.1
13.2	13.2
13.3	13.3
13.4	13.4
13.5	13.5
13.6	13.6
13.7	13.7
13.8	13.8
13.9	13.9

SECTION 7 : SCREEN EDITOR.

- 7.0 INTRODUCTION
- 7.1 CxxxxEDIT.Pyy
- 7.2 CxxxxECAS.Pyy
- 7.3 CxxxxDCAS.Pyy
- 7.4 CxxxxETBL.Pyy
- 7.5 CxxxxDTBL.Pyy
- 7.6 CxxxxEFLD.Pyy
- 7.7 CxxxxSFLD.Pyy
- 7.8 CxxxxCPRM.Pyy

SECTION 8 : ALARM SYSTEM.

- 8.0 INTRODUCTION
- 8.1 CxxxxALRM.Pyy
- 8.2 CxxxxASYS.Pyy
- 8.3 CxxxxALOG.Pyy

SECTION 9 : TOTAL.

- 9.0 INTRODUCTION
- 9.1 CxxxxTOTL.Pyy

SECTION 10 : FAIL.

- 10.0 INTRODUCTION
- 10.1 CxxxxFAIL.Pyy

SECTION 11 : DISK & FILE SYSTEM.

- 11.0 INTRODUCTION
- 11.1 CxxxxDISK.Pyy
- 11.2 CxxxxFILE.Pyy

SECTION 12 : DATA BASE.

- 12.0 INTRODUCTION
- 12.1 CxxxxDBAS.Pyy
- 12.2 CxxxxFILL.Pyy

SECTION 13 : DATA BASE MONITOR & CONTROLS.

- 13.0 INTRODUCTION
- 13.1 CxxxxANIN.Pyy
- 13.2 CxxxxDCIN.Pyy
- 13.3 CxxxxANOP.Pyy
- 13.4 CxxxxDCOP.Pyy
- 13.5 CxxxxPCON.Pyy
- 13.6 CxxxxCTRL.Pyy

25 : A

Q.4  
I.4  
S.4  
E.4  
B.4

25 :

Q.2  
I.  
S.2

T.2

I.  
Q.2  
I.2  
S.2  
E.2  
B.2

B.2

Q.2  
I.2  
S.2  
E.2  
B.2  
C.2

15 : DEC

4.0 40.0  
3.9 39.0  
3.8 38.0  
3.7 37.0  
3.6 36.0  
3.5 35.0  
3.4 34.0  
3.3 33.0

15 : DEC

I.3  
S.3  
E.3

I.1

15.1  
20.2

---

SECTION 14 : DISPLAY GENERATOR.

14.0 INTRODUCTION  
14.1 CxxxxGGEN.Pyy  
14.2 CxxxxDISG.Pyy  
14.3 CxxxxESCR.Pyy  
14.4 CxxxxDGEN.Pyy

---

SECTION 15 : GRAPHICS.

15.0 INTRODUCTION  
15.1 CxxxxGRPH.Pyy  
15.2 CxxxxGRPH.EXT

---

SECTION 16 : TRENDS.

16.0 INTRODUCTION  
16.1 CxxxxTRND.Pyy  
16.2 CxxxxLINE.Pyy  
16.3 CxxxxLNGR.Pyy  
16.4 CxxxxDLGR.Pyy

---

SECTION 17 : REPORT GENERATOR.

17.0 INTRODUCTION  
17.1 CxxxxREPG.Pyy  
17.2 CxxxxPRTG.Pyy  
17.3 CxxxxERPT.Pyy  
17.4 CxxxxRGEN.Pyy  
17.5 CxxxxPERD.Pyy

---

SECTION 18 : TEST.

18.0 INTRODUCTION  
18.1 CxxxxTEST.Pyy  
18.2 CxxxxDREM.Pyy  
18.3 CxxxxDMST.Pyy  
18.4 CxxxxDIOR.Pyy  
18.5 CxxxxIORK.Pyy

---

SECTION 19 : PRINT

19.1 INTRODUCTION  
19.2 CxxxxPRNT.Pyy  
19.3 CxxxxPBAS.Pyy

---

SECTION 20 : HISTORICAL DATA

20.1 INTRODUCTION  
20.2 CxxxxHSTM.Pyy

3 - 12

33  
12

2.5  
0.2  
0.5  
0.2

295,  
235,  
195,  
195,  
555,  
255,  
295,  
295,  
295,  
295,  
135,

SECTION 21 : CRT AND PRINTER MACROS.

SECTION 22 : AQUA.

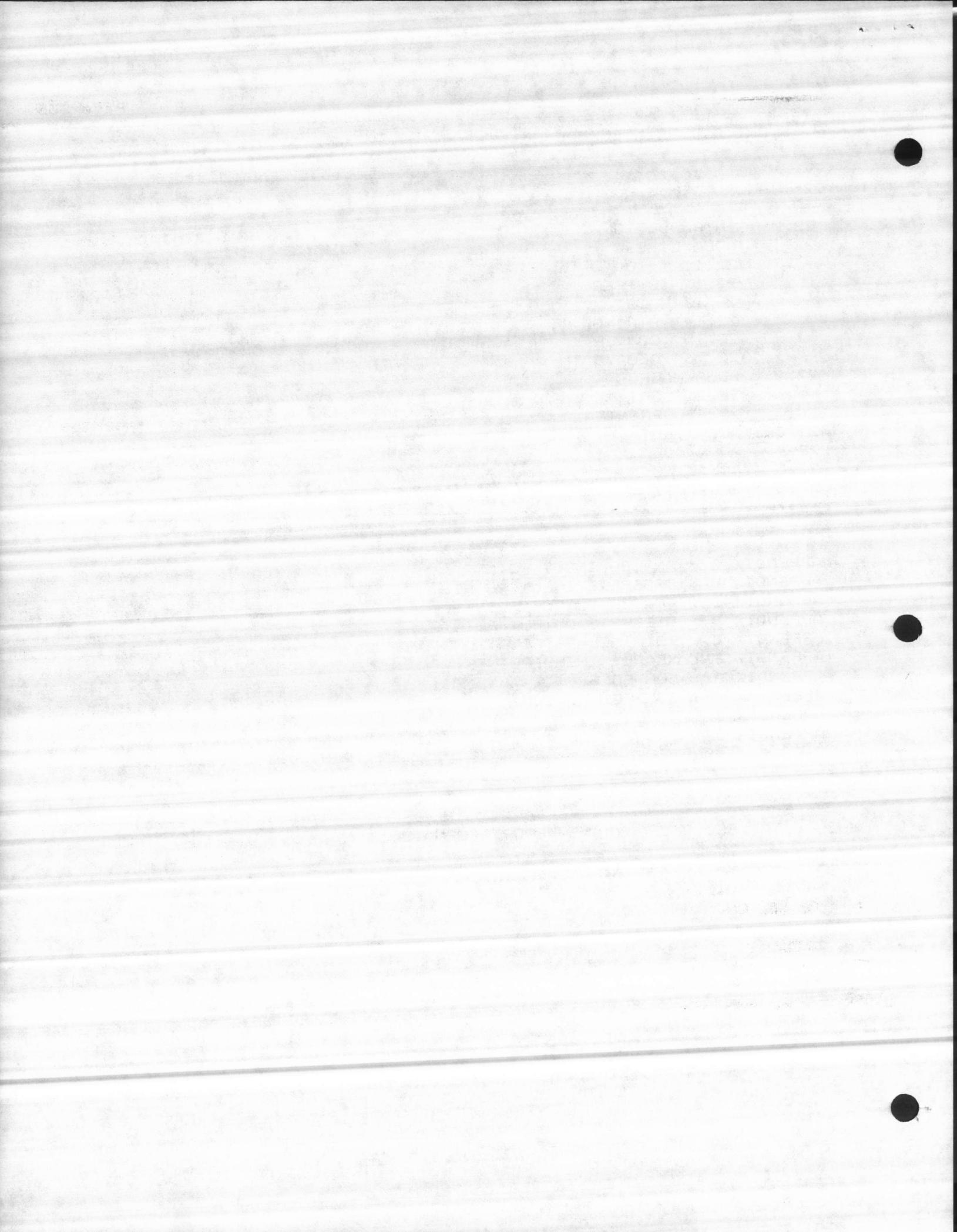
22.0 INTRODUCTION  
22.1 AQUA/LIBSRC  
    22.11 IO  
    22.12 BUFFER  
    22.13 LIST  
    22.14 STRING  
    22.15 CONVER  
    22.16 AQUA/MATH  
22.2 AQUA/MACROS  
    22.21 CRT  
    22.22 PRINTER  
22.3 AQUA/TEST  
22.4 AQUA/JOB  
22.5 AQUA/INC  
22.6 AQUA/LIB

SECTION 23 : APPENDIXES.

APPENDIX A : ANALOG INPUT POINTS.  
APPENDIX B : DISCRETE INPUT POINTS.  
APPENDIX C : ANALOG OUTPUT POINTS.  
APPENDIX D : DISCRETE OUTPUT POINTS.  
APPENDIX E : POINT CLASSIFICATIONS.  
APPENDIX F : YEARLY TREND POINTS.  
APPENDIX G : CONTROLS.  
APPENDIX H : CONTROL LEVELS.  
APPENDIX I : REPORTS.  
APPENDIX J : GROUP DISPLAYS.

SECTION 24 : DIAGNOSTIC NOTES & TESTS.

SECTION 25 : TABLES, DISPLAY AND REPORT FORMATS.



AQUATROL DIGITAL SYSTEMS  
Arden Hills, MN.  
COPYRIGHT 1986

SECTION No. 0

I N T R O D U C T I O N

BY

Mohamed E. Fayad

REVIEWED

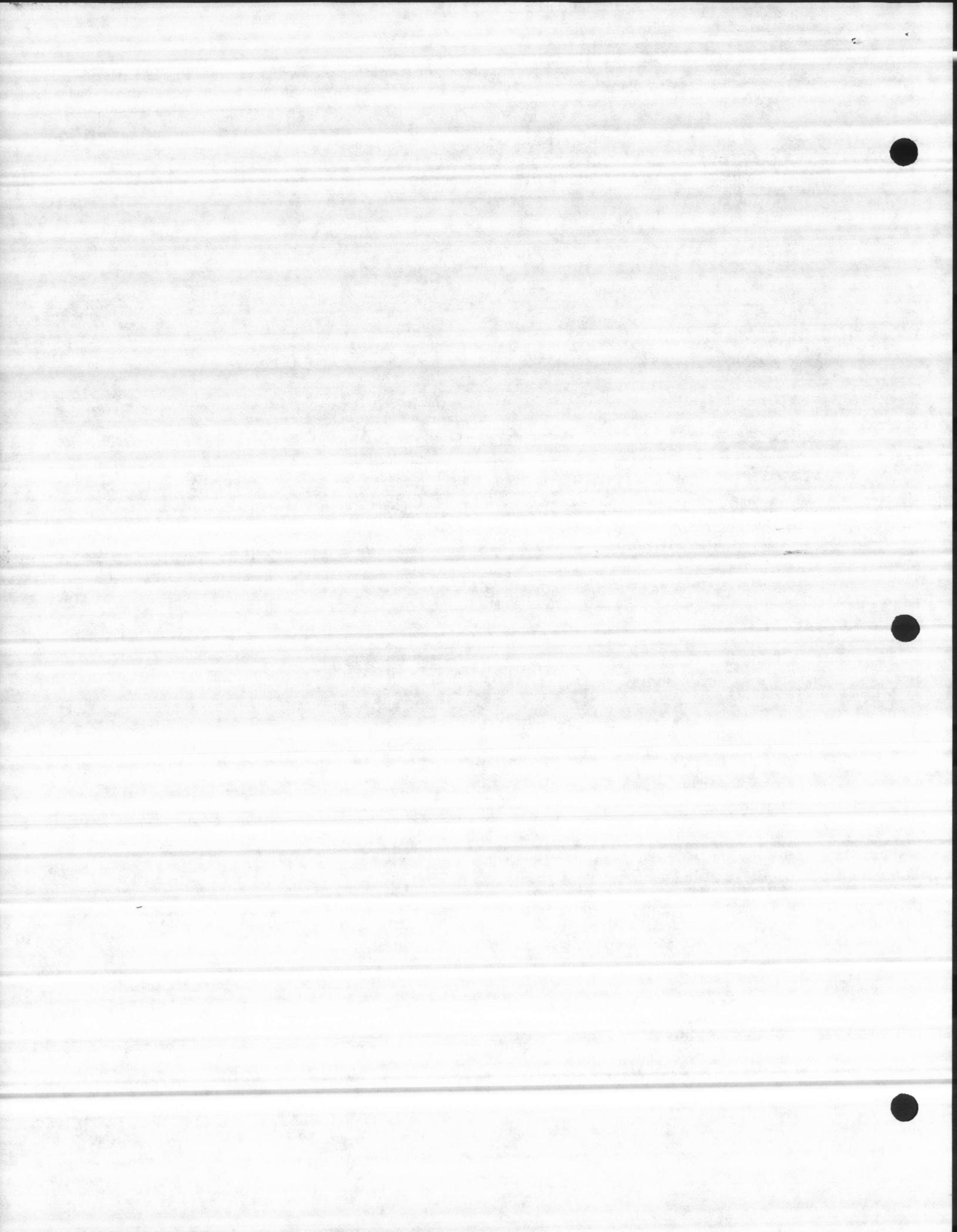
BY

Jerry Knoth

DATE : Oct 10, 1985

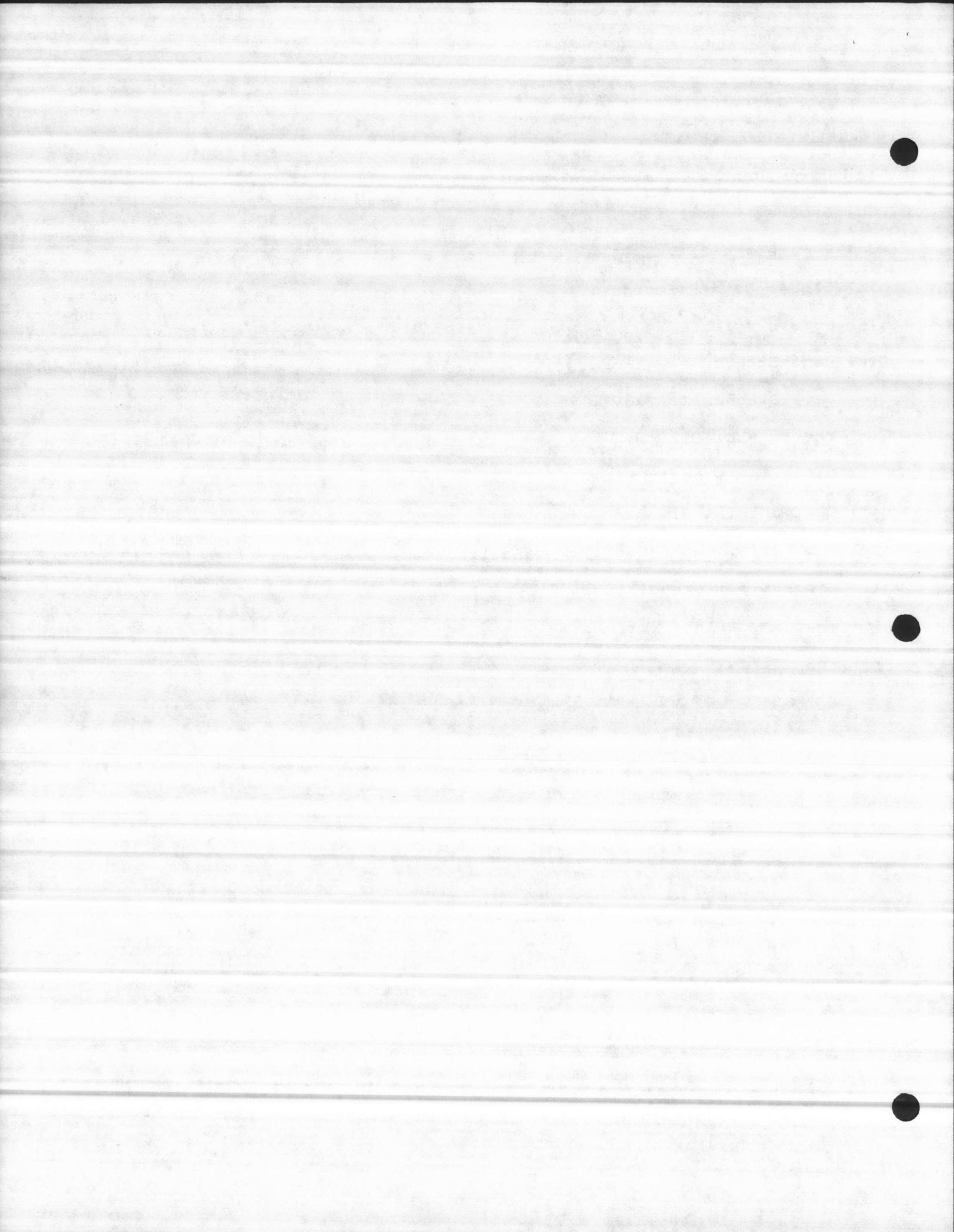
-----  
JOB # : C4758  
-----

HOLCOMB BLVD, W.W.T.P. CAMP LEJEUNE, NC.



0.1 DOCUMENTATION DESIGN :

- =====
- (a) Dewey Decimal Number and Module Name.
  - (b) General Description.
  - (c) Parameters
  - (d) Global Data Reference.
  - (e) Local Variables.
  - (f) Data Structure Introduced.
  - (g) Data Diagram (If any).
  - (h) Module Flow Diagram
  - (i) Algorithm & Logic Flow
  - (j) Text entries Diagram (If any)
  - (k) Who Calls, Entry Codition & Calling Parameters.
  - (l) Error & exit conditions
  - (n) What it calls / Externals?



**0.2 STANDARD FUNCTIONS & TOP\_DOWN DESIGN :**

<b>main group</b>	<b>sub group functions</b>	<b>functions</b>	<b>module</b>
ALARMS & EVENTS	Alarms Display	<ul style="list-style-type: none"> <li>- display alarms</li> <li>- Auto alarms option</li> <li>- Two different alarm condition colors RED for Critical &amp; MAGENTA for Non-Critical</li> <li>- When alarm occurs, the message will blink</li> <li>- Add alarm to the alarm structure</li> <li>- Clear alarm</li> <li>- Alarms saved every 15 min. in the power up file.</li> </ul>	ALRM.Pyy FAIL.Pyy
PRINT ALARMS & PRINT EVENTS		<ul style="list-style-type: none"> <li>- Print alarm or signal event.</li> <li>- Emphasize print for alram msg.</li> <li>- No emphasise print for event message.</li> </ul>	FAIL.Pyy FAIL.Pyy FAIL.Pyy
ALARM SYSTEM		<ul style="list-style-type: none"> <li>- Display alarm system page.</li> <li>- Look up Sonalerts.</li> <li>- Look up Dialers.</li> </ul>	ASYS.Pyy FAIL.Pyy FAIL.Pyy
ALARM LOG		<ul style="list-style-type: none"> <li>- Displays current daily alarms</li> <li>- Prints current daily alarms</li> </ul>	ALOG.Pyy ALOG.Pyy
<b>main group</b>	<b>sub group functions</b>	<b>functions</b>	<b>module</b>
DATA BASE		<ul style="list-style-type: none"> <li>- Displays the menu driver for all the group included in the data base.</li> </ul>	DBAS.Pyy

The data base contains the following:

## 1) ANALOG INPUT POINTS

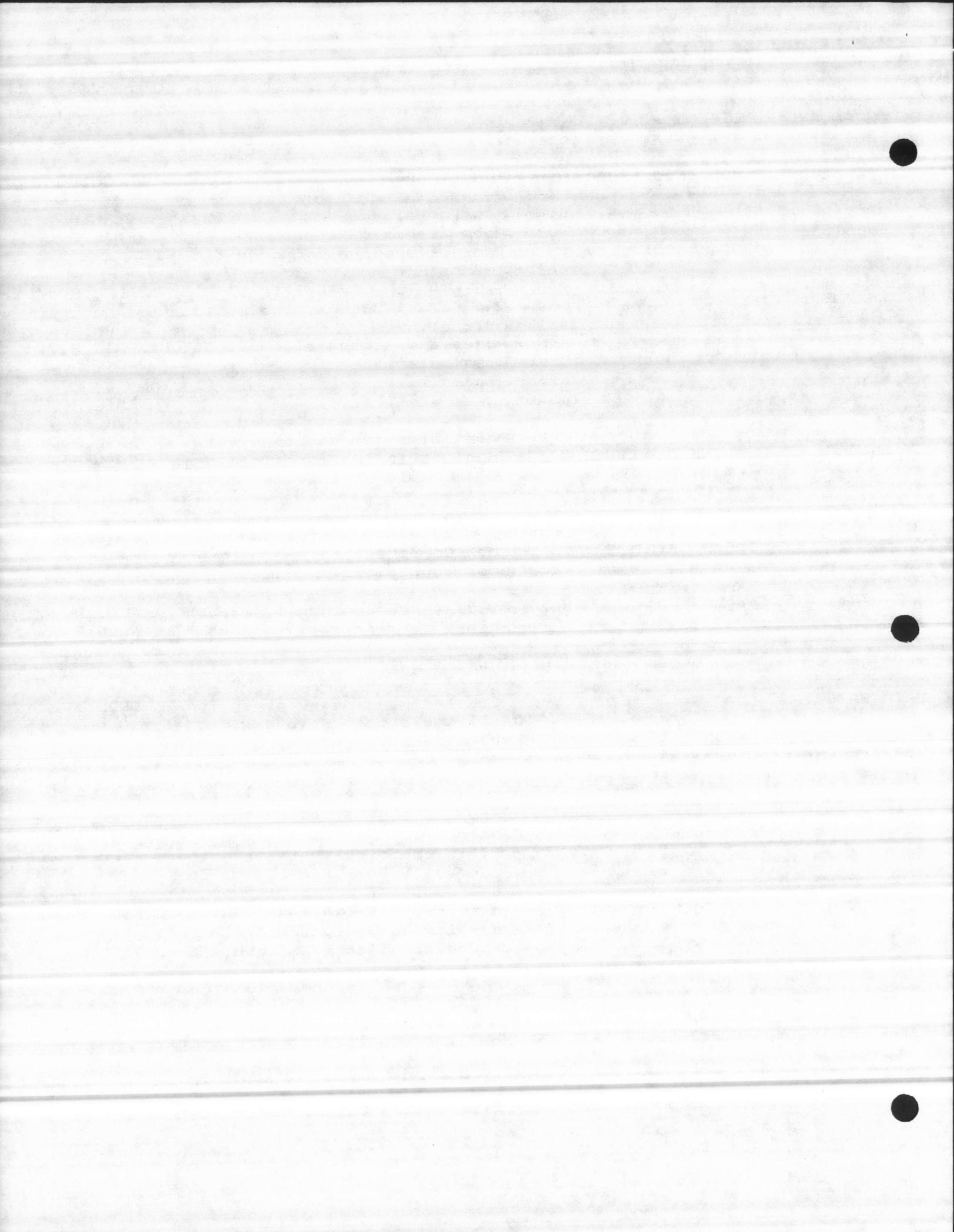
- Displays all the information related to the analog input point.
- Fills the display table with current information for each analog input point.
- (historically) In case of displaying historical file fills the display table with historical data.
- The ability to modify analog input structure by editing the display table at any time.

DBAS.Pyy  
DTBL.Pyy

FILL.Pyy

FILL.Pyy

EDIT.Pyy



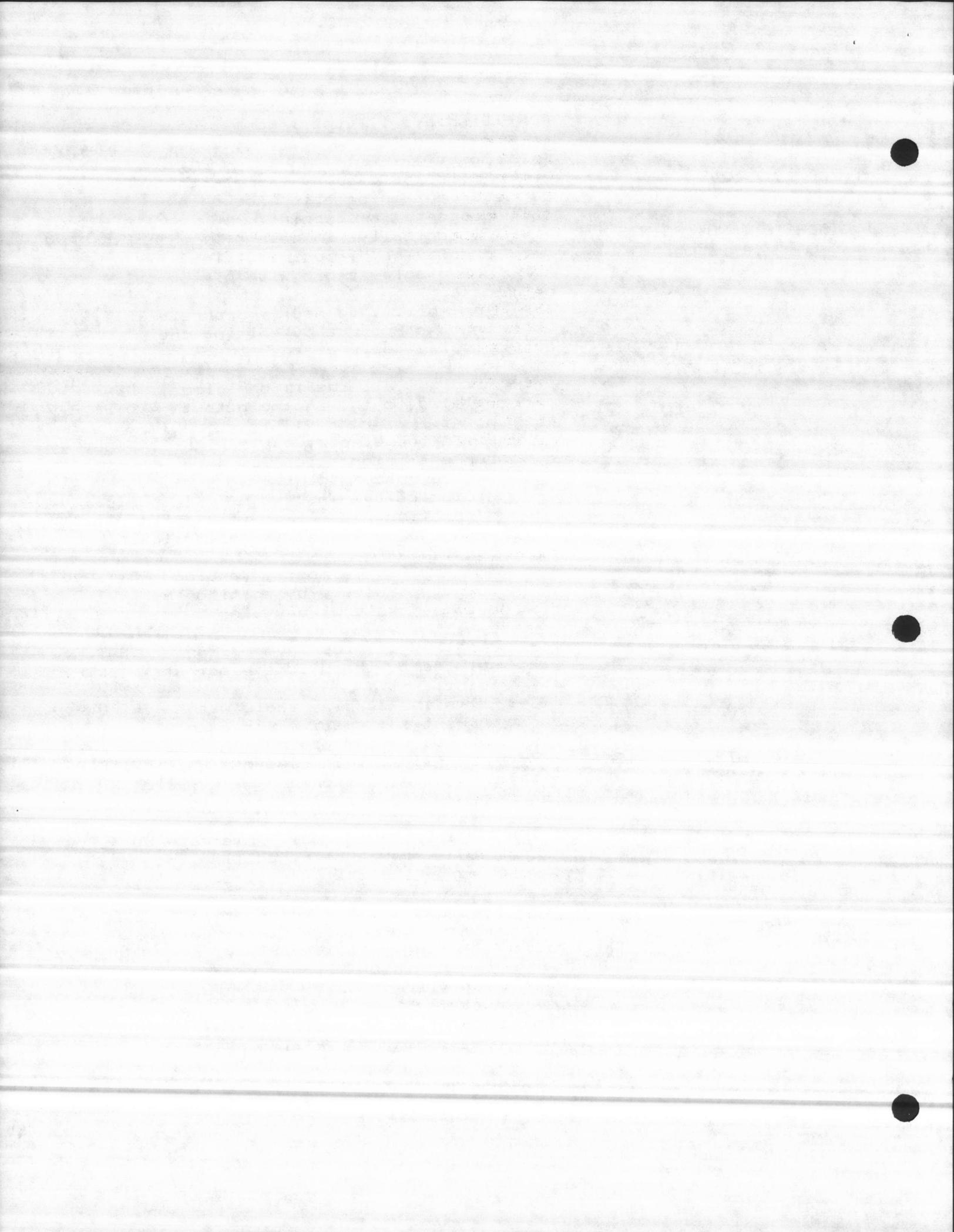
- Displays all the information related to the discrete input point. DBAS.Pyy DTBL.Pyy
  - Fills the display table with current information for each discrete input point. FILL.Pyy
  - (historically) In case of displaying historical file fills the display table with historical data. FILL.Pyy
  - The ability to modify discrete EDIT.Pyy input structure by editing the display table at any time.
- 
- Displays all the information related to the analog output point. DBAS.Pyy DTBL.Pyy
  - Fills the display table with current information for each analog output point. FILL.Pyy
  - (historically) In case of displaying historical file fills the display table with historical data. FILL.Pyy
  - The ability to modify analog EDIT.Pyy output structure by editing the display table at any time.
- 
- Displays all the information related to the discrete output point. DBAS.Pyy DTBL.Pyy
  - Fills the display table with current information for each discrete output point. FILL.Pyy
  - (historically) In case of displaying historical file fills the display table with historical data. FILL.Pyy
  - The ability to modify discrete EDIT.Pyy output structure by editing the display table at any time.
- 
- Displays all the information related to the year trend point. DBAS.Pyy DTBL.Pyy
  - Fills the display table with current information for each yearly trend point. FILL.Pyy
  - (historically) In case of displaying historical file fills the display table with historical data. FILL.Pyy
  - The ability to modify yearly trend structure by editing the display table at any time. EDIT.Pyy

## 2) DISCRETE INPUT POINTS

## 3) ANALOG OUTPUT POINTS

## 4) DISCRETE OUTPUT POINTS

## 5) ANALOG YEARLY TRENDS



## 6) CONTROLS

- Displays all the information related to the control section. DBAS.Pyy
- Fills the display table with current information for each control section.
- (historically) In case of displaying historical file fills the display table with historical data.
- The ability to modify control section structure by editing the display table at any time.
- The ability to alternate one or two pumps and to assign pumps to any step in the control section.

## 7) LEVEL CONTROL POINTS

- Displays all the information related to the controlled pump. LVEL.Pyy
- Fills the display table with current information for each controlled pump.
- (historically) In case of displaying historical file fills the display table with historical data.
- The ability to modify pump point structure by editing the display table at any time.

THE DATA BASE SYSTEM could include as many points as desired, also the data base can be expanded to hold any number of data groups. By calling our sale department for additions.

more data base groups:

REAL POINTS  
 LAB DATA POINTS  
 CALCULATION POINTS  
 IO PM550 POINTS  
 IO GOULD POINTS .... ETC.

===== ALL THE ABOVE MODULES ARE COMBINED INTO ONE MODULE =====

DBAS.Pyy

main group

sub group functions

functions

module

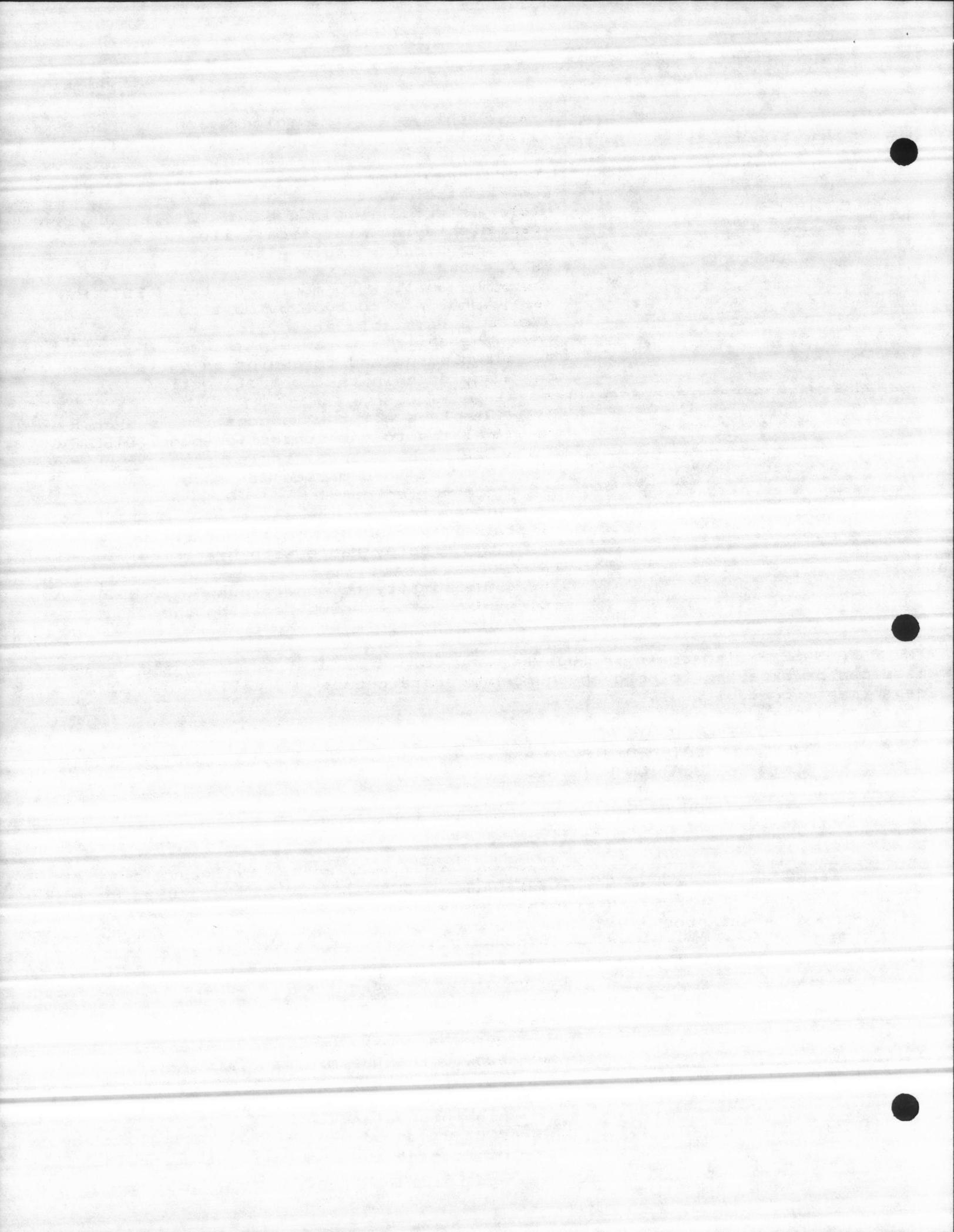
MONITORING

&

CONTROLS

Monitor  
 Analog In.

- Activate/Deactivate Option ANIN.Pyy
- Using the scan time to scan through the analog input points.
- Monitoring the value changes in every scan.
- Monitoring the min/max value in every scan.
- check for low low /low/high high high alarms.
- BCD/HEX options
- Monitoring the analog input according to the source.



Monitor  
Discrete In.

- Activate/Deactivate Option. DCIN.Pyy
- Using the scan time to scan through the discrete input point.
- Monitoring the sonalert, computer pulse signal (computer Not Fail/Fail).
- Monitoring the value change in each scan.
- Monitoring the number of the starts.
- Monitoring the data fail signal.
- Monitoring the discrete input point according to the source.

Monitor  
Analog Out.

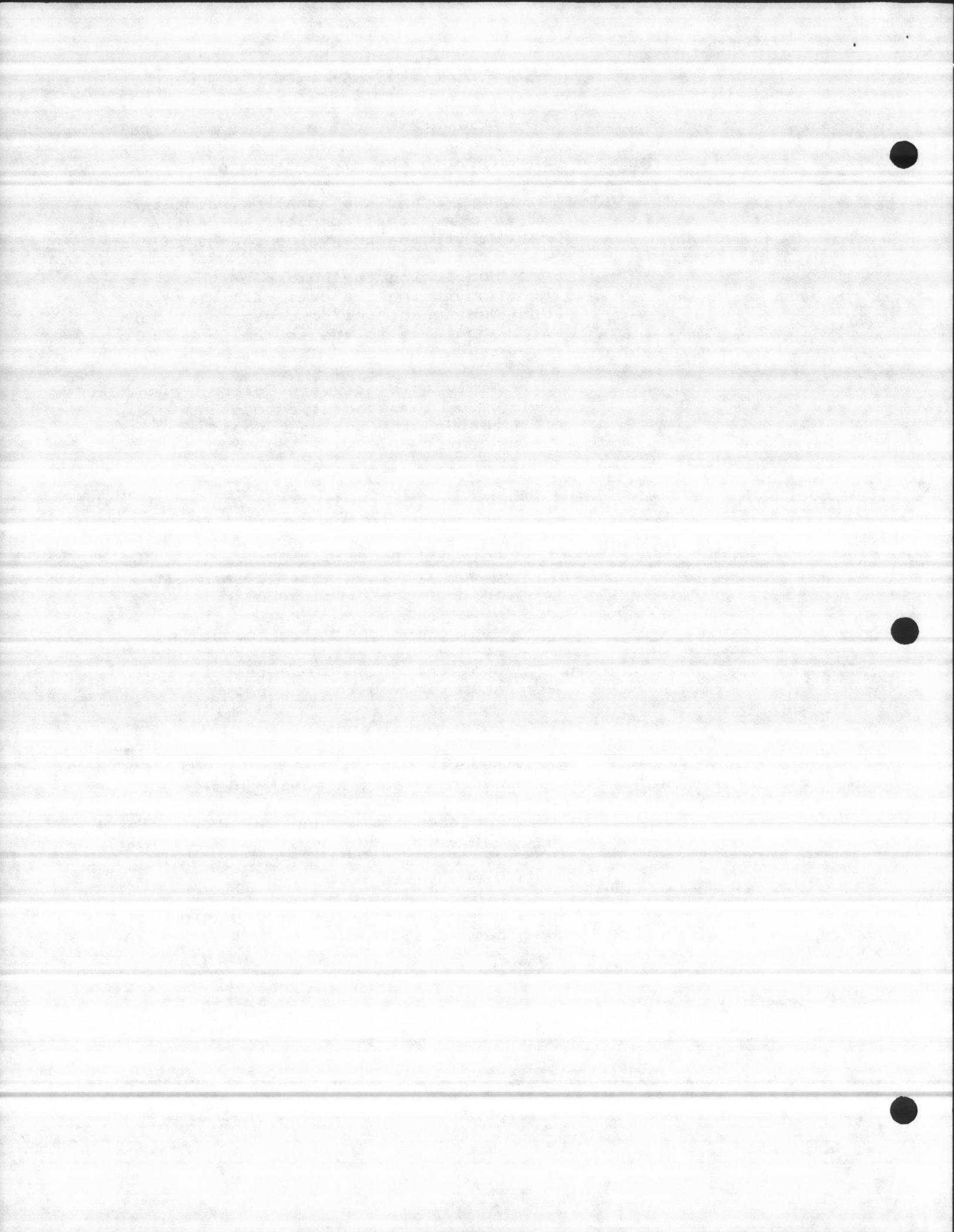
- Activate/Deactivate Option. ANOP.Pyy
- Using the scan time to scan through the analog output point.
- Monitoring the value change in each scan.
- Monitoring the analog output pt according to the source.

Monitor  
Discrete Out.

- Activate/Deactivate Option. DCOP.Pyy
- Using the scan time to scan through the discrete output point.
- Monitoring the value change in each scan.
- Monitoring the discrete output pt according to the source.

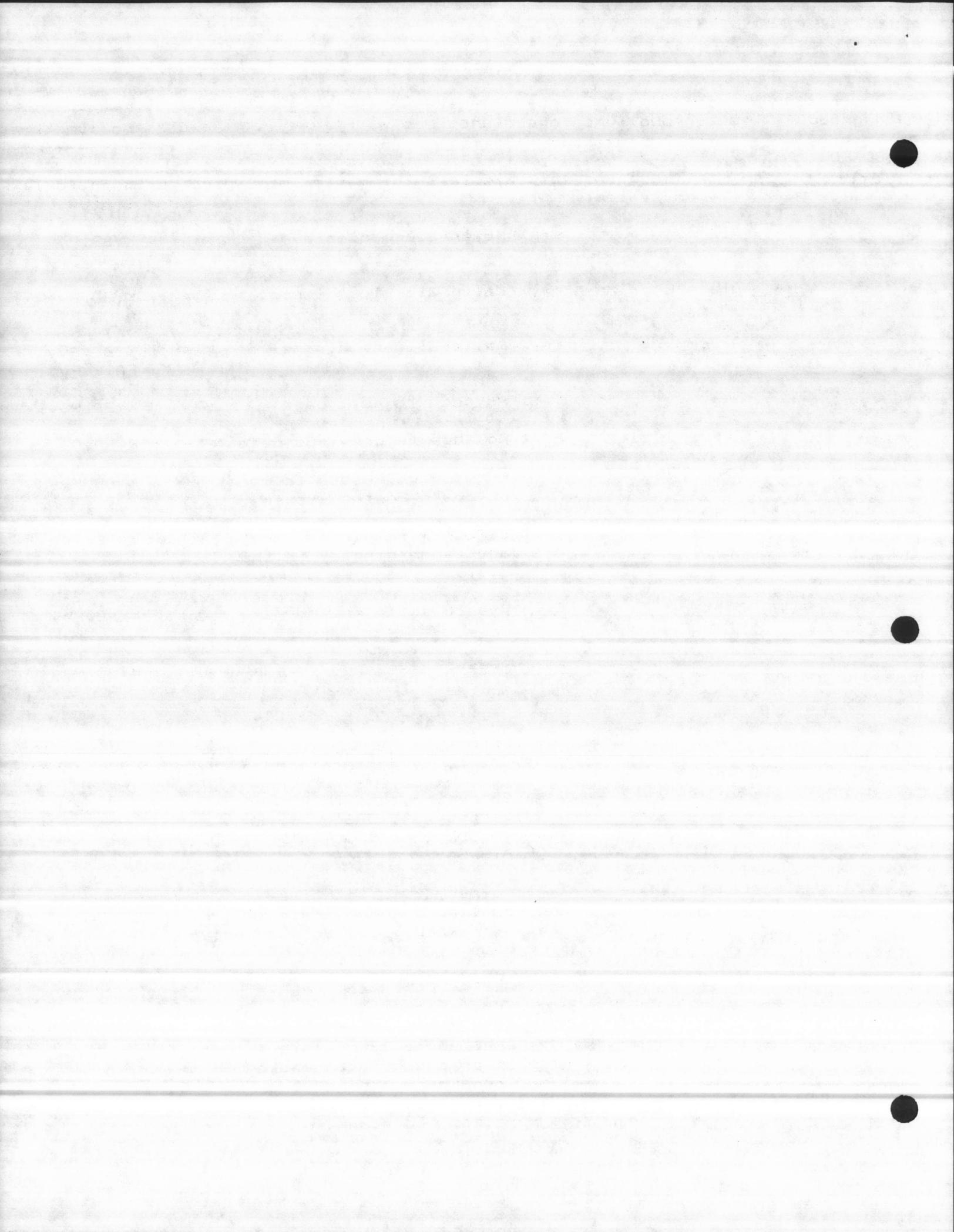
Controls

- Auto/manual control. PCON.Pyy
- Monitoring max # of steps with CTRL.Pyy setpoints. KYBD.Pyy
- Perform start a pump manually & Auto.
- Perform stop a pump manually & Auto.
- Perform select the control point
- User definable.
- Managable.
- Perform alternate more than one pump
- Perform matrix control using as many pumps as you wish (10 maximum).



main group	sub group functions	functions	module
BOARD			
	_ the keyboard task displays the master menu page designed for your system.		
	Alarms	Keyin alarm display	KYBD.Pyy ALRM.Pyy
	Alarm Log	Keyin alarmlog display	KYBD.Pyy ALOG.Pyy
	Alarm Sil	Keyin silence alarms	KYBD.Pyy FAIL.Pyy
	Next	Advance to next page	KYBD.Pyy
	Previous	Advance to previous page	KYBD.Pyy
	Page num	Enter Page Number to locate any page in the system	KYBD.Pyy
	Select Id	Select Point Id to locate any taged display in the system	KYBD.Pyy
	Master Menu	Displays the master menu	KYBD.Pyy
	Keyboard Task	=====	KYBD.Pyy

main group	sub group functions	functions	module
DISPLAY GENERATOR			
		To design, construct and display any GROUP DISPLAY, TREND DISPLAY or GRAPHIC DISPLAY.	
	CRT Generator	<ul style="list-style-type: none"> <li>- An easy menu interface to construct a group display, a trend display or a graphic display.</li> <li>- Access CRT Display &amp; Print File</li> <li>- Cross Hair</li> <li>- Displays Graphic Parameters.</li> <li>- Draw up to 27 graphic symbols.</li> <li>- Graphic generator editor.</li> <li>- Trend generator editor.</li> <li>- Construct Screen.</li> <li>- Screen management.</li> </ul>	GGEN.Pyy EDIT.Pyy GRPH.Pyy LINE.Pyy  GGEN.Pyy



Display Generator - Key display current group menu DISG.Pyy  
 - Keyin display historical group menu.  
 - display the constructed group, trend graphic.

Edit Screen - Full screen editor used for developing user definable CRT display. ESCR.Pyy

Display Generator Static DGEN.Pyy

**main group**  
-----  
**REPORT GENERATOR**

<b>sub group functions</b>	<b>functions</b>	<b>module</b>
Report Generator	To design, format and construct any report or data log. - An easy menu interface to construct a report	REPG.Pyy EDIT.Pyy
Print A Report To The Line Printer	- Read report disk File. - Save disk file. - Construct report. - Screen management.	PRTG.Pyy
Edit Screen	- Full screen editor used for developing user definable formatted report.	ERPT.Pyy

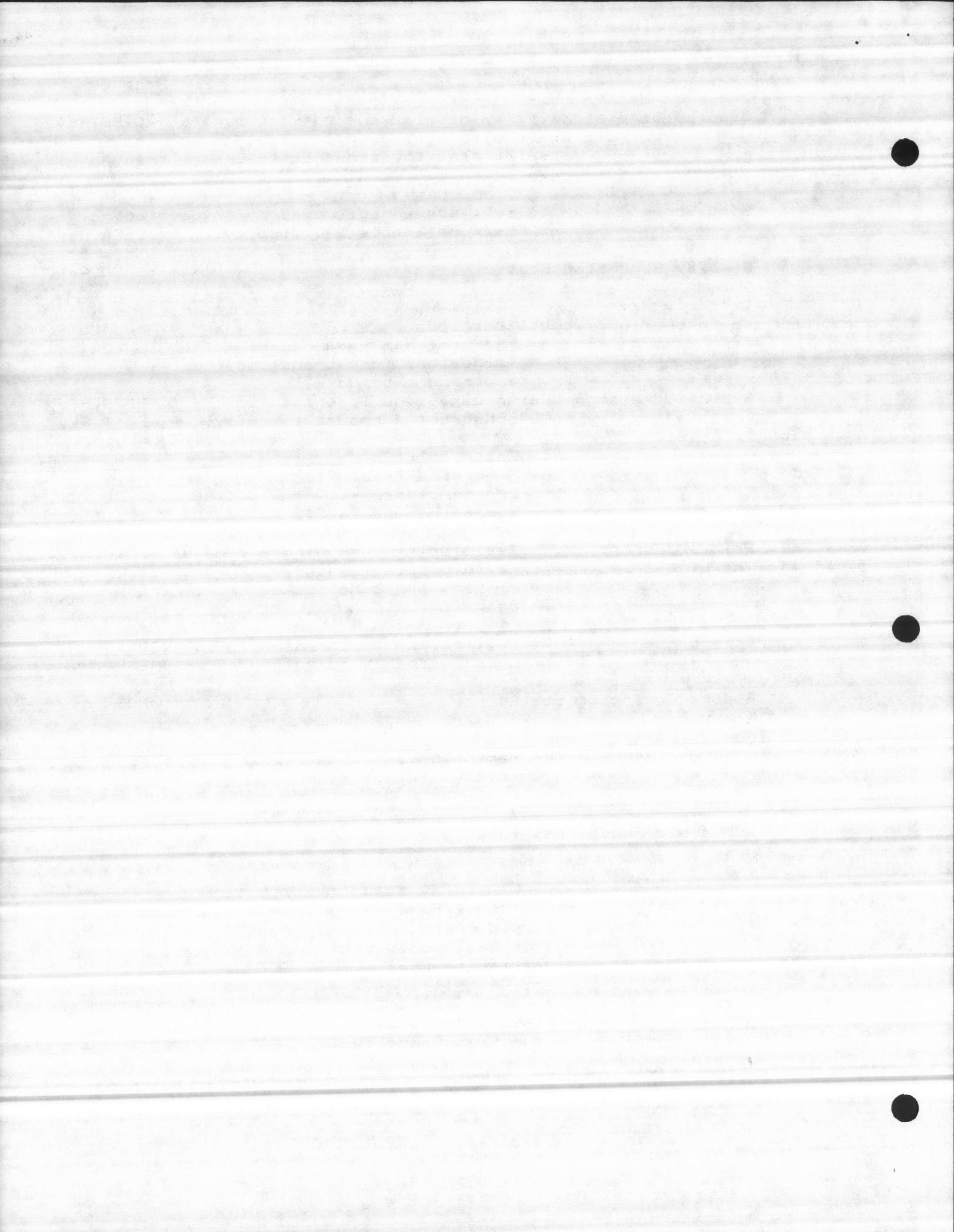
Report Generator Static RGEN.Pyy

Periodic Report Option

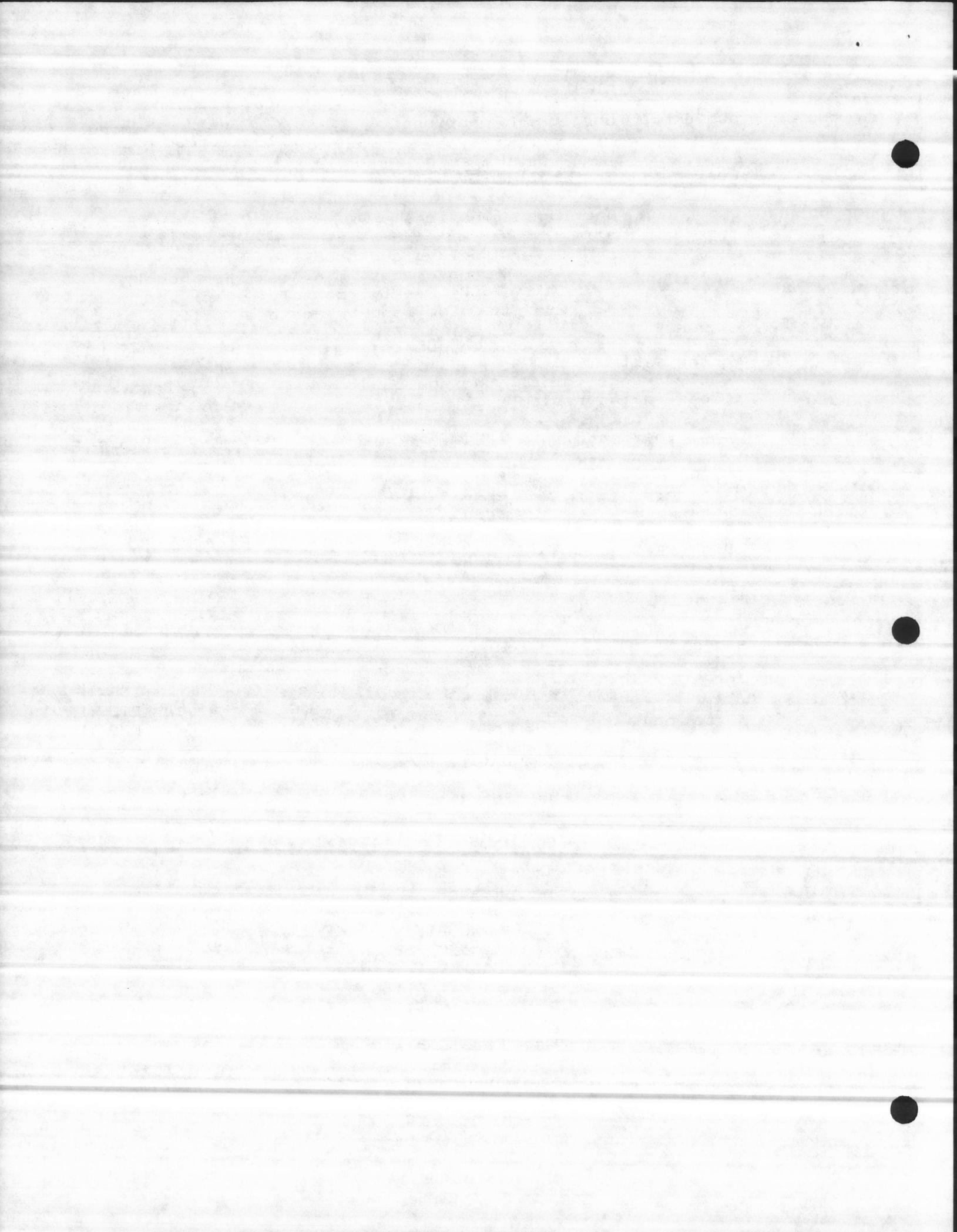
- Periodic task with a base timer, PERD.Pyy interval(cycle timer), print timer and enable/disable periodic flag.

**main group**  
-----  
**SCREEN EDIT SYSTEM**

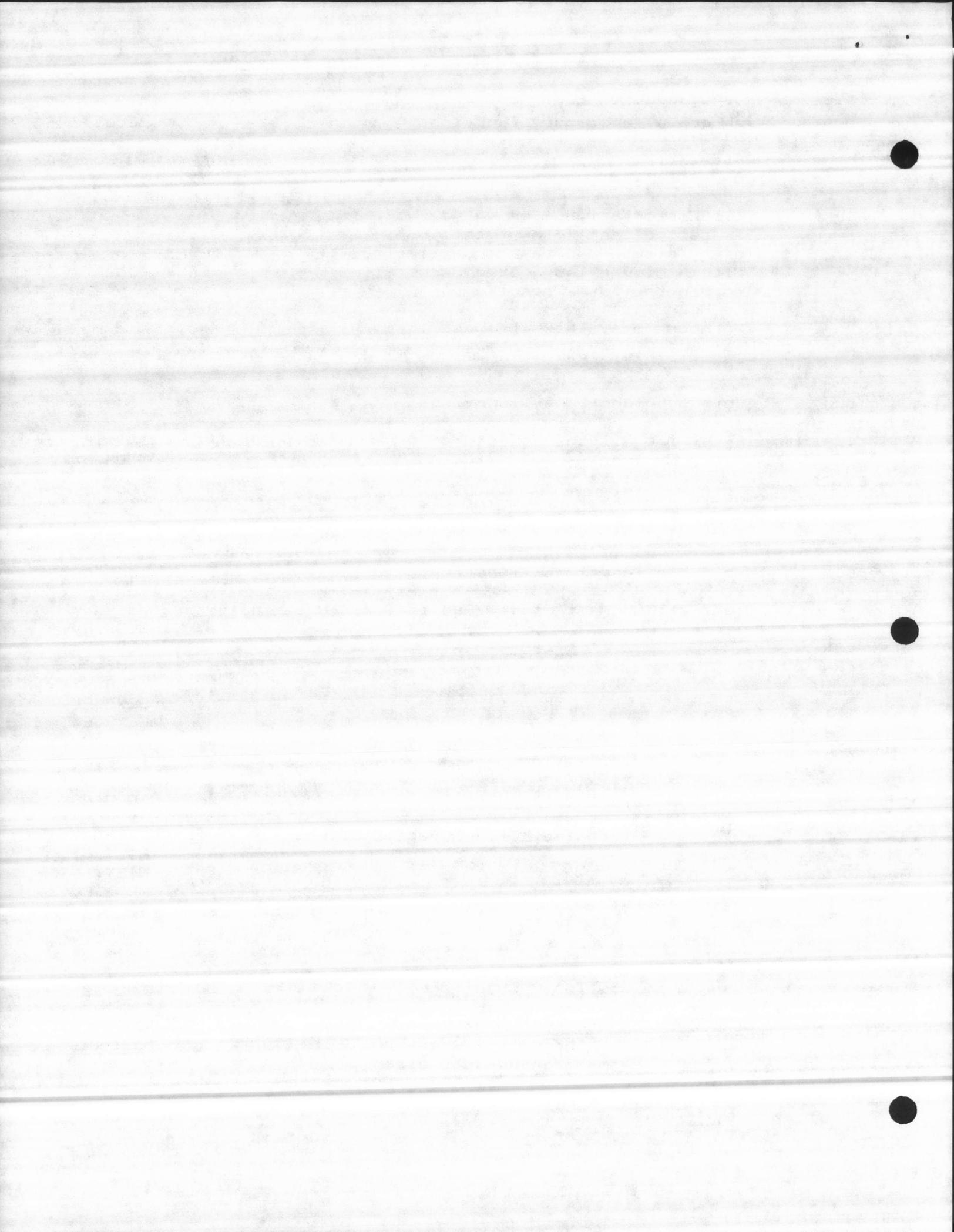
<b>sub group functions</b>	<b>functions</b>	<b>module</b>
Screen Editor	- Display case - Edit Case - Display Table - Edit Table - Print Change in the numeric and strings. - Select Field - Edit Field - Read Character.	EDIT.Pyy DCAS.Pyy ECAS.Pyy DTBL.Pyy ETBL.Pyy CPRM.Pyy SFLD.Pyy EFLD.Pyy
Edit Time		
Edit New Password		



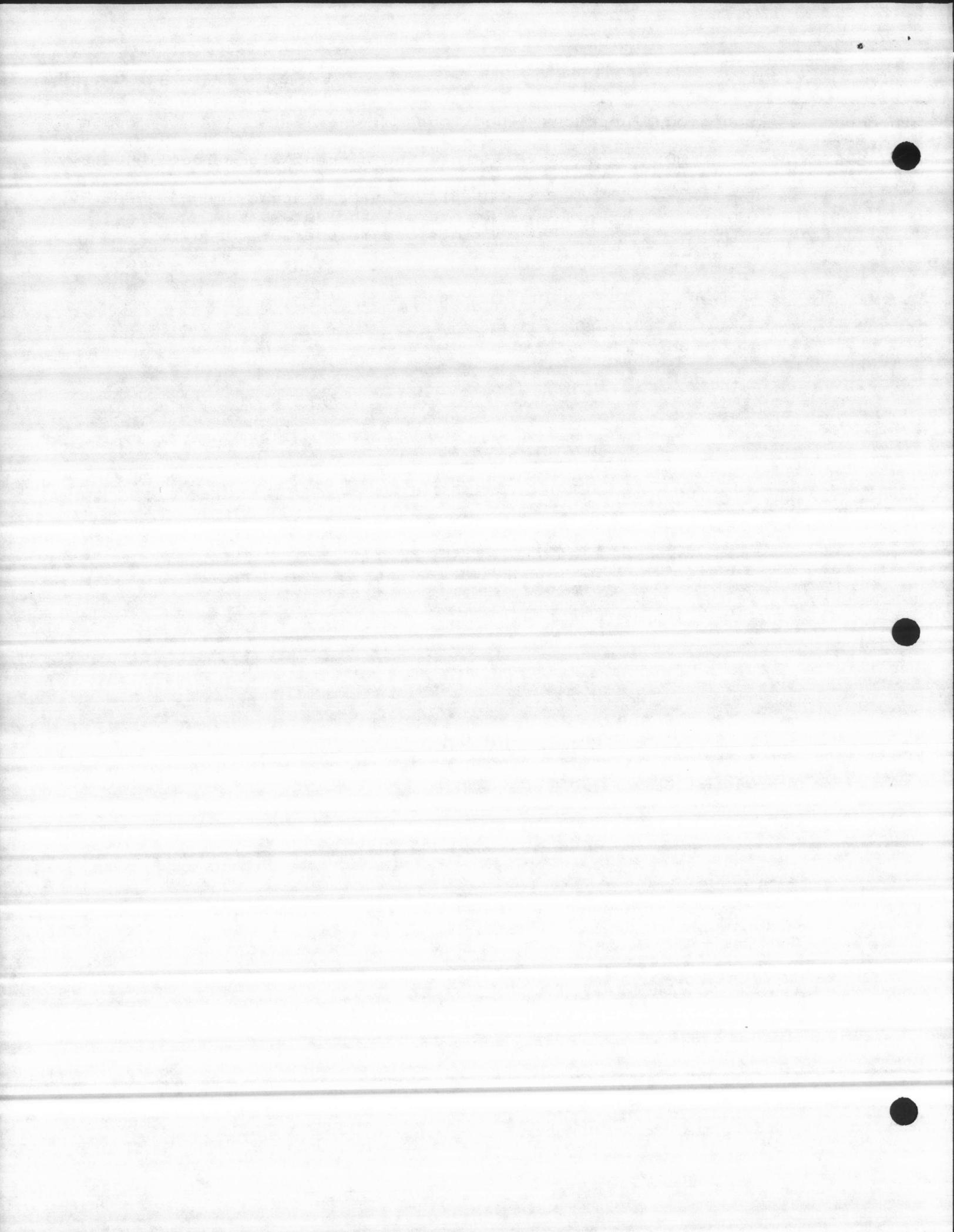
main group	sub group functions	functions	module
GRAPHICS		Design all The Graphic Symbols	GRPH.Pyy GRPH.EXT
main group	sub group functions	functions	module
TRENDS		displays and holds numeric values for averages, min, max. For hourly Daily, weekly, monthly, yearly. user defined trends are included also.	
		Trends For all the analog points in the system.	TRND.Pyy
		Yearly trends for all the analog points in the system	LINE.Pyy
		Draw Trend	
		Draw line trend.	DLGR.Pyy LNGR.Pyy
main group	sub group functions	functions	module
HISTORICAL		Data Base Alarms Displays Trends Group displays Historical reports	DBAS.Pyy ALOG.Pyy DISG.Pyy TRND.Pyy DISG.Pyy REPG.Pyy HSTM.Pyy
main group	sub group functions	functions	module
TEST SYSTEM		Display I/O rack  Display Master Page Display Remotes Reset Alarms in the entire system.	IORK.Pyy DIOR.Pyy DMST.Pyy DREM.Pyy TEST.Pyy
main group	sub group functions	functions	module
DISK & FILE SYSTEM	DISK	- Create Directory - Delete File - Write File - Read File - Write Block - Read Block - Read Directory Entry - Copy File - Format Floppy - Rename File - Add Serial - Read Serial - Copy File	DISK.Pyy
	FILE	- Display directory for the harddisk and floppy. - Rename File - Format Floppy - Load File - Delete File - Save File	FILE.Pyy



main group	sub group functions	functions	module
-----	-----	-----	-----
TOTAL SYSTEM	Write File Copy File Link Report Fill Historical Buffer Six second task (flow total) Thirty six second task (run time total) Fifteen Minute Task (save Pwrup File) Two Minute Task (Trend) One Hour Task (trends) Two Hour Total Task (trends) Near End Of Day Task End Of Day Task		TOTL.Pyy
-----	-----	-----	-----
main group	sub group functions	functions	module
-----	-----	-----	-----
INITIALIZATION	Initailize All the structure Create Regions Create Segments Create Directory Read Power Up File Create Tasks		INIT.Pyy MODS.EXT
-----	-----	-----	-----
main group	sub group functions	functions	module
-----	-----	-----	-----
DATA GLOBAL & EXTERNAL	Holds all Current Global Public Structures in the system.		GLOB.Pyy GLOB.EXT TPUB.Pyy TPUB.EXT GEXR.Pyy GEXR.EXT TABL.PUB TABL.EXT
-----	-----	-----	-----
main group	sub group functions	functions	module
-----	-----	-----	-----
HISTORICAL GLOBAL & EXTERNAL	Holds all Historical Global Public Structures in the system.		HIST.Pyy HIST.EXT HPUB.Pyy HPUB.EXT HEXR.Pyy HEXR.EXT
-----	-----	-----	-----
main group	sub group functions	functions	module
-----	-----	-----	-----
OPTION SYSTEM	Displays all the system options (Auto Options) for daily, weekly, monthly, alarms log and auto alarm display.		OPTP.Pyy



main group	sub group functions	functions	module
POINT SYSTEM	ALARMLOG	- Prints the current alarm log.	ALOG.Pyy
	Data Base	- Displays an easy access menu - Prints all the data base - Prints one member of the data base	PBAS.Pyy



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 1

L I T E R A L S

BY

Mohamed E. Fayad

REVIEWED

BY

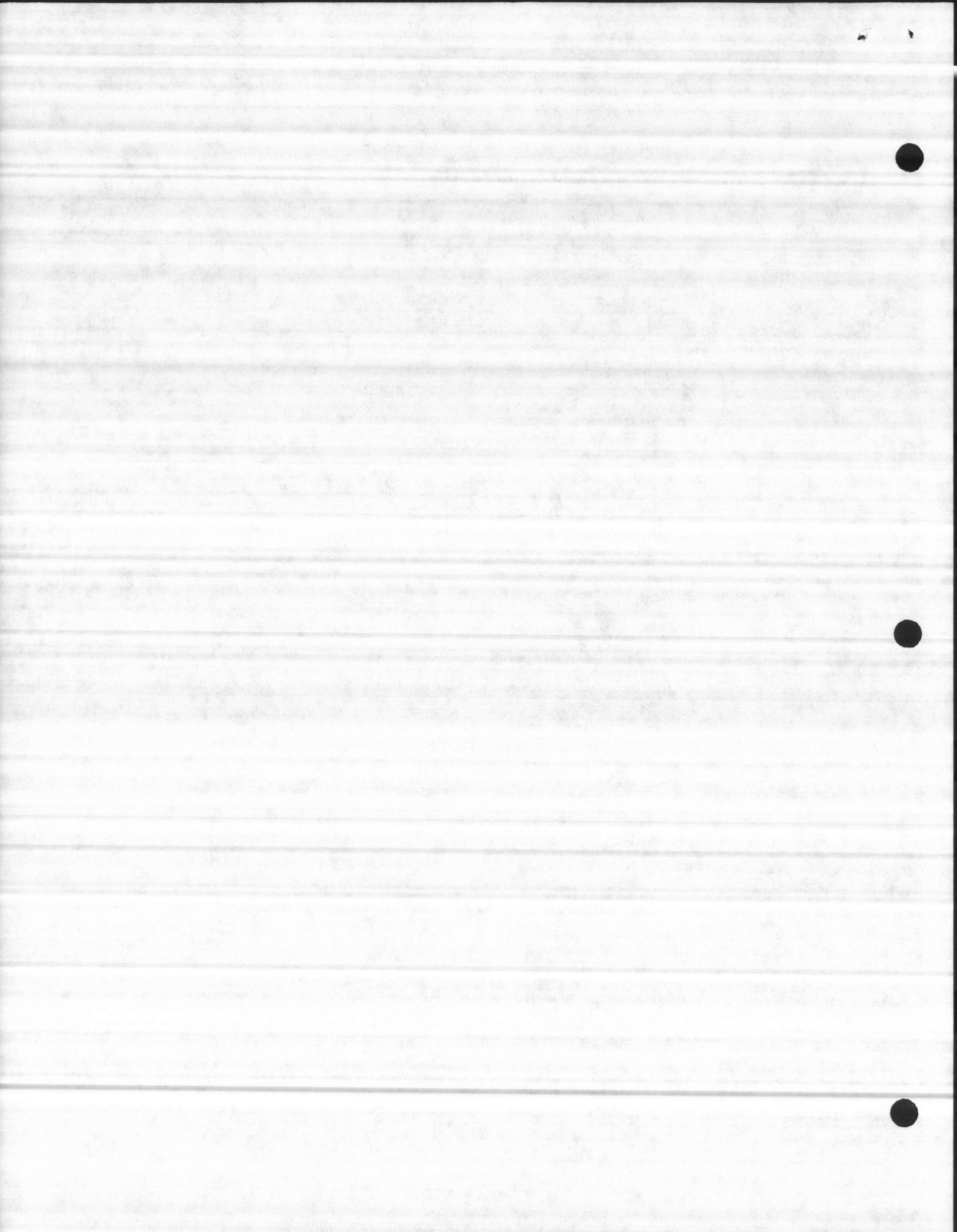
Jerry Knoth

DATE : Oct 10, 1985

J # : C4758

HOLCOMB BLVD, W.W.T.P. CAMP LEJEUNE, NC.

The literals - Written by Mohamed Fayad.



1.0 INTRODUCTION :

=====

Defines literals for the system. The literals differ from system to another. In this document, the literals for Camp Lejeune system will be defined as follow.

1.1 MODULE NAME : Cxxxx.LIT

=====

- I / O LITERALS -

The number of the i/o ports is 24 ports for the hardware representation.

The max number of remotes in the system is 24 remotes.

- DATA BASE LITERALS -

The number of the analog input points plus spares for future use is 38 analogs.

The number of the discrete input points plus spares for future use is 360 discretes.

The number of the analog output points plus spares for future use is 1 analog output. (This system has no analog outputs).

The number of the discrete output points plus spares for future use is 120 discrete outputs.

The max number of level controls in the system is 60 points.

The max number of controls in the system is 5 points.

The max number of step pumps in the system is 30 pumps.  
(Which means you can assign 30 pumps to any control.)

- GROUP DISPLAY & REPORT LITERALS -

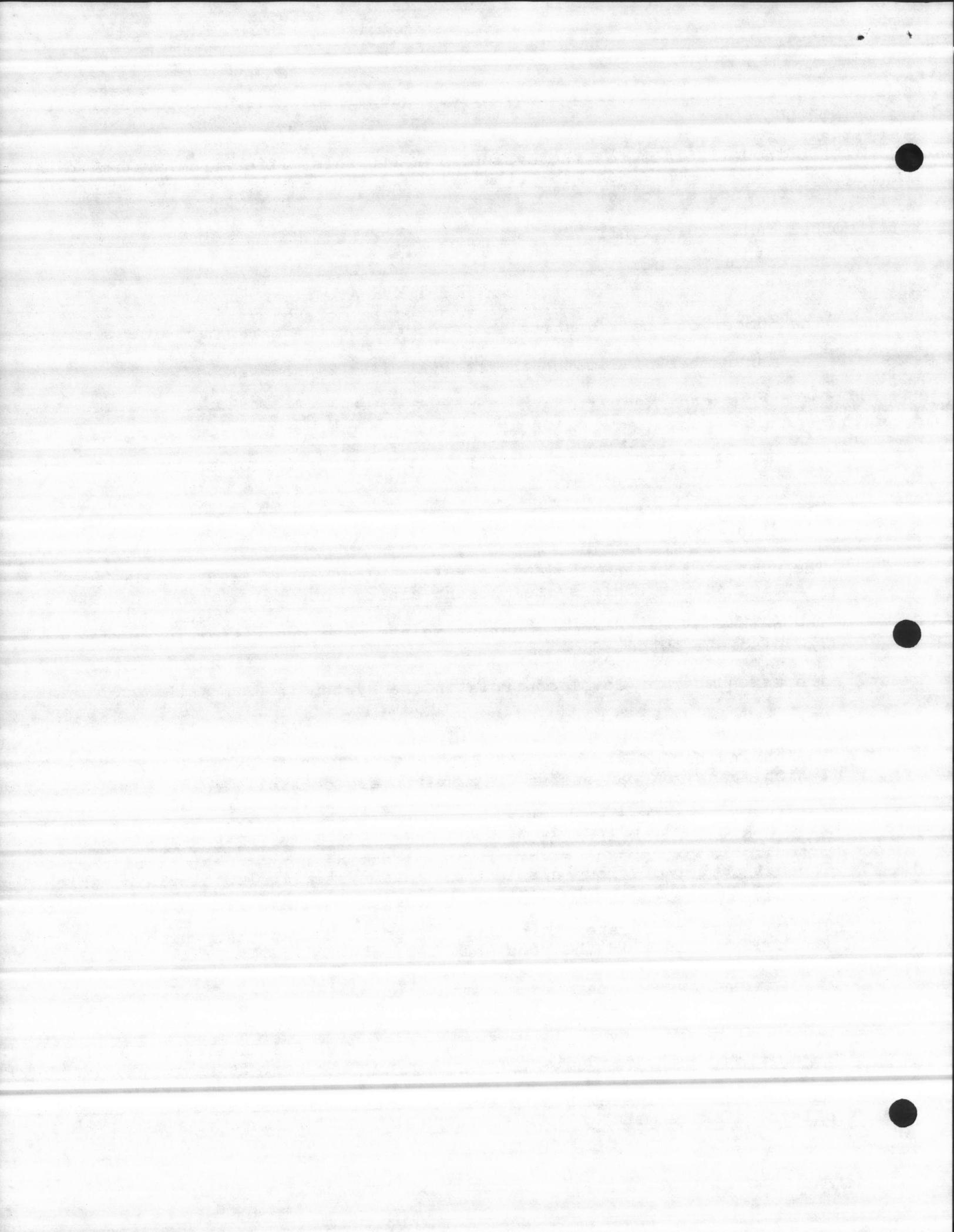
The max number of group displays in this system is 100 displays. which you can construct in the construct display.

The max number of reports in this system is 25 reports. which you can construct in the construct report.

- POWER UP FILE LITERALS -

The max number of elements in the power up file is 24 elements. which is written to the hard disk. The power up file name is CxxxxPWRUP.

The max number of elements is 401 elements. which is used to create a large disk segment.



- O T H E R L I T E R A L S -

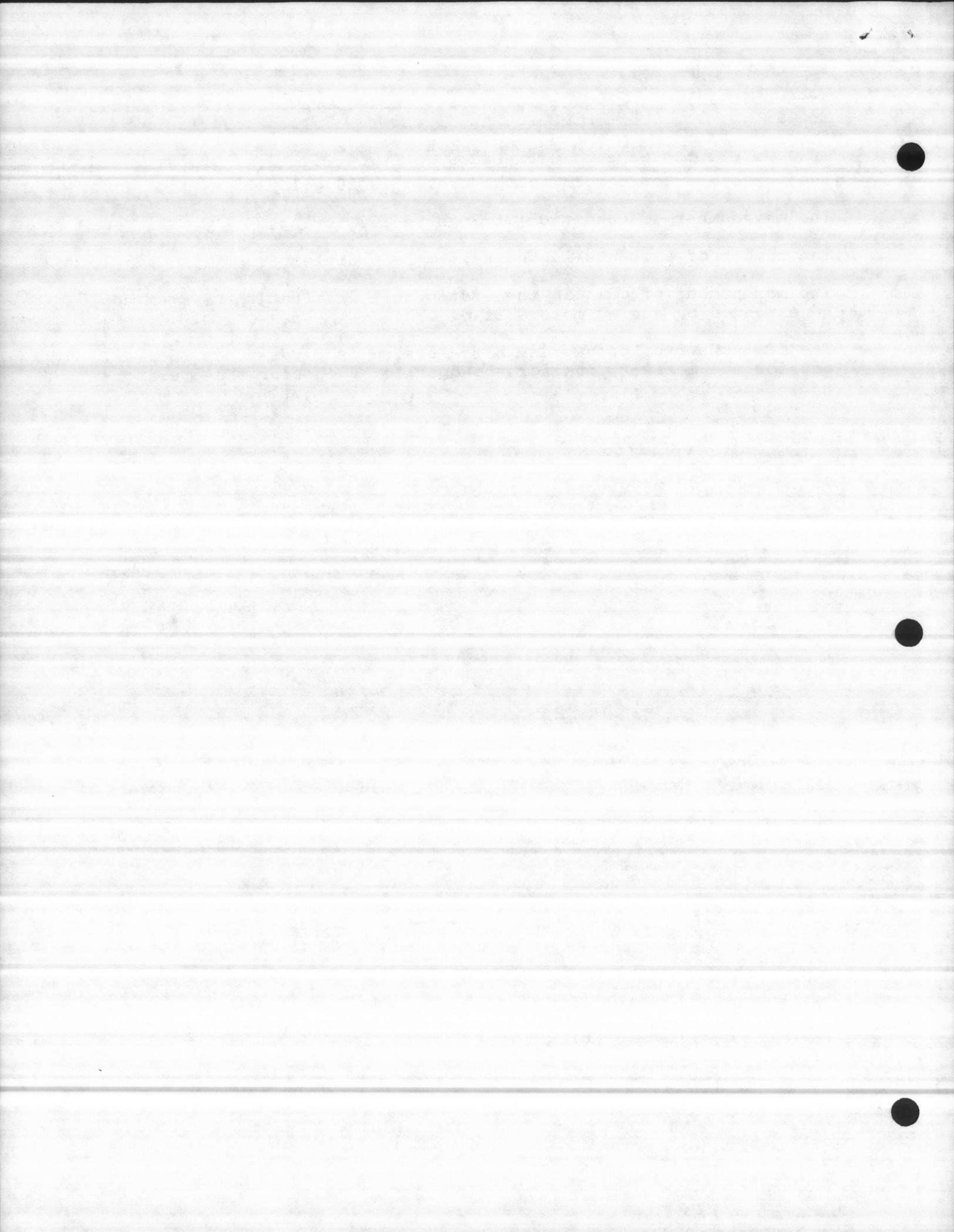
The max number of colors in the system is 16 colors.

The max number of condition strings is 100 strings.

The number of elements in the system is 1 dialer.

The max number of CRTs in the system is 1 CRT (but the system could be expanded by adding more CRTs or LPs).

The max number of passwords in the system is 3 passwords (For ex: one for the manager, one for the operator & one for the operator assistance.).



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 2

G L O B A L

AND

E X T E R N A L

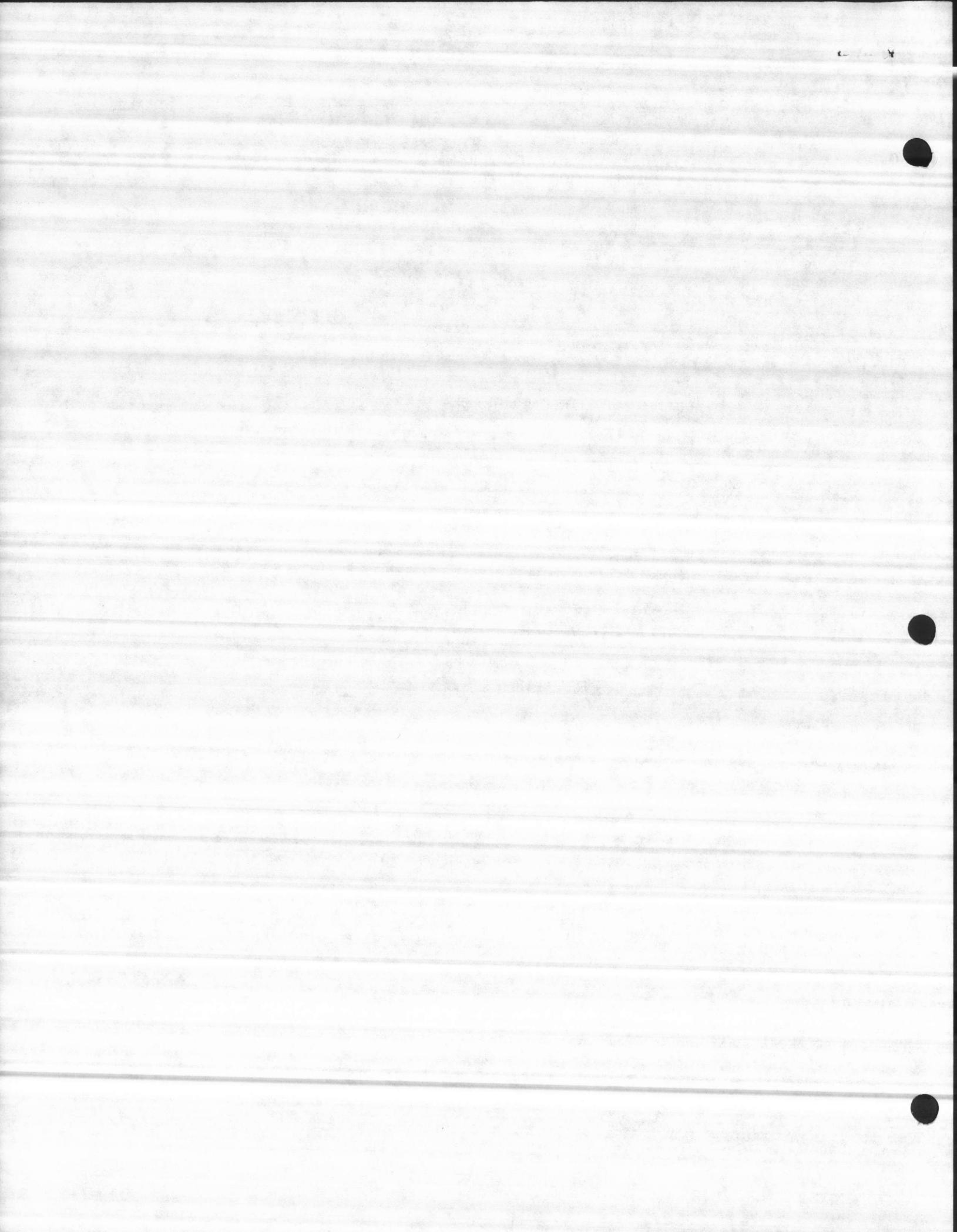
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



INTRODUCTION :

Global and external files hold all the public and external variables, tokens and structures

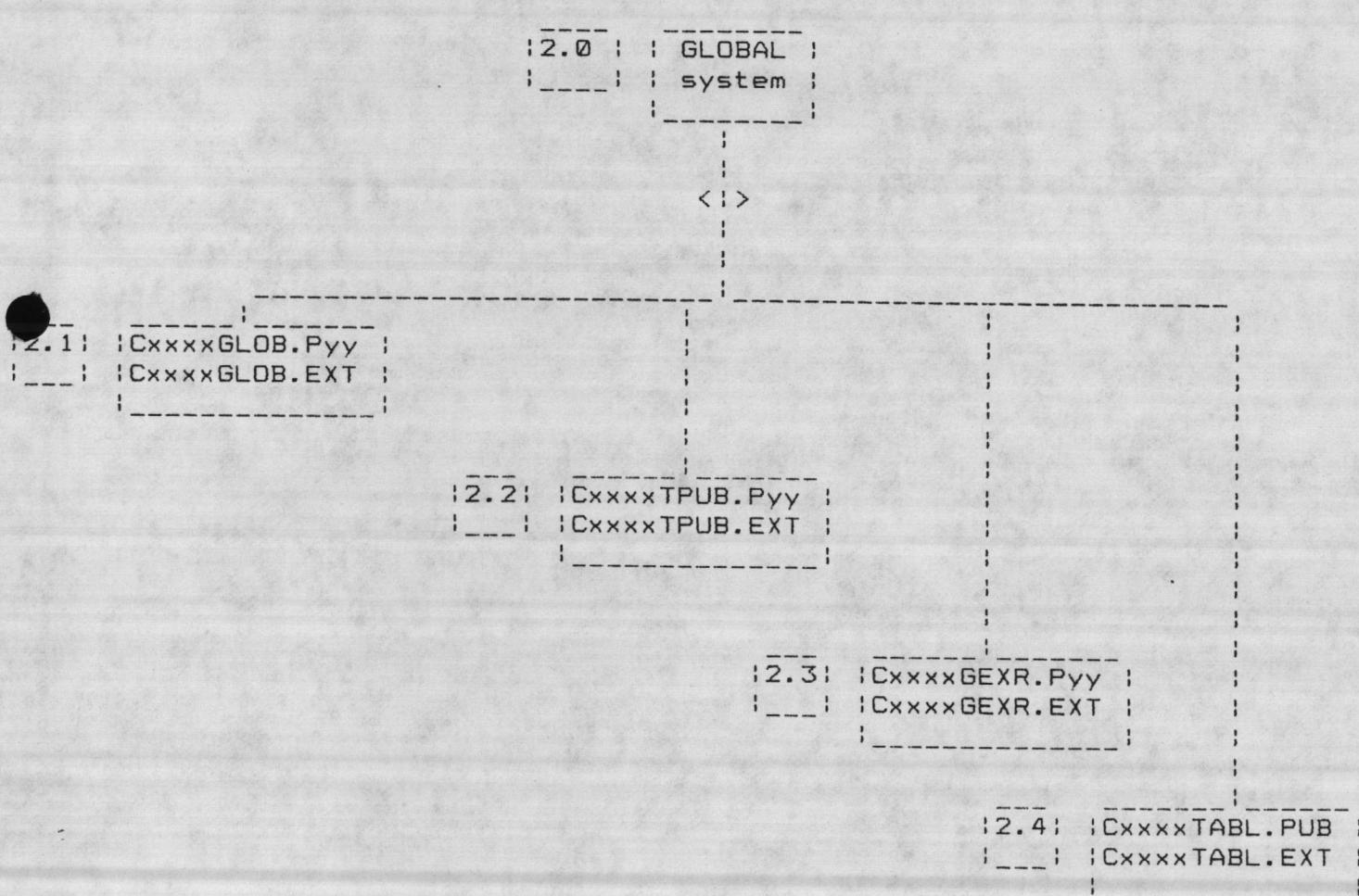
DATE : Oct 10, 1985

JOB # : C4758

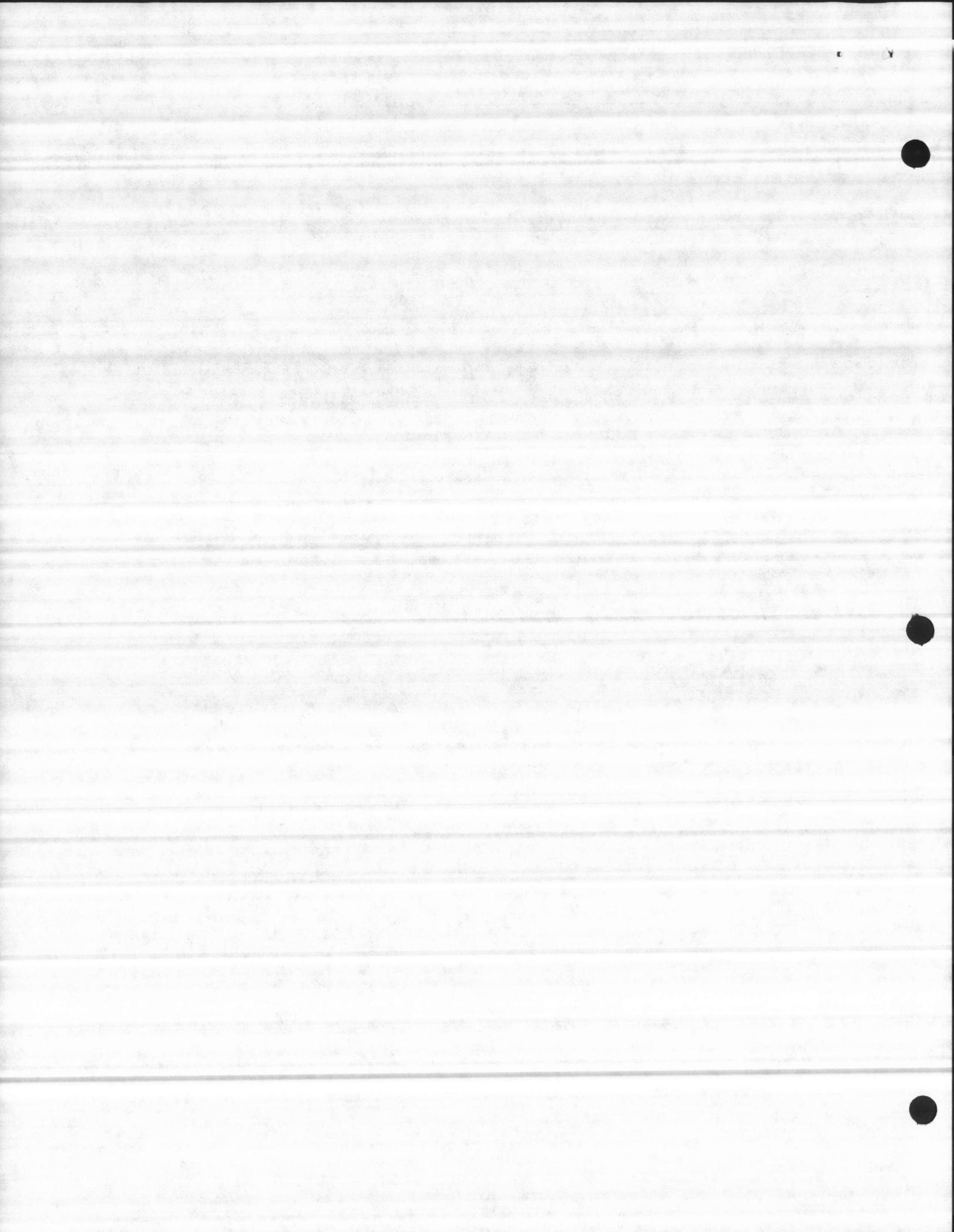
HOLCOMB BLVD, W.W.T.P. CAMP LEJEUNE, NC.

The glob & external - written by Bob Ryan, Mohamed Fayad, Peter Wollenzien.

GLOBAL SYSTEM :



Where Cxxxx = C4758 & Pyy = P86



(2.1) MODULE NAME : CxxxxGLOB.Pyy & CxxxxGLOB.EXT

=====

DESCRIPTION :

=====

The module CxxxxGLOB.Pyy holds all the public data structure, variables, and tokens of the following and the module CxxxxGLOB.EXT holds all the external data of the CxxxxGLOB.Pyy :

1) I/O RACK VARS, STRUCTURE.

2) CRT DISPLAY AND EDITOR VARIABLES.

3) PRINTER VARIABLES.

4) DISK VARIABLES.

5) ALARM VARIABLES.

6) FAIL INTERFACE VARIABLES.

7) TOTAL PROC VARIABLES.

8) POWER UP FILE SAVE TIME.

9) SYSTEM PASSWORD.

10) SYSTEM OPTION VARIABLES.

11) GROUP DISPLAY STRUCTURE.

12) REPORT STRUCTURE.

13) SYSTEM CONDITION STRUCTURE.

14) DISCRETE INPUT.

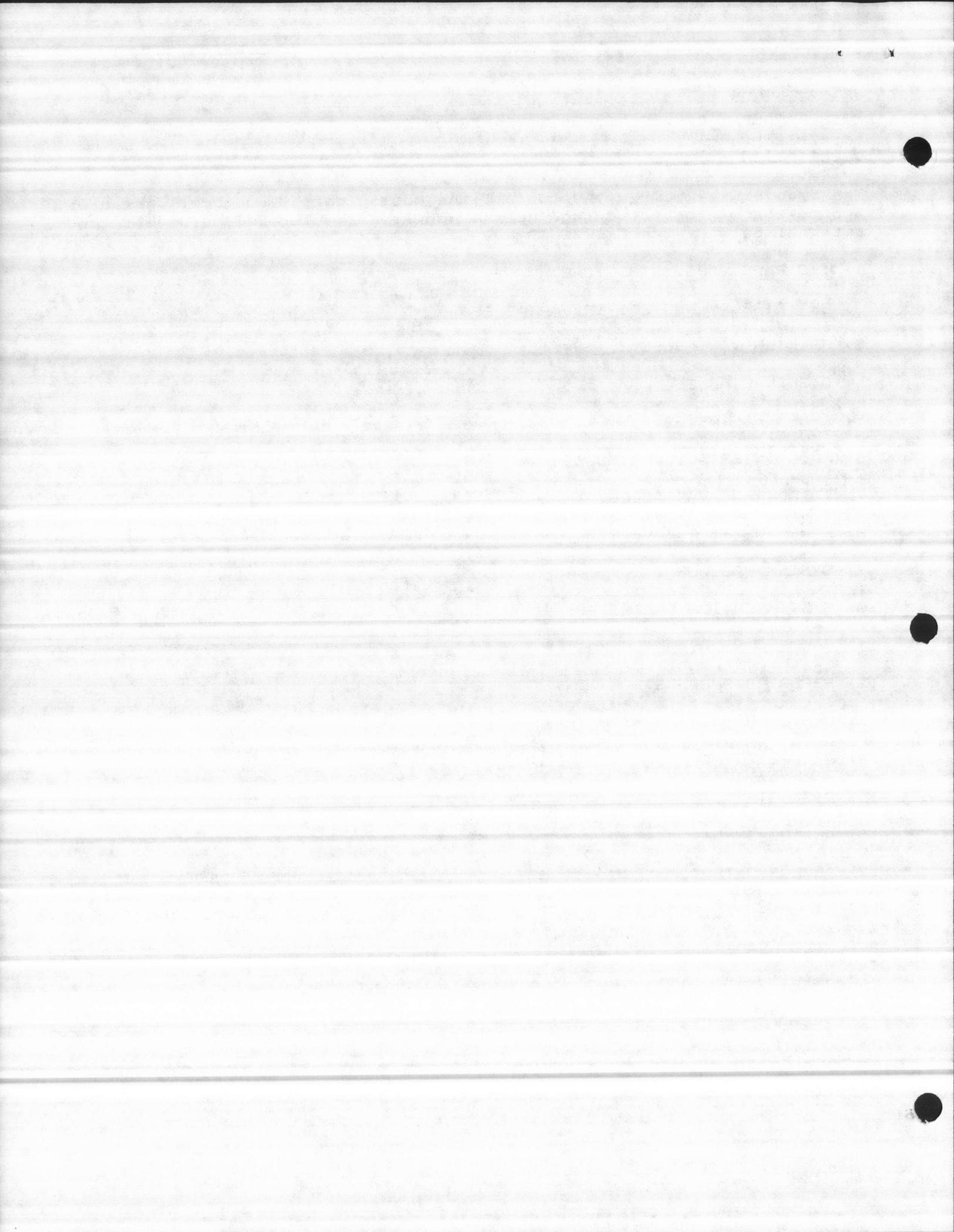
15) DISCRETE OUTPUT.

16) ANALOG INPUT.

17) ANALOG OUTPUT.

18) FILE NAME STRUCTURE.

19) DISK HEADER DECLARATION.



---



---



---

(1) I/O RACK PUBLIC VARIABLES, TOKENS, AND STRUCTURES

---



---

```

DECLARE io_ds_reg_t          TOKEN PUBLIC;
DECLARE (input_rack_list_t,
        output_rack_list_t) TOKEN PUBLIC;

DECLARE io_ports (NUM_IO_PORTS_PLUS1) STRUCTURE
    (mode           WORD,
     scan_time      WORD,
     scan_time_count WORD,
     data_change    WORD,
     current_value  WORD) PUBLIC INITIAL (0, 0, 0, 0, 0);

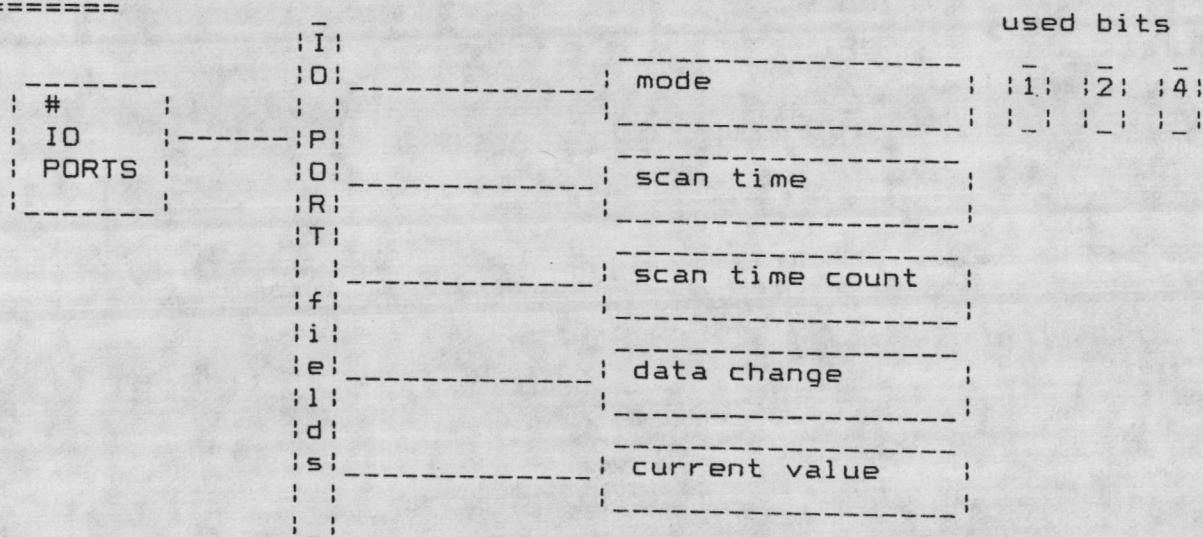
```

STRUCTURE SIZE = 10 \* 25 = 250 BYTES.

---

DATA DIAGRAM :

---



DESCRIPTION :

---

io\_ds\_reg\_t - A token, used for a region protection.

input\_rack\_list\_t - A token, used for list entry in input.

output\_rack\_list\_t - A token, used for list entry in output.

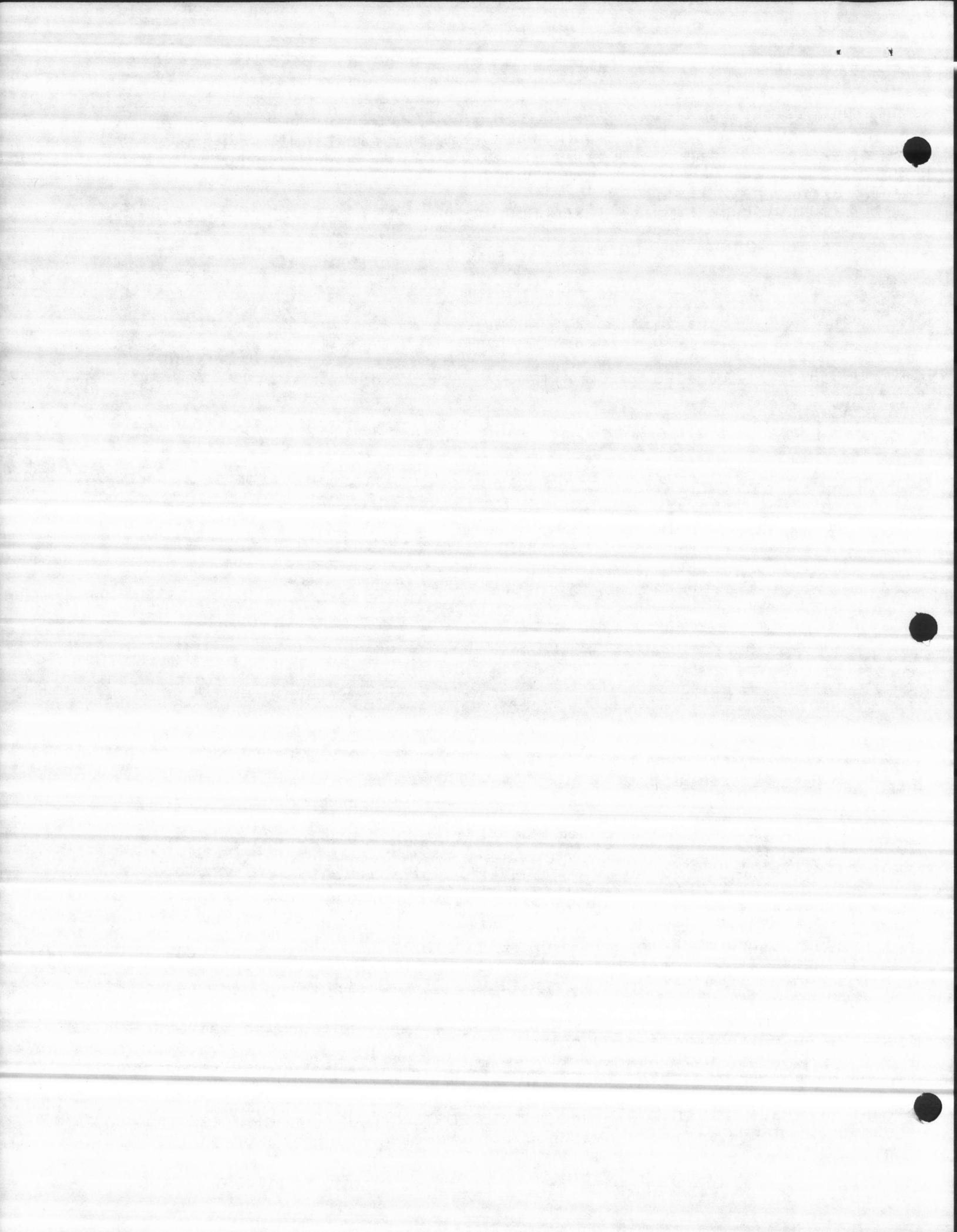
I/O Card Rack Interface:

---

All hardware card rack interfacing is done with an independent task. This task runs off of a tabular data structure which allows the user to define many of the parameters associated with a single io rack port.

The total io card rack system consists of a standard io card rack data structure, an editable standard crt display screen, and input/output task. The standard input/output task will use cross-reference indices to access data in the primary io data structure.

Global literals UNM\_IO\_PORTS and NUM\_IO\_PORTS\_PLUS1 need to be defined for every system. These literals are used to dimension the primary io data structure.



The entries in the io\_ports data structure have the following meaning:

- ode* - A word value whose bit have the following meaning:

	0	1
1H	non-existent	/ port exist
2H	input port	/ output port
4H	auto input	/ manual input

- scan\_time* - A word value holding the number of seconds an input is to be read or an output is to be refreshed. An output will be ported any time its data\_change flag is set. This scan time on an output will be used primarily for refreshing of the data.
- scan\_time\_count* - A word value used to hold the actual elapsed time between task runing times.
- data\_change* - A word value that is set to 0FFFFH any time an input read in is different from the current value. For output this word must be set by some other task and is used to signify that port action output is needed.
- current\_value* - A word value that holds the last input value or the last value to be output.

---

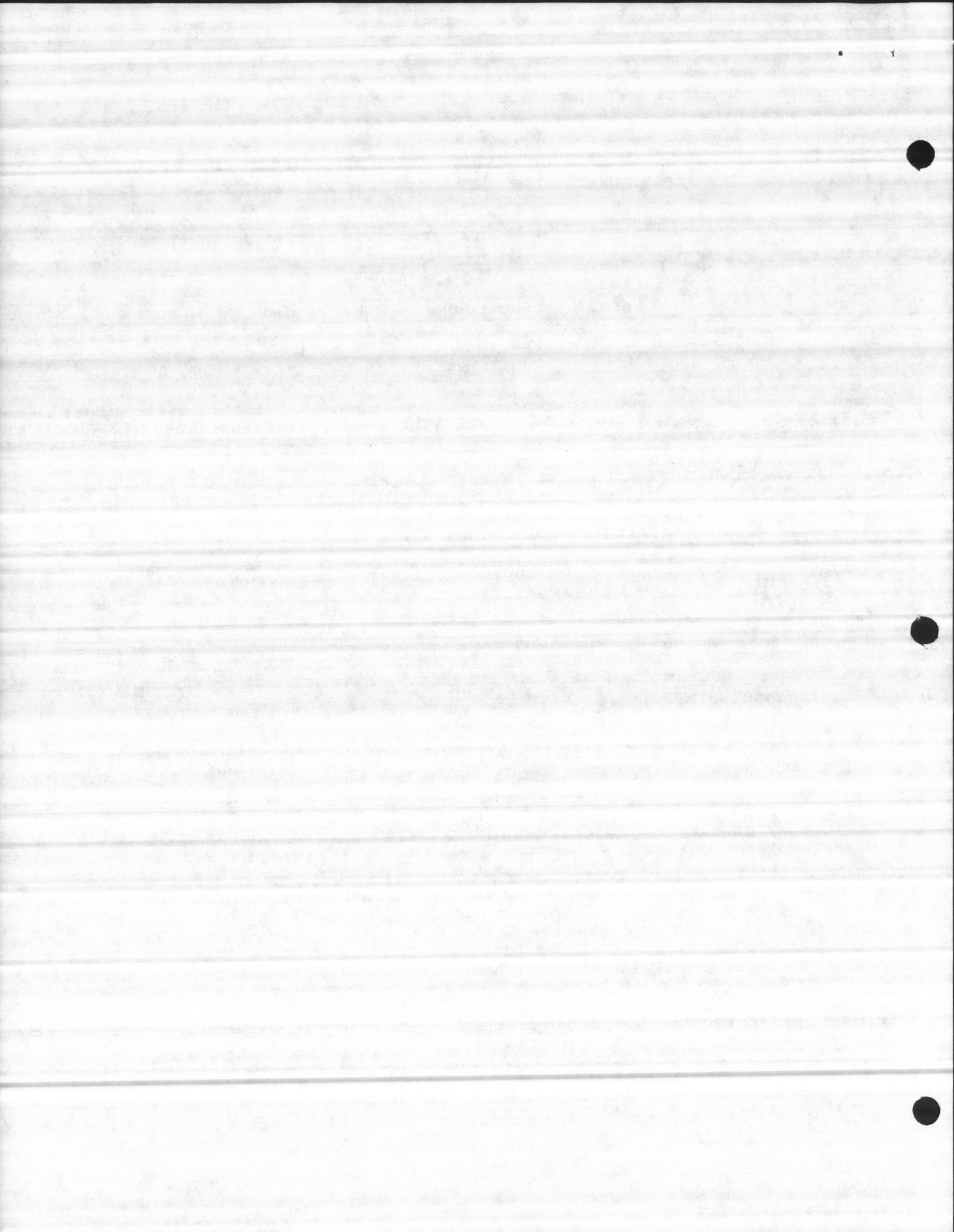
## (2) GLOBAL CRT DISPLAY AND EDIT VARIABLES

---

```
DECLARE alarms_display_flag      (3) BYTE PUBLIC INITIAL (0,0,0),
crt_base_display_type        (3) BYTE PUBLIC INITIAL (0,0,0),
hist_display_flag            (3) BYTE PUBLIC INITIAL (0,0,0),
page_limit                   (3) WORD PUBLIC INITIAL (1,1,1),
alarms_page_limit           (3) WORD PUBLIC INITIAL (1,1,1),
page                         (3) WORD PUBLIC INITIAL (1,1,1),
new_page                      (3) WORD PUBLIC INITIAL (1,1,1),
alarms_display_page         (3) WORD PUBLIC INITIAL (1,1,1),
prompt_entry_count          (3) BYTE PUBLIC INITIAL (0,0,0);
```

```
DECLARE crt_buf_t                (3) TOKEN    PUBLIC,
crt_table_seg_t              (3) TOKEN    PUBLIC,
prompt_entry_buf_t            (3) TOKEN    PUBLIC,
edit_buf_t                    (3) TOKEN    PUBLIC,
table_reg_t                  (3) TOKEN    PUBLIC,
crt_edit_flag                (3) BYTE     PUBLIC,
crt_table_p                  (3) POINTER  PUBLIC,
crt_param_seg_t              (3) TOKEN    PUBLIC,
crt_param_p                  (3) POINTER  PUBLIC,
entry                        (3) STRUCTURE (count    BYTE,
                                         asc (95) BYTE) PUBLIC;
```

TRY STRUCTURE SIZE = 3 \* 96 = 207 BYTES.



---

DESCRIPTION :

=====

alarms\_display\_flag (3) - Three bytes value used as an alarms display flags, One for each CRT.

crt\_base\_display\_typ(3) - Three bytes value used to identify data base type, One for each CRT.

hist\_display\_flag (3) - Three bytes value used as historical display flags, One for each CRT.

page\_limit (3) - Three word values hold the maximum number of pages for a multipage display. One for each CRT.

alarms\_page\_limit (3) - Three word values hold the maximum number of pages for a alarms multipage display. One for each CRT.

page (3) - Three word values hold the current page displayed. One for each CRT.

new\_page (3) - Three word values hold the new page number. One for each CRT.

alarms\_display\_page (3) - Three word values hold the current alarm page displayed. One for each CRT.

prompt\_entry\_count (3) - Three word values hold the prompt entry count. One for each CRT.

crt\_buf\_t (3) - Three tokens to the crt buffer. One Specified for each CRT.

crt\_table\_seg\_t (3) - Three tokens to the crt table segment. Used to Create or Delete Segment for each CRT.

prompt\_entry\_buf\_t (3) - Three tokens to the prompt entry buffer.

edit\_buf\_t (3) - Three tokens to edit buffer.

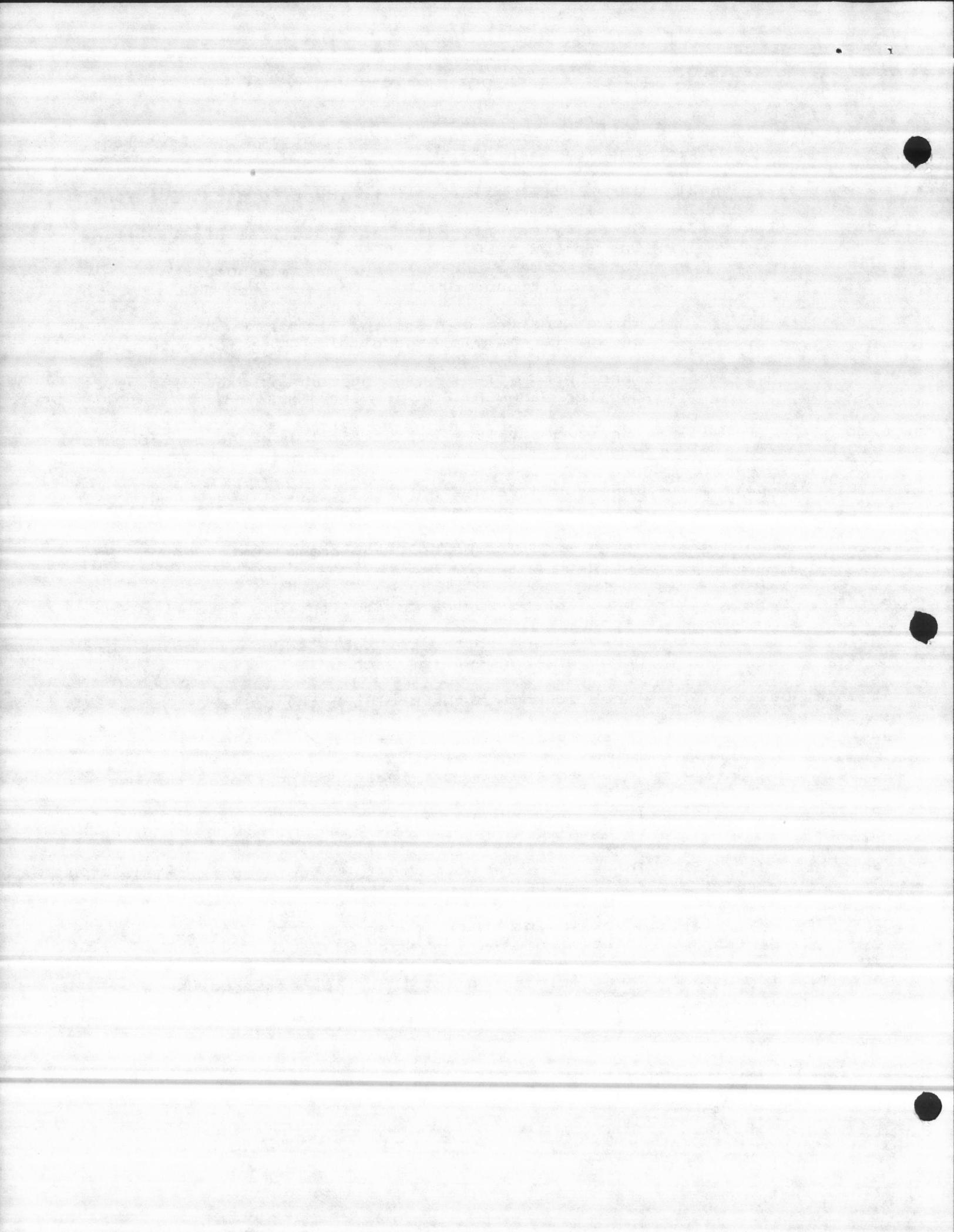
table\_reg\_t (3) - Three tokens for crt table region protection. One for each CRT.

crt\_edit\_flag (3) - Three word values hold the crt edit flag. One for each CRT.

crt\_table\_p (3) - Three pointers point to the displayed crt table. One for each CRT.

crt\_param\_seg\_t (3) - Three tokens to crt param segment. Used to Create and Delete Segments for each CRT.

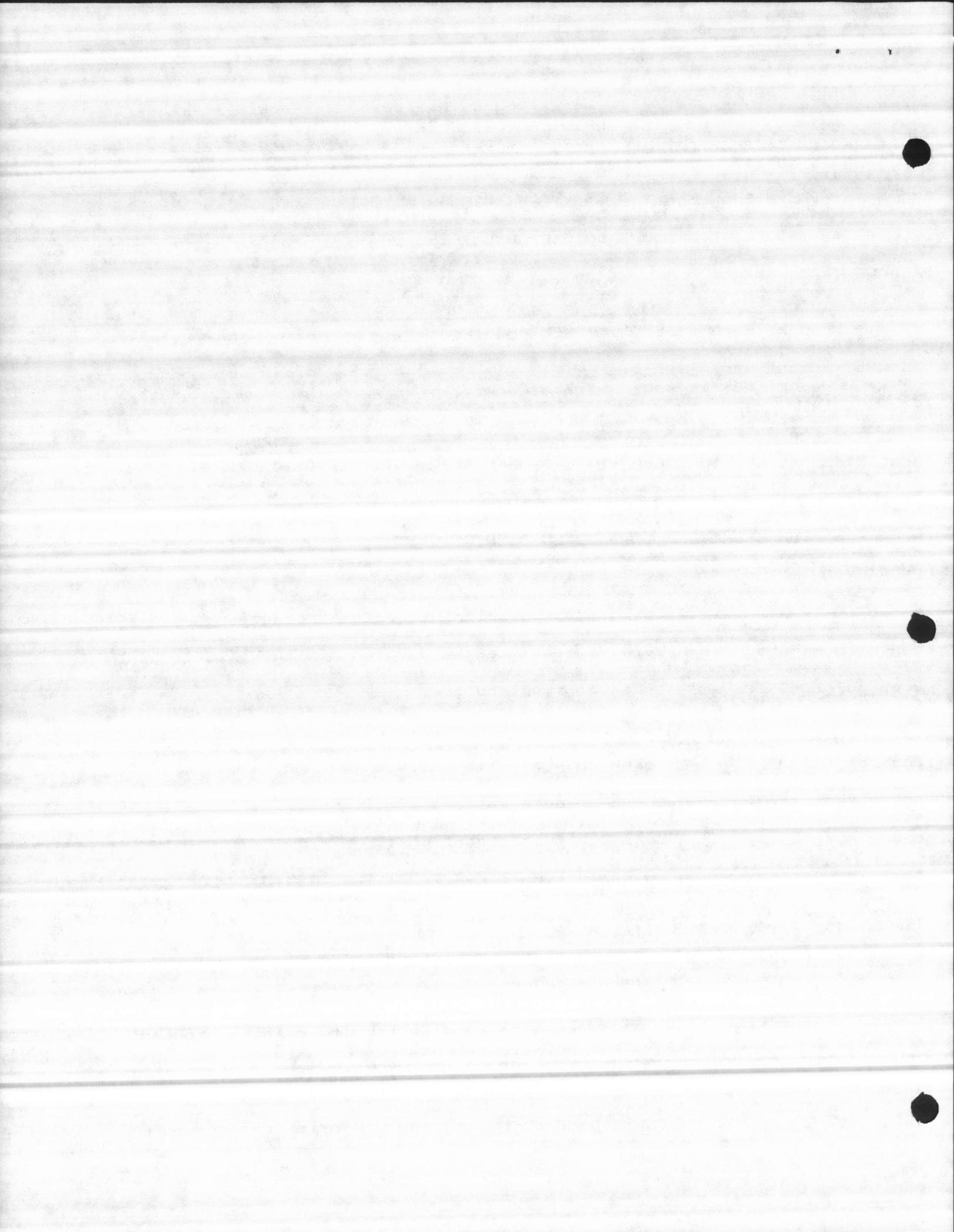
crt\_param\_p (3) - Three pointers point to crt param structure.



===== (3) PRINTER VARIABLES =====

```
DECLARE (lp_reg_t) TOKEN PUBLIC;
DECLARE (event_line_count) BYTE PUBLIC INITIAL (66);
DECLARE (event_buf_t, report_buf_t) TOKEN PUBLIC;
DECLARE (alog_mbx_t,
        dump_mbx_t,
        report_mbx_t,
        report_result_mbx_t) TOKEN PUBLIC;
DECLARE (report_param_seg_t,
        report_buffer_seg_t) TOKEN PUBLIC;
DECLARE (report_param_reg_t) TOKEN PUBLIC;
DECLARE report_printer_busy_flag BYTE PUBLIC INITIAL (0);
DECLARE report_reg_t TOKEN PUBLIC;
DECLARE period_start_sem_t TOKEN PUBLIC;

DESCRIPTION :
=====
lp_reg_t - A token - used for print region protection.
event_line_count - A byte initially 66 holds a count line value.
event_buf_t - A token to event buffer.
report_buf_t - A token to report buffer.
alog_mbx_t - A token to alarm log mail box.
dump_mbx_t - A token to dump data base mail box.
report_mbx_t - A token to report mail box.
report_result_mbx_t - A token to report result mail box.
report_param_seg_t - A token to report param segment.
report_buffer_seg_t - A token to report buffer segment.
report_param_reg_t - A token to report param region.
report_printer_busy_flag - A byte used as a switch (true/false) to show
                           the printer is busy or not.
report_reg_t - A token - used for the report region protection.
period_start_sem_t - A token used to create the periodic start report
                      semaphore.
```



## ===== (4) DISK VARIABLES =====

```
DECLARE (disk_mbx_t,  
        disk_result_mbx_t) TOKEN PUBLIC;
```

```
DECLARE (disk_param_seg_t,  
        disk_buffer_seg_t) TOKEN PUBLIC;
```

```
DECLARE (disk_param_reg_t) TOKEN PUBLIC;
```

disk\_mbx\_t - A token to disk mail box.  
disk\_result\_mbx\_t - A token to disk result mail box.

disk\_param\_seg\_t - A token to disk param segment.  
disk\_buffer\_seg\_t - A token to disk buffer segment.

disk\_param\_reg\_t - A token - Used for disk param region protection.

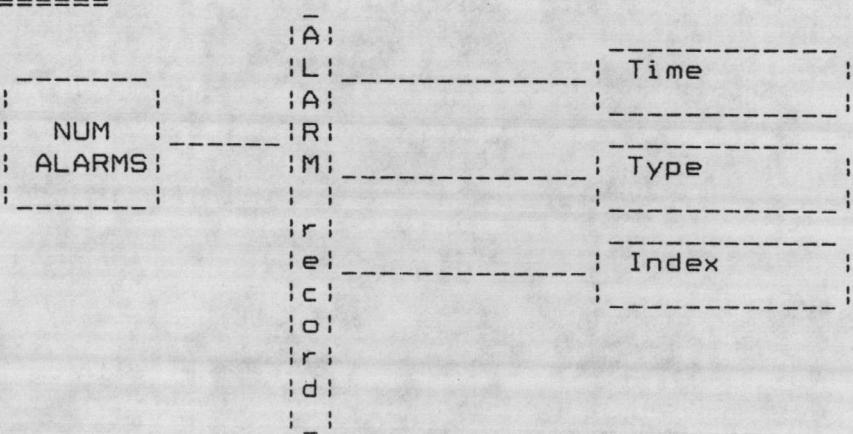
## ===== (5) ALARM VARIABLES =====

```
DECLARE alarm_display_type (4) WORD PUBLIC DATA (0,0,0,0);
```

```
DECLARE alarms (150) STRUCTURE (time DWORD,  
                                type BYTE,  
                                index WORD) PUBLIC;
```

ALARMS STRUCTURE SIZE = 150 \* 7 = 1050 BYTES.

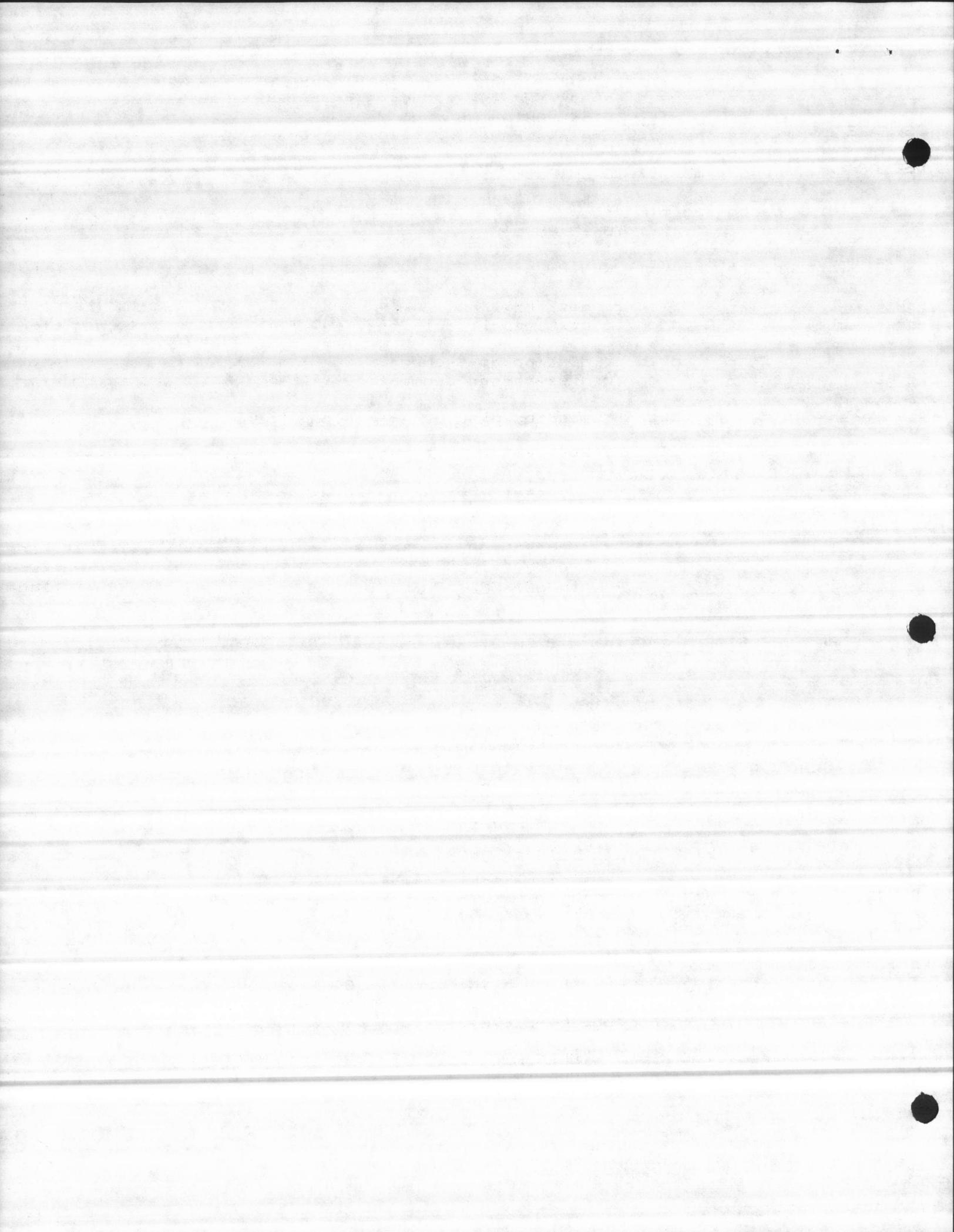
DATA DIAGRAM :



```
DECLARE alarm_variables (3) WORD PUBLIC INITIAL (0, 0, 0),  
                        last_ack_alarm WORD PUBLIC AT (@alarm_variables(1)),  
                        alarm_count WORD PUBLIC AT (@alarm_variables(2));
```

```
DECLARE (alarm_reg_t) TOKEN PUBLIC;
```

```
DECLARE (alarms_update_list_t) TOKEN PUBLIC;
```



```

DECLARE sonalert(2) BYTE PUBLIC INITIAL (0,0);

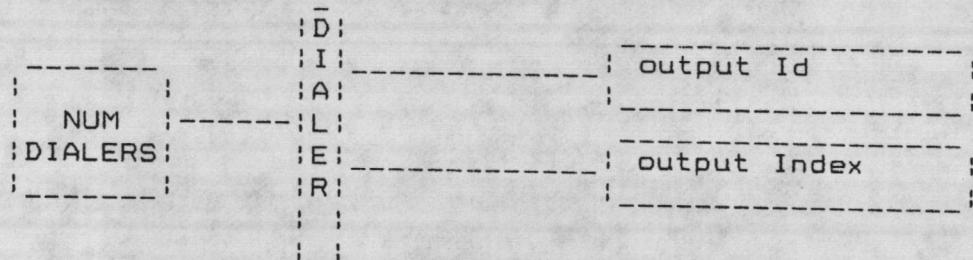
DECLARE dialer_count (NUM_DIALERS_PLUS1) WORD PUBLIC INITIAL (0);

DECLARE dialers (NUM_DIALERS_PLUS1) STRUCTURE (output_id     BYTE,
                                              output_index WORD)
                                              PUBLIC INITIAL (0FFH, 0);

```

DIALERS STRUCTURE SIZE = 2 \* 3 = 6 BYTES.

DATA DIAGRAM :



Camp Lejuene Sonalert structure includes:

- (1) = common alarm sonalert
- (2) = computer fail pulsed minimum of every six seconds

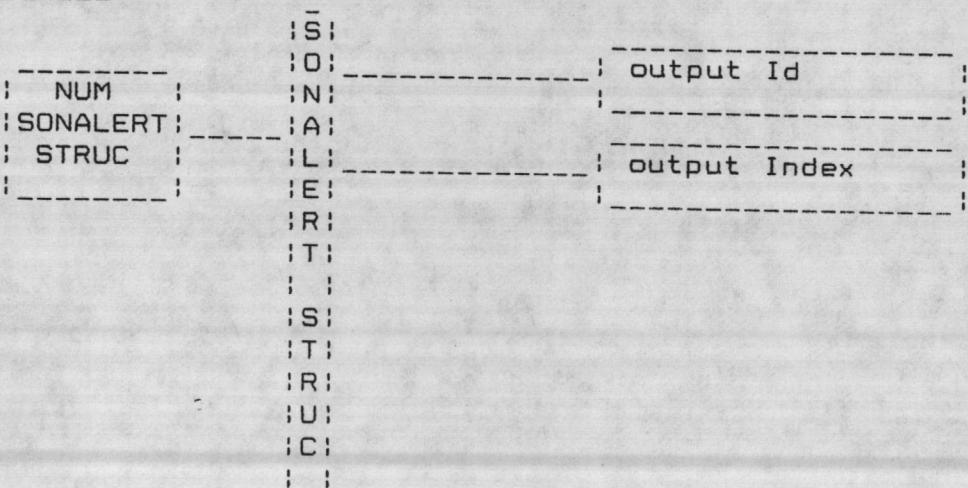
```

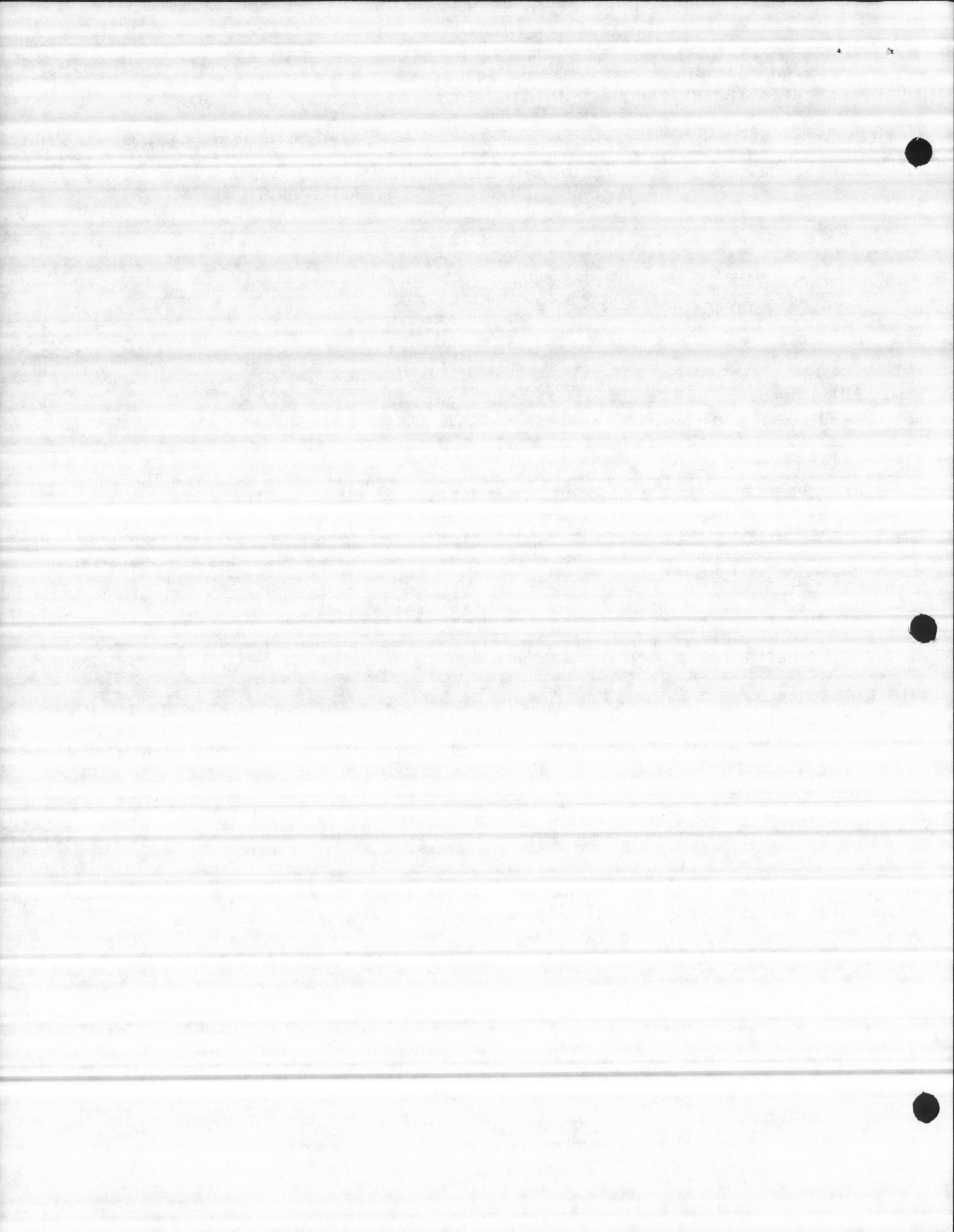
DECLARE sonalert_struc (3) STRUCTURE (output_id     BYTE,
                                       output_index WORD)
                                       PUBLIC INITIAL (0FFH, 0);

```

SONALERT STRUCTURE SIZE = 3 \* 3 = 6 BYTES.

DATA DIAGRAM :





The following is a description of all the variables and the entries of each structure in that section :

alarm\_disp STRUCTURE :

This structure used in the alarm section displays.

point\_asc => 10 bytes ascii point id number. used to access specific alarm section or structure elements. Allows the operator to index to an alarm section by entering the point\_asc.  
The point\_asc is also used by the keyboard SELECT command to select any alarm sectin in the system.

desc\_asc => 24 bytes ascii alarm section header and description.

Variables :

alarm\_display\_type - Four word value shows the alarm display types.

alarms STRUCTURE :

time - A dword value holds the full time when the alarm occurs  
type - A byte value holds the alarm type.  
index - A word value holds the alarm index.

Variables :

alarm\_variables - Two words holds the addresses for last\_ack\_alarm and alarm count.

last\_ack\_alarm - A word value holds the last acknowledge alarm index.  
alarm\_count - A word value holds the alarm count

alarm\_reg\_t - A token to the alarm region.

alarms\_update\_list\_t - A token to the alarms updated list entry.

sonalert - Two bytes hold the number sequence for the 2 sonalert.

dialer\_count - six words hold the dialer count.

dialers STRUCTURE :

output\_id - A byte value contains the index to the structure type of output source.

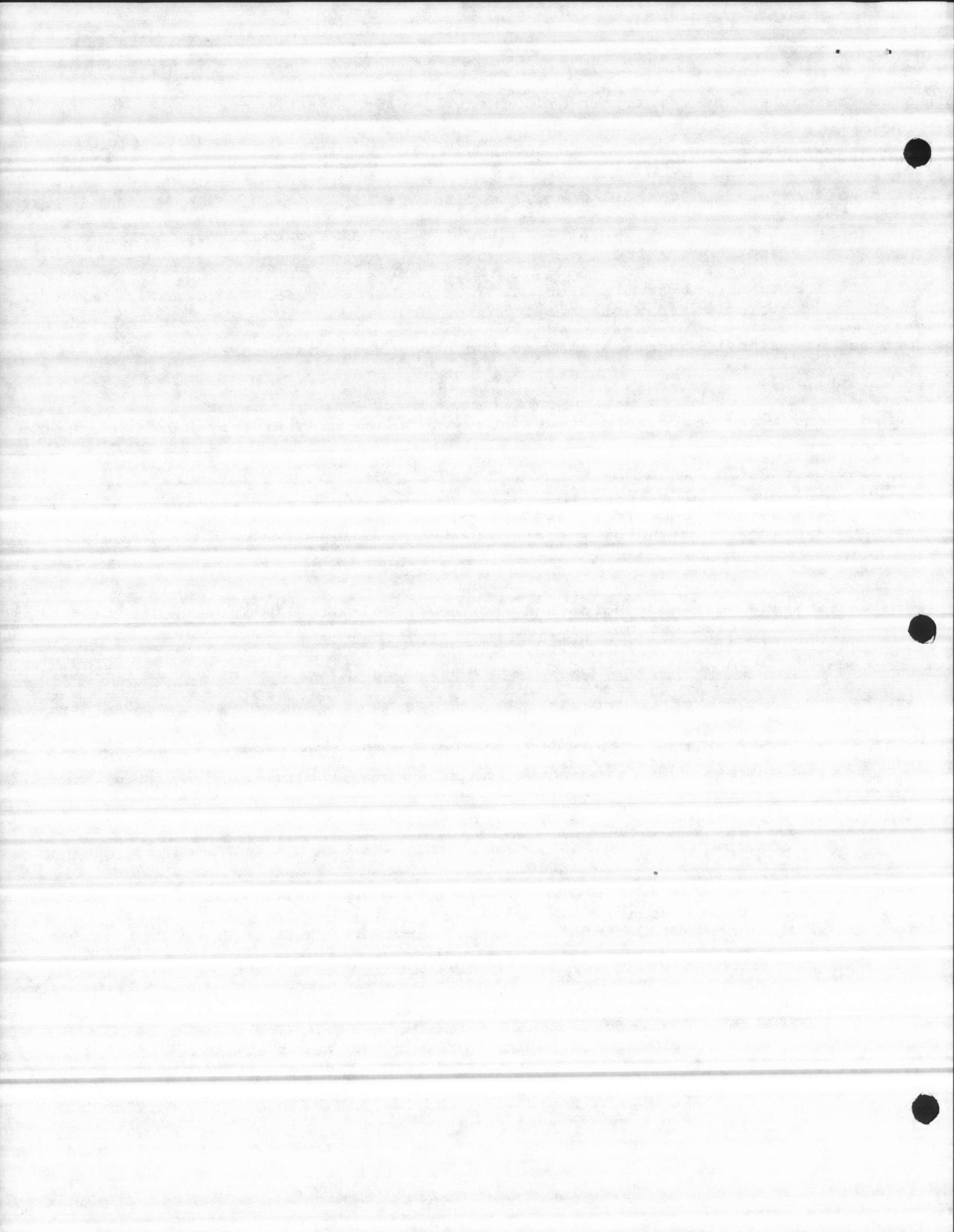
output\_index - A word value contains the index to the ouput source.

sonalert\_struct STRUCTURE

output\_id - A byte value contains the index to the structure type of output source.

output\_index - A word value contains the index to the ouput source.

- (1) = acknowledge input
- (2) = computer fail pulsed minimum of every six seconds
- (3) = common alarm sonalert

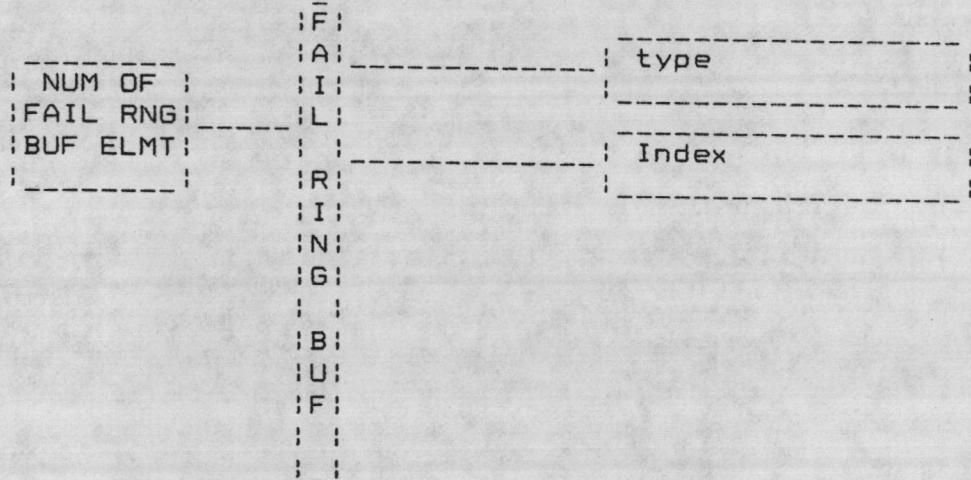


## ===== FAIL INTERFACE VARIABLES =====

```
=====
DECLARE fail_ring_buf (10) STRUCTURE (type BYTE,
                                         index WORD) PUBLIC;
```

FAIL RING BUFFER STRUCTURE SIZE = 10 \* 3 = 30 BYTES.

DATA DIAGRAM :



```
=====
DECLARE (send_fail_ring_buf_count, recv_fail_ring_buf_count)
        BYTE PUBLIC INITIAL (0,0);
```

```
=====
DECLARE (send_fail_ring_buf_reg_t) TOKEN PUBLIC;
```

```
=====
DECLARE (fail_sem_t) TOKEN PUBLIC;
```

**fail\_ring\_buf STRUCTURE:**

**type** - A byte holds the fail type.

**index** - A word holds the index.

**Variables :**

**send\_fail\_ring\_buf\_count** - A byte holds the send count.

**recv\_fail\_ring\_buf\_count** - A byte holds the receive count.

**send\_fail\_ring\_buf\_reg\_t** - A token to the send fail ring buffer region.  
**fail\_sem\_t** - A token to the fail semaphore.

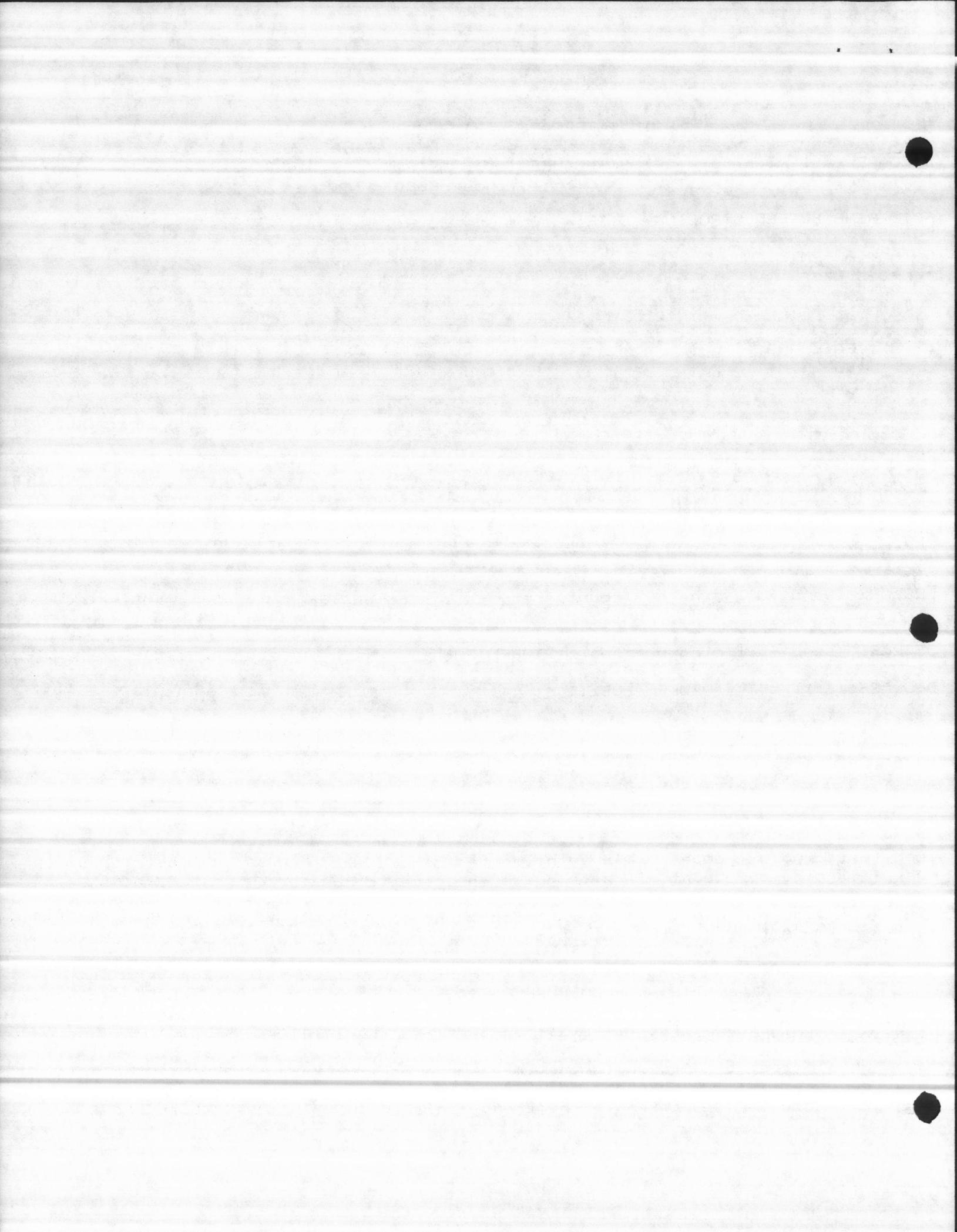
## ===== TOTAL PROCEDURE VARIABLES =====

```
=====
DECLARE (total_update_list_t, total_reg_t, trend_reg_t) TOKEN PUBLIC;
```

**total\_update\_list\_t** - A token to the total updated list entry.

**total\_reg\_t** - A token to the total region.

**trend\_reg\_t** - A token to the trend region.



```
=====
=                   POWER UP FILE SAVE TIME
=====
= CLARE power_up_save_time (2) DWORD PUBLIC INITIAL (0);
=====
```

power\_up\_save\_time - A dword holds the time when the power up file saved

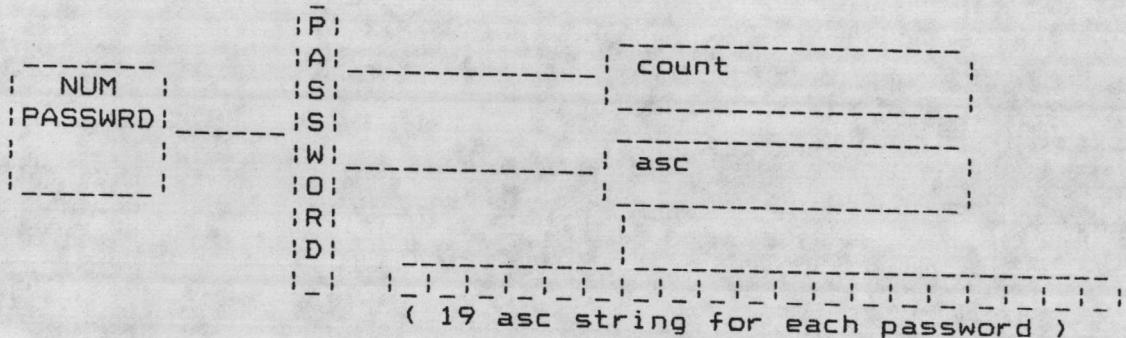
```
=====
=                   SYSTEM PASSWORD STRUCTURE
=====

```

```
DECLARE password (4) STRUCTURE (count BYTE,
                                 asc(19) BYTE) PUBLIC INITIAL (0);
=====
```

PASSWORD STRUCTURE SIZE = 4 \* 20 = 60 BYTES.

DATA DIAGRAM :



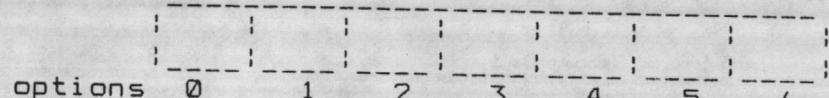
password STRUCTURE

count - A byte holds the asc count of the password  
 asc(19) - 19 bytes password asc.

```
=====
=                   SYSTEM OPTIONS STRUCTURE
=====

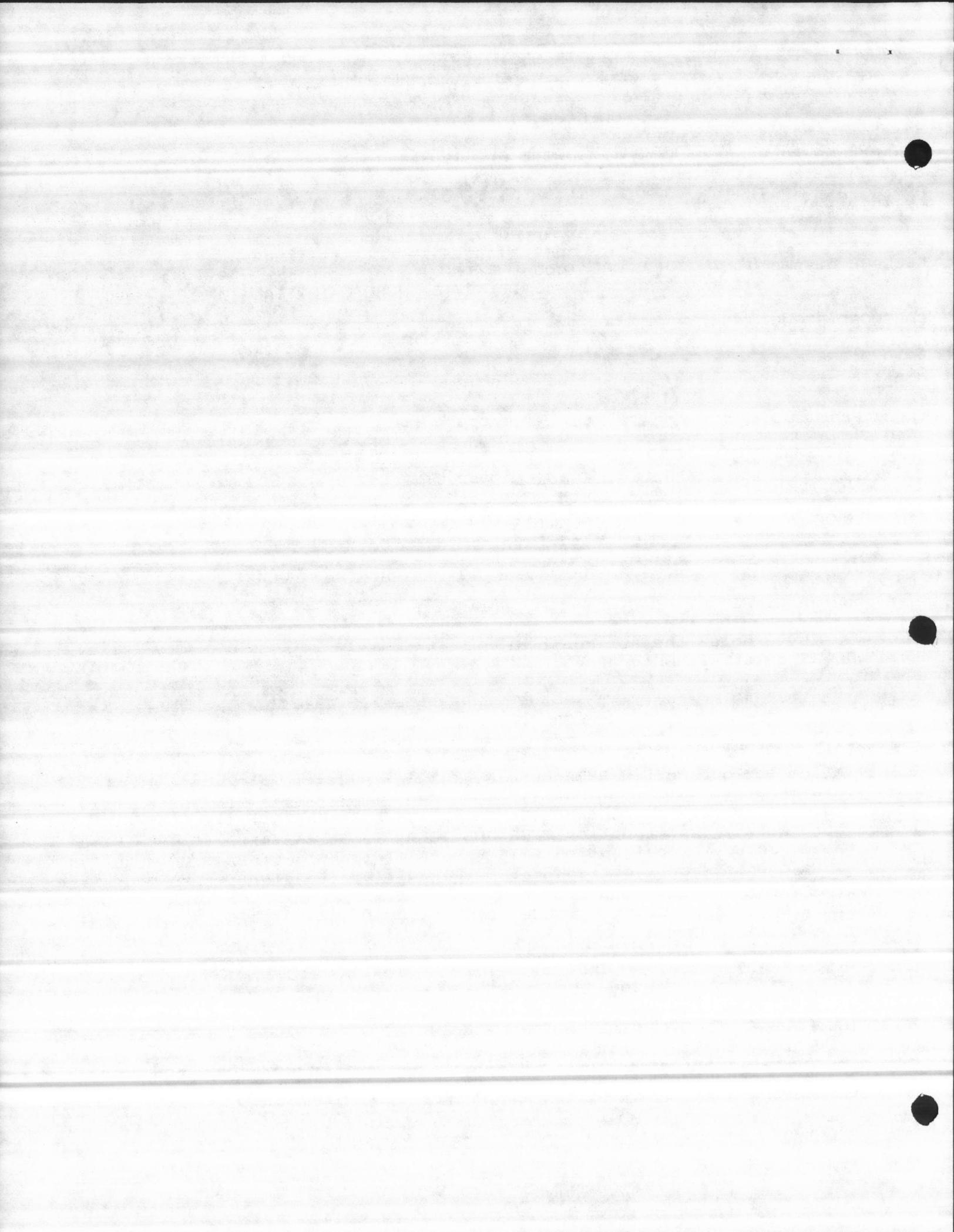
```

```
DECLARE options (7) BYTE PUBLIC INITIAL (0),
          alarm_lockout_option BYTE PUBLIC AT (@options(1)),
          alarm_auto_display_option BYTE PUBLIC AT (@options(2)),
          auto_daily_report_option BYTE PUBLIC AT (@options(3)),
          auto_weekly_report_option BYTE PUBLIC AT (@options(4)),
          auto_monthly_report_option BYTE PUBLIC AT (@options(5)),
          auto_events_report_option BYTE PUBLIC AT (@options(6));
=====
```



options (7) BYTE - This means we have six options:

- 1) alarm\_lockout\_option
- 2) alarm\_auto\_display\_option
- 3) auto\_daily\_report\_option
- 4) auto\_weekly\_report\_option
- 5) auto\_monthly\_report\_option
- 6) auto\_events\_report\_option



## ===== GROUP DISPLAY STRUCTURE =====

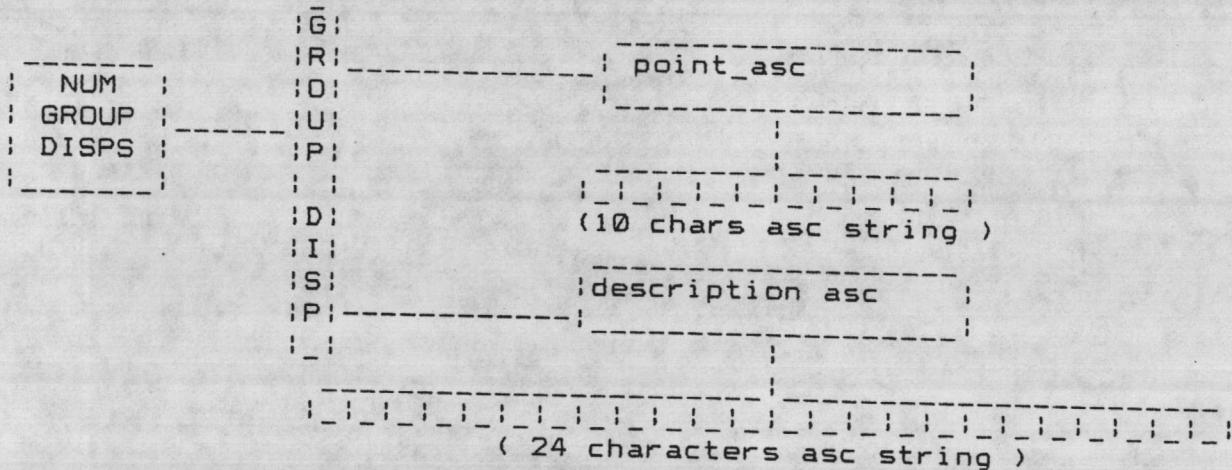
```

DECLARE group_disp (NUM_GROUP_DISP_PLUS1) STRUCTURE
    (point_asc (10) BYTE,
     desc_asc (24) BYTE) PUBLIC INITIAL(
     'Spare      ,
     'Spare group display      );

```

GROUP DISP STRUCTURE SIZE = 101 \* 34 = 3434 BYTES.

DATA DIAGRAM :



The group disp structure used in the display generator and displays.

point\_asc => 10 bytes ascii point id number. used to access specific group display or structure elements. Allows the operator to index to group display by entering the point\_asc.  
The point\_asc is also used by the keyboard SELECT command to select any group display in the system.

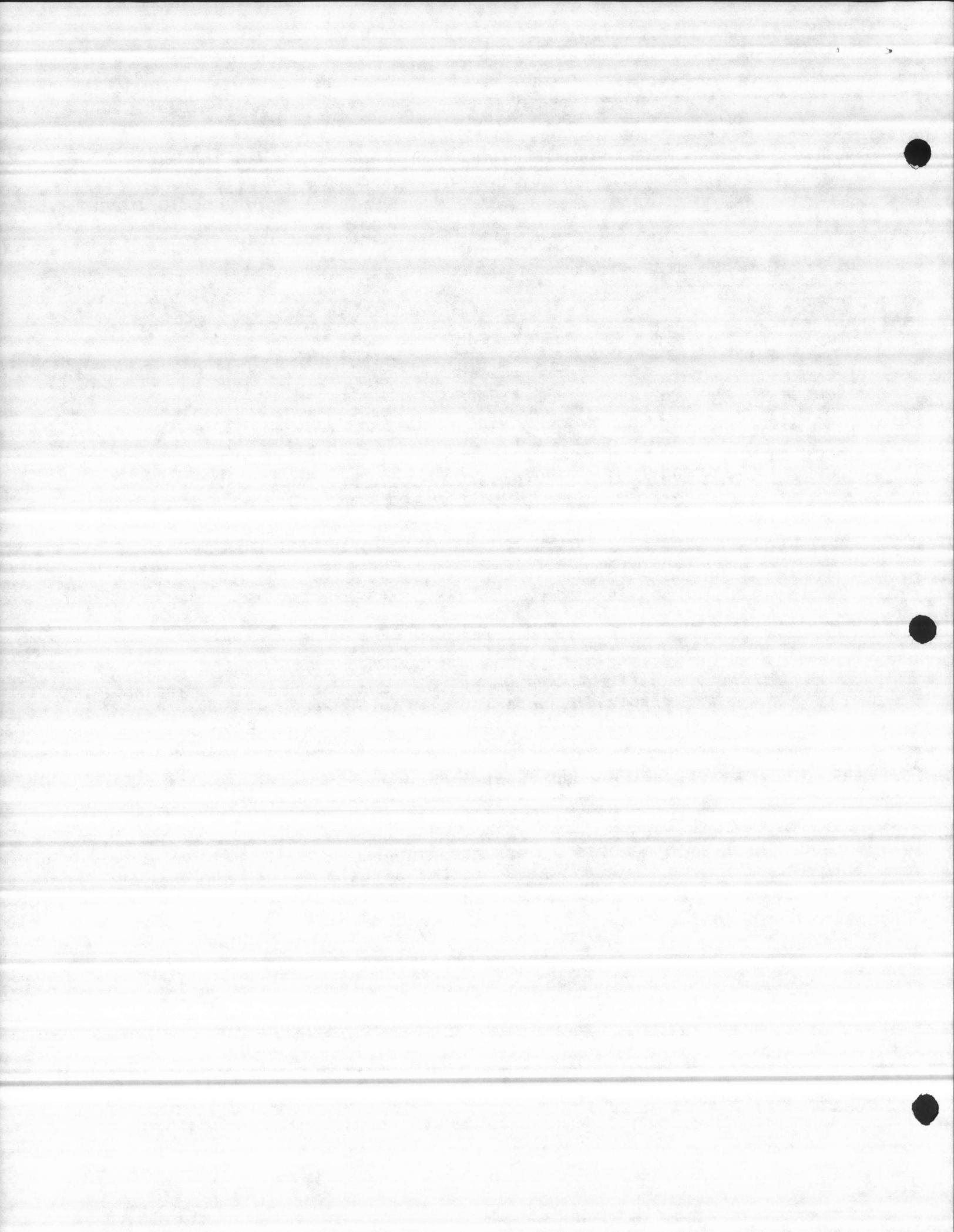
desc\_asc => 24 bytes ascii group display header and description.

## ===== REPORT DISPLAY STRUCTURE =====

```

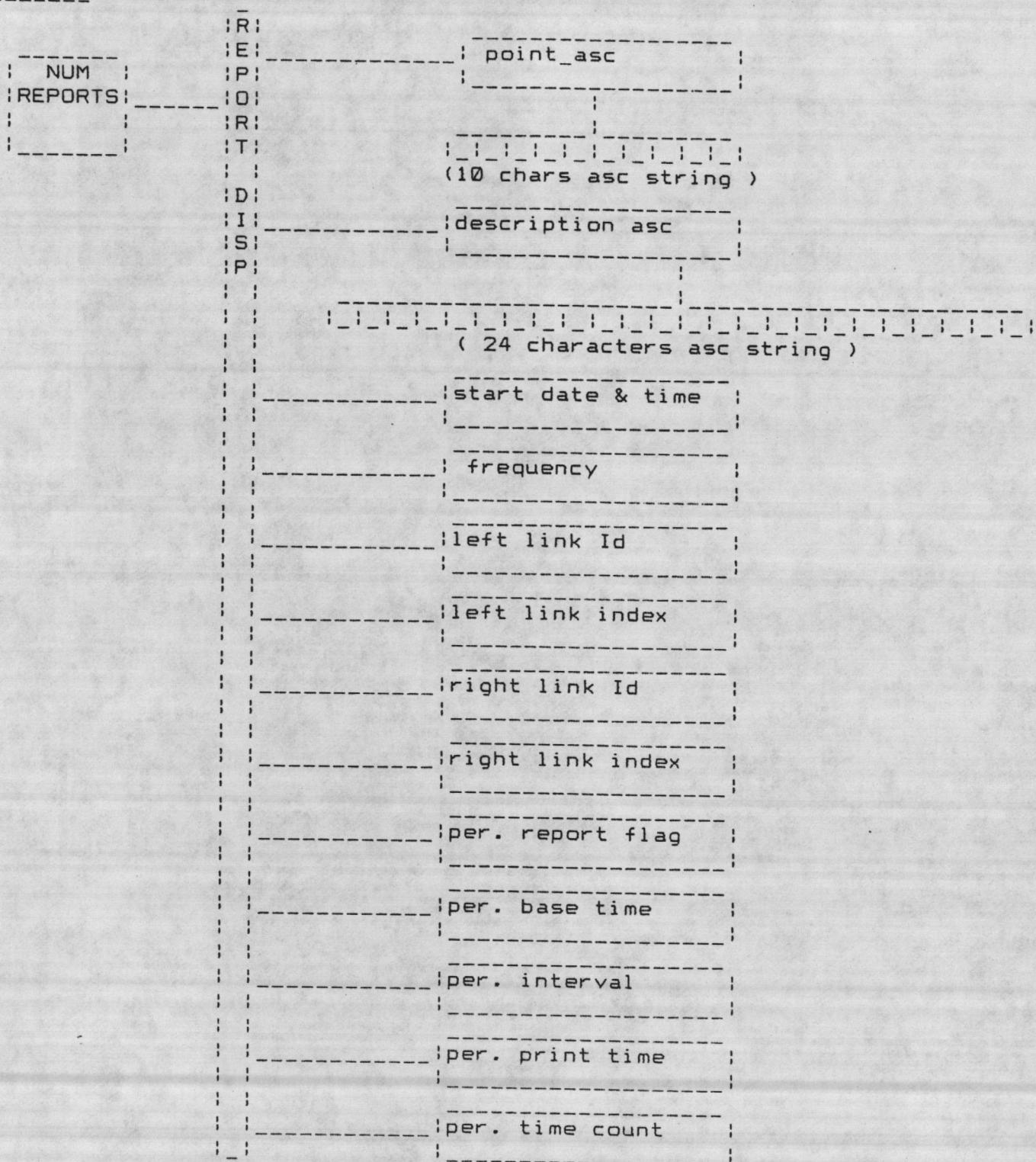
DECLARE report_disp (NUM_REPORTS_PLUS1) STRUCTURE
    (point_asc (10)     BYTE,
     desc_asc (24)     BYTE,
     start_date_time  DWORD,
     frequency        WORD,
     left_link_id     BYTE,
     left_link_index  WORD,
     right_link_id    BYTE,
     right_link_index WORD,
     per_report_flags WORD,
     per_base_time    DWORD,
     per_interval     DWORD,
     per_print_time   DWORD,
     per_time_count   DWORD)
    PUBLIC INITIAL ('Spare      ,
     'Spare report page      ,
     0,0,0FFH,0,0FFH,0, 0, 0, 0, 0, 0, 0);

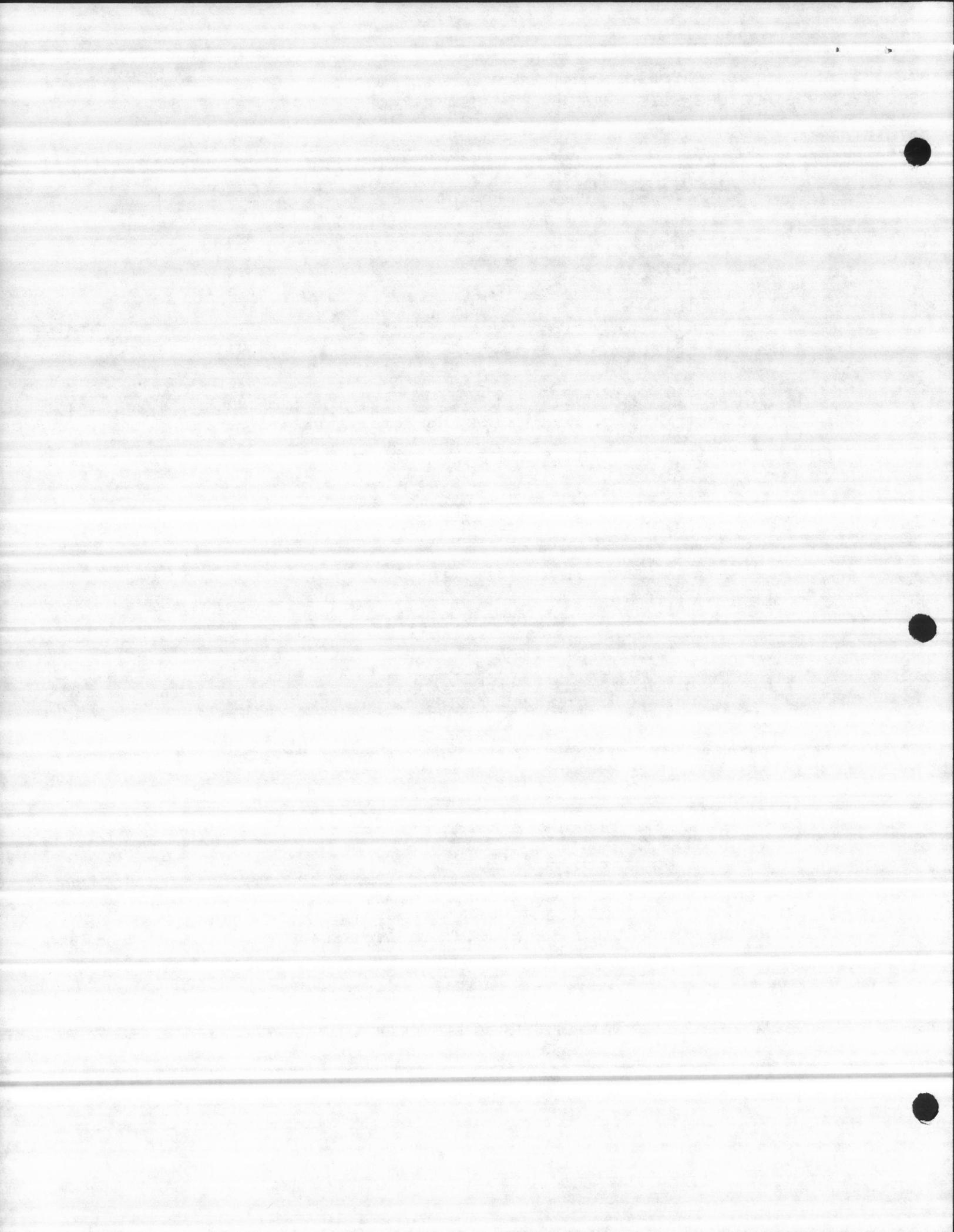
```



REPORT DISP STRUCTURE SIZE = 26 \* 64 = 1164 BYTES.

DATA DIAGRAM :





```
-----  
DECLARE report_disp_sem_t (NUM_REPORTS_PLUS1) TOKEN PUBLIC;
```

The report\_disp Structure used in the report generator (which includes any kind of report you wish to have) and print report.

point\_asc => 10 bytes ascii point id number. used to access specific report or print the report. Allows the operator to index to report or print the report by entering the point\_asc. The point\_asc is also used by the keyboard SELECT command to select any group display in the system.

desc\_asc => 24 bytes ascii report header, and description.

start\_date\_time => A dword holds the start date and time.

frequency => A word value holds the report frequency which it could be daily, periodic, weekly, monthly or yearly.

left\_link\_id => A byte value contains the index to the structure type of left link source.

left\_link\_index => A word value contains the index to the left link source.

right\_link\_id => A byte value contains the index to the structure type of right link source.

right\_link\_index => A word value contains the index to the right link source.

per\_report\_flags => A word flag if periodic report wanted the flag sould be set YES otherwise is equal to NO.

per\_base\_time => A dword holds the periodic report start time.

per\_interval => A dword value holds the time interval for the periodic report.

per\_print\_time => A dword value holds the specified print time of the periodic report

per\_time\_count => A dword holds the seconds counter until periodic report printed and it reset to zero.

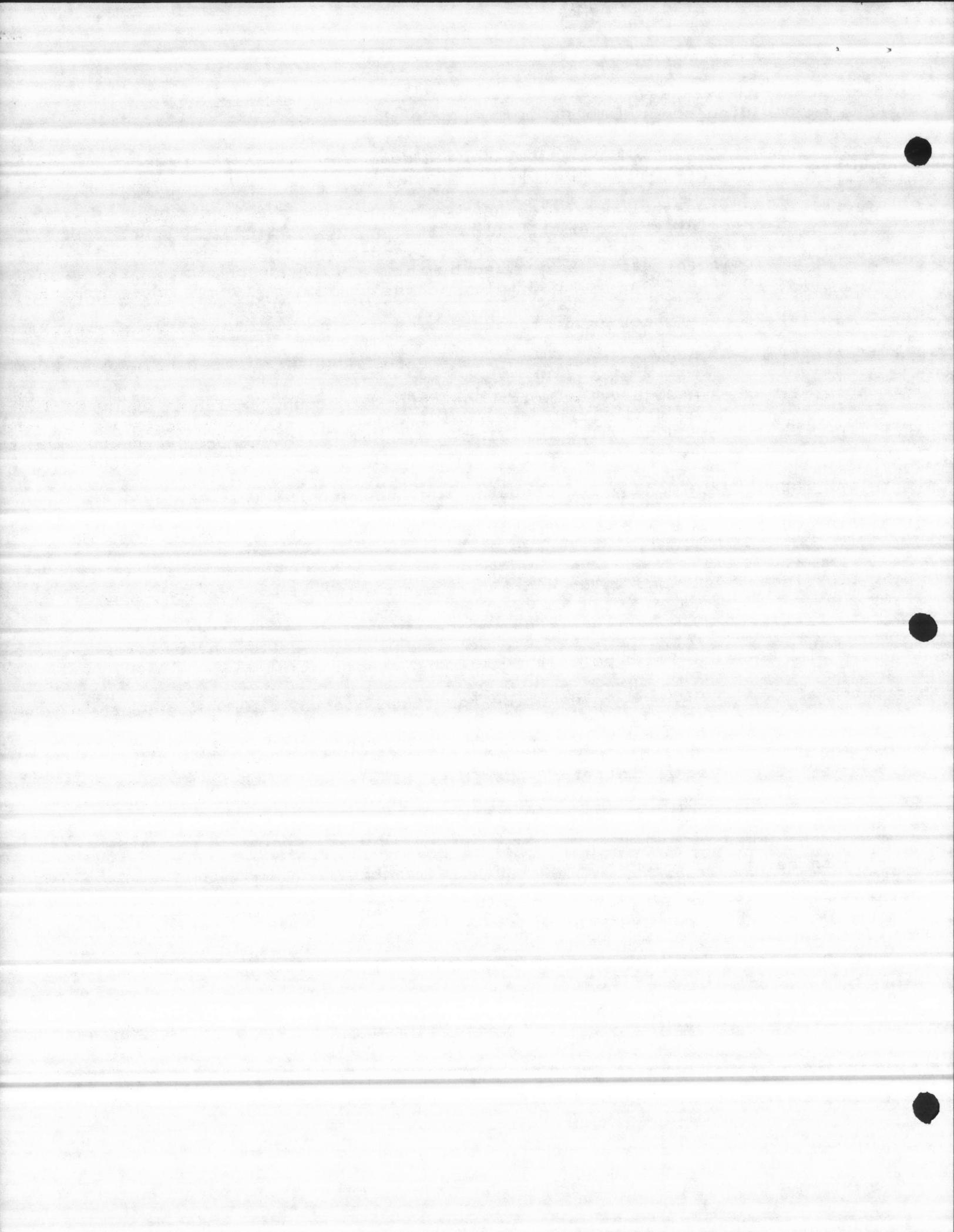
#### Variables :

```
=====  
report_disp_sem_t - A token to report disp semaphore for each report generated.  
=====
```

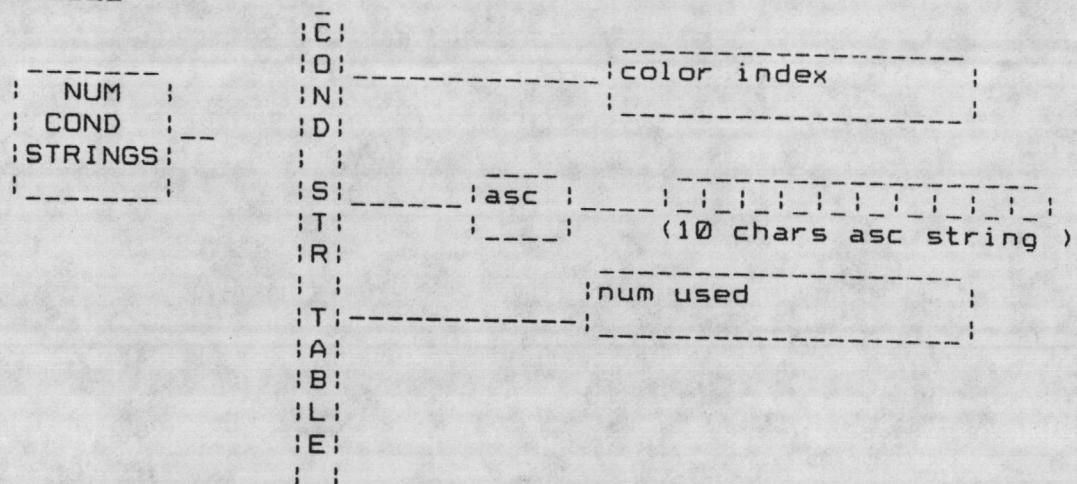
#### ===== SYSTEM CONDITION STRUCTURE =====

```
=====  
DECLARE cond_str_table (NUM_COND_STRINGS_PLUS1) STRUCTURE  
  (color_index BYTE,  
   asc      (10) BYTE,  
   num_used WORD) PUBLIC INITIAL  
   (7,           ',0);  
=====
```

COND STR TABLE STRUCTURE SIZE = 101 \* 13 = 1313 BYTES.  
=====

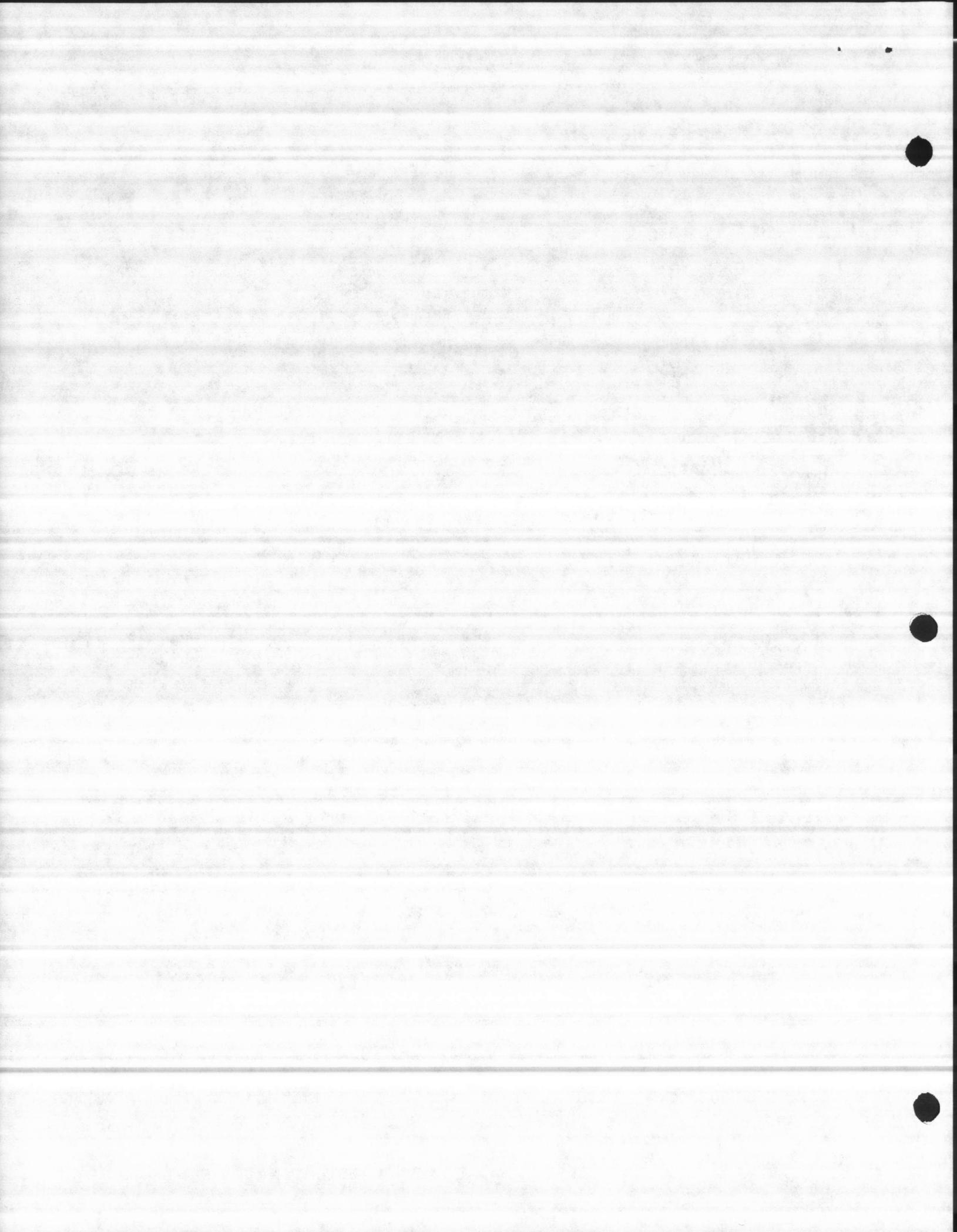


## DATA DIAGRAM :



## cond\_srt\_table STRUCTURE :

color\_index - A byte holds the condition color index.  
asc (10) - 10 byte ascii represent the condition string.  
num\_used - A word hold the num\_used of the condition.



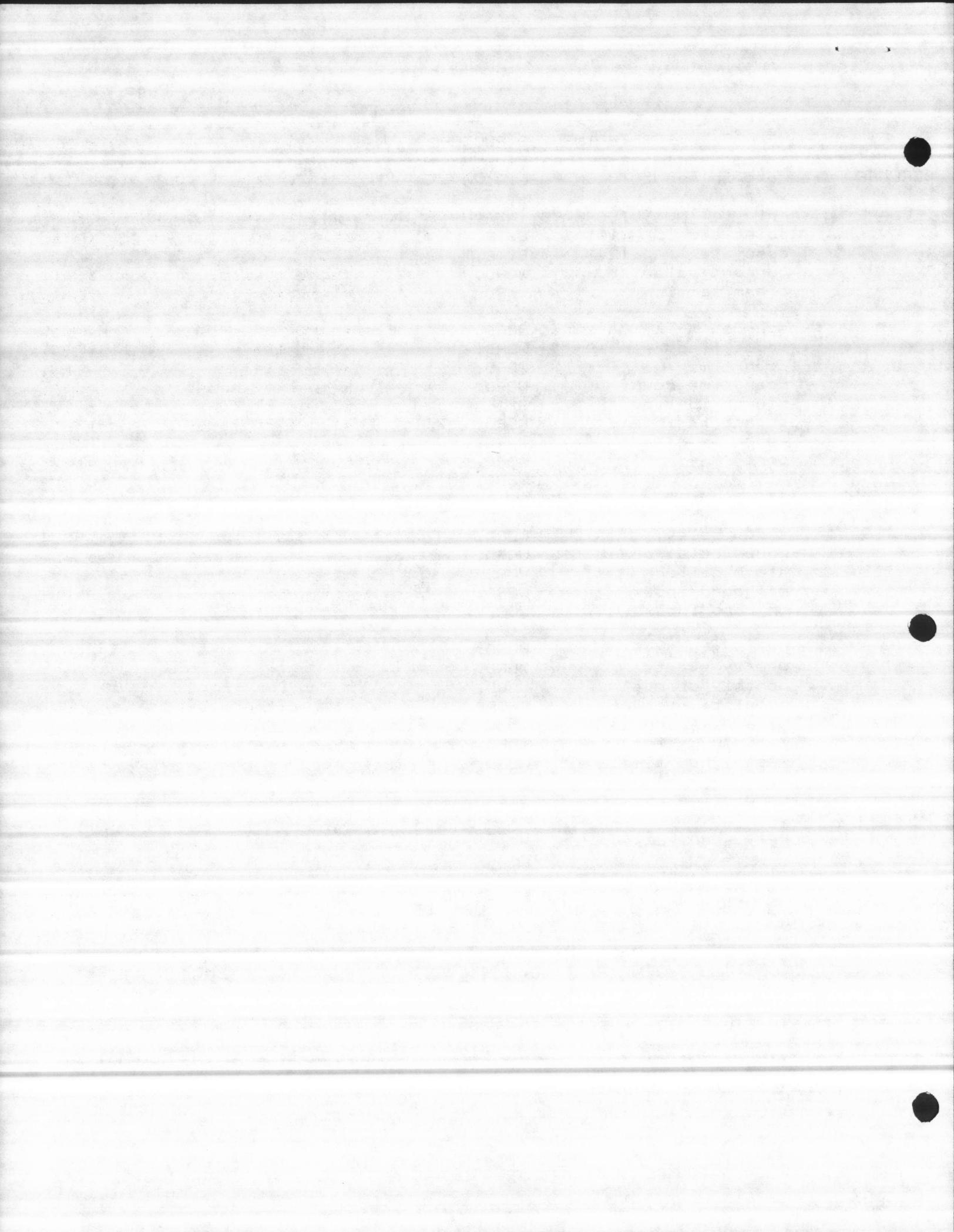
## DISCRETE INPUT STRUCTURE &amp; VARS

```
DECLARE discrete_in_reg_t TOKEN PUBLIC;

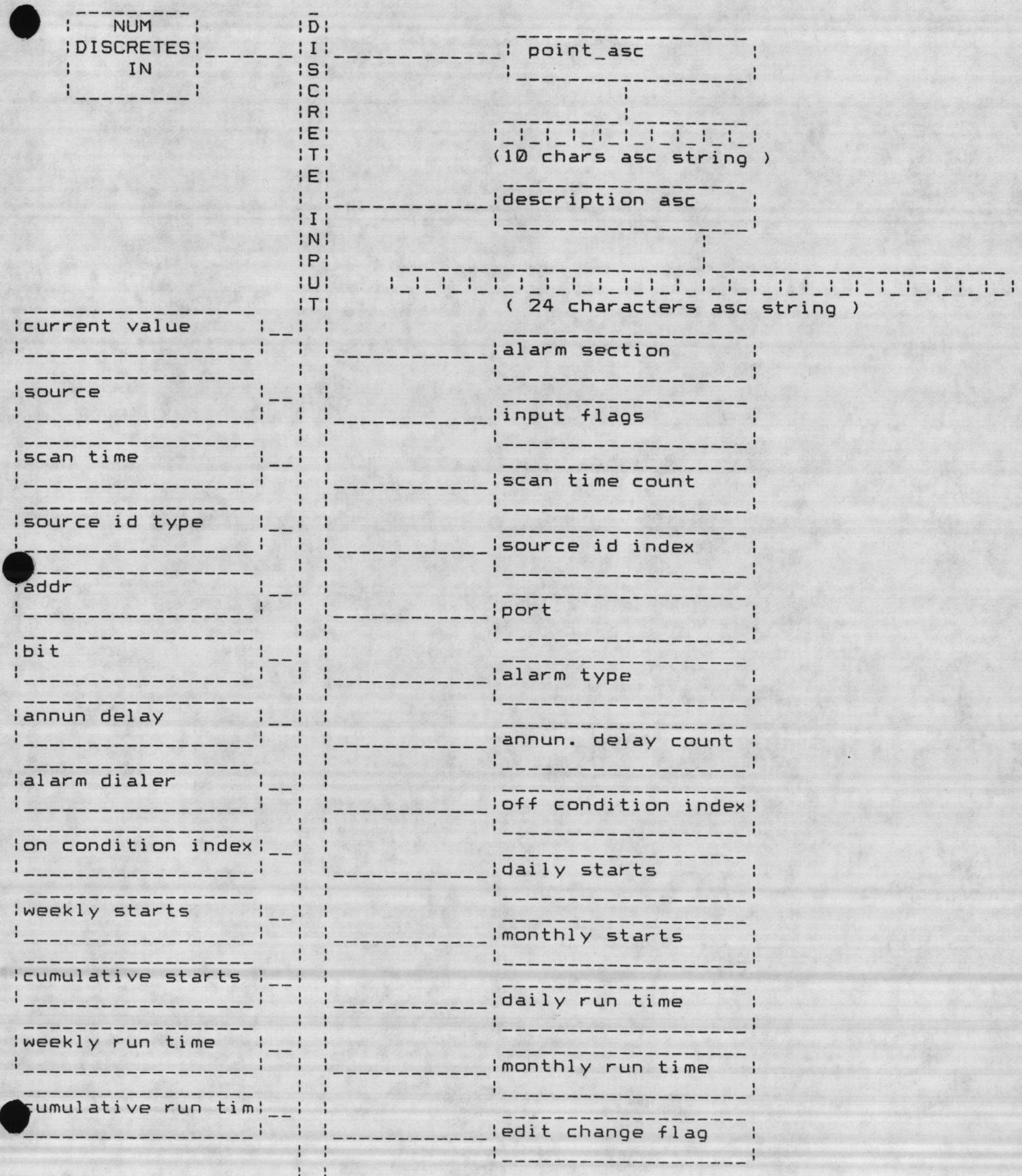
DECLARE discrete_in (NUM_DISCRETES_IN_PLUS1) STRUCTURE
    (point_asc      (10)      BYTE,
     desc_asc       (24)      BYTE,
     current_value           WORD,
     alarm_section           WORD,
     source             WORD,
     input_flags         WORD,
     scan_time          WORD,
     scan_time_count     WORD,
     source_id_type      BYTE,
     source_id_index     WORD,
     addr               WORD,
     port               WORD,
     bit                WORD,
     alarm_type          WORD,
     annun_delay         WORD,
     annun_delay_count   WORD,
     alarm_dialer        WORD,
     off_cond_index      WORD,
     on_cond_index       WORD,
     daily_starts        WORD,
     weekly_starts       WORD,
     monthly_starts      WORD,
     cumulative_starts   DWORD,
     daily_run_time      WORD,
     weekly_run_time     WORD,
     monthly_run_time    DWORD,
     cumulative_run_time DWORD,
     edit_change_flag    BYTE)

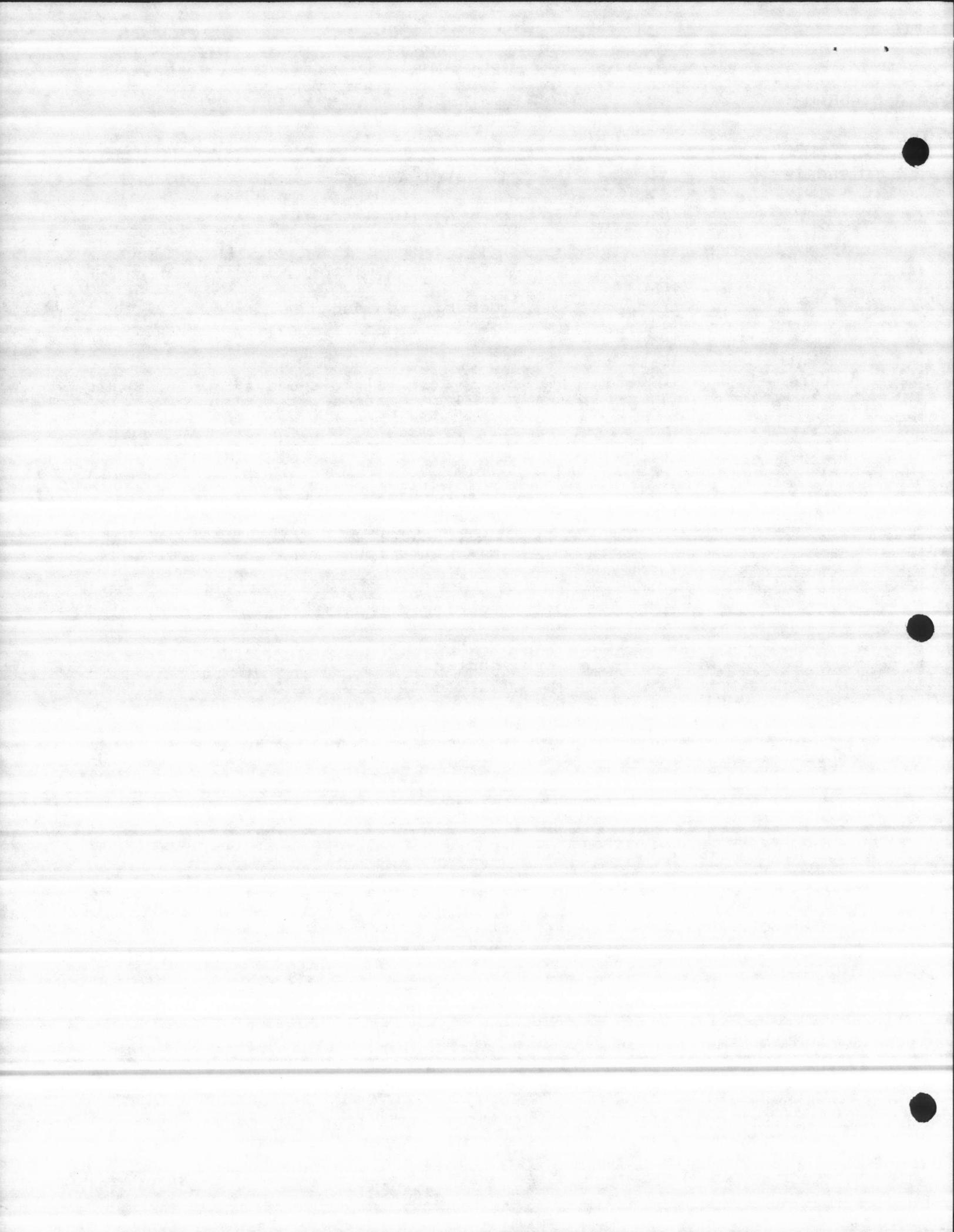
PUBLIC INITIAL ('Spare      ',
                'discrete input point    ',
                0, 1,
                0, 0, 0, 0, OFFH, 0, 0H,
                0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0,
                0, 0, 0, 0);
```

DISCRETE IN STRUCTURE SIZE = 361 \* 90 = 32490 BYTES.



## DATA DIAGRAM :





-----  
discrete\_in\_reg\_t - A token to discrete input region.

The description and the use of each element entry in the discrete\_in structure:

point\_asc => 10 bytes ascii point id number, used to access specific discrete point or structure elements. Allows the operator to index discrete input by entering point id. The point\_asc is also used by the keyboard SELECT command to select any disc point in the system.

desc\_asc => 24 bytes ascii point description.

current\_value => a word value set to 1H if discrete input is on or zero if off (edit allowed if discrete is deactivated).

alarm section => a word value represent the alarm section.

source => a word value that indicates which option of input devices was chosen. A list of sources :

- 0 - undefined point.
- 1 - telemetry input.
- 2 - local rack input.
- 4 - analog alarm input.
- 5 - boolean calculation input.
- 6 - TI PM550 input
- 7 - Gould Micro84 input
- 8 - manual input.

input\_flags => Word bitmapped flags as follows :

bit	0 - off	1 - on
---	-----	-----
1	- Deactivate	/ Activate
2	- normally open	/ normally closed
10	- not data fail	/ data fail
100	- not alarm point	/ alarm point
200	- not printed event	/ printed event
400	- no start accum	/ number start accumulated
800	- no run tim accm	/ run time accumulated.

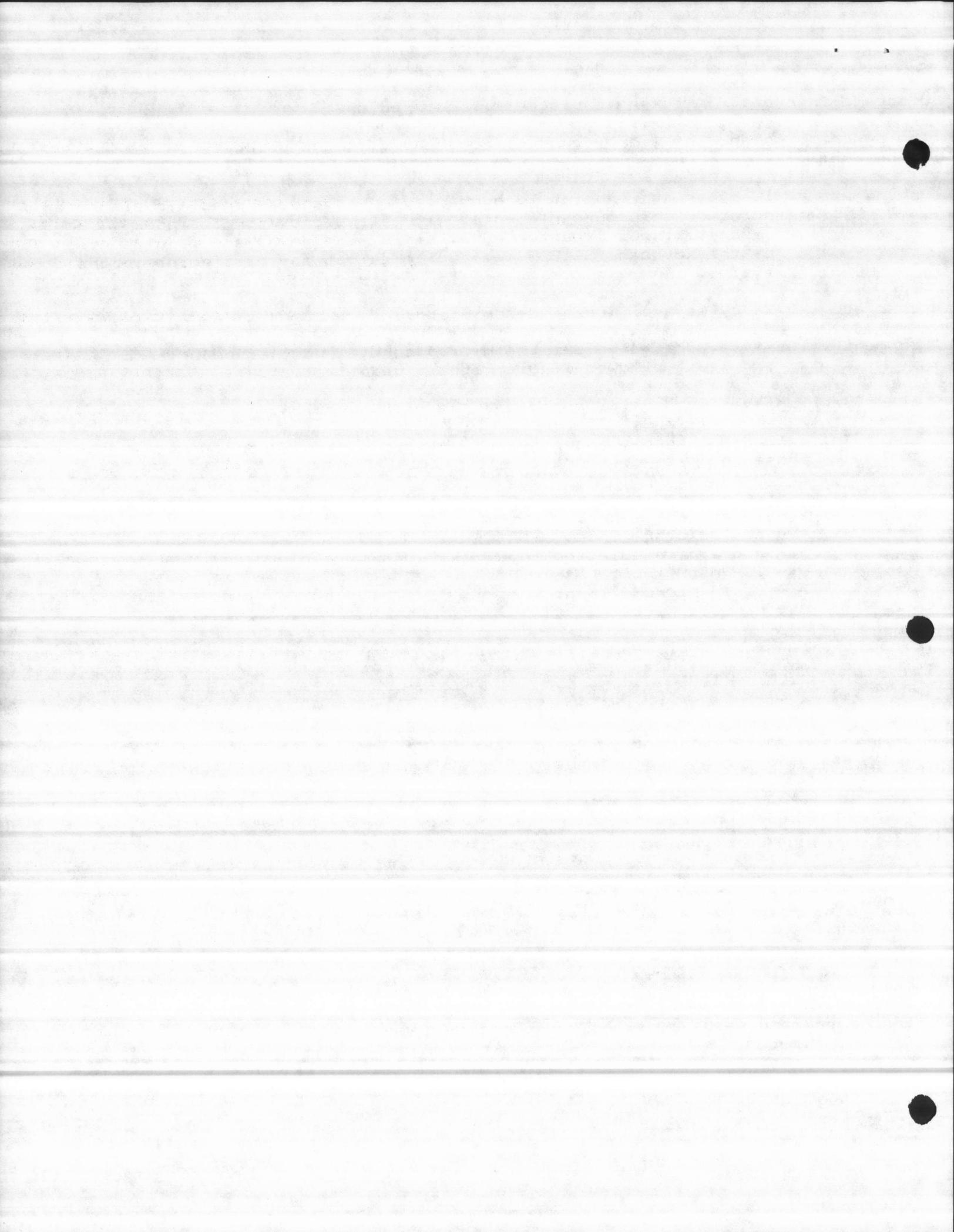
scan\_time => a word value, containing the number of seconds between input scan (defined with a six sec resolution)

scan\_time\_count => variable time hold the elapsed time between actual scans.

source\_id\_type => a byte value contains the index to the structure type.

source\_id\_index => a word value contains the index to the source.

addr => a word value contains the remote telemetry address of the contact input.



port => a word value contains the i/o port #.

bit => a word value contains the contact input bit.

alarm\_type => word value that can be used to designate different alarm priorities that require specific alarm functions.

annun\_delay => word value containing the number of scans a contact alarm violation or event change is tolerated before annunciated. Some jobs require time delay in seconds.

annun\_delay\_count => a word value used to accumulate the number of scans or the time between initial change in state and alarm annunciation.

alarm\_dialer => word value set to the selected dialer channel to be activated in an alarm condition. A zero signifies alarm dialer disabled.

off\_cond\_index => word index to a contact condition string which describes the on condition of the discrete input.

on\_cond\_index => word index to a contact condition string which describes the off condition of the discrete input.

daily\_starts => word holding the number of starts (change of state 0 to 1) which have occurred during the day.

weekly\_starts => word holding the number of starts (change of state 0 to 1) which have occurred during the week.

monthly\_starts => word holding the number of starts (change of state 0 to 1) which have occurred during the month.

cumulative\_starts=> word holding the number of starts (change of state 0 to 1) which have occurred during since value reset. This variable can be used to count pump starts since last maintenence for a user resetable time period (i.e. like yearly starts).

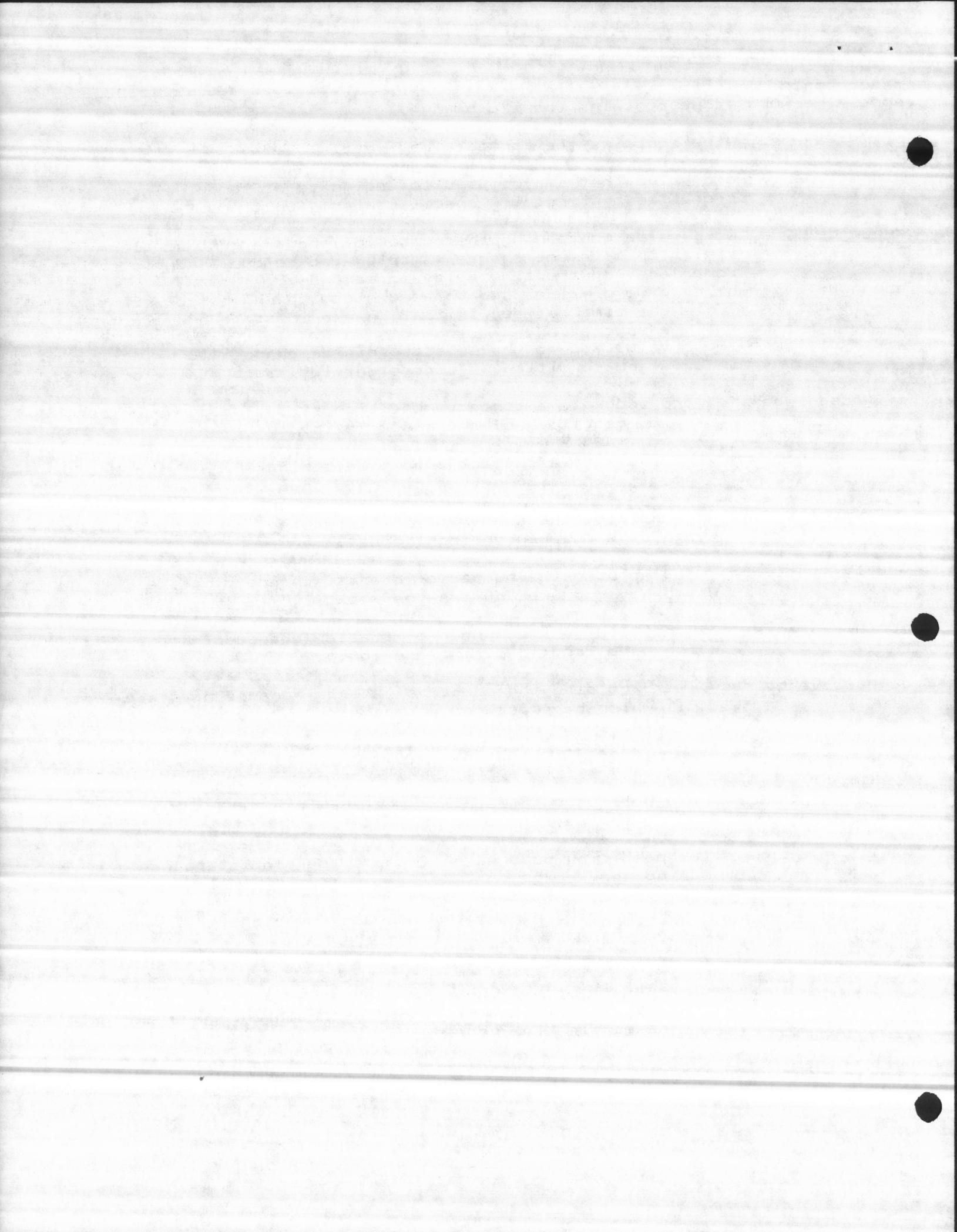
daily\_run\_time => word holding hundredth of hours the high state was active during the day.

weekly\_run\_time => word holding hundredth of hours the high state was active during the week.

monthly\_run\_time => word holding hundredth of hours the high state was active during the month.

cumulative\_run\_time=> word holding hundredth of hours the high state was active since value reset. This variable can be used to count pump runtimes since last maintenence or for a resetable time period.

edit\_change\_flag => a byte, to be set if any numeric value change in this structure, otherwise is reset.



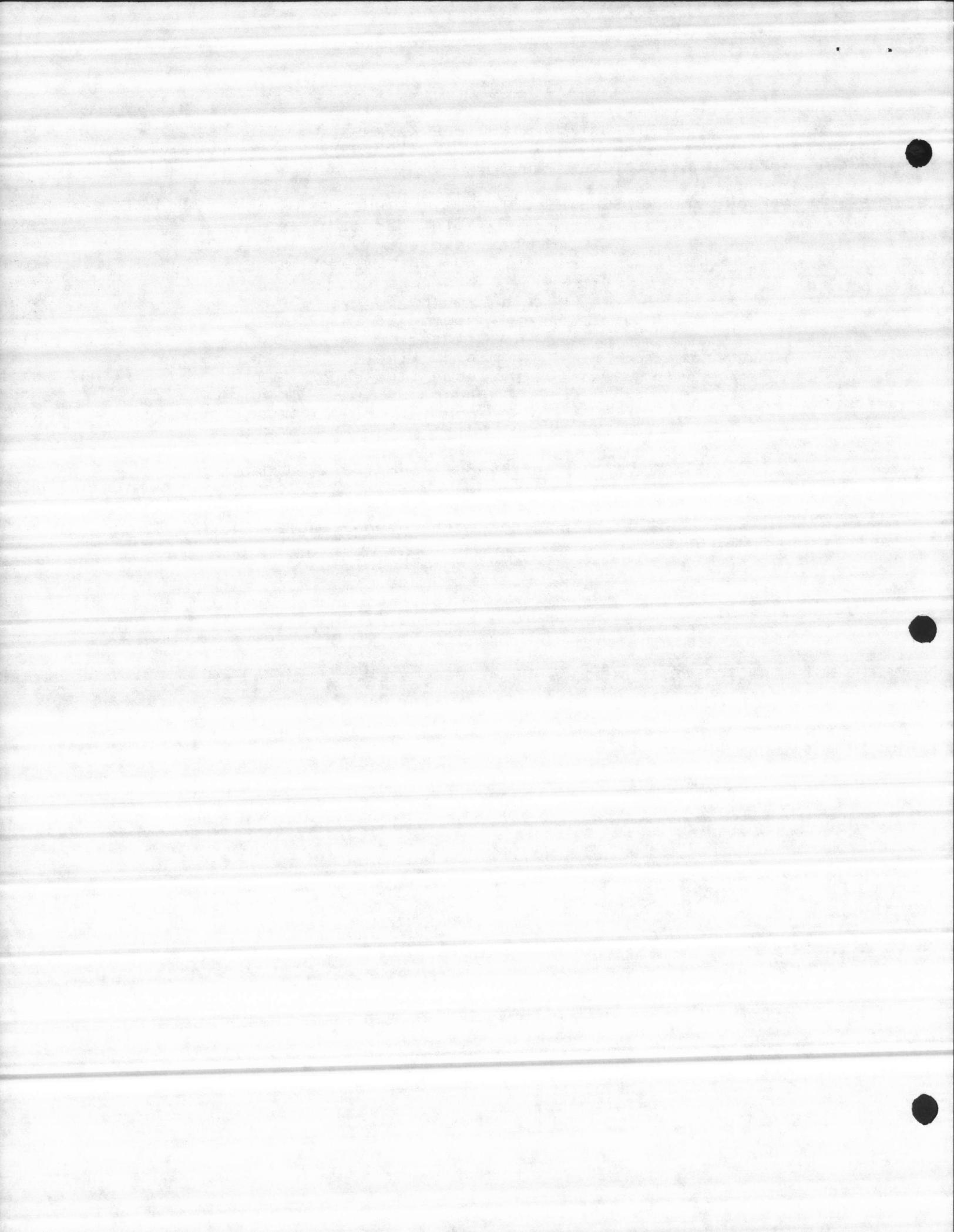
=====  
DISCRETE OUTPUT STRUCTURE & VARS  
=====

DECLARE discrete\_out\_reg\_t TOKEN PUBLIC;

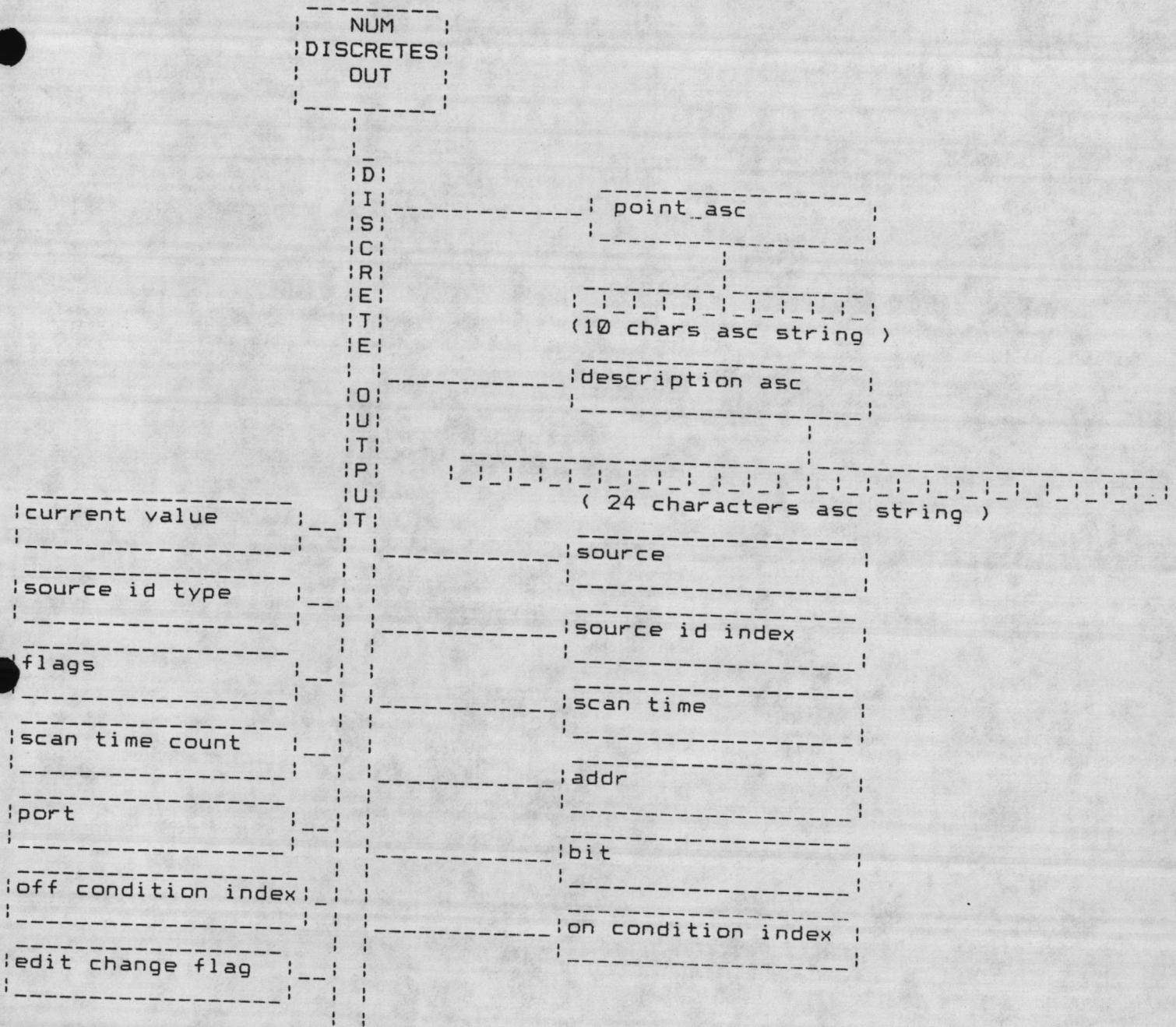
```
DECLARE discrete_out (NUM_DISCRETES_OUT_PLUS1) STRUCTURE
    (point_asc (10)     BYTE,
     desc_asc (24)    BYTE,
     current_value    WORD,
     source           WORD,
     source_id_type   BYTE,
     source_id_index  WORD,
     flags            WORD,
     scan_time        WORD,
     scan_time_count  WORD,
     addr             WORD,
     port             WORD,
     bit              WORD,
     off_cond_index   WORD,
     on_cond_index    WORD,
     edit_change_flag BYTE)

PUBLIC INITIAL (
    'Spare      ',
    'discrete output point  ',
    0, 0, 0FFH, 0,
    0, 60, 0, 0, 0, 0, 0, 0, 0, 0);
```

DISCRETE OUT STRUCTURE SIZE = 121 \* 58 = 7018 BYTES.



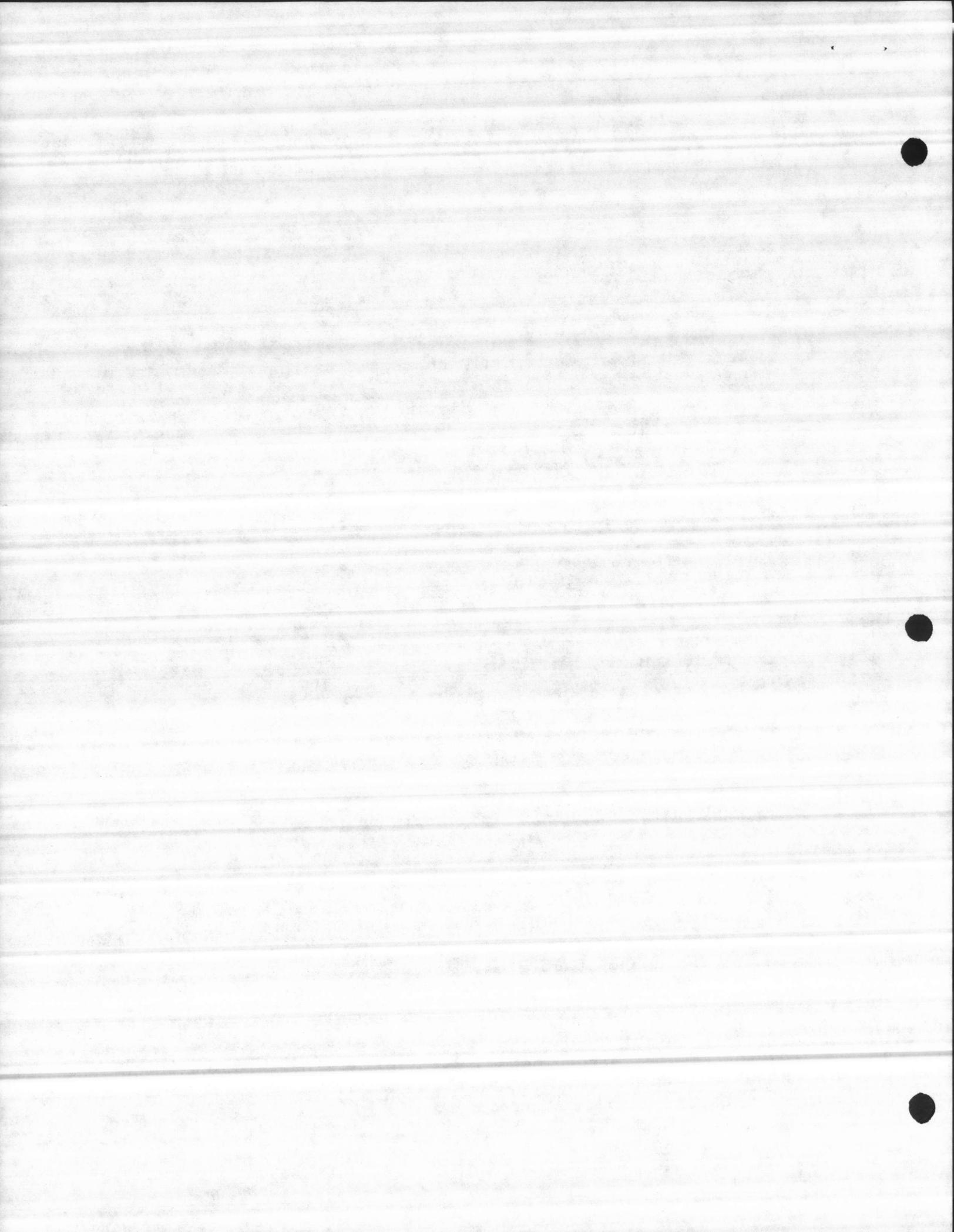
## DATA DIAGRAM :



`discrete_out_reg_t` - A token, is used for discrete output region protection.

The description and the use of each element entry in the `discrete_out` structure :

`point_asc` => 10 bytes ascii point id number used to access specific discrete output record or structure elements. Allows the operator to index discrete output by entering tag name. The `point_asc` is also used by the keyboard SELECT command to select any discrete output point in the system.



desc\_asc => 24 bytes ascii point description.

current\_value => a word value (On/Off)

source=> a word value that indicates which option of sources were chosen. A list could be :

- 0 - Undefined point
- 1 - Telemetry
- 2 - Local Rack
- 3 - Calculated
- 4 - Manual

source\_id\_type => a byte value contains the index to the structure.

source\_id\_index => a word value contains the index to the source.

flags => Word bitmapped flags as follows :

bit	0 - off	1 - on
1	Deactivate	/ Activate
2	normally closed	/ normally open
4	( not used)	
8	( not used)	
10	( not used)	
20	( not used)	
40	( not used)	
80	( not used)	
100	( not used)	
200	( not used)	
400	( not used)	
800	( not used)	

scan\_time => a word value, containing the number of seconds between output scans (defined with a six second resolution).

scan\_time\_count => variable time hold the elapsed time between actual scans.

addr => a word value contains the remote telemetry address of the contact output.

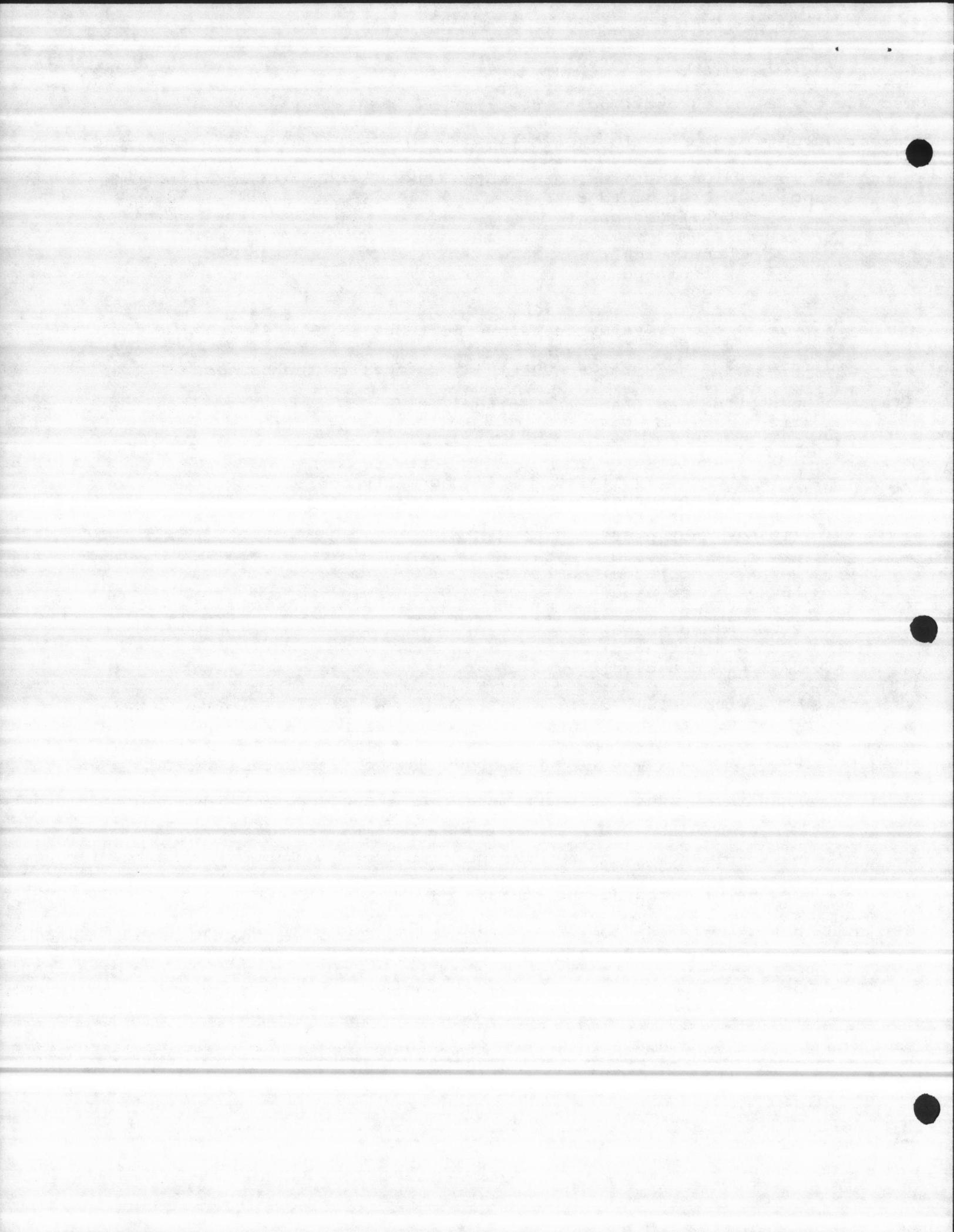
port => a word value contains the i/o port #.

bit => a word value contains the contact output bit.

off\_cond\_index => a word index to a discrete condition string which describes the off condition of the discrete output.

on\_cond\_index => a word index to a discrete condition string which describes the on condition of the discrete output.

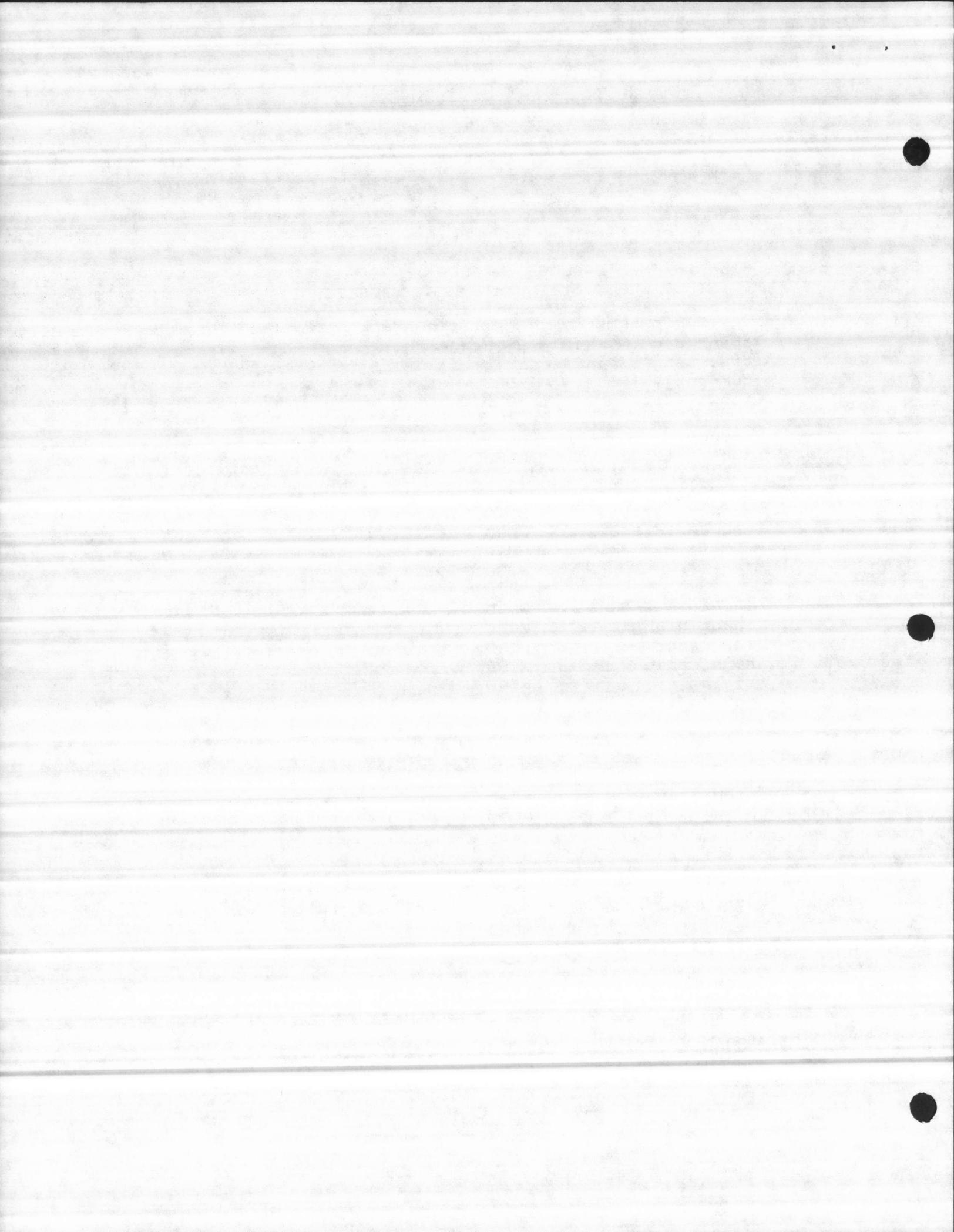
edit\_change\_flag => a byte, to be set if any numeric value change in this structure, otherwise is reset.



```
=          ANALOG INPUT STRUCTURE & VARS
=====
DECLARE  (analog_in_reg_t,
          analog_min_max_reg_t,
          analog roc_reg_t,
          analog_average_reg_t,
          analog_total_reg_t)      TOKEN PUBLIC;

DECLARE  disp_analog_in (NUM_ANALOGS_IN_PLUS1)
          STRUCTURE (total_scale WORD) PUBLIC;

DECLARE  analog_in (NUM_ANALOGS_IN_PLUS1) STRUCTURE
          (point_asc      (10)   BYTE,
           desc_asc       (24)   BYTE,
           current_value    WORD,
           input_status     WORD,
           source          WORD,
           input_flags      WORD,
           scan_time        WORD,
           scan_time_count  WORD,
           source_id_type   BYTE,
           source_id_index  WORD,
           addr            WORD,
           port             WORD,
           input_low_range  WORD,
           input_high_range WORD,
           low_range         WORD,
           high_range        WORD,
           scale            WORD,
           eng_units         WORD,
           alarm_deadband   WORD,
           low_low_alarm    WORD,
           low_alarm         WORD,
           high_alarm        WORD,
           high_high_alarm  WORD,
           daily_min         WORD,
           daily_min_time   DWORD,
           daily_max         WORD,
           daily_max_time   DWORD,
           weekly_min        WORD,
           weekly_min_time  DWORD,
           weekly_max        WORD,
           weekly_max_time  DWORD,
           monthly_min       WORD,
           monthly_min_time DWORD,
           monthly_max       WORD,
           monthly_max_time DWORD,
```



```
roc_time_period      WORD,
roc_alarm_setpoint   WORD,
last_roc_sample      WORD,
roc_time_count       WORD,

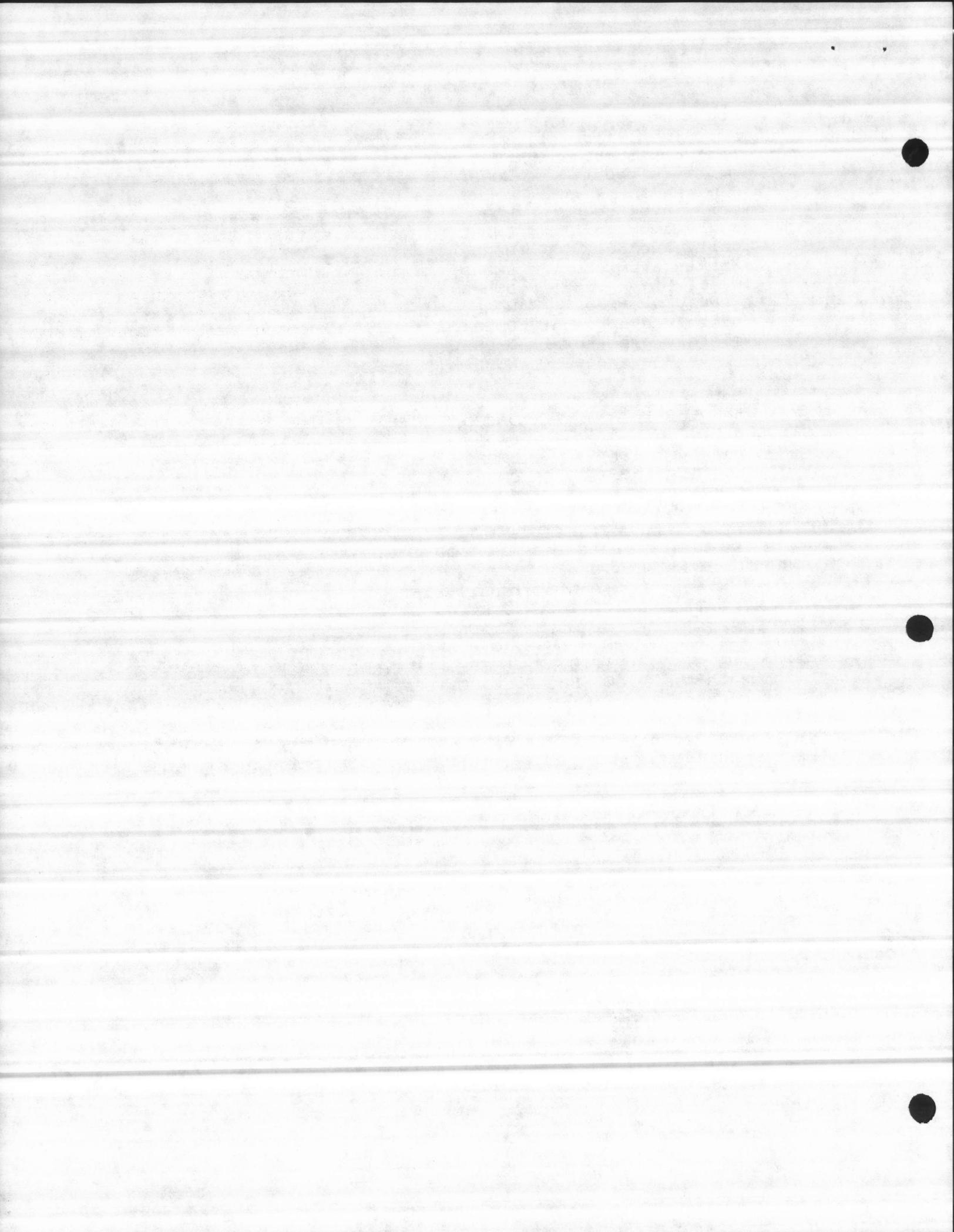
daily_aver_count     WORD,
daily_aver_total     DWORD,
daily_average        WORD,
weekly_aver_count    WORD,
weekly_aver_total    WORD,
weekly_average       WORD,
monthly_aver_count   WORD,
monthly_aver_total   WORD,
monthly_average      WORD,

total_type           WORD,
sub_total            DWORD,
daily_total          WORD,
weekly_total         WORD,
monthly_total        WORD,
cumulative_total     WORD,

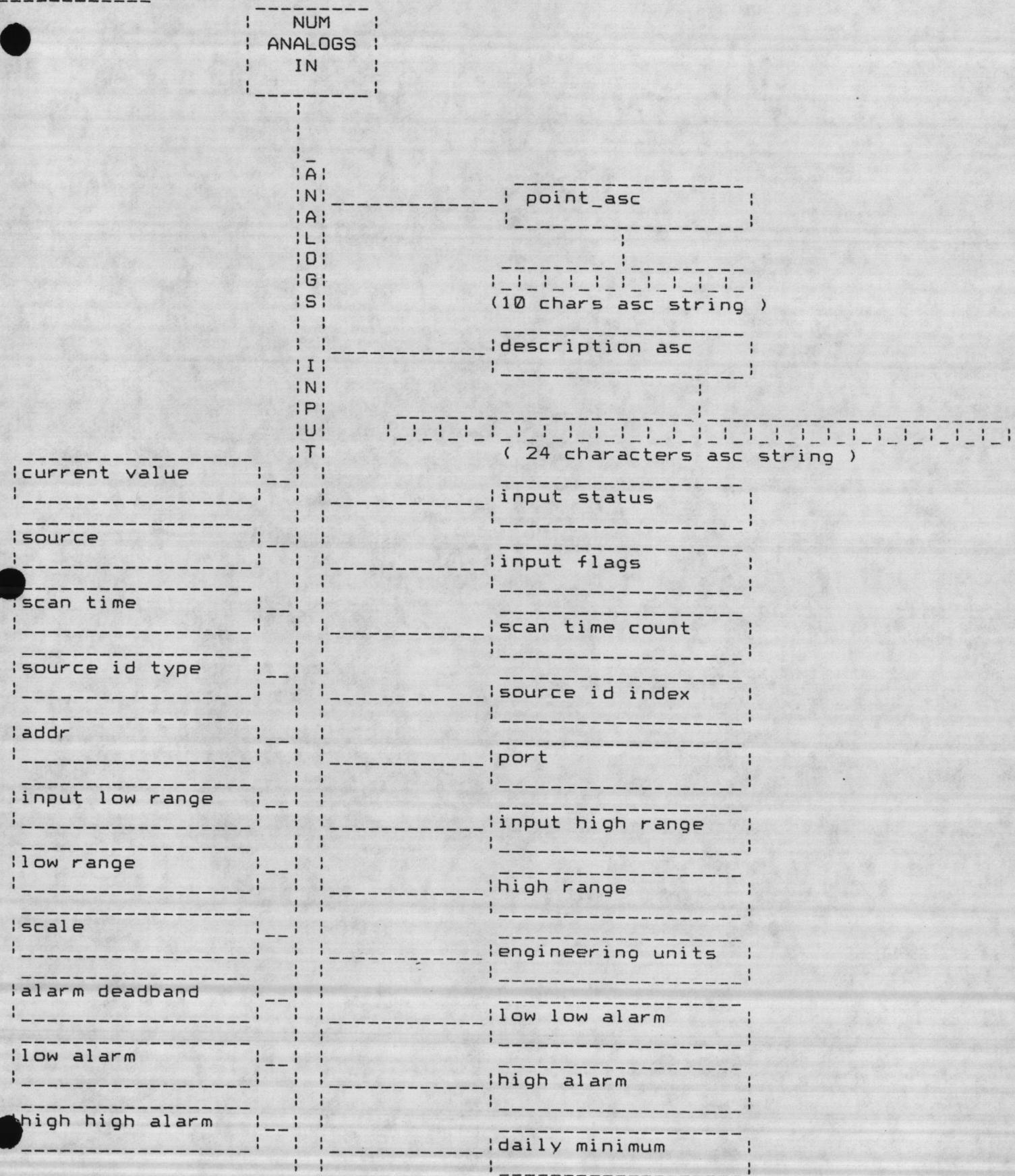
edit_change_flag     BYTE)

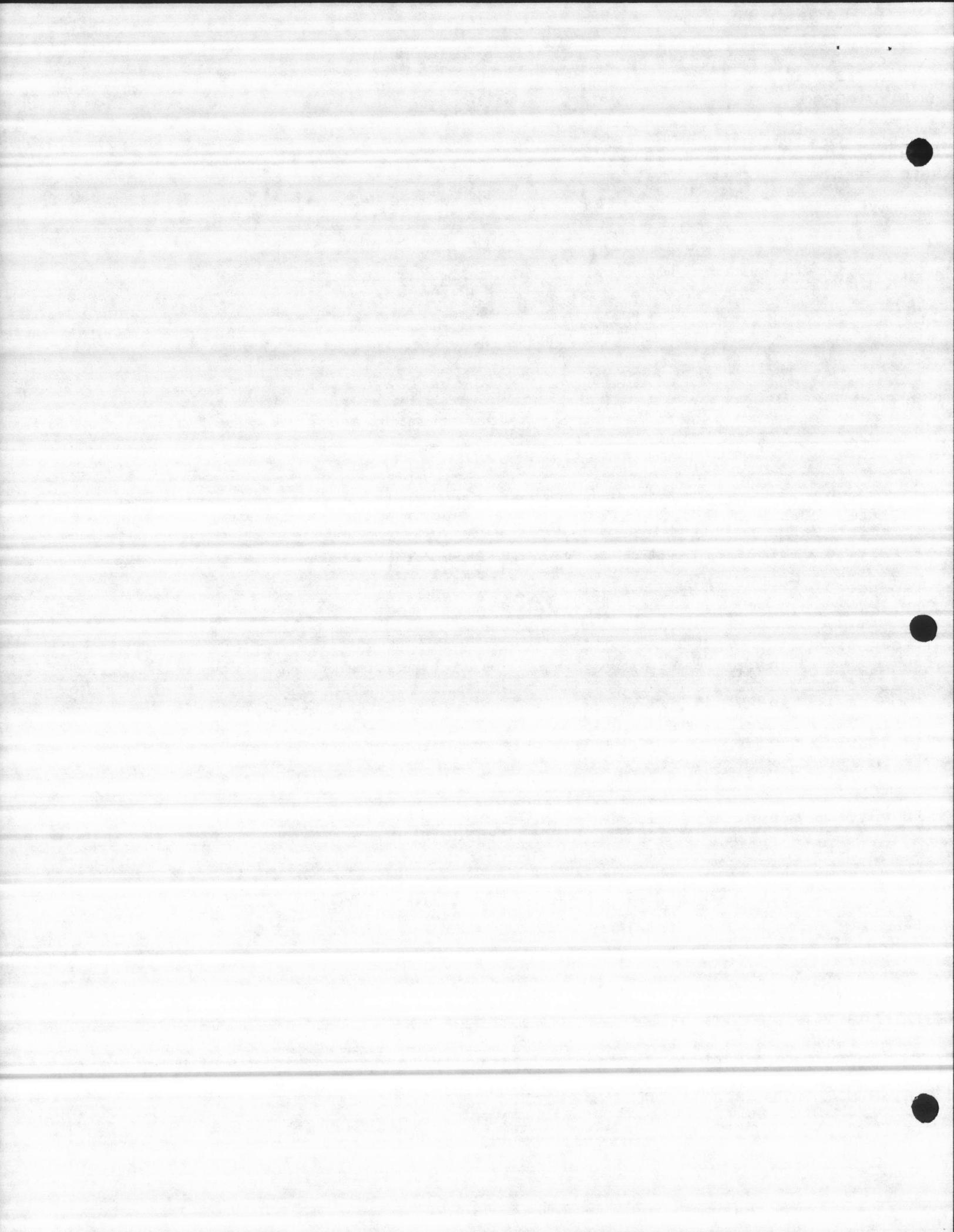
PUBLIC INITIAL ('Spare      ,
                 'Analog input point      ,
                 0, 1H,
/* input info */ 0, 0, 0, 0, 0FFH, 0, 0, 0,
/* analog info */ 0, 0FFFH, 0, 0FFFH, 0, 0,
/* alrm sp info */ 1, 0, 0, 0FFFH, 0FFFH,
/* minmax info */ 0FFFH, 0, 0, 0, 0FFFH, 0, 0, 0,
/* rate of chg */ 0FFFFH, 0FFFFH, 0, 0,
/* average info */ 0, 0, 0, 0, 0, 0, 0, 0, 0,
/* total info */ 0, 0, 0, 0, 0, 0);
```

ANALOG IN STRUCTURE SIZE = 34 \* 170 = 5780 BYTES.



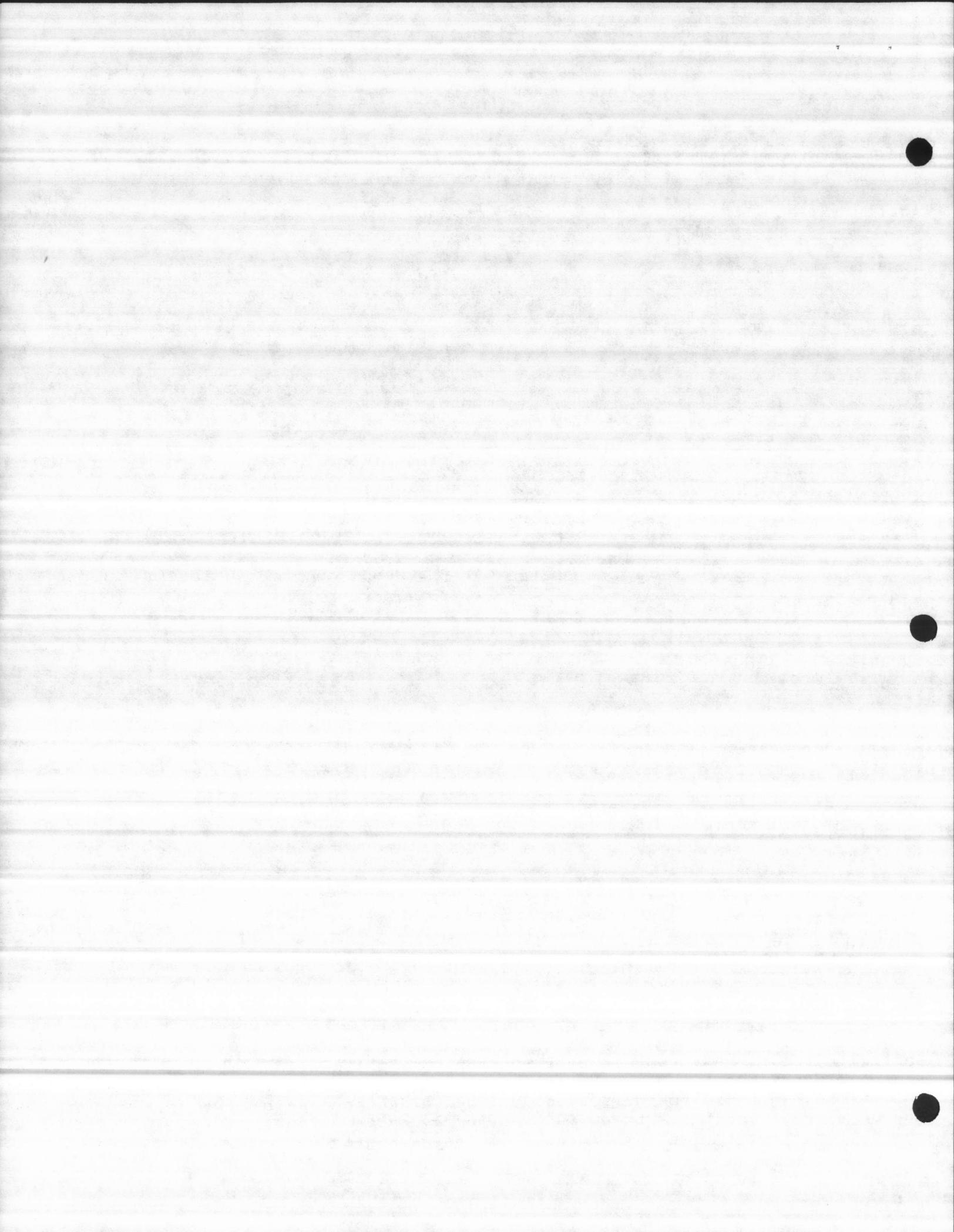
## DATA DIAGRAM :





daily minimum time	-----  daily maximum
daily maximum time	-----  weekly minimum
weekly min time	-----  weekly maximum
weekly max time	-----  monthly minimum
monthly min time	-----  monthly maximum
monthly max time	-----  ROC time period
ROC alarm setpoint	-----  last roc sample
roc time count	-----  daily average count
daily aver. total	-----  daily average
weekly aver. count	-----  weekly aver. total
weekly average	-----  monthly aver. count
monthly aver.total	-----  monthly average
total type	-----  sub total
daily total	-----  weekly total
monthly total	-----  cumulative total
edit change flag	-----

analog\_in\_reg\_t - A token, is used for analog input region protection.  
analog\_min\_max\_reg\_t - A token, is used for min/max region.  
analog\_roc\_reg\_t - A token is used for rate of change region.  
analog\_average\_reg\_t - A token is used for average region.  
analog\_total\_reg\_t - A token is used for total region.



disp\_analog STRUCTUR :

total\_scale => a word holds the scale for the total for each analog.  
analog\_in STRUCTURE :

The description and the use of each element entry in the analog\_in structure  
point\_asc => 10 bytes ascii point id number. used to access specific analog  
point or structure elements. Allows the operator to index  
analog input by entering lab name. The point\_asc is also used  
by the keyboard SELECT command to select any analog point in the  
system.

desc\_asc => 24 bytes ascii point description.

current\_value => a word holds a digital input value after input processing  
(edit allowed if A/D is deactivated or in manual mode).

input\_status => word value set as following bit mask. This word will be  
accessed by a discrete element for alarm/event.

source => a word value that indicates which option of input devices was  
chosen. A list of sources :

- 0 - undefined point.
- 1 - telemetry input.
- 2 - local rack input.
- 3 - Calculation point.
- 4 - manual input.
- 5 - TI PM550 input
- 6 - Gould Micro84 input

input\_flags => Word bitmapped flags as follows :

bit	0 - off	1 - on
1	- Deactivate	/ Activate
2	- Hex input format	/ BCD input format
100	- no alarm gen	/ alarm generation
200	- no min/max gen	/ min/max generation
400	- no roc alarm	/ roc alarm
800	- no averagin	/ averaging enabled
1000	- no totalization	/ totalization enabled.

scan\_time => a word value, containing the number of seconds between input  
scan (defined with a six sec resolution)

scan\_time\_count => variable time hold the elapsed time between actual scans.

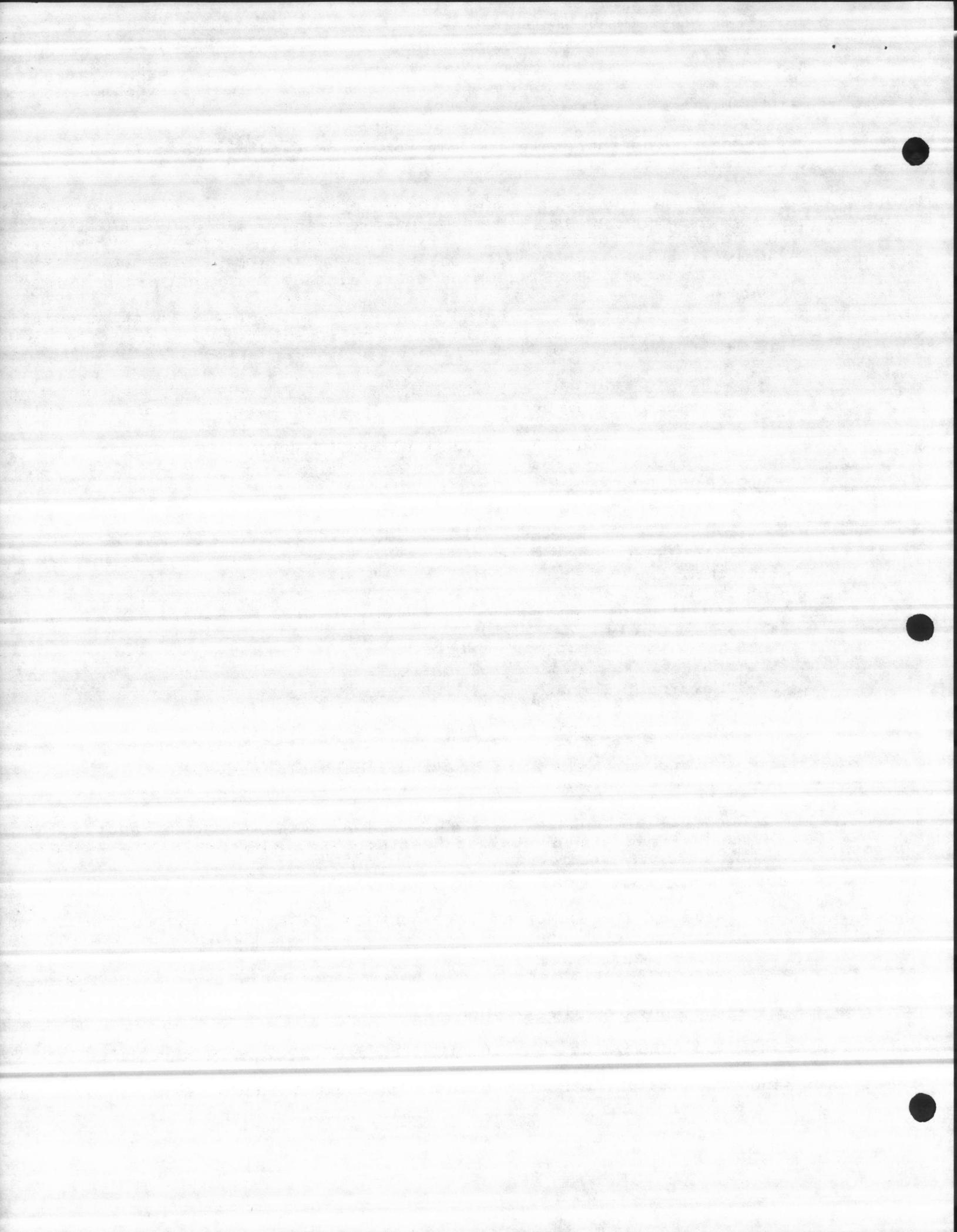
source\_id\_type => a byte value contains the index to the structure type.  
source\_id\_index => a word value contains the index to the source.

addr => a word value contains the remote telemetry address of the analog  
input.

port => a word value contains the i/o port #.

input\_low\_range => a word value contains the low digital input range value.

input\_high\_range=>a word value contains the high digital input range value.



input\_low\_range => a word value contains the low output range after the linear input scaling.

input\_high\_range => a word value contains the high output range after the linear input scaling.

scale => 2's complement word value containing the exponential power of ten scale for the digital value.

eng\_units => a word index to the eng. units string table, which is defined as follows :

0 - null	1 - Inches	2 - MGD
3 - GPM	4 - Kwh	5 - Feet
6 - Kcfh	7 - Celsius	8 - mg/l
9 - Cfm	10 - PSI	11 - % ....

alarm\_deadband => word value contains the allowed deadband for low and high alarms

low\_low\_alarm => word value contains the low alarm setpoint.

low\_alarm => word value contains the low low alarm setpoint.

high\_high\_alarm => word value contains the high high alarm setpoint.

high\_alarm => word value contains the high alarm setpoint.

daily\_min\_time => standard time dword containing the time of occurrence of the daily minimum.

daily\_max => a word value contains the maximum processed digital input for the day.

daily\_max\_time => standard time dword containing the time of occurrence of the daily maximum.

weekly\_min => a word value contains the minimum processed digital input for the day.

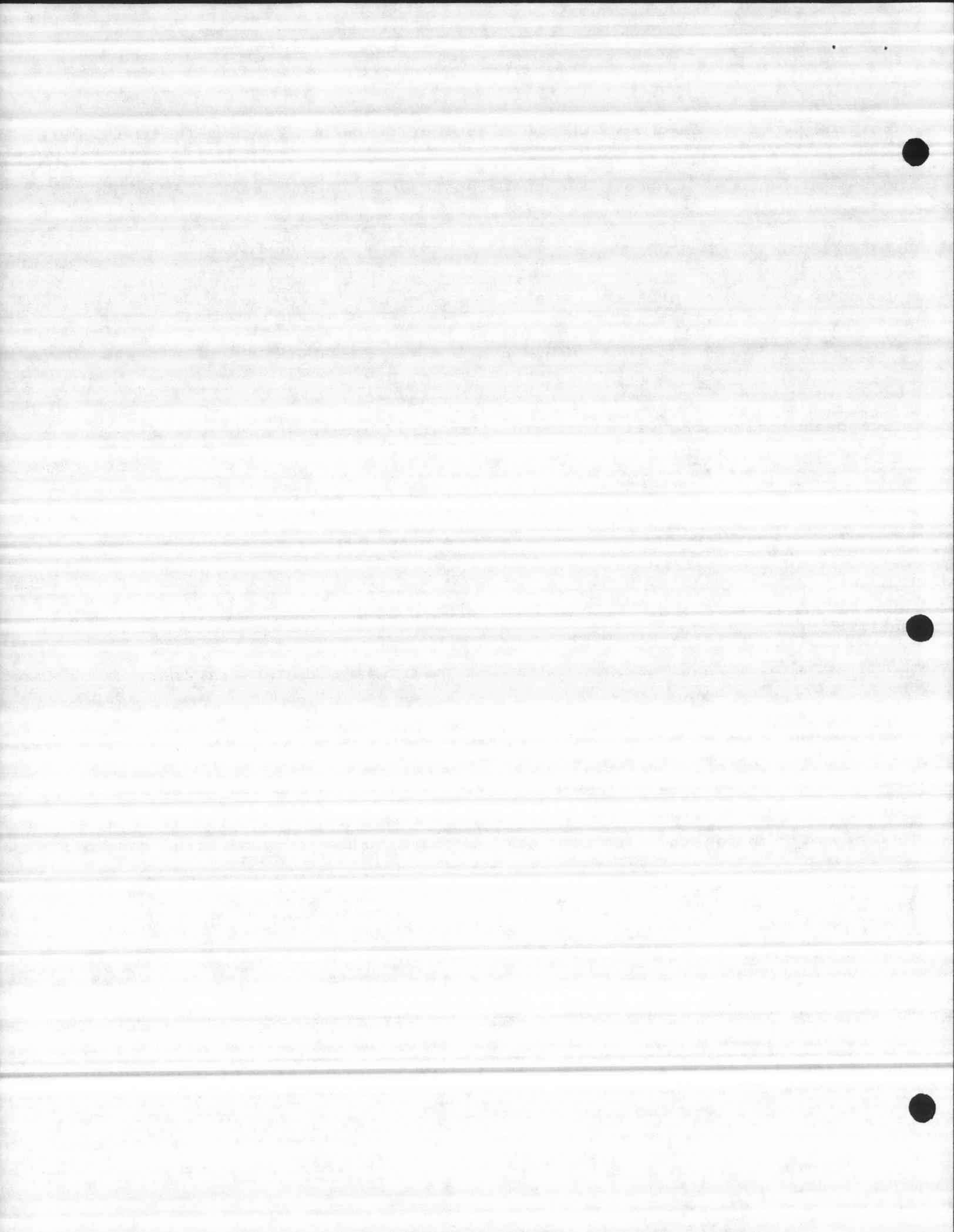
weekly\_min\_time => standard time dword containing the time of occurrence of the weekly minimum.

weekly\_max => a word value contains the maximum processed digital input for the day.

weekly\_max\_time => standard time dword containing the time of occurrence of the weekly maximum.

monthly\_min => a word value contains the minimum processed digital input for the day.

monthly\_min\_time => standard time dword containing the time of occurrence of the monthly minimum.



monthly\_max => a word value contains the maximum processed digital input for the day.

monthly\_max\_time => standard time dword containing the time of occurrence of the monthly maximum.

roc\_time\_period => word value contains the time period to measure for rate of change. units are tenths of a minutes. This means that a sample is taken and every time period another sample is taken and compared with the previous sample.

roc\_alarm\_setpoint => word used to indicate a maximum allowable rate of change An alarm bit is set in the status word if the absolute value difference between the current sample and the last sample exceeds this setpoint.

last\_roc\_sample => word value holding the last examined rate of change sample.

roc\_time\_sample => word value used to accumulate the time between samples.

daily\_aver\_count => word value holding the number of averaging values taken for the daily average determination. Otherwise it is equal to the frequency if the time interval is equal to daily.

daily\_aver\_total => a dword value holding the total accumulation of all the values in the day

daily\_average => the current average value for the week, which is calculated by dividing the daily\_aver\_total by the daily\_aver\_count.

weekly\_aver\_count => word value holding the number of averaging values taken for the weekly average determination. Otherwise it is equal to the frequency if the time interval is equal to weekly.

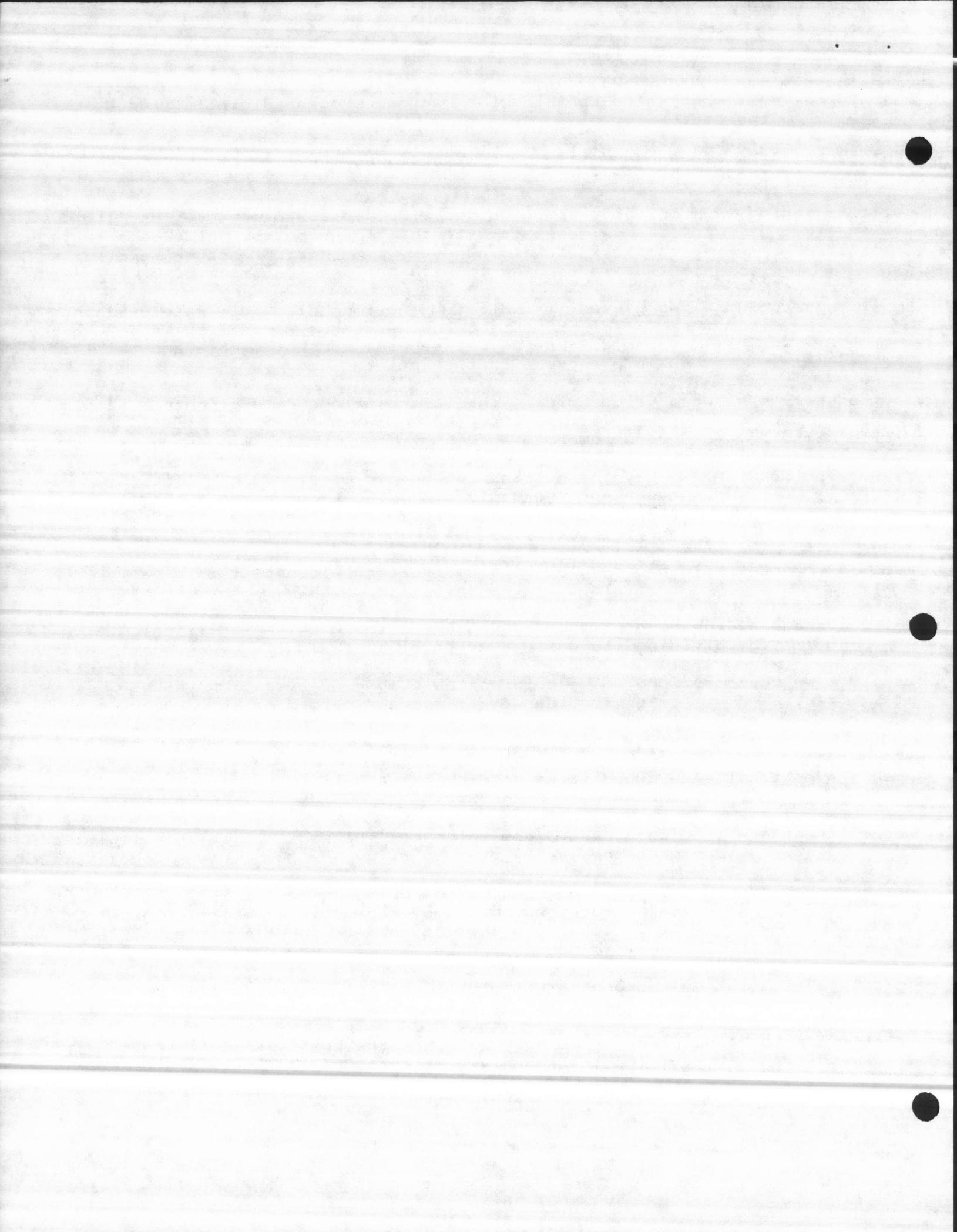
weekly\_aver\_total => a dword value holding the total accumulation of all the values in the week

weekly\_average => the current average value for the week, which is calculated by dividing the weekly\_aver\_total by the weekly\_aver\_count.

monthly\_aver\_count => word value holding the number of averaging values taken for the monthly average determination. Otherwise it is equal to the frequency if the time interval is equal to monthly.

monthly\_aver\_total => a dword value holding the total accumulation of all the values in the month

monthly\_average=>the current average value for the month, which is calculated by dividing the monthly\_aver\_total by the monthly\_aver\_count.



-----  
total\_type => word index to type of totalization required by the digital  
input rate, for example Mgd to KGal.

sub\_total => a dword value holding the remainder of any totalization.  
This variable assures accuracy in flow totalization.

daily\_total => a dword the totalization in the specified total type for the  
day.

weekly\_total => a dword the totalization in the specified total type for the  
week.

monthly\_total => dword the totalization in the specified total type for the  
month.

cumulative\_total => a dword the totalization in the specified total type of  
a user defined period. the user must manually reset this  
variable every wanted time period.

source\_id\_type => a byte value contains the index to the structure type.

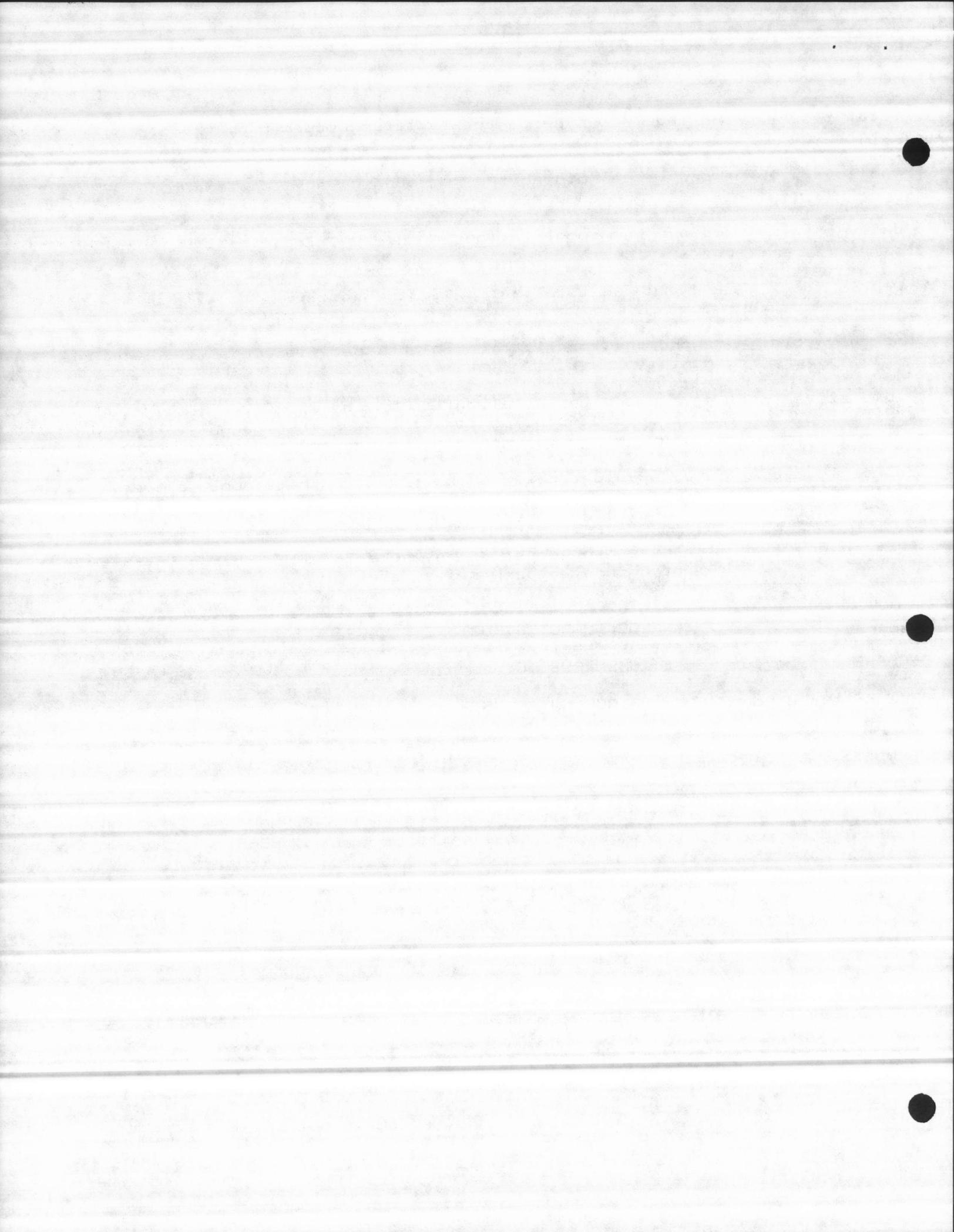
source\_id\_index => a word value contains the index to the source.

edit\_change\_flag => a byte, to be set if any numeric value change in this  
structure, otherwise is reset.

===== ANALOG OUTPUT STRUCTURE & VARS =====

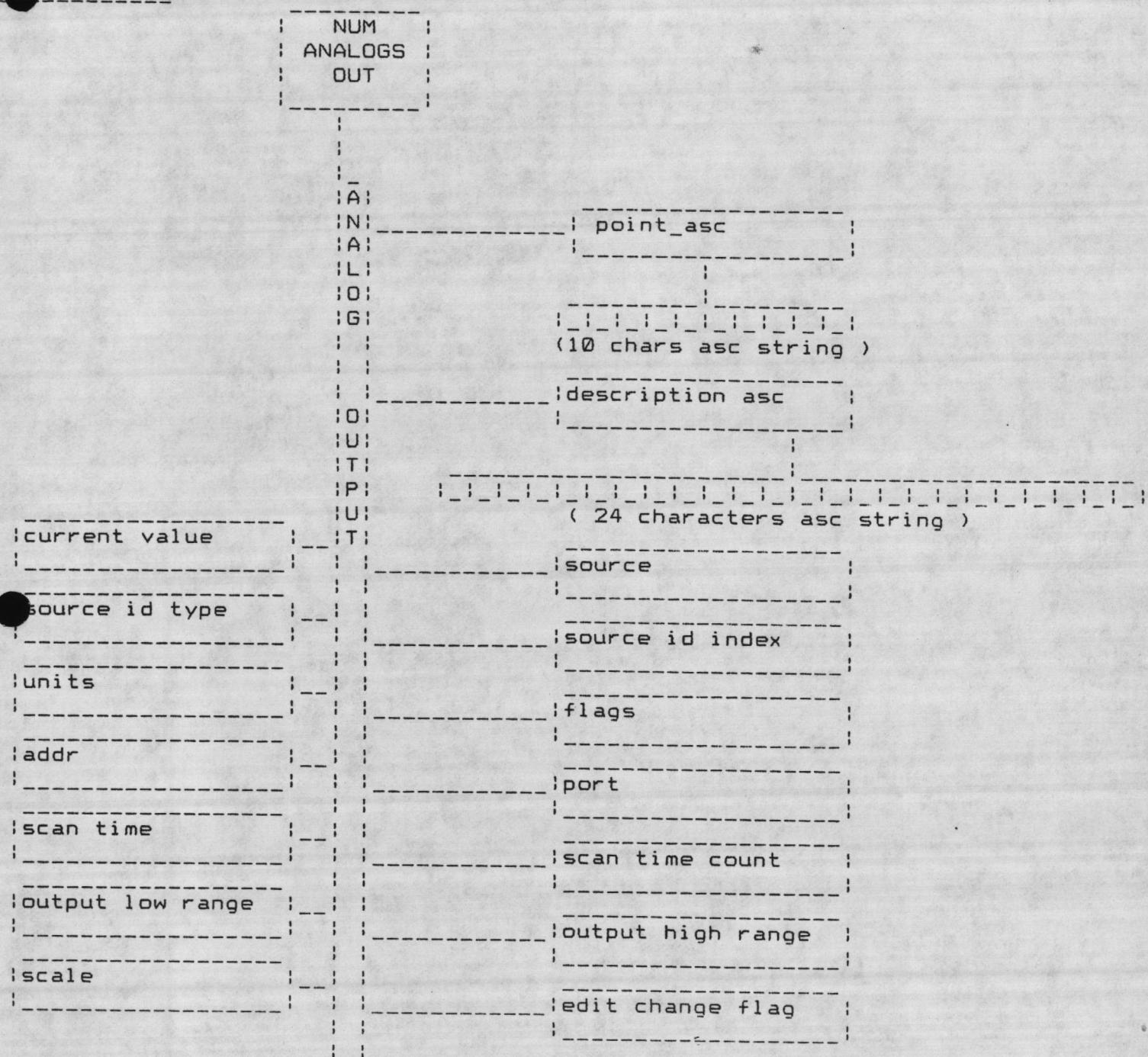
DECLARE analog\_out\_reg\_t TOKEN PUBLIC;

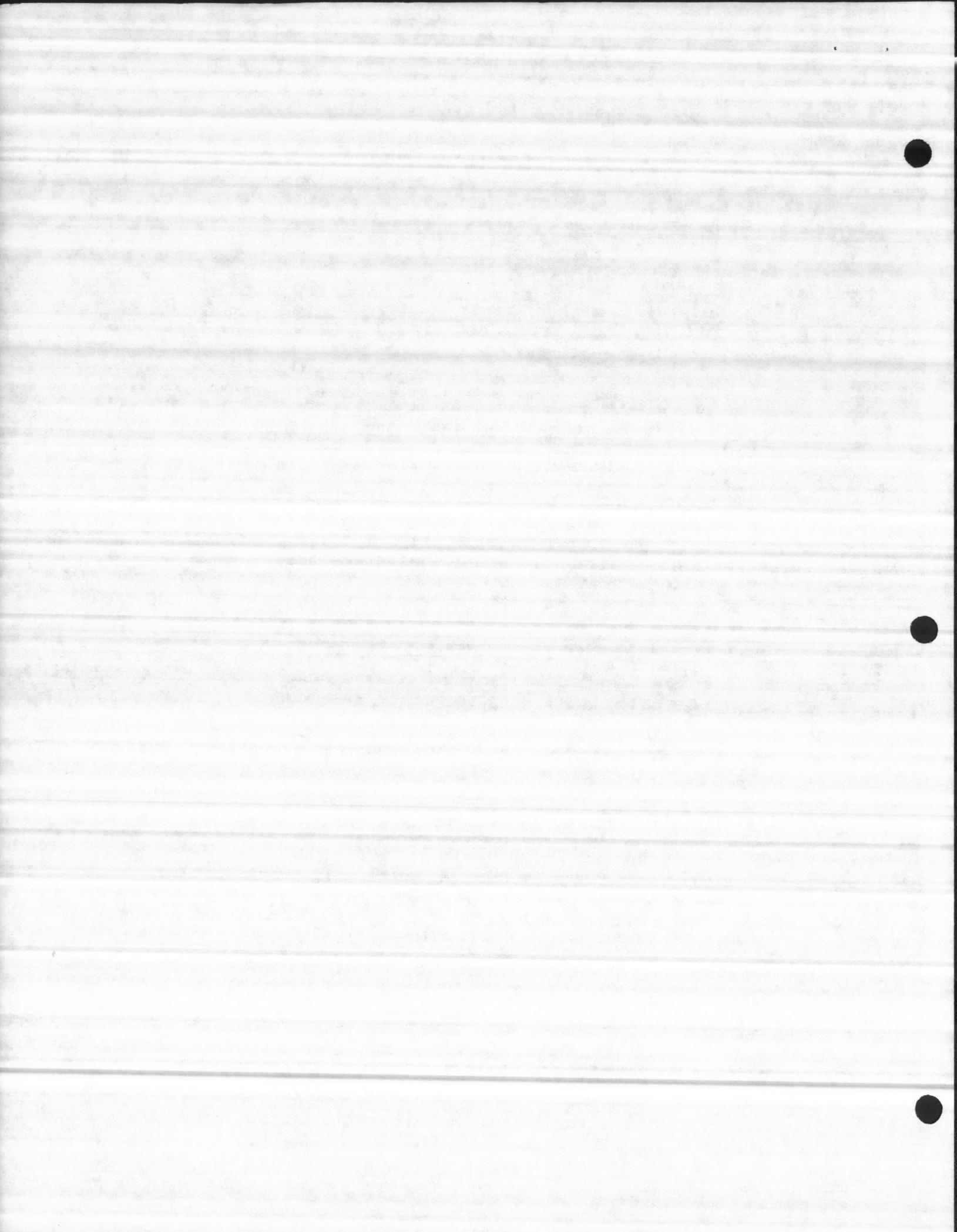
DECLARE analog\_out (NUM\_ANALOGS\_OUT\_PLUS1) STRUCTURE  
(point\_asc (10) BYTE,  
desc\_asc (24) BYTE,  
value WORD,  
source WORD,  
source\_id\_type BYTE,  
source\_id\_index WORD,  
units WORD,  
flags WORD,  
addr WORD,  
port WORD,  
scan\_time WORD,  
scan\_time\_count WORD,  
output\_low\_range WORD,  
output\_high\_range WORD,  
  
scale WORD,  
  
edit\_change\_flag BYTE)  
  
PUBLIC INITIAL (  
'Spare ',  
'Analog output point ',  
0,1,0FFH,0,  
0, 0, 0, 0, 60,0,0,0FFFFH,0,0);



ANALOG OUT STRUCTURE SIZE = 2 \* 60 = 120 BYTES.

DATA DIAGRAM :





analog\_out\_reg\_t => A token used for the analog output region protection.

~~analog\_out STRUCTURE :~~

The description and the use of each element entry in the analog\_out structure

point\_asc => 10 bytes ascii point id number. used to access specific analog output record or structure elements. Allows the operator to index analog output by entering tag name. The point\_asc is also used by the keyboard SELECT command to select any analog point in the system.

desc\_asc => 24 bytes ascii point description.

value => a word value, the numeric data (manually input or calculated).

source=> a word value that indicates which data type was chosen. A list could be :

- 0 - Undefined point
- 1 - Telemetry
- 2 - Local Rack
- 3 - Calculated
- 4 - Manual

source\_id\_type => a byte value contains the index to the structure type.

source\_id\_index => a word value contains the index to the source.

units => a word index to the eng. units string table (as defined in analog input)

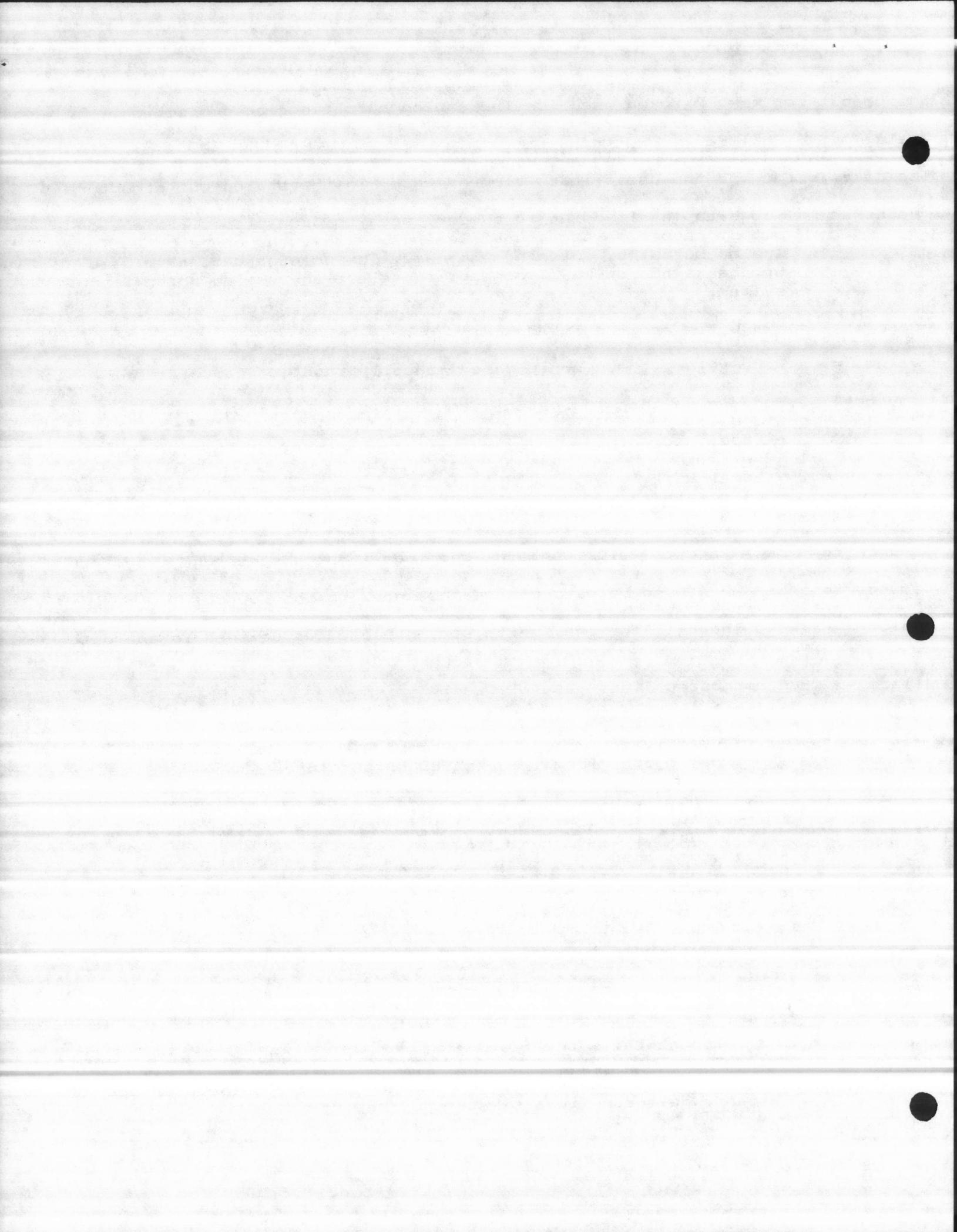
flags => Word bitmapped flags as follows :

bit	0 - off	1 - on
1	- Deactivate	/ Activate
2	- Hex output format	/ BCD output format
4	- ( not used )	
8	- ( not used )	
10	- ( not used )	
20	- ( not used )	
40	- ( not used )	
80	- ( not used )	
100	- ( not used )	
200	- ( not used )	
400	- ( not used )	
800	- ( not used )	

addr => a word value contains the remote telemetry address of the contact output.

port => a word value contains the i/o port #.

scan\_time => a word value, containing the number of seconds between output scans (defined with a six second resolution).



scan\_time\_count => variable time hold the elapsed time between actual scans.  
output\_low\_range => a word value contains the low range value.  
output\_high\_range => a word value contains the high range value.  
edit\_change\_flag => a byte, to be set if any numeric value change in this structure, otherwise is reset.

=====

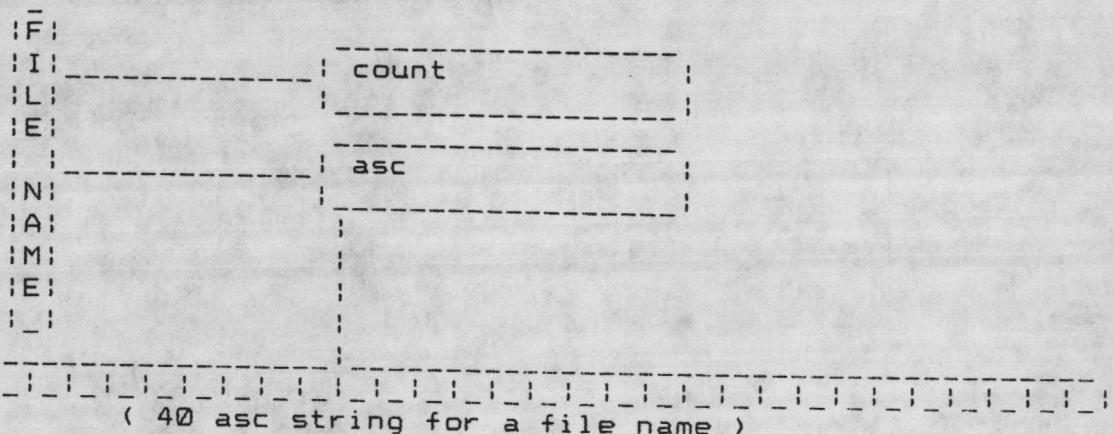
= FILE NAME STRUCTURE =

=====

DECLARE file\_name STRUCTURE (count BYTE,  
asc (40) BYTE) PUBLIC INITIAL (0);

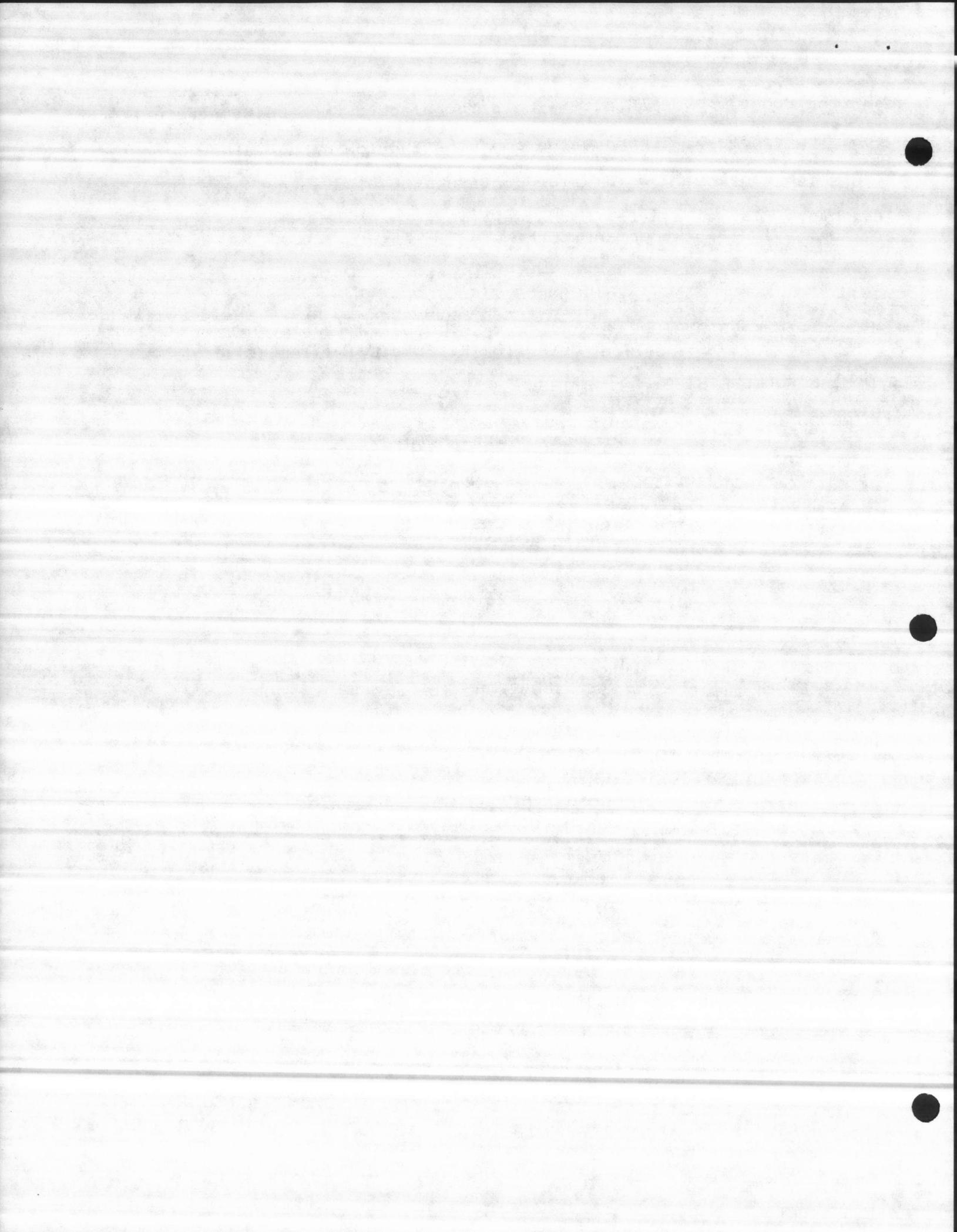
FILE NAME STRUCTURE SIZE = 41 BYTES.

DATA DIAGRAM :



file\_name STRUCTURE :

count - A byte value holds the file name count.  
asc (40) BYTE - Is a 20 byte holds the file name asc.



## DISK HEADER DECLARATIONS

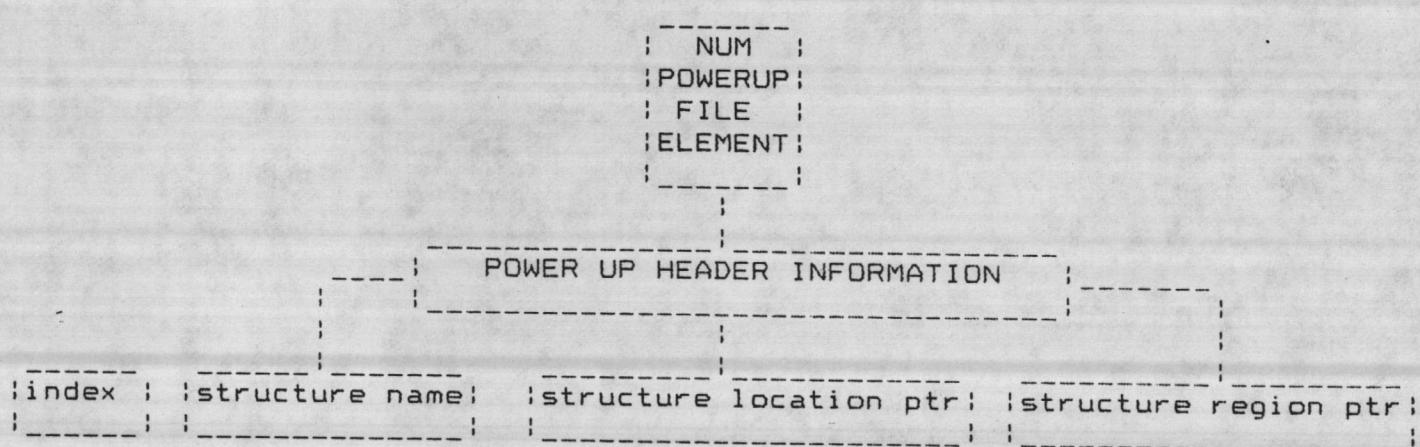
```
DECLARE power_up_header_info (NUM_POWER_UP_FILE_ELEMENTS) STRUCTURE
    (struc_name (6)     BYTE,
     struc_loc_P       POINTER,
     struc_reg_P       POINTER) PUBLIC INITIAL (
```

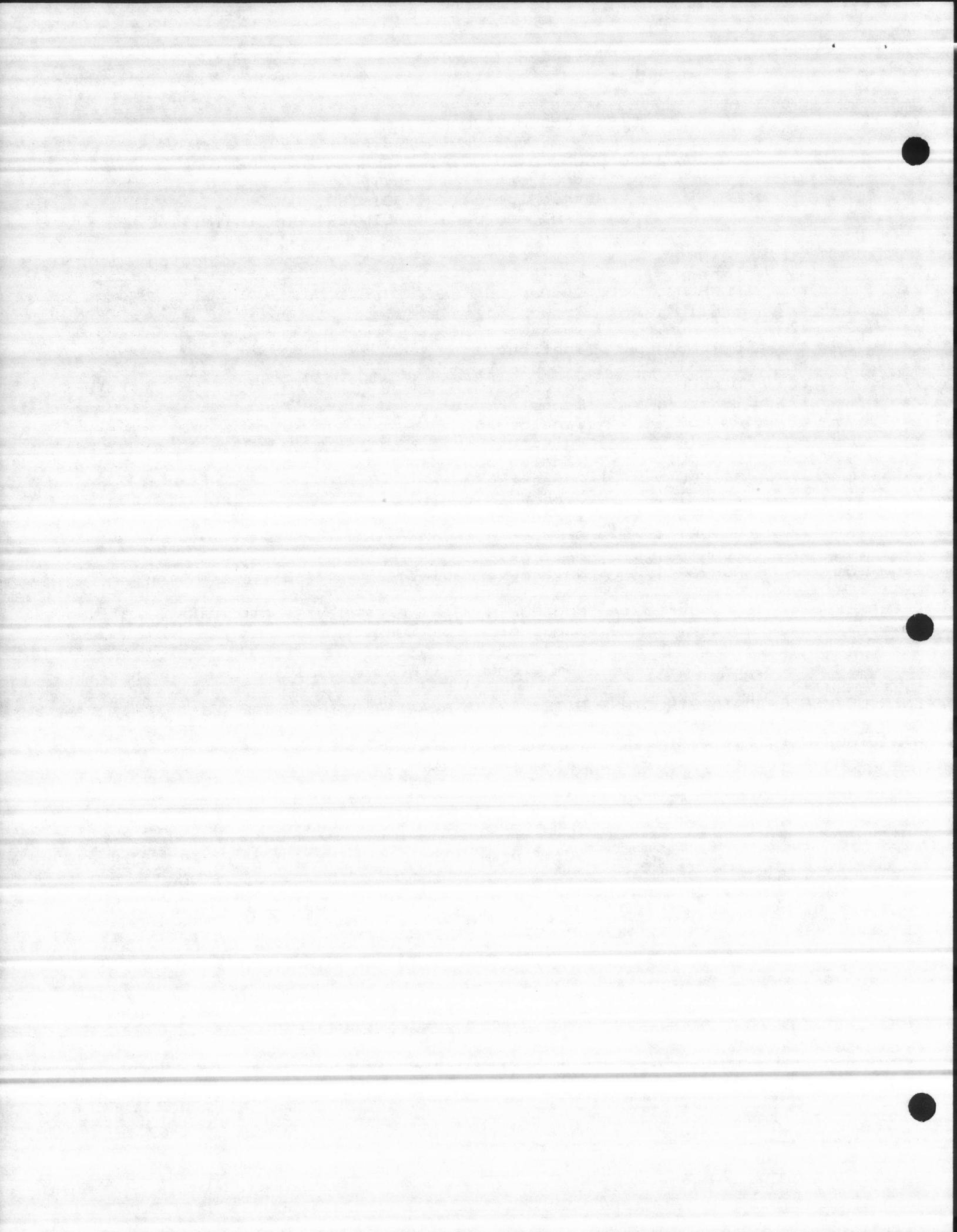
/* index	name	location	region	*/
/* 0 */	'PWRHDR'	@power_up_header_data,	0,	*/
/* 1 */	'PWRTIM'	@power_up_save_time,	0,	
/* 2 */	'PSSWRD'	@password,	0,	
/* 3 */	'OPTIONS'	@options,	0,	
/* 4 */	'IOPORT'	@io_ports,	0,	
/* 5 */	'DIALER'	@dialers,	0,	
/* 6 */	'SONALE'	@sonalert_struc,	0,	
/* 7 */	'ANALOG'	@analog_in,	0,	
/* 8 */	'DISCIN'	@discrete_in,	0,	
/* 9 */	'ANALOU'	@analog_out,	0,	
/* 10 */	'DISCOU'	@discrete_out,	0,	
/* 11 */	'CONDST'	@cond_str_table,	0,	
/* 12 */	'GROUPD'	@group_disp,	0,	
/* 13 */	'REPORT'	@report_disp,	0,	
/* 14 */	'ALARMS'	@alarms,	0,	
/* 15 */	'ALRVAR'	@alarm_variables,	0);	

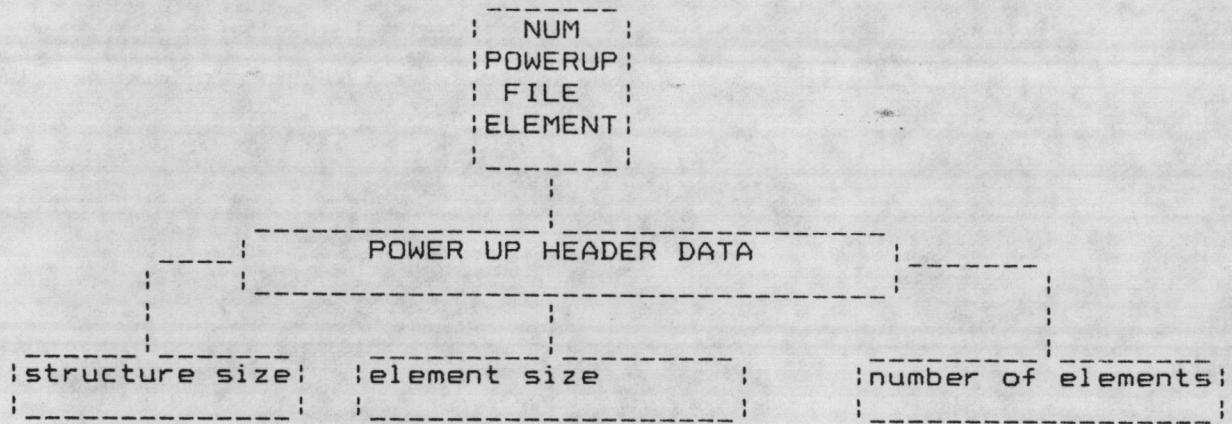
```
DECLARE power_up_header_data (NUM_POWER_UP_FILE_ELEMENTS) STRUCTURE
    (struc_size      WORD,
     element_size    WORD,
     num_elements    WORD) PUBLIC;
```

POWER UP HEADER DATA STRUCTURE SIZE = 24 \* 6 = 144 BYTES.

DATA DIAGRAM :







`power_up_header_info STRUCTURE :`

This structure holds the following information:

`struc_name (6)` - six bytes created name for each structure in the power up file.

`struc_loc_p` - A pointer points to the structure.

`struc_reg_p` - A pointer points to the structure protected region if any otherwise is 0.

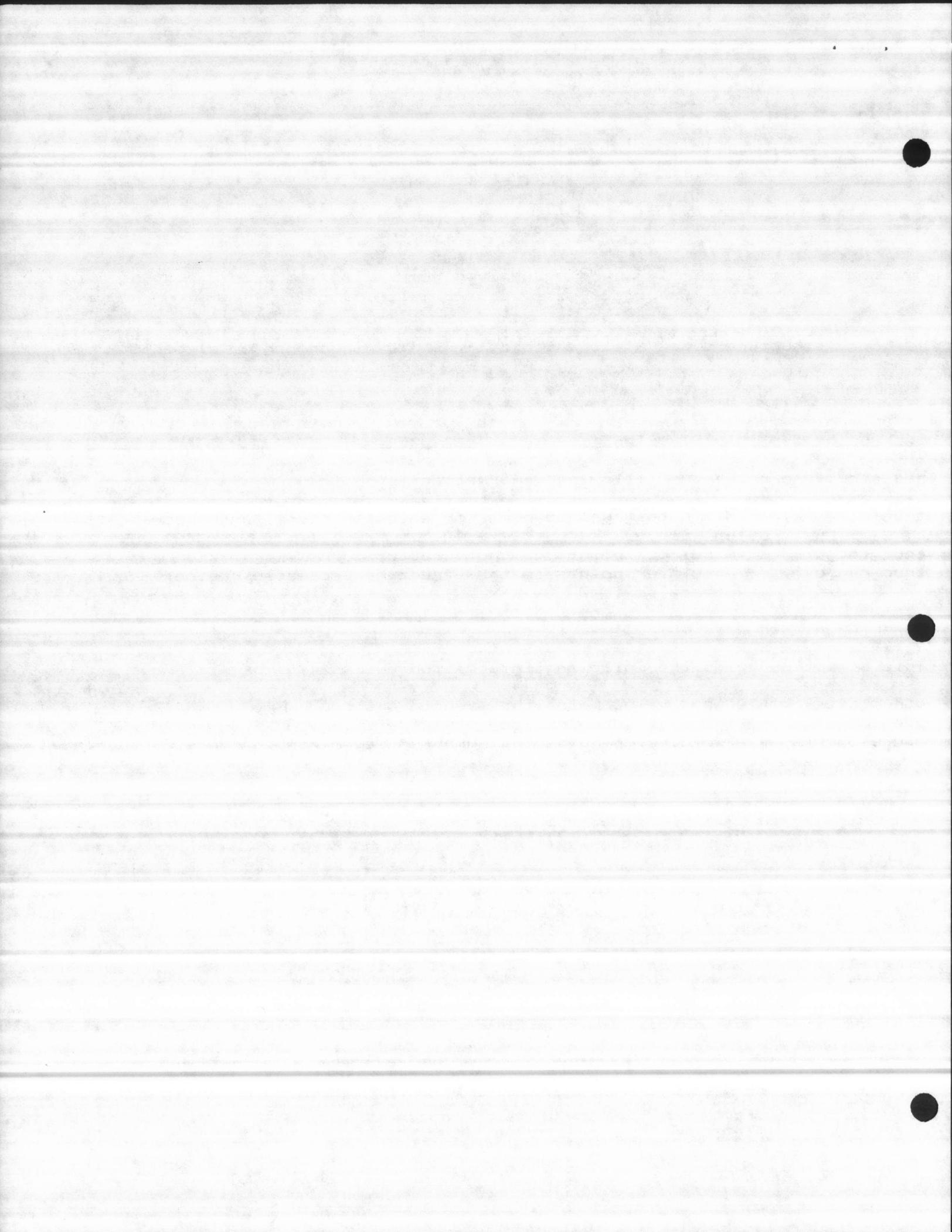
`power_up_header_data STRUCTURE :`

This structure holds the following information:

`struc_size` - A word value holds the structure size.

`element_size` - A word value holds the element size.

`num_elements` - A word value holds the number of elements in the structure.



(2.2) MODULE NAME : CxxxxTPUB.Pyy & CxxxxTPUB.EXT

=====

DESCRIPTION :

=====

The CxxxxTPUB.Pyy module holds all the public data structure, variables, and tokens, & the CxxxxTPUB.EXT holds all the external for the CxxxxTPUB.pyy of the following :

TREND DATA STORAGE AND

RETRIEVAL STRUCRURES.

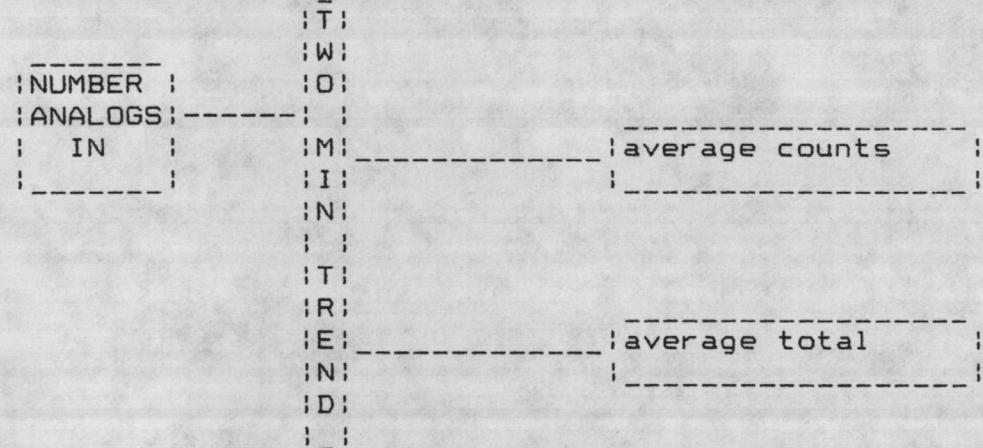
Note that - all the data stucture in this section holding the current trends for analog input only.

```
DECLARE two_min_trend_data (NUM_ANALOGS_IN_PLUS1) STRUCTURE
    (average_counts    WORD,
     average_total     DWORD) PUBLIC INITIAL (0,0);
```

TWO MIN STRUCTURE SIZE = 34 \* 6 = 204 BYTES.

DATA DIAGRAM :

=====

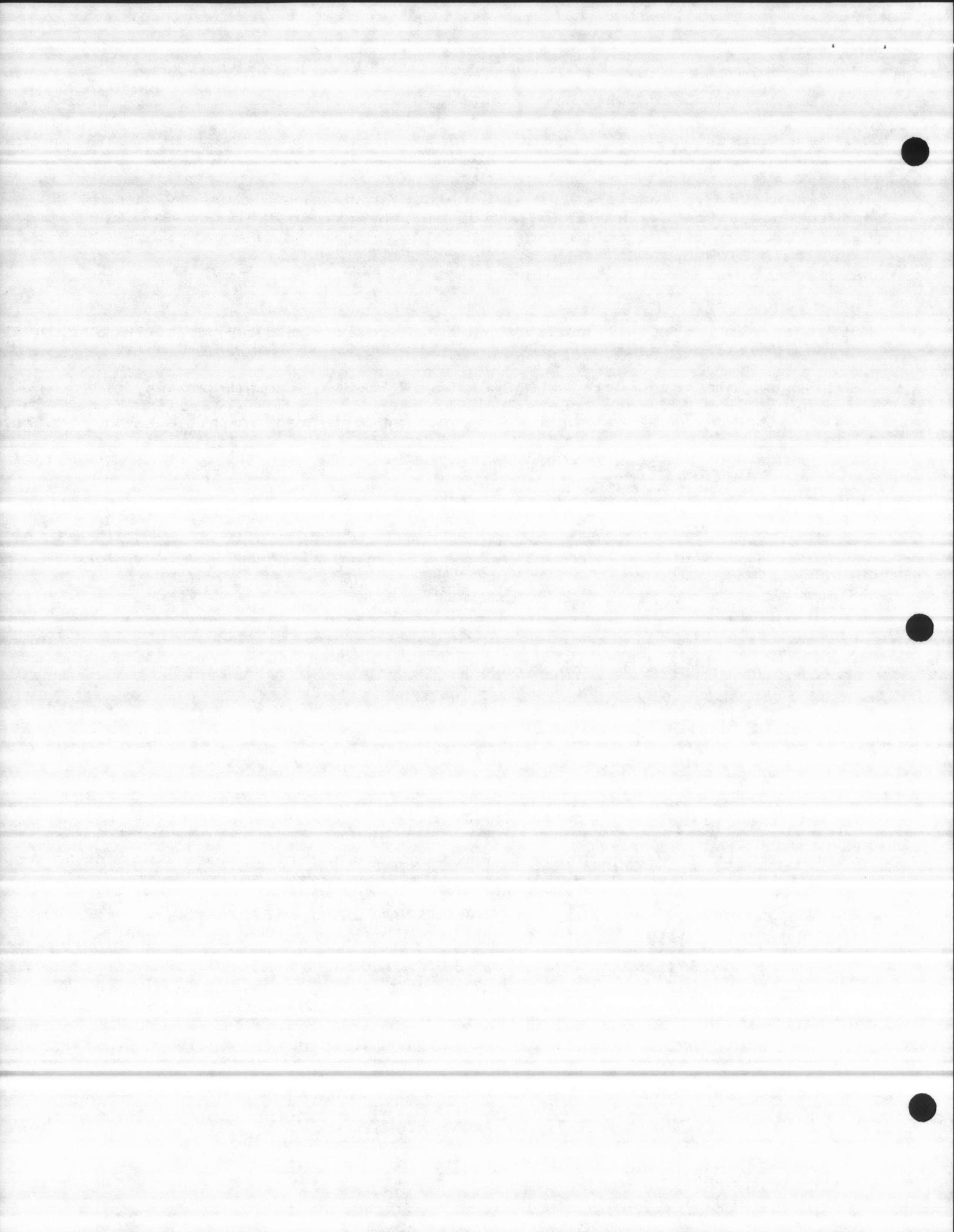


two\_min\_trend\_data STRUCTURE :

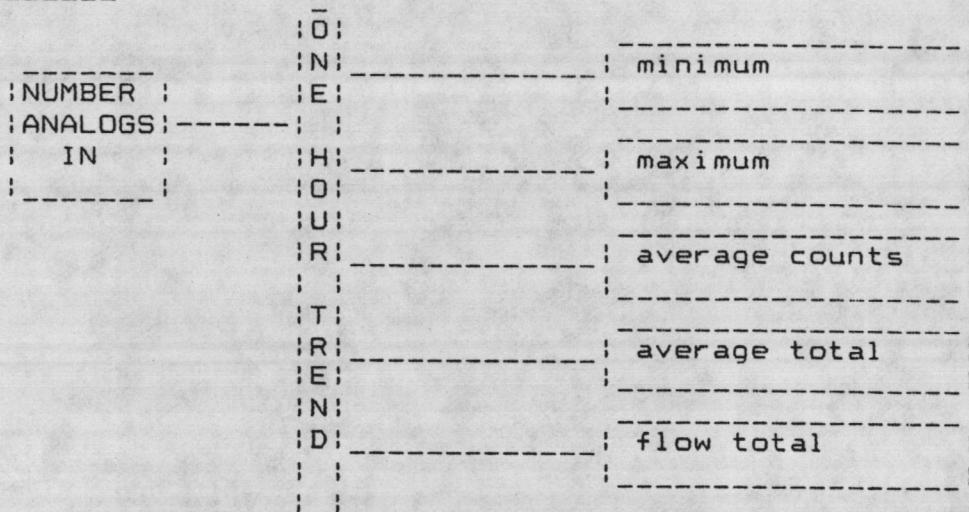
average\_counts => a word holds the average count for each analog input.  
 average\_total => a dword holds the average total for analog input.

```
DECLARE one_hour_trend_data (NUM_ANALOGS_IN_PLUS1) STRUCTURE
    (min           WORD,
     max           WORD,
     average_counts WORD,
     average_total  DWORD,
     flow_total     DWORD) PUBLIC INITIAL (0,0,0,0,0);
```

ONE HOUR TREND STRUCTURE SIZE = 34 \* 14 = 476 BYTES.



DATA DIAGRAM :



one\_hour\_trend\_data STRUCTURE :

```

min  => a word holds the minimum value for each analog input.
max  => a word holds the maximum value for each analog input.
average_counts => a word holds the average_count.
average_total  => a dword holds the calculated average_total.
flow_total    => a dword holds the flow_total for each analog input.
  
```

```

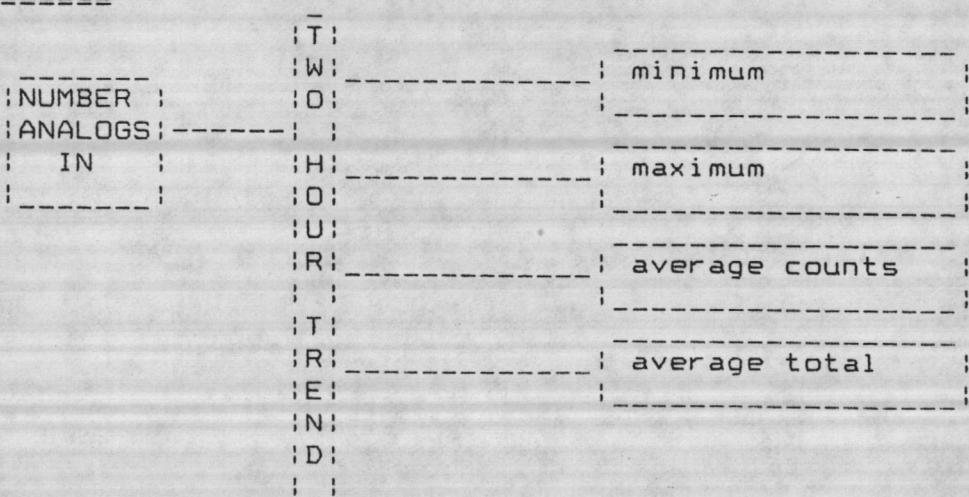
DECLARE two_hour_trend_data (NUM_ANALOGS_IN_PLUS1) STRUCTURE
  (min          WORD,
   max          WORD,
   average_counts WORD,
   average_total  DWORD) PUBLIC INITIAL (0,0,0,0);
  
```

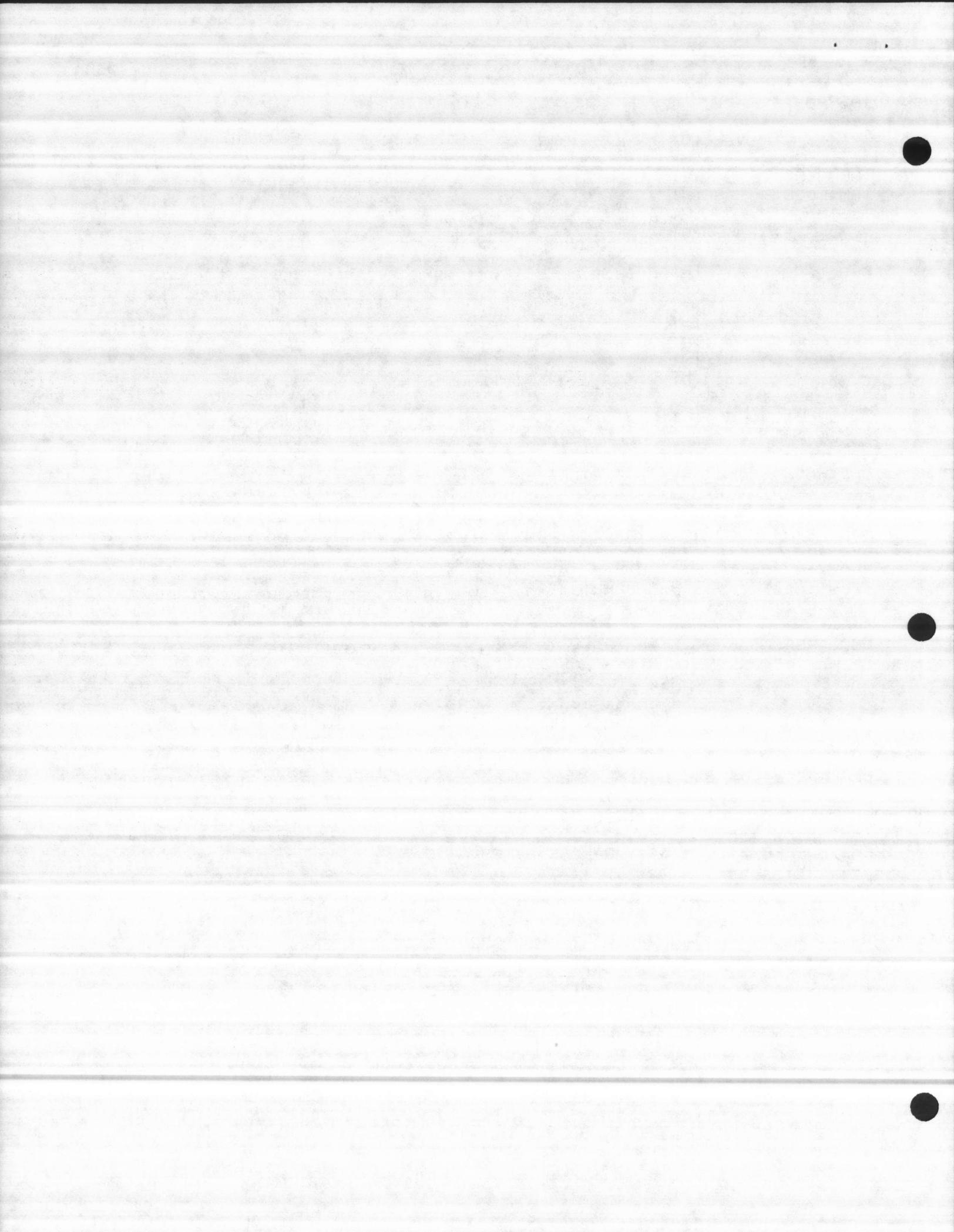
```
DECLARE day_trend_buffer (NUM_ANALOGS_IN_PLUS1) WORD PUBLIC INITIAL (0);
```

TWO HOUR TREND STRUCTURE SIZE = 34 \* 10 = 340 BYTES.

DAY TREND BUFFER SIZE = 34 \* 2 = 68 BYTES.

DATA DIAGRAM :





two\_hour\_trend\_data STRUCTURE :

min => a word holds the minimum value for each analog input.  
 max => a word holds the maximum value for each analog input.  
 average\_counts => a word holds the average\_count.  
 average\_total => a dword holds the calculated average\_total.

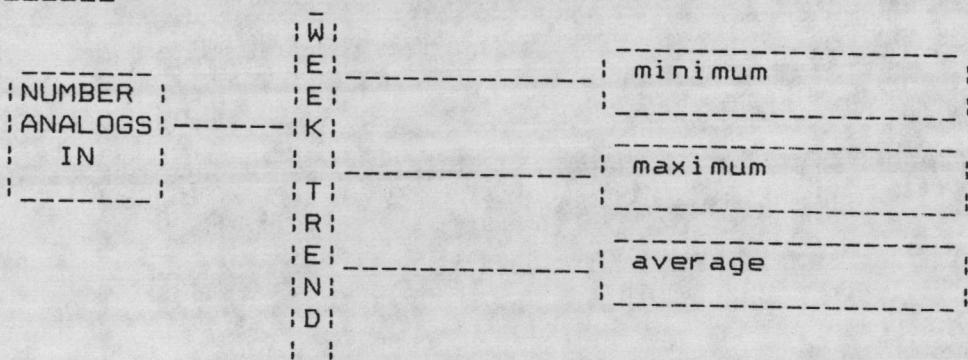
Variables :

day\_trend\_buffer => NUM\_ANALOGS\_IN\_PLUS1 words used to store the  
 last collected trend sample during the day.

DECLARE week\_trend\_buffer (NUM\_ANALOGS\_IN\_PLUS1) STRUCTURE  
 (min WORD,  
 max WORD,  
 average WORD) PUBLIC INITIAL (0FFFFH,0,0);

WEEK TREND BUFFER STRUCTURE SIZE = 34 \* 6 = 204 BYTES.

DATA DIAGRAM :

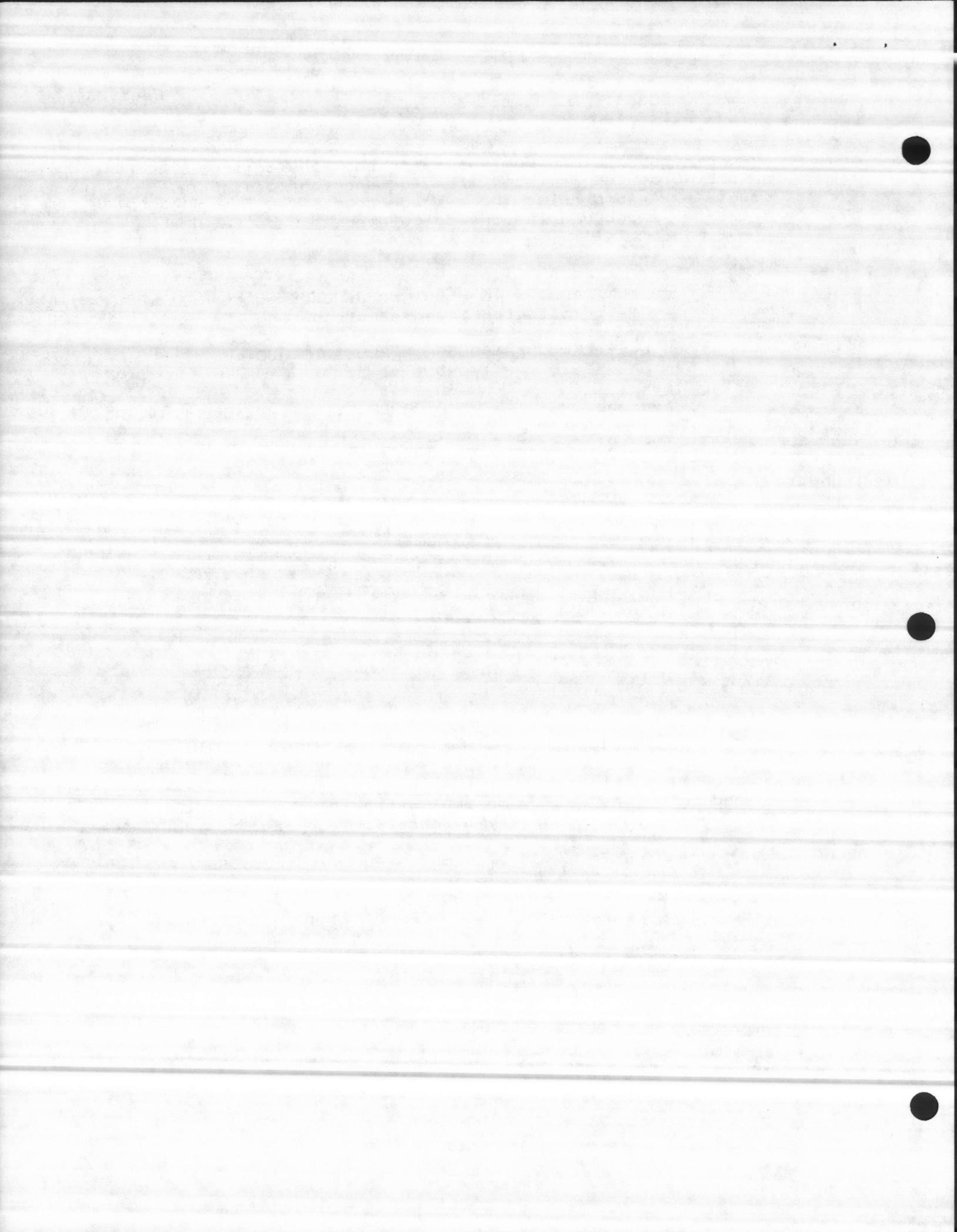


week\_trend\_buffer STRUCTURE :

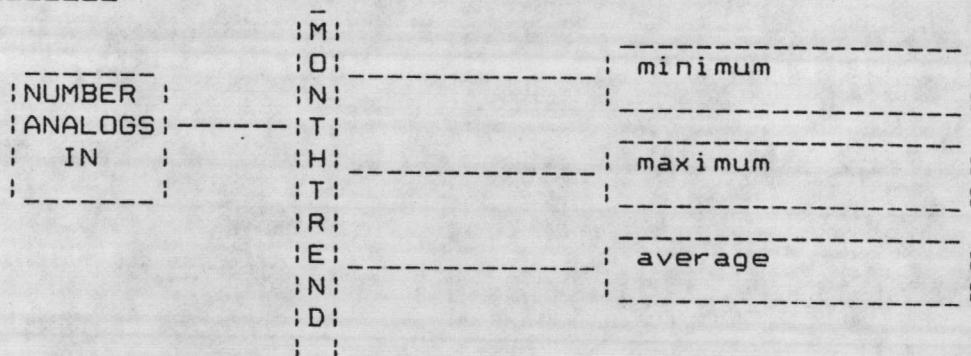
min => a word holds the weekly minimum value for each analog input.  
 max => a word holds the weekly maximum value for each analog input.  
 average => a word holds the weekly average for each analog input.

DECLARE month\_trend\_buffer (NUM\_ANALOGS\_IN\_PLUS1) STRUCTURE  
 (min WORD,  
 max WORD,  
 average WORD) PUBLIC INITIAL (0FFFFH,0,0);

MONTH TREND BUFFER STRUCTURE SIZE = 34 \* 6 = 204 BYTES.



DATA DIAGRAM :



month\_trend\_buffer STRUCTURE :

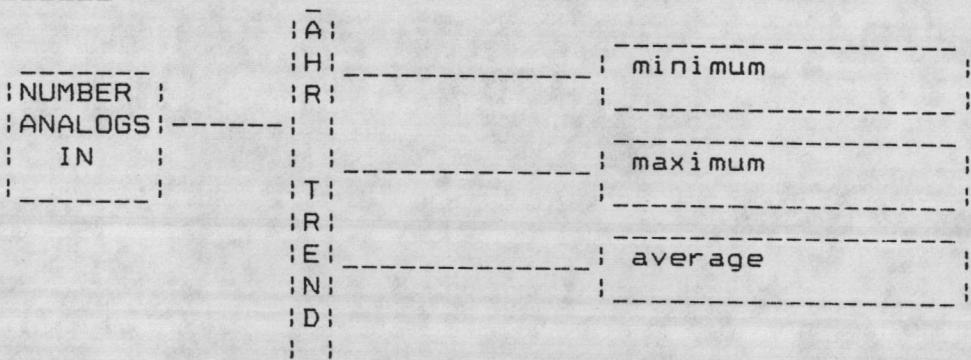
min => a word holds the monthly minimum value for each analog input.  
 max => a word holds the monthly maximum value for each analog input.  
 average => a word holds the monthly average for each analog input.

```

DECLARE ahour_trend_buffer (NUM_ANALOGS_IN_PLUS1) STRUCTURE
  (min           WORD,
   max           WORD,
   average       WORD) PUBLIC INITIAL (0FFFFH,0,0);
  
```

AHOUR TREND BUFFER SIZE = 34 \* 6 = 204 BYTES.

DATA DIAGRAM :



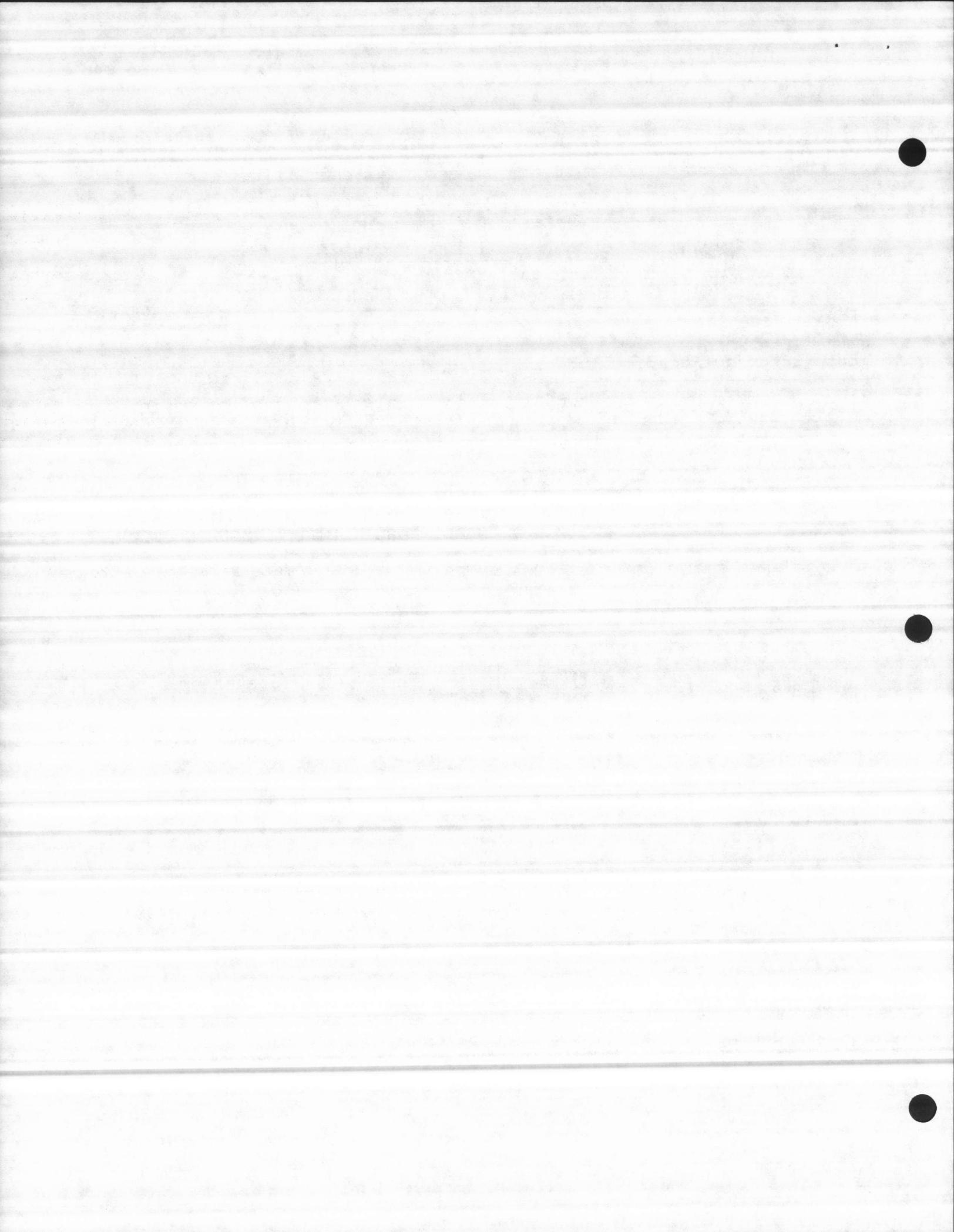
ahour\_trend\_buffer STRUCTURE :

min => a word holds the minimum value for each analog input.  
 max => a word holds the maximum value for each analog input.  
 average => a word holds the average for each analog input.

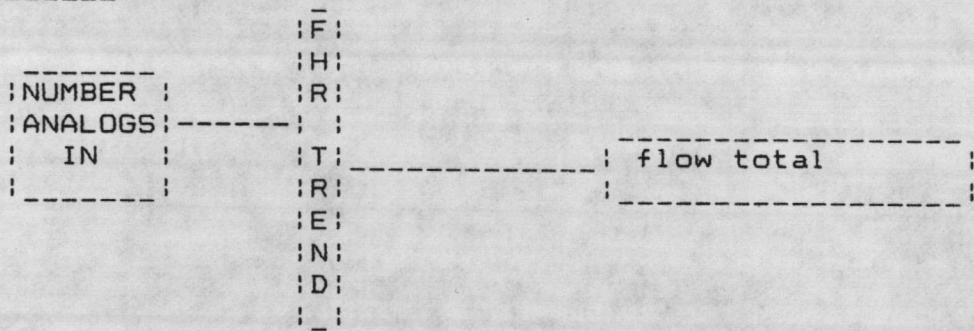
```

DECLARE fhour_trend_buffer (NUM_ANALOGS_IN_PLUS1) STRUCTURE
  (flow_total      WORD) PUBLIC INITIAL (0);
  
```

FHOUR TREND BUFFER SIZE = 34 \* 2 = 68 BYTES.



DATA DIAGRAM :



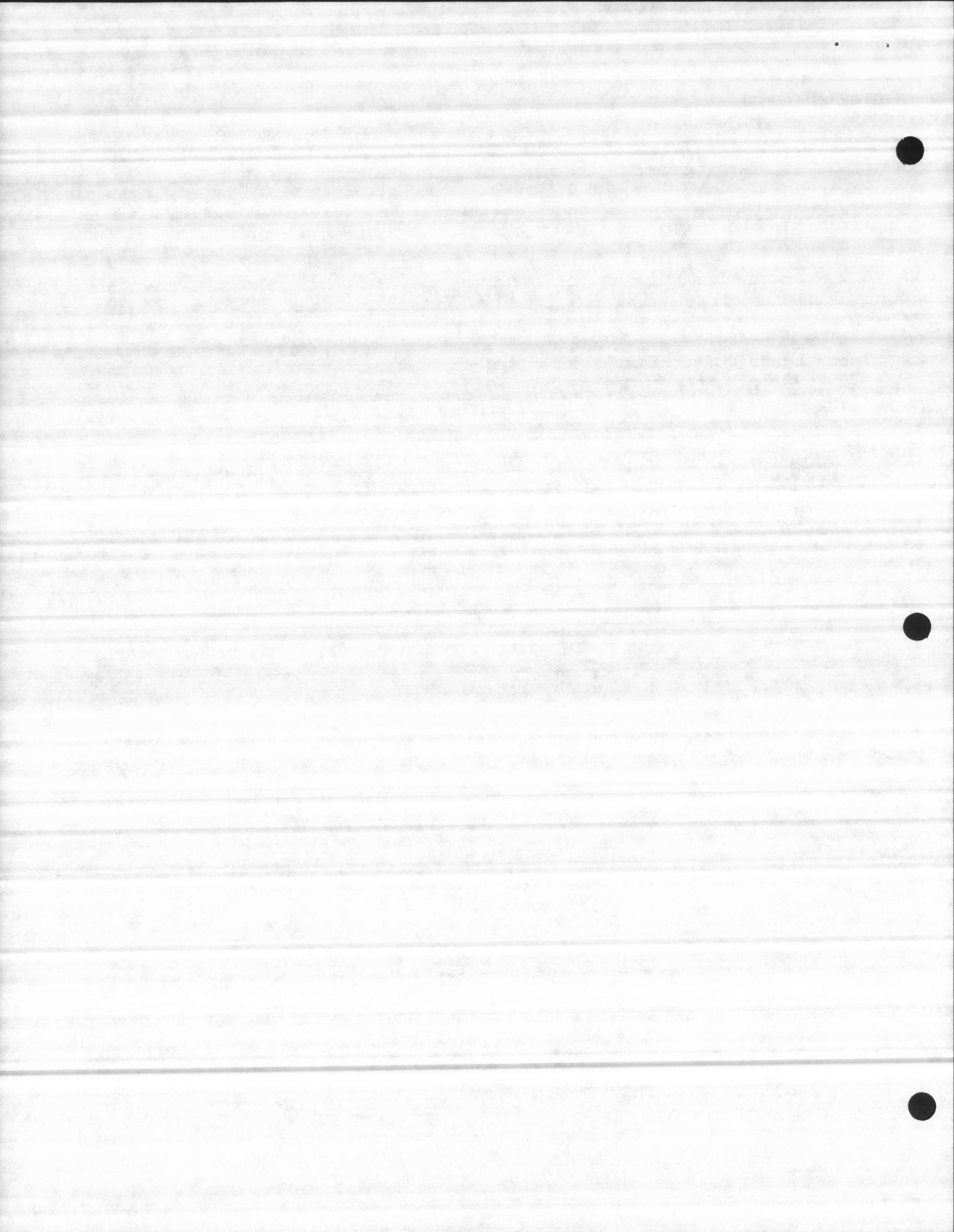
fhour\_trend\_buffer STRUCTURE :

flow\_total => a word holds the period flow total.

DECLARE trend\_data (722) INTEGER PUBLIC;

Variables :

trend\_data => 722 integers



(2.3) MODULE NAME : CxxxxGEXR.Pyy & CxxxxGEXR.EXT

=====

DESCRIPTION :

=====

The CxxxxGEXR.Pyy module holds all the public data structure, variables, and tokens, & the CxxxxGEXR.EXT holds all the external for the CxxxxGEXR.Pyy. BOTH MODULE HOLD THE CURRENT YEARLY DATA, CONTROL, LEVEL CONTROL AND STEP PUMPS.

=====

= YEARLY ANALOGS =

=====

```
DECLARE yearly_scale (NUM_ANALOGS_IN_PLUS1) WORD PUBLIC;
```

```
DECLARE yearly_analog (NUM_ANALOGS_IN_PLUS1) STRUCTURE
```

```
    (point_asc (10)      BYTE,  
     desc_asc   (24)      BYTE,  
     source_id_type    BYTE,  
     source_id_index   WORD,
```

```
     monthly_avg   (13) WORD,
```

```
     monthly_tot   (13) DWORD,  
     minimum_tot   DWORD,  
     maximum_tot   DWORD,
```

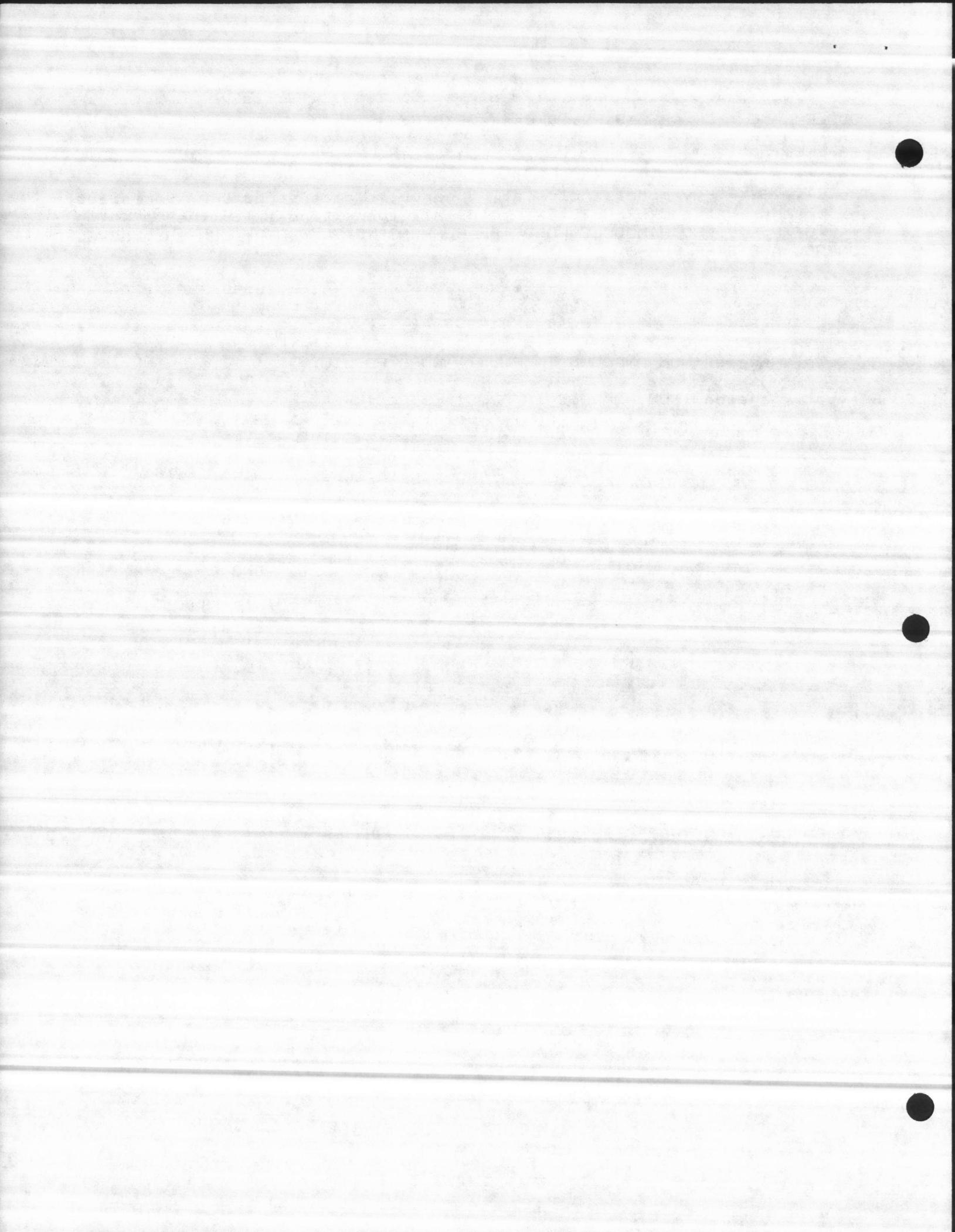
```
     edit_change_flag BYTE)
```

```
PUBLIC INITIAL (
```

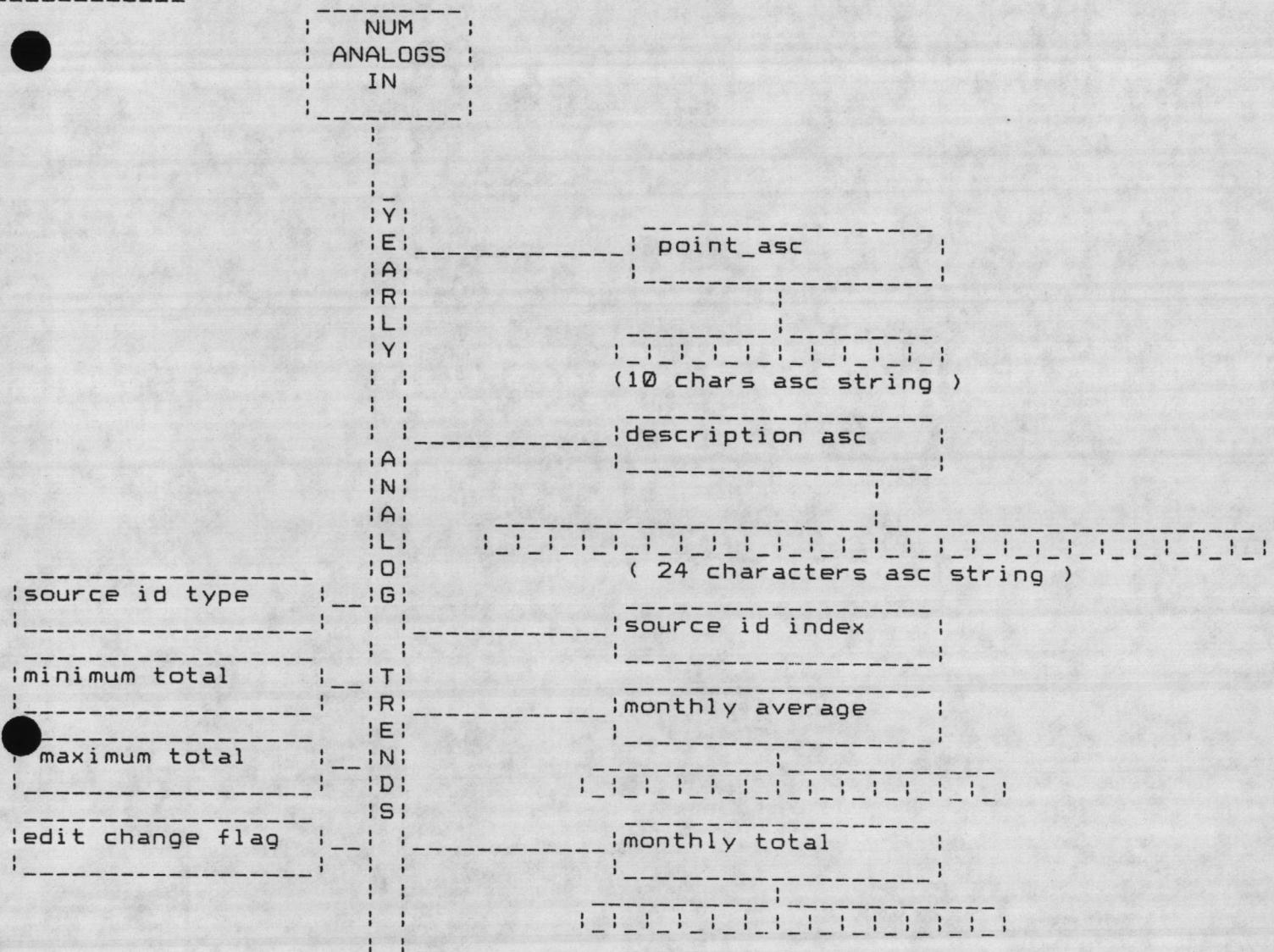
```
    'Spare      ',  
    'Yearly      data buffer  ',  
    0FFH,0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
    0FFFFFFFFFFH, 0,  
    0);
```

YEARLY ANALOG STRUCTURE SIZE = 34 \* 124 = 4216 BYTES.

-----



## DATA DIAGRAM :



yearly\_scale => 30 words for yearly scale.

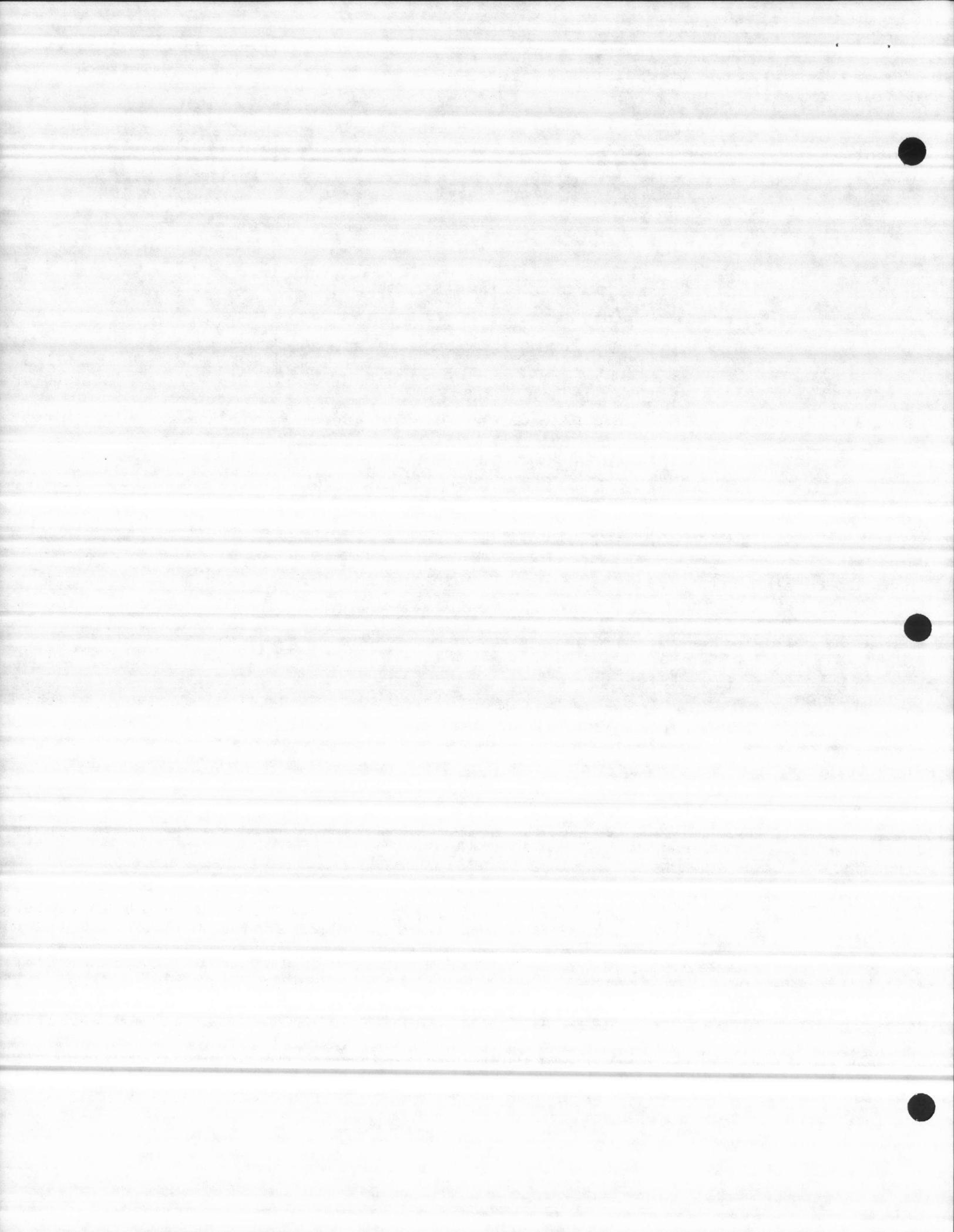
yearly\_analog STRUCTURE :

point\_asc => 10 bytes ascii point id number. used to access specific analog trend or structure elements. Allows the operator to index analog trend point by entering analog name. The point\_asc is also used by the keyboard SELECT command to select any analog trend point in the system.

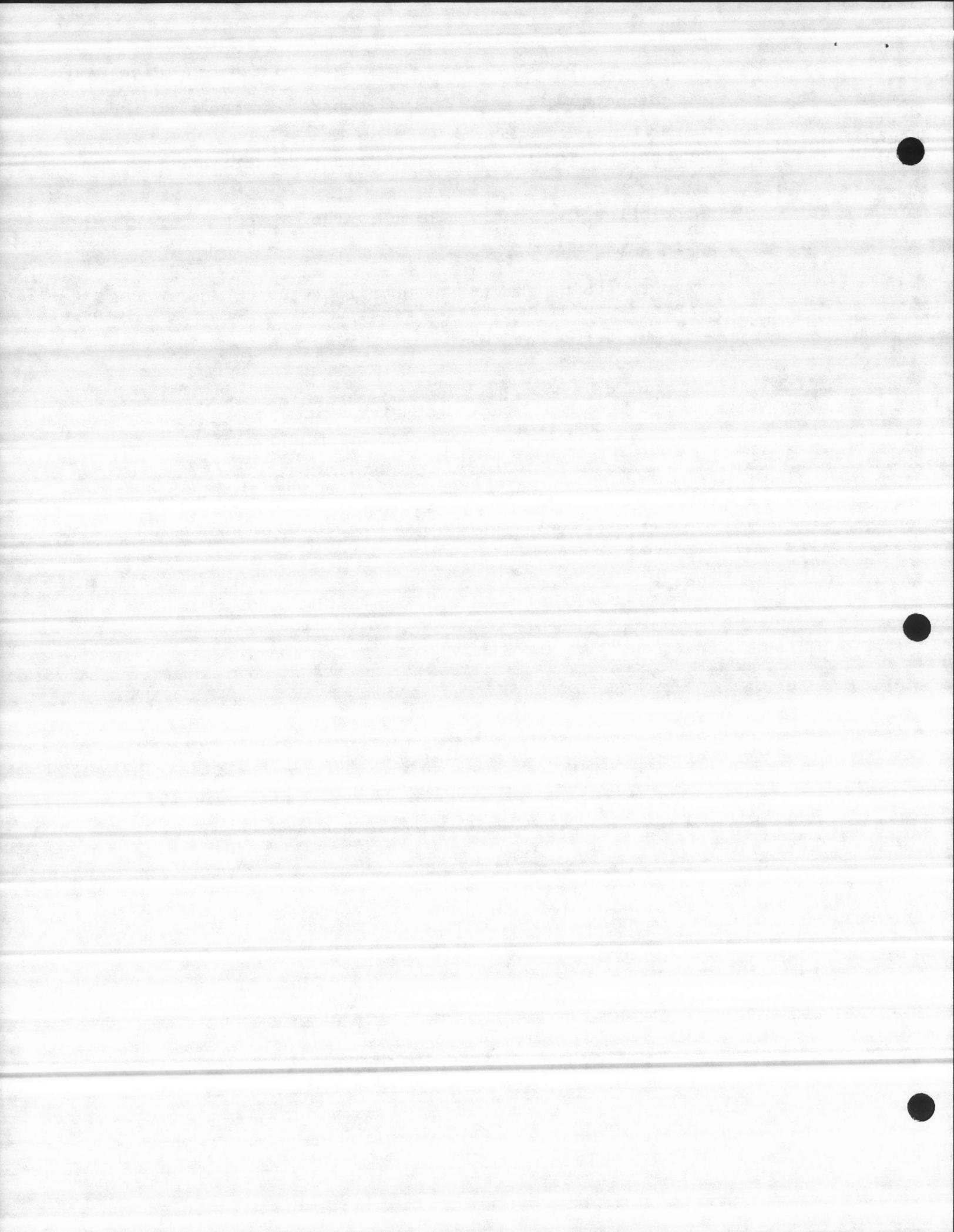
desc\_asc => 24 bytes ascii point description.

source\_id\_type => a byte value contains the index to the structure type.

source\_id\_index => a word value contains the index to the source.

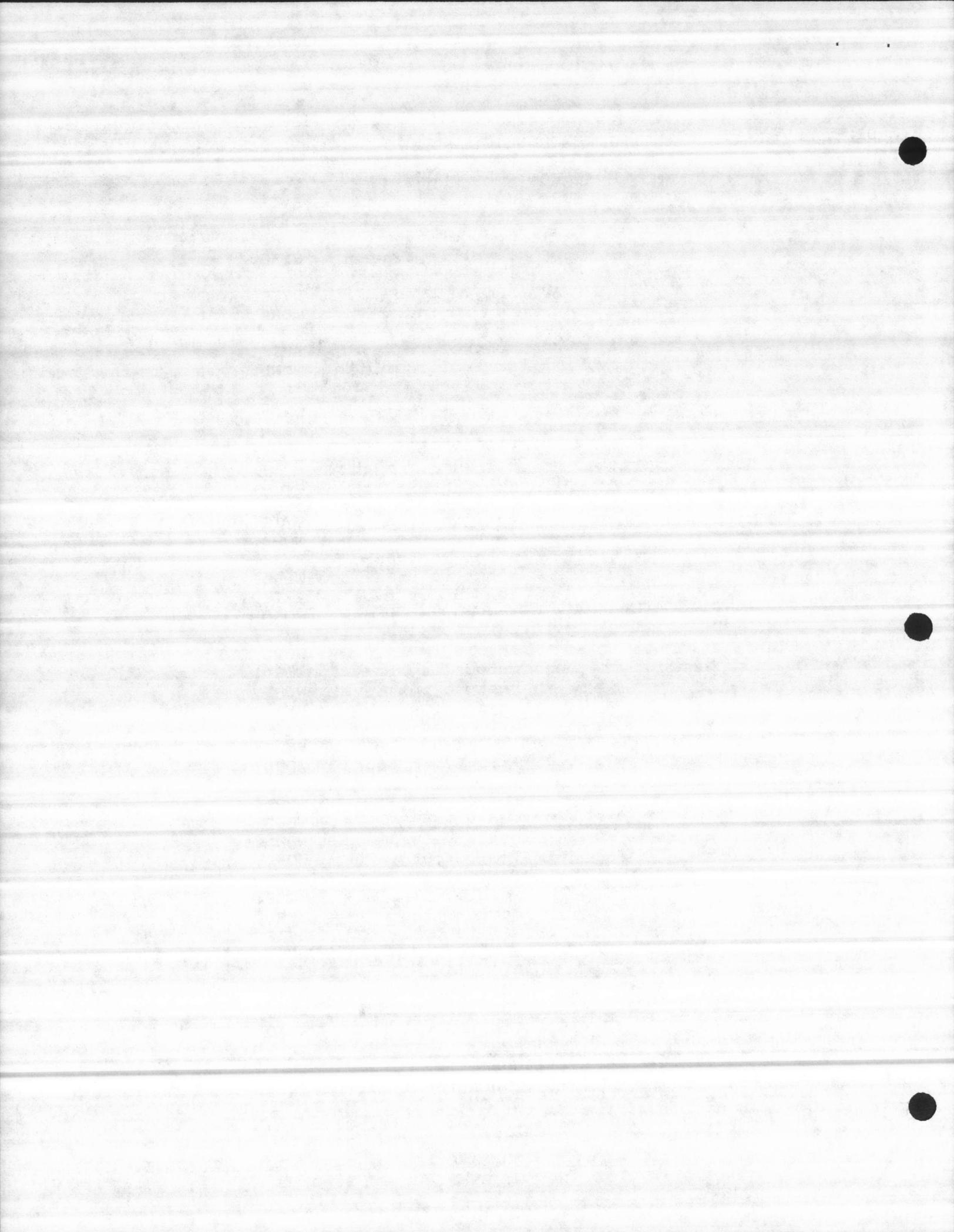


-----  
monthly\_avg => 13 words holds all the 12 plus1 monthly averages during the year.  
monthly\_tot => 13 dwords holds all the 12 plus1 monthly totals during the year.  
minimum\_tot => a dword holds the yearly minimum value.  
maximum\_tot => a dword holds the yearly maximum value.  
edit\_change\_flag => a byte, to be set if any numeric value change in this structure, otherwise is reset.



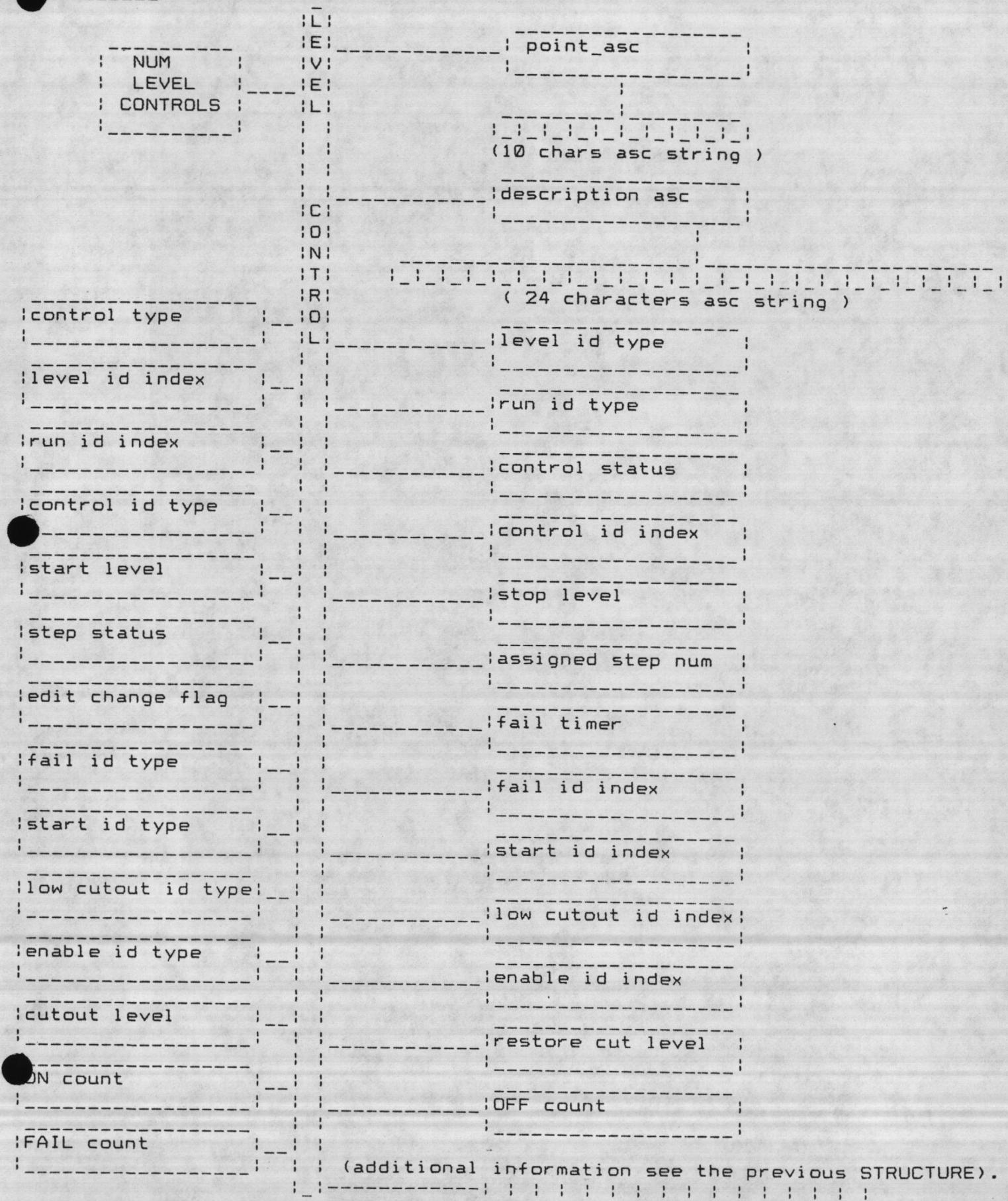
```
=          LEVEL CONTROL
=====
DECLARE level_control_reg_t TOKEN PUBLIC;

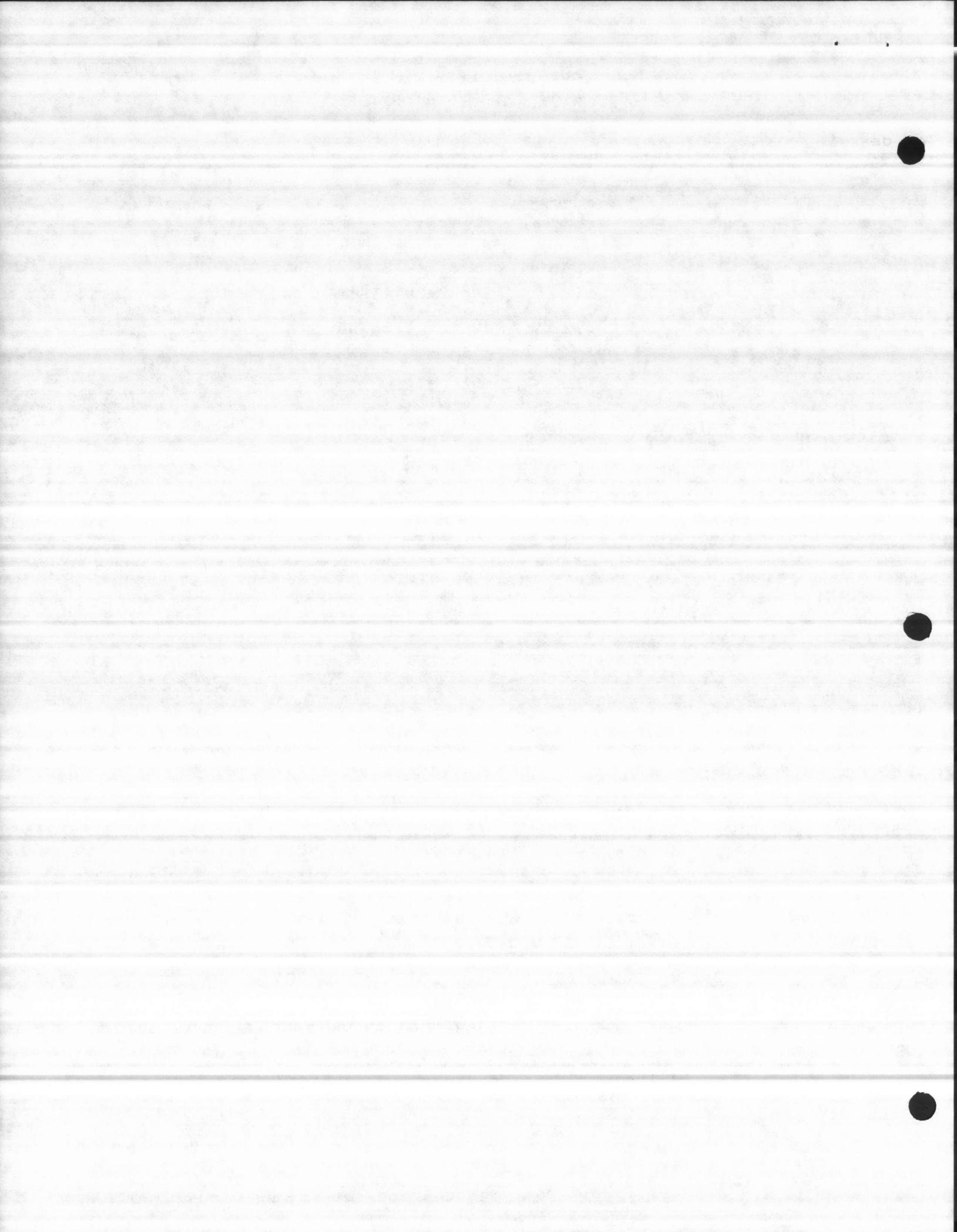
DECLARE Level_Control (NUM_LEVEL_CONTROLS_PLUS1)
    STRUCTURE (point_asc      (10)      BYTE,
                desc_asc       (24)      BYTE,
                control_type   WORD,
                level_id_type  BYTE,
                level_id_index WORD,
                run_id_type    BYTE,
                run_id_index   WORD,
                control_status WORD,
                control_id_type BYTE,
                control_id_index WORD,
                start_level     WORD,
                stop_level      WORD,
                step_status     WORD,
                assigned_step_num BYTE,
                edit_change_flag BYTE,
                fail_timer      WORD,
                fail_id_type    BYTE,
                fail_id_index   WORD,
                start_id_type   BYTE,
                start_id_index  WORD,
                low_cutout_type BYTE,
                low_cutout_index WORD,
                enable_id_type  BYTE,
                enable_id_index WORD,
                cutout_level    WORD,
                restore_cut_level WORD,
                on_count        BYTE,
                off_count       BYTE,
                fail_count      BYTE,
                fail_timer_count WORD,
                level_value     WORD,
                run_value       WORD,
                start_value     WORD,
                low_cutout_value WORD,
                fail_value      WORD,
                enable_value    WORD,
                option_flags    WORD,
                alt_id_type     BYTE,
                alt_id_index    WORD,
                low_low_id_type BYTE,
                low_low_id_index WORD,
                low_low_value   WORD)
PUBLIC INITIAL
('spare lev',
'Spare level control set',
0, 0FFH, 0, 0FFH, 0, 0, 0FFH, 0,
0, 0, 0, 0, 0, 0, 0,
0FFH, 0, 0FFH, 0, 0FFH, 0, 0FFH, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0FFH, 0, 0FFH, 0, 0);
```



LEVEL CONTROL STRUCTURE SIZE = 61 \* 101 = 6161 BYTES.

DATA DIAGRAM :





---

level\_control STRUCTURE :

The description and the use of each element entry in the level\_control structure.

point\_asc => 10 bytes ascii point id number. used to access specific level control point or struc elements. Allows the operator to index level control by entering lab name. The point\_asc is also used by the keyboard SELECT command to select any level control point in the system.

desc\_asc => 24 bytes ascii point description.

control type => a word value that indicates the scan status

0 - Deactivated.

1 - Activated.

level\_id\_type => a byte value contains the index to the structure type.

level\_id\_index => a word value contains the index to the level source.

run\_id\_type => a byte value contains the index to the structure type.

run\_id\_index => a word value contains the index to the pump run source.

control status => a word holds the status of the control  
(editable field) manual / auto.

control\_id\_type => (unused).

control\_id\_index => (unused).

start\_level => (unused).

stop\_level => (unused).

stop\_status => (unused).

assigned\_step\_num => (unused)

edit\_change\_flag => a byte, to be set if any numeric value change in this structure, otherwise is reset.

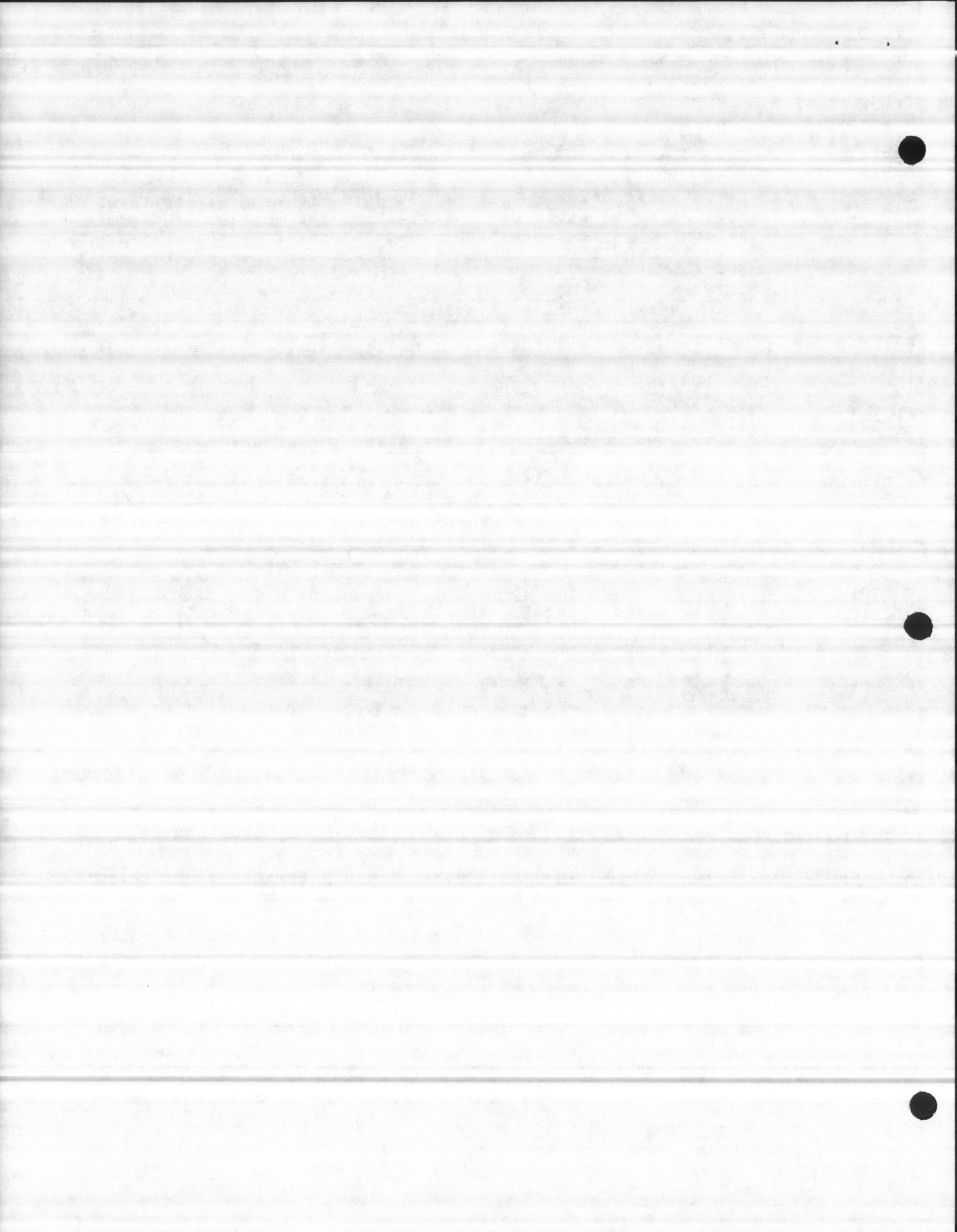
fail\_timer => a word value holds the fail timer value.

fail\_id\_type => a byte value contains the index to the structure type.

fail\_id\_index => a word value contains the index to the pump fail source.

start\_id\_type => a byte value contains the index to the structure type.

start\_id\_index => a word value contains the index to the pump start source.



low\_cutout\_type => a byte value contains the index to the structure type.  
 low\_cutout\_index => a word value contains the index to the low cutout source.  
 enable\_id\_type => a byte value contains the index to the structure type.  
 enable\_id\_index => a word value contains the index to the enable pump source.  
 cutout\_level => a word value contains the cutout setpoint.  
 restore\_cut\_level => a word value contains the restore cutout setpoint.  
 on\_count => a byte value contains the manual require status.  
 off\_count => a byte value contains the auto require status.  
 fail\_count => a byte value contains the emergency power setpoint.  
 fail\_timer\_count => a word value holds a counter for the fail\_timer.  
 level\_value => (unused).  
 run\_value => a word value holds the current value of the discrete run signal.  
 start\_value => a word value holds the current value of the start signal.  
 low\_cutout\_value => a word holds the current value of the low cutout signal.  
 fail\_value => a word holds the current value of the discrete fail signal.  
 enable\_value => a word value holds the current value of the enable pump.  
 option\_flags => a word value holds the pump HOA status.  
 alt\_id\_type => a byte value contains the index to the structure type.  
 alt\_id\_index => a word value contains the index to pump HOA source.  
 low\_low\_id\_type => (unused).  
 low\_low\_id\_index=> (unused).  
 low\_low\_value => (unused).

---

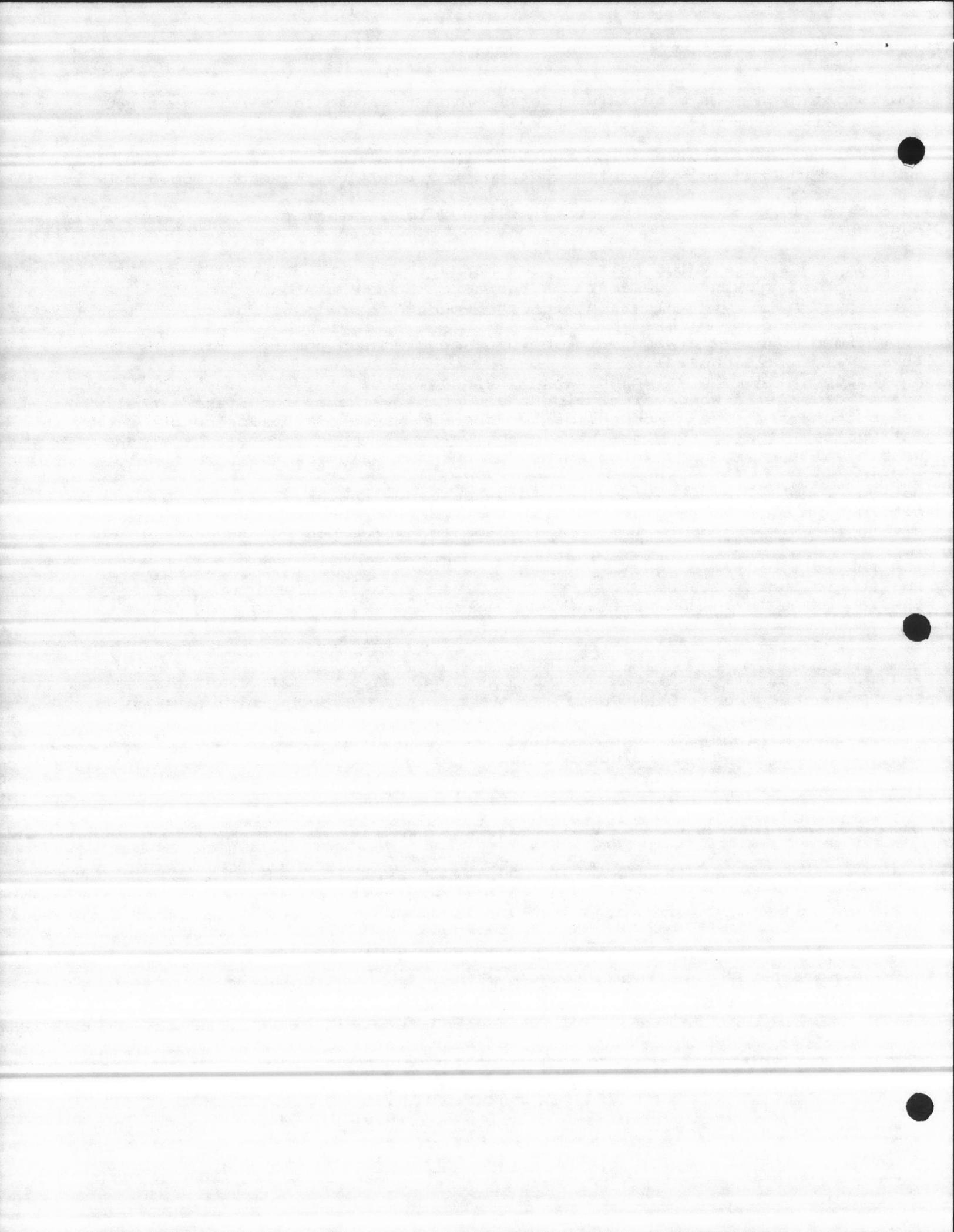
 = CONTROLS =
 

---

```
DECLARE control_reg_t TOKEN PUBLIC;
```

```
DECLARE Control (NUM_CONTROLS_PLUS1)
```

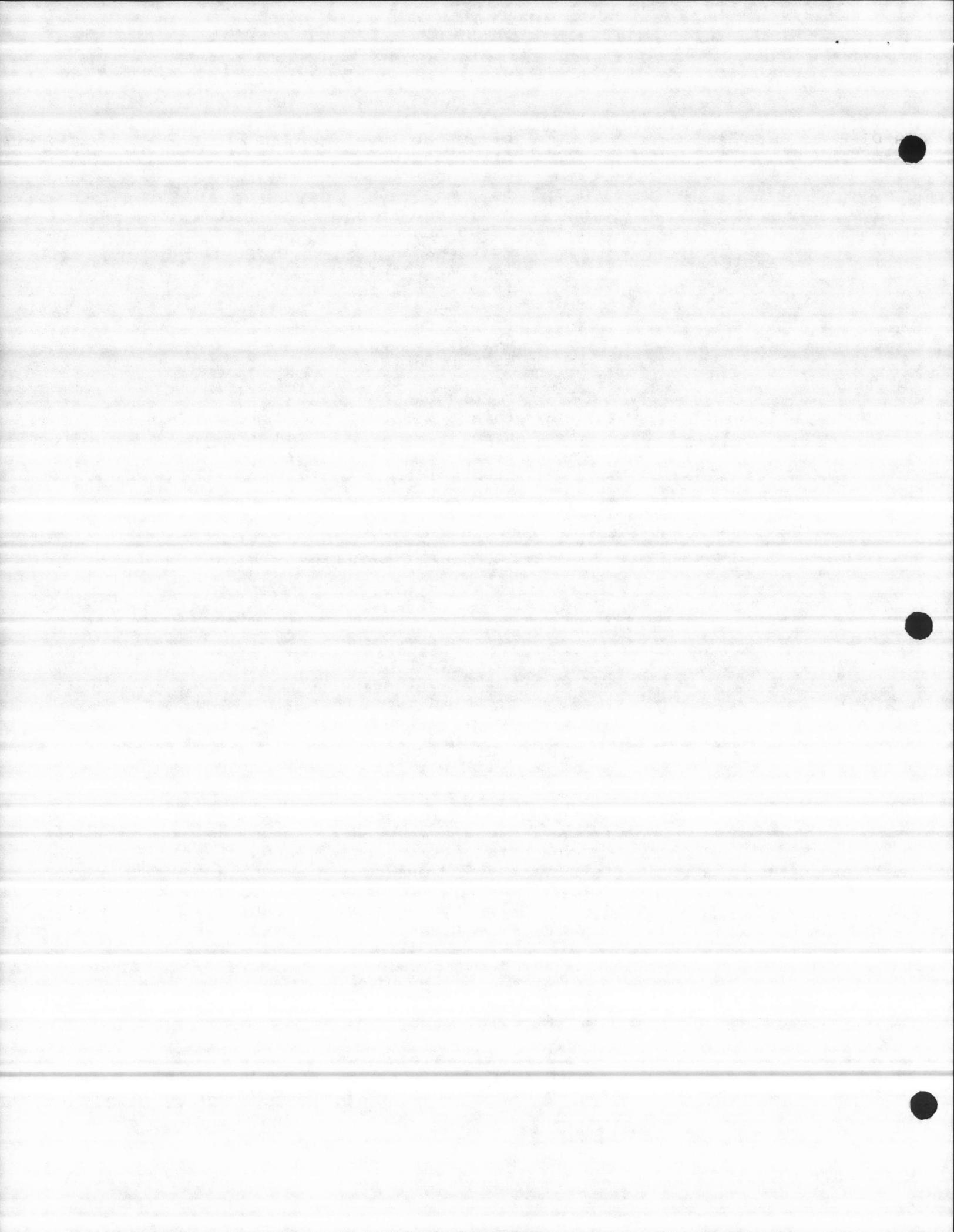
STRUCTURE	(point_asc (10) BYTE,
	desc_asc (24) BYTE,
	level1_id_type BYTE,
	level1_id_index WORD,
	max_num_steps BYTE,
	step_num1 BYTE,
	on_step1_level WORD,
	off_step1_level WORD,
	step1_status WORD,
	step_num2 BYTE,
	on_step2_level WORD,
	off_step2_level WORD,
	step2_status WORD,
	step_num3 BYTE,
	on_step3_level WORD,
	off_step3_level WORD,
	step3_status WORD,
	step_num4 BYTE,
	on_step4_level WORD,
	off_step4_level WORD,
	step4_status WORD,



```
step_num5           BYTE,
on_step5_level     WORD,
off_step5_level    WORD,
step5_status       WORD,
step_num6           BYTE,
on_step6_level     WORD,
off_step6_level    WORD,
step6_status       WORD,
edit_change_flag   BYTE,
level2_id_type    BYTE,
level2_id_index   WORD,
level3_id_type    BYTE,
level3_id_index   WORD,
select_id_type    BYTE,
select_id_index   WORD,
level1_value      WORD,
level2_value      WORD,
level3_value      WORD,
step1_timer        WORD,
step2_timer        WORD,
step3_timer        WORD,
step4_timer        WORD,
ctrl_options_flag WORD,
spare1             BYTE,
step5_timer        WORD,

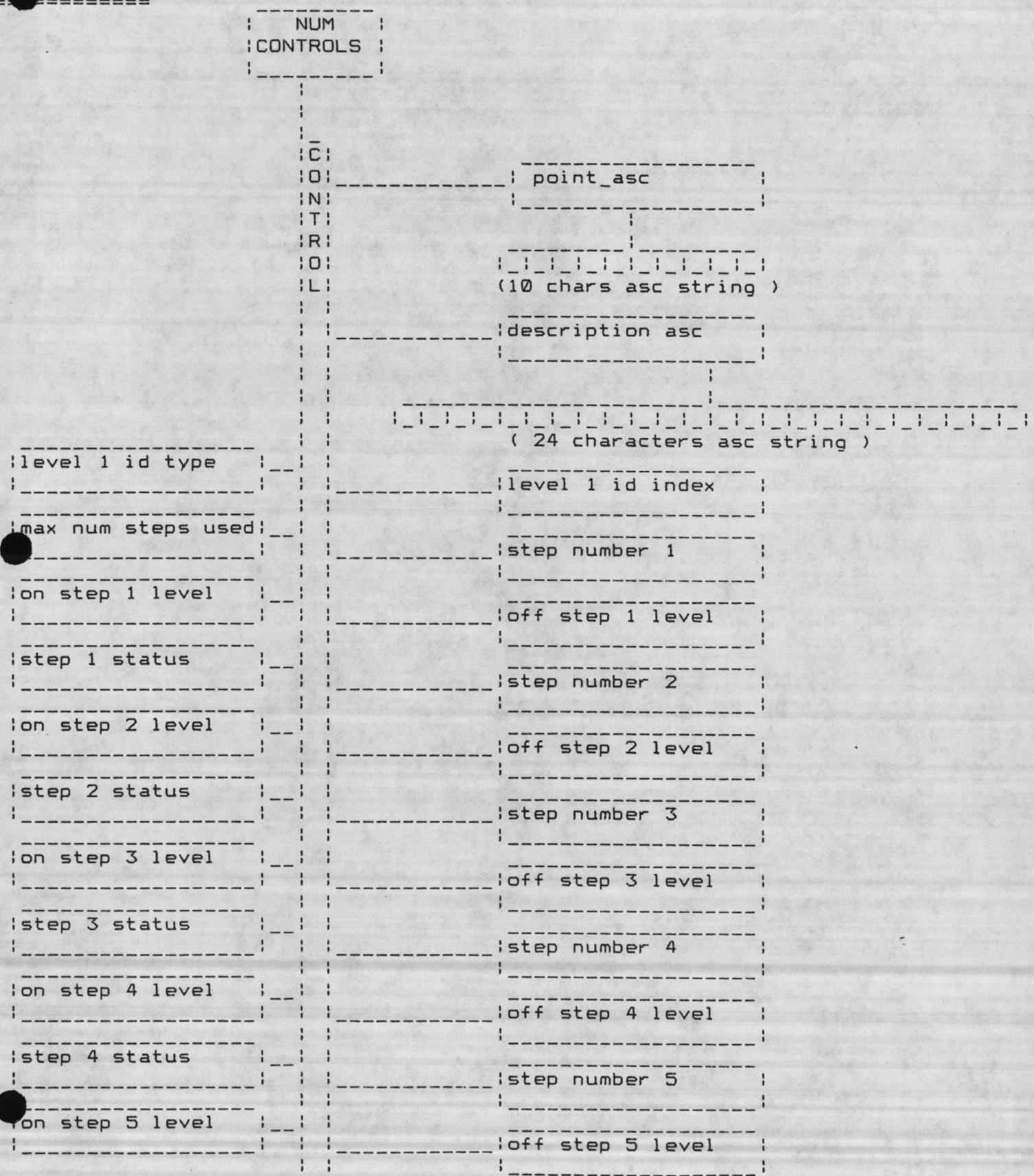
first_id_type      BYTE,
first_id_index     WORD,
second_id_type     BYTE,
second_id_index    WORD,
third_id_type      BYTE,
third_id_index     WORD,
forth_id_type      BYTE,
forth_id_index     WORD,

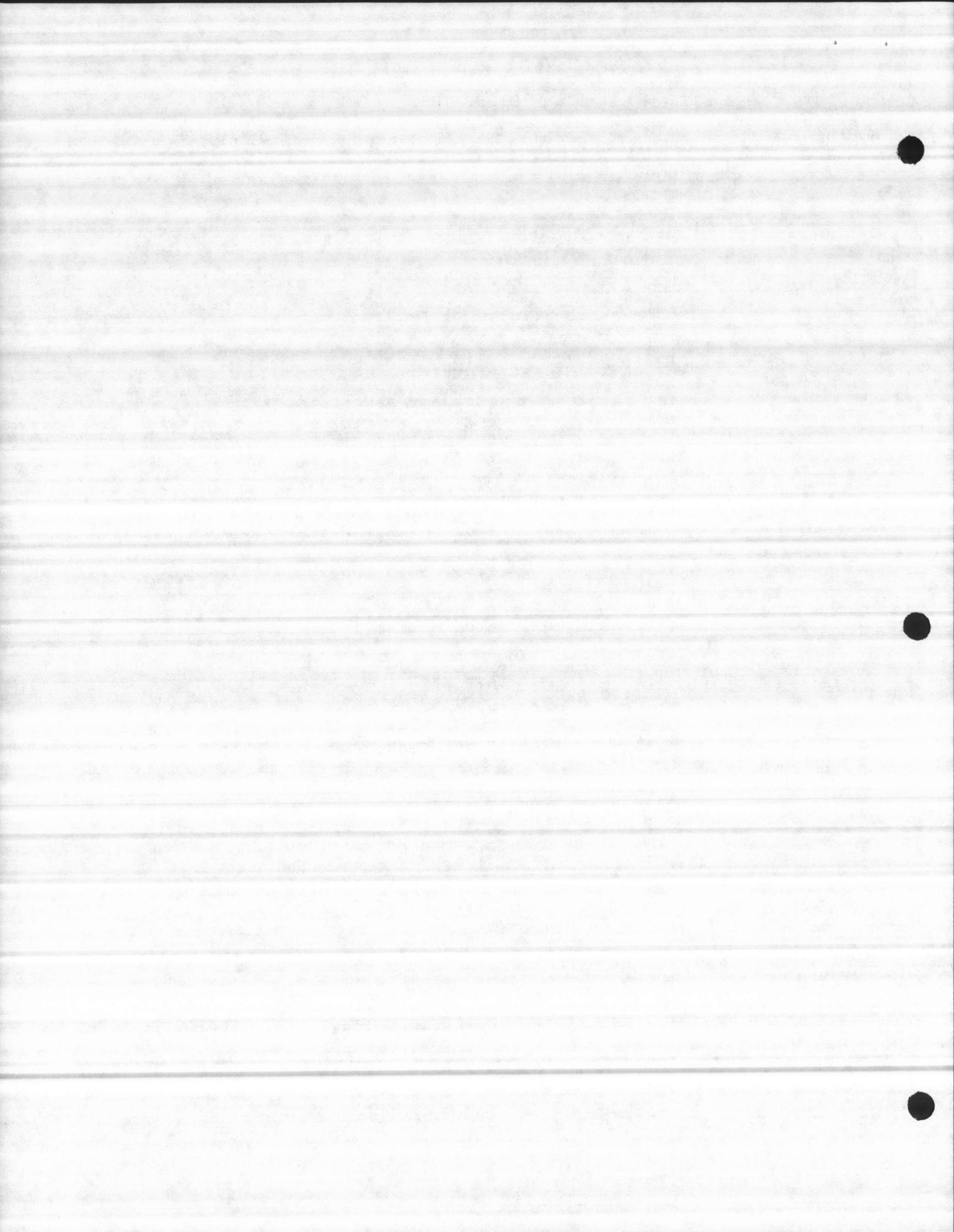
num_ctrl_pts       BYTE,
step6_timer        WORD,
step1_timer_count WORD,
step2_timer_count WORD,
step3_timer_count WORD,
step4_timer_count WORD,
step5_timer_count WORD,
step6_timer_count WORD,
on_off_scale      WORD)
PUBLIC INITIAL
('spare ctrl',
'Spare control elements ', 0FFH, 0, 0,
1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0,
4, 0, 0, 0, 5, 0, 0, 0, 6, 0, 0, 0,
0, 0FFH, 0, 0FFH, 0, 0FFH, 0,
0, 0, 0, 0, 0, 0,
0, 0FFH, 0, 0FFH, 0, 0FFH, 0,
0FFH, 0, 0FFH, 0, 0, 0, 0, 0, 0, 0, 0);
```



CONTROL STRUCTURE SIZE = 6 \* 136 = 816 BYTES.

DATA DIAGRAM :





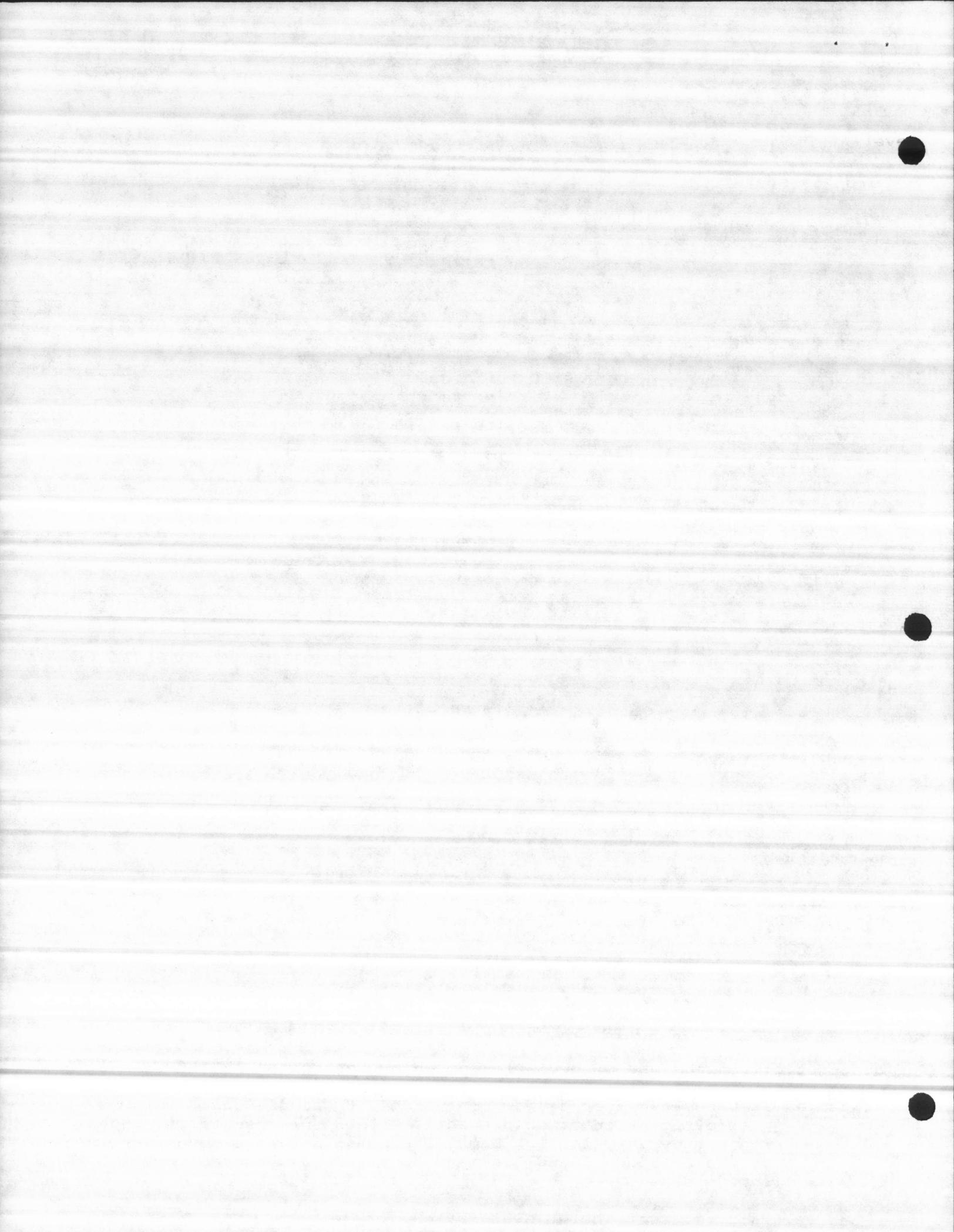
step 5 status	step number 6
on step 6 level	off step 6 level
step 6 status	edit change flag
level 2 id type	level 2 id index
level 3 id type	level 3 id index
select id type	select id index
level 1 value	level 2 value
level 3 value	step1 timer
step 2 time	step 3 timer
step 4 timer	control option flag
spare1	step 5 timer
first id type	first id index
second id type	second id index
third id type	third id index
forth id type	forth id index
	(additional info. see previous structure)

## control STRUCTURE :

The description and the use of each element entry in the control structure.

point\_asc => 10 bytes ascii point id number. used to access specific control point or struc elements. Allows the operator to index control by entering lab name. The point\_asc is also used by the keyboard SELECT command to select any control point in the system.

desc\_asc => 24 bytes ascii point description.



level1\_id\_type => a byte value contains the index to the structure type.

level1\_id\_index => a word value contains the index to the level source.

max\_num\_steps => a byte value contains the maximum number of steps used in the control.

step\_num1 => a byte value contains the index to step number 1.  
on\_step1\_level => a word value contains the step No. 1 on setpoint.  
off\_step1\_level => a word value contains the step No. 1 off setpoint.  
step1\_status => a word value contains the status of step 1 (OFF/ON).

step\_num2 => a byte value contains the index to step number 2.  
on\_step2\_level => a word value contains the step No. 2 on setpoint.  
off\_step2\_level => a word value contains the step No. 2 off setpoint.  
step2\_status => a word value contains the status of step 2 (OFF/ON).

step\_num3 => a byte value contains the index to step number 3.  
on\_step3\_level => a word value contains the step No. 3 on setpoint.  
off\_step3\_level => a word value contains the step No. 3 off setpoint.  
step3\_status => a word value contains the status of step 3 (OFF/ON).

step\_num4 => a byte value contains the index to step number 4.  
on\_step4\_level => a word value contains the step No. 4 on setpoint.  
off\_step4\_level => a word value contains the step No. 4 off setpoint.  
step4\_status => a word value contains the status of step 4 (OFF/ON).

step\_num5 => a byte value contains the index to step number 5.  
on\_step5\_level => a word value contains the step No. 5 on setpoint.  
off\_step5\_level => a word value contains the step No. 5 off setpoint.  
step5\_status => a word value contains the status of step 5 (OFF/ON).

step\_num6 => a byte value contains the index to step number 6.  
on\_step6\_level => a word value contains the step No. 6 on setpoint.  
off\_step6\_level => a word value contains the step No. 6 off setpoint.  
step6\_status => a word value contains the status of step 6 (OFF/ON).

edit\_change\_flag => a byte, to be set if any numeric value change in this structure, otherwise is reset.

level2\_id\_type => a byte value contains the index to the structure type.

level2\_id\_index => a word value contains the index to the level2 source.

level3\_id\_type => a byte value contains the index to the structure type.

level3\_id\_index => a word value contains the index to the level3 source.

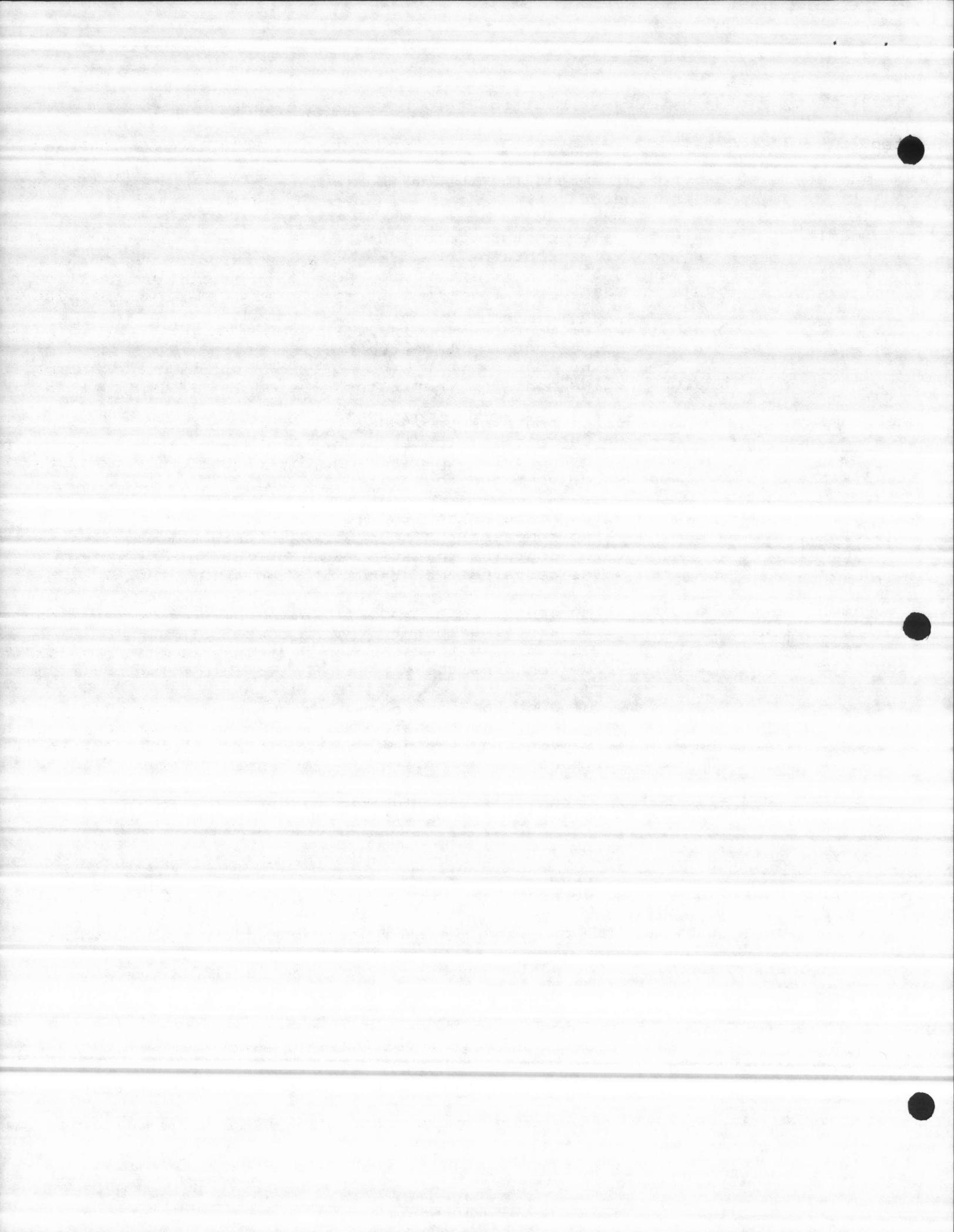
select\_id\_type => a byte value contains the index to the structure type.

select\_id\_index => a word value contains the index to the select source.

level1\_value => a word value contains the current value of level1 source.

level2\_value => a word value contains the current value of level2 source.

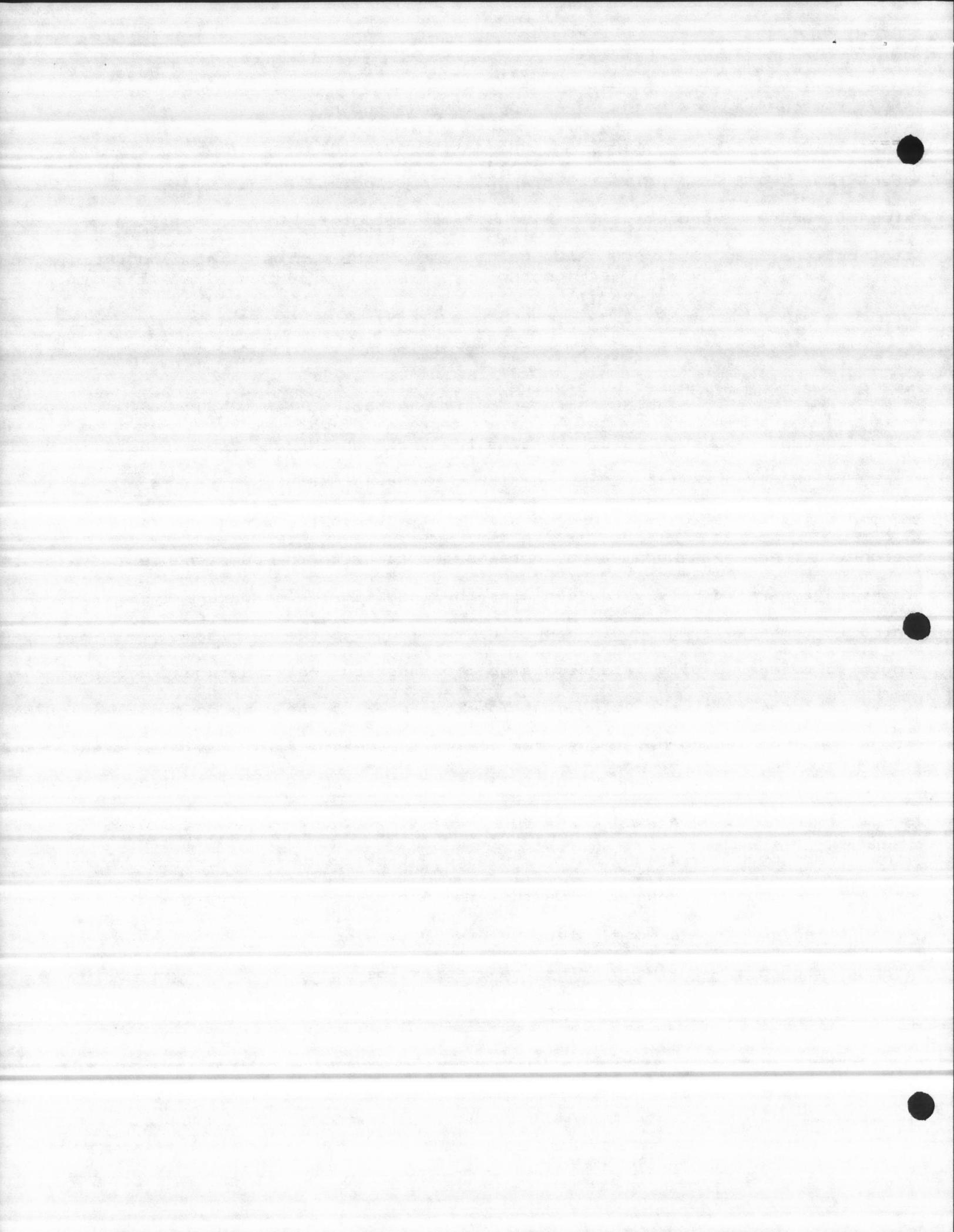
level3\_value => a word value contains the current value of level3 source.



-----  
step1\_timer => a word holds the step 1 timer setpoint.  
step2\_timer => a word holds the step 2 timer setpoint.  
step3\_timer => a word holds the step 3 timer setpoint.  
step4\_timer => a word holds the step 4 timer setpoint.  
ctrl\_options\_flag => a word value holds some of the options flag associates with each control.

ON	OFF
-----	-----
1H alternate	/ no alternation.

spare1 => unused byte value.  
step5\_timer => a word holds the step 5 timer setpoint.  
first\_id\_type => a byte value contains the index to the structure type.  
first\_id\_index => a word value contains the index to the pump source.  
second\_id\_type => a byte value contains the index to the structure type.  
second\_id\_index => a word value contains the index to the pump source.  
third\_id\_type => a byte value contains the index to the structure type.  
third\_id\_index => a word value contains the index to the pump source.  
forth\_id\_type => a byte value contains the index to the structure type.  
forth\_id\_index => a word value contains the index to the pump source.  
step6\_timer => a word holds the step 6 timer setpoint.  
step1\_timer\_count => a word holds the step 1 time counter.  
step2\_timer\_count => a word holds the step 2 time counter.  
step3\_timer\_count => a word holds the step 3 time counter.  
step4\_timer\_count => a word holds the step 4 time counter.  
step5\_timer\_count => a word holds the step 5 time counter.  
step6\_timer\_count => a word holds the step 6 time counter.  
on\_off\_scale => a word holds the step setpoints scale.



## STEP PUMPS

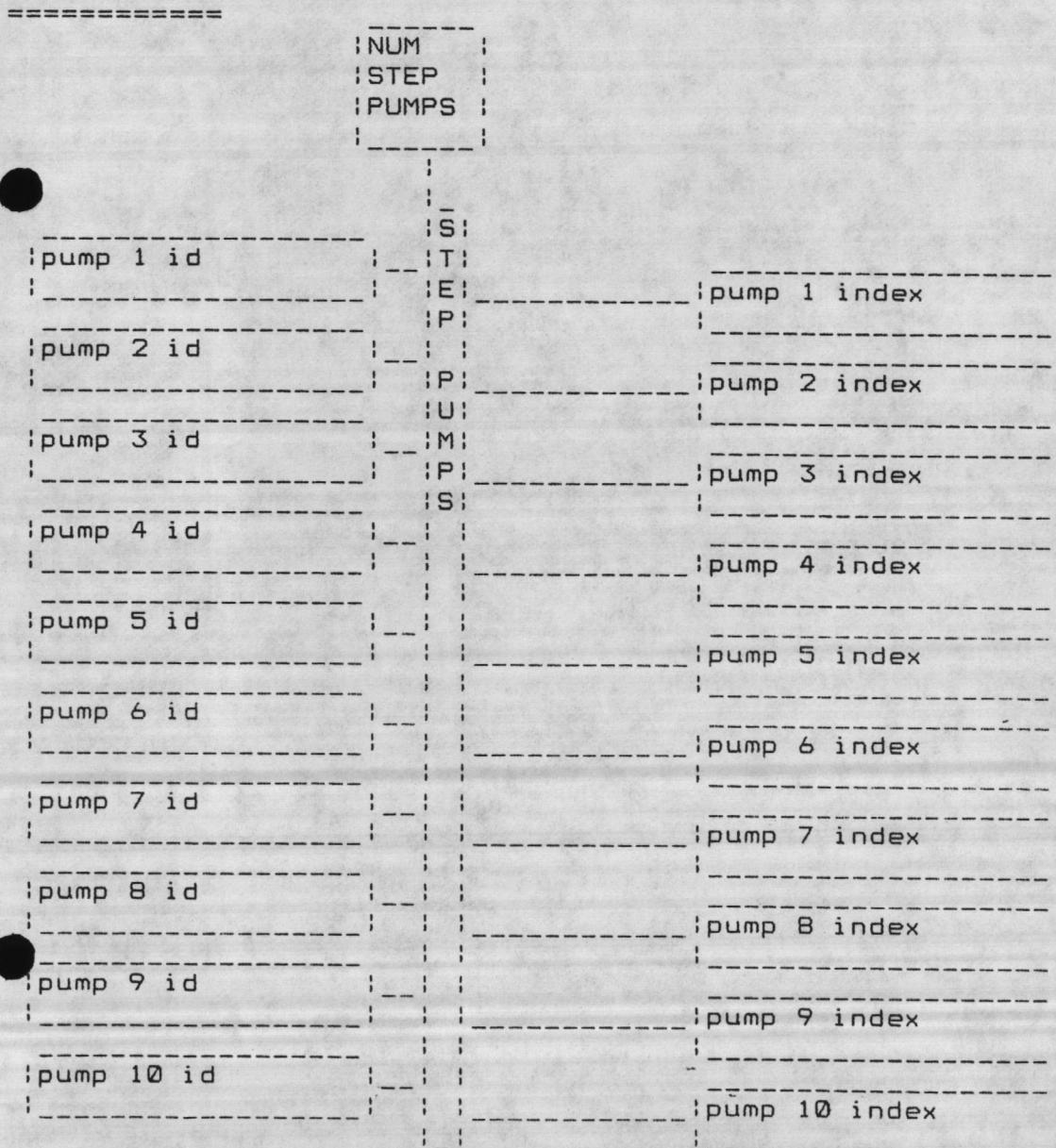
```

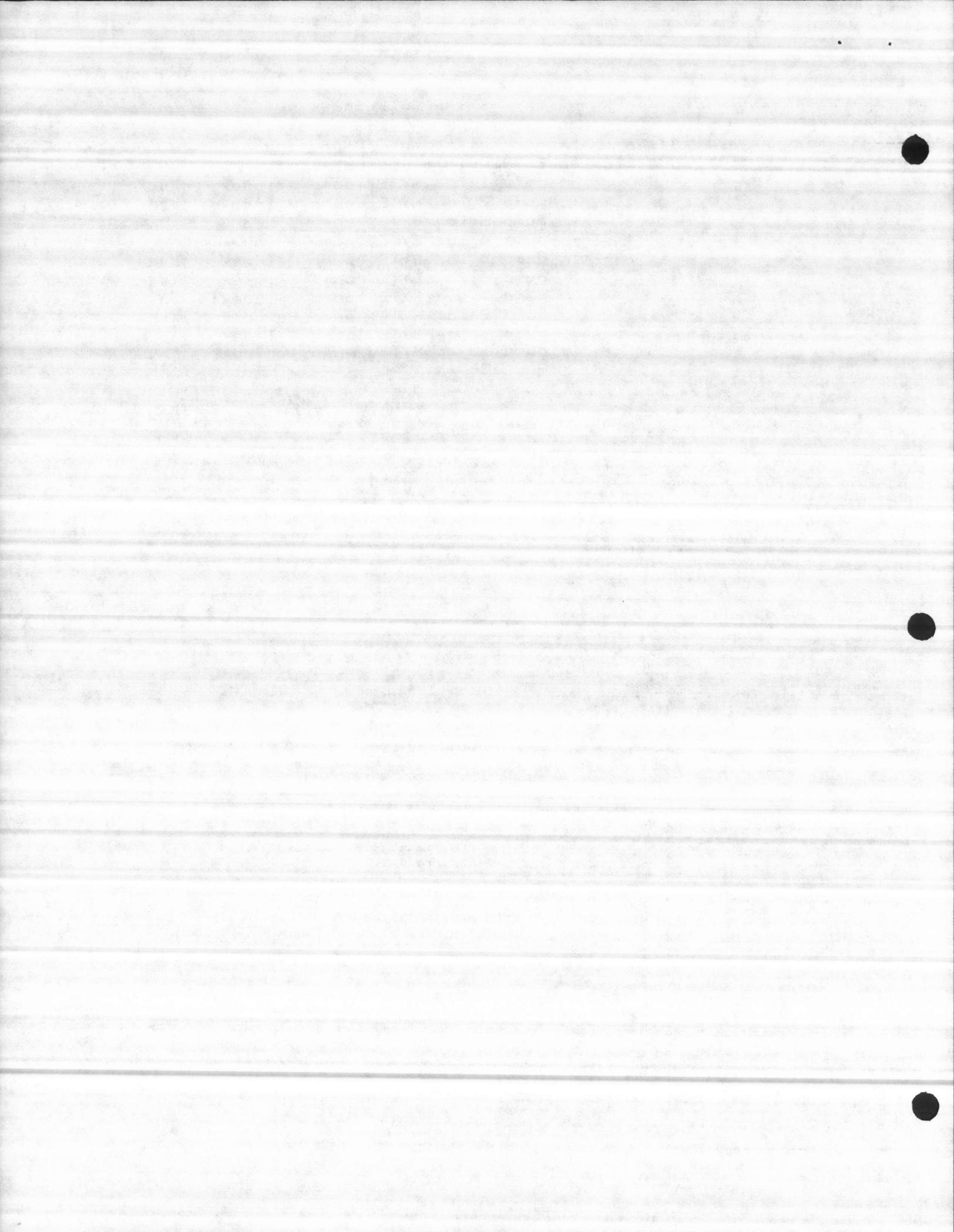
DECLARE step_pumps (NUM_STEP_PUMPS_PLUS1) STRUCTURE
    (pump_1_id     BYTE, pump_1_index      WORD,
     pump_2_id     BYTE, pump_2_index      WORD,
     pump_3_id     BYTE, pump_3_index      WORD,
     pump_4_id     BYTE, pump_4_index      WORD,
     pump_5_id     BYTE, pump_5_index      WORD,
     pump_6_id     BYTE, pump_6_index      WORD,
     pump_7_id     BYTE, pump_7_index      WORD,
     pump_8_id     BYTE, pump_8_index      WORD,
     pump_9_id     BYTE, pump_9_index      WORD,
     pump_10_id    BYTE, pump_10_index     WORD)
PUBLIC INITIAL
( 0FFH, 0, 0FFH, 0, 0FFH, 0, 0FFH, 0,
  0FFH, 0, 0FFH, 0, 0FFH, 0, 0FFH, 0,
  0FFH, 0, 0FFH, 0);

```

STEP\_PUMPS STRUCTURE SIZE = 31 \* 30 = 930 BYTES.

DATA DIAGRAM :





step\_pumps STRUCTURE :

pump\_1\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_1\_index => a word value contains the index to the pump 1.

pump\_2\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_2\_index => a word value contains the index to the pump 2.

pump\_3\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_3\_index => a word value contains the index to the pump 3.

pump\_4\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_4\_index => a word value contains the index to the pump 4.

pump\_5\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_5\_index => a word value contains the index to the pump 5.

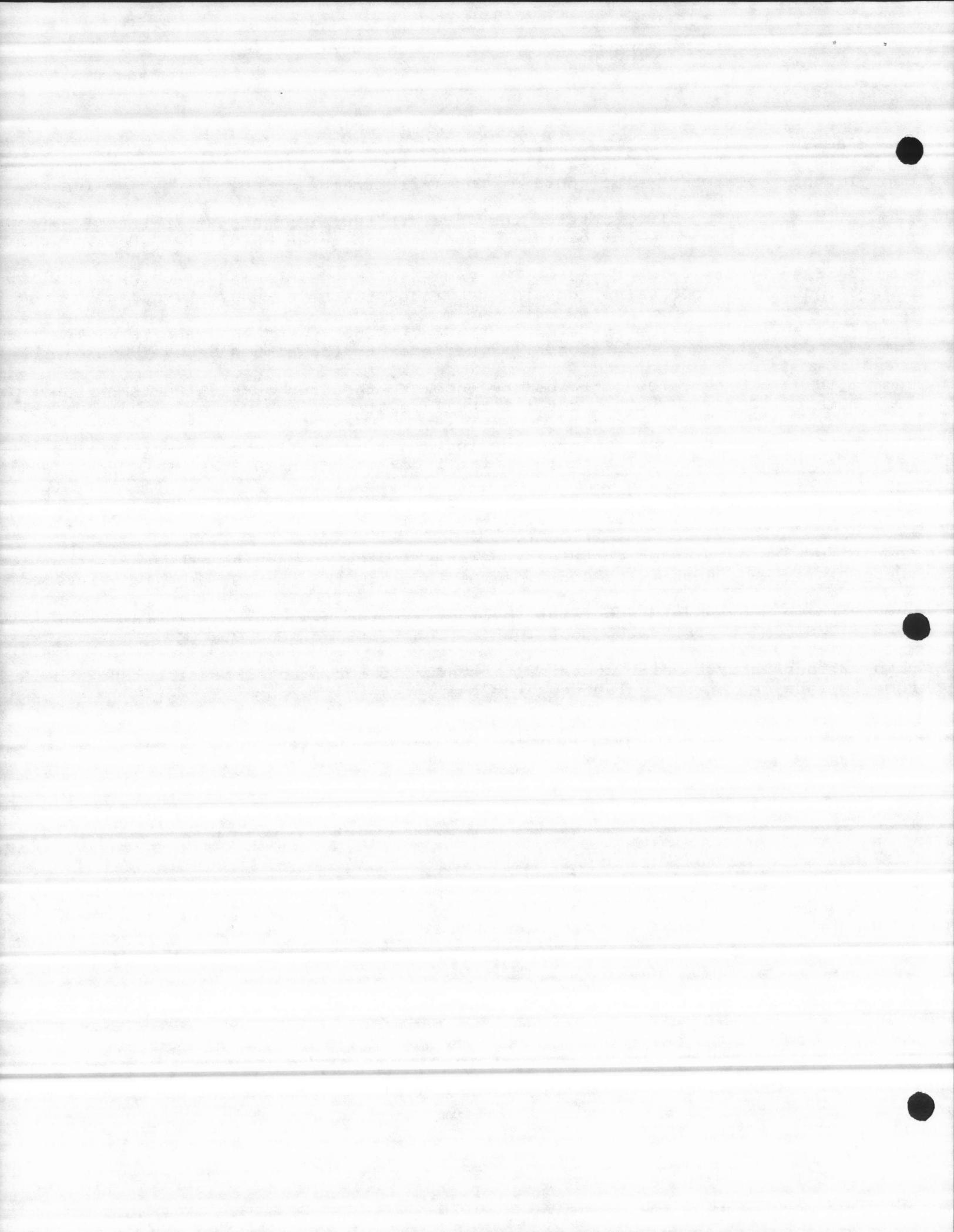
pump\_6\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_6\_index => a word value contains the index to the pump 6.

pump\_7\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_7\_index => a word value contains the index to the pump 7.

pump\_8\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_8\_index => a word value contains the index to the pump 8.

pump\_9\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_9\_index => a word value contains the index to the pump 9.

pump\_10\_id => a byte value contains the index to the step\_pumps structure type.  
pump\_10\_index => a word value contains the index to the pump 10.



(2.4) MODULE NAME : CxxxxTABL.PUB & CxxxxTABL.EXT

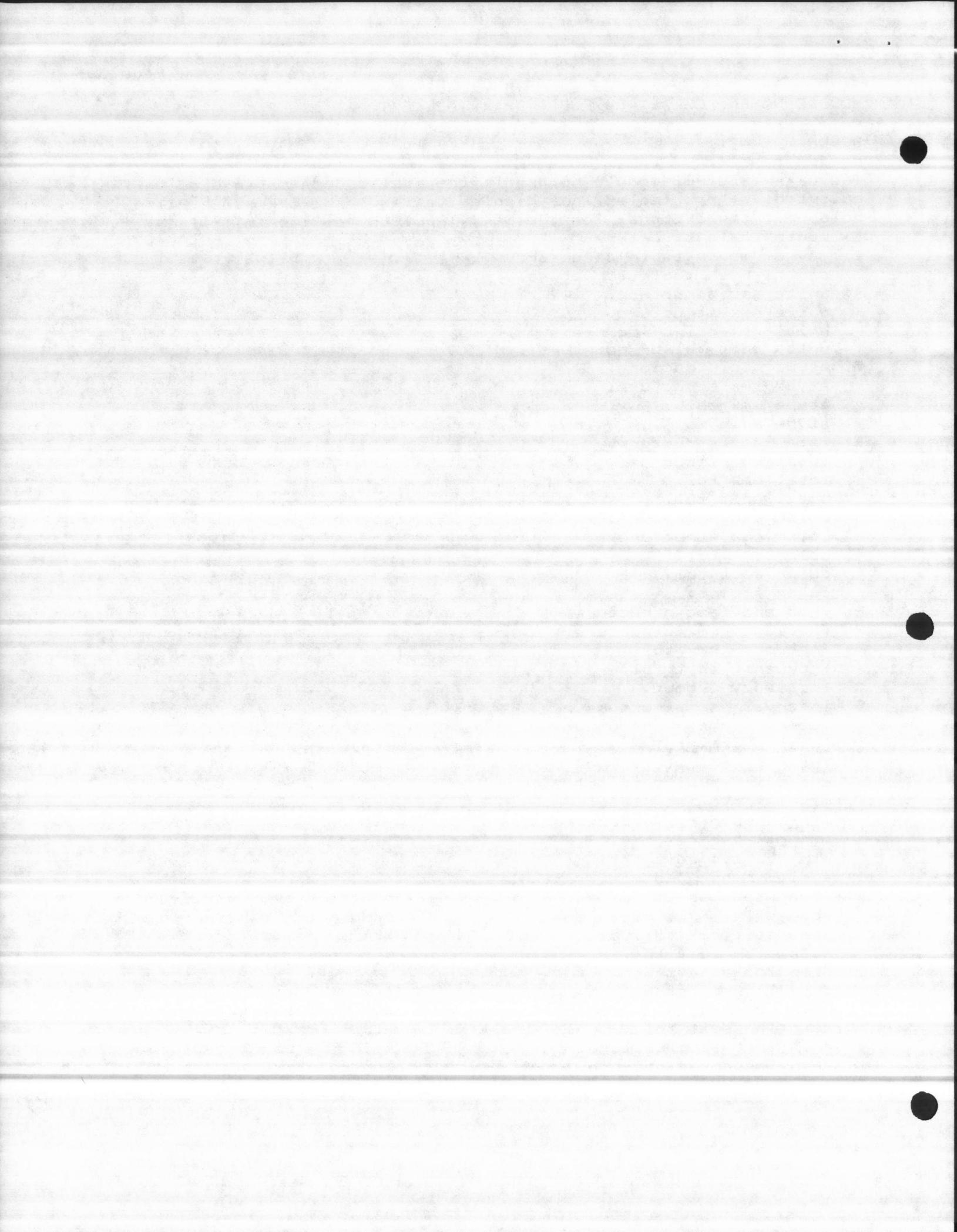
=====

DESCRIPTION :

=====

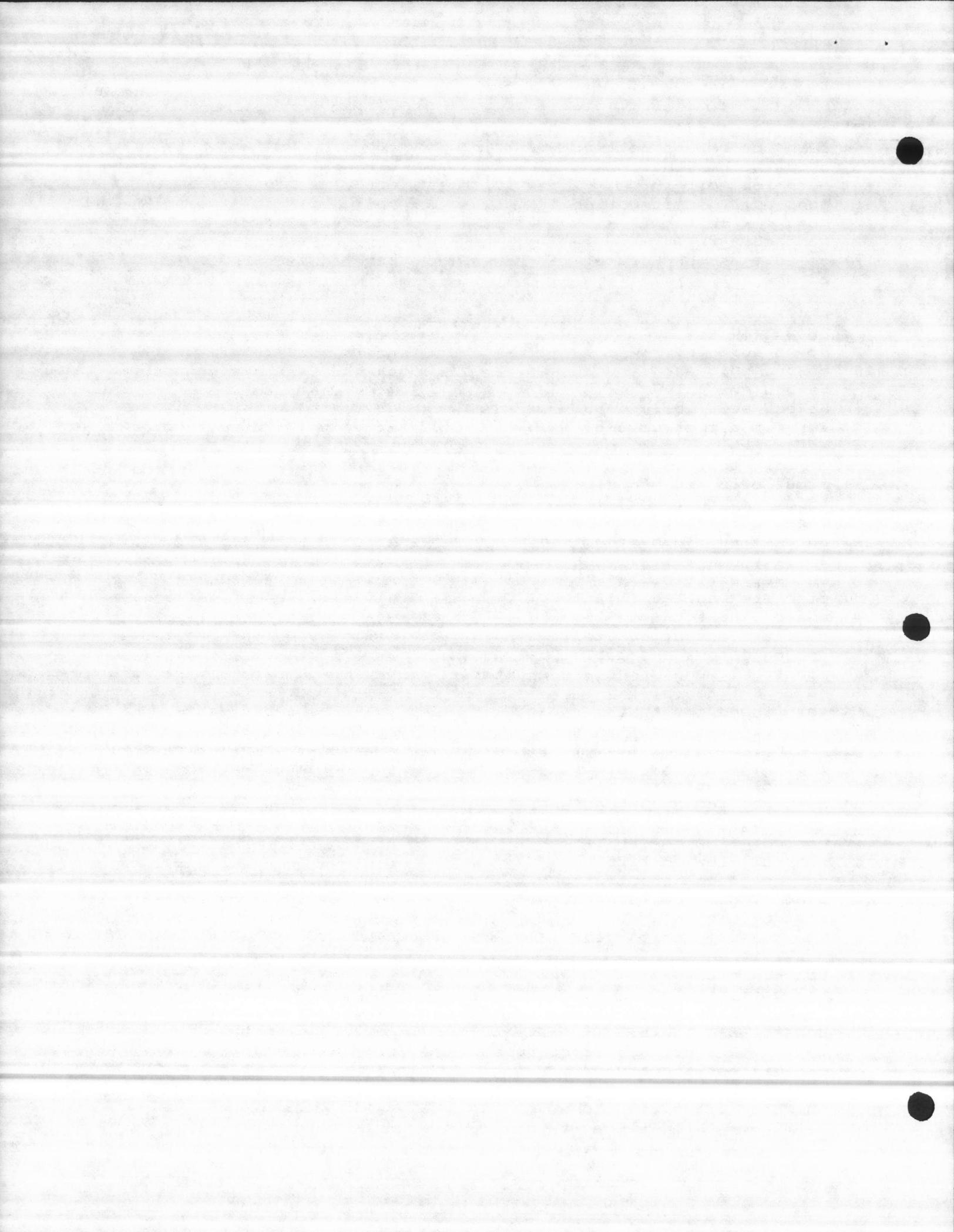
The CxxxxTABL.PUB module holds all the public data structure, variables, edit\_tables, rmx\_tables and display tables & the CxxxxTABL.EXT holds all the external for the CxxxxTABL.PUB of the following :

- 1) month\_table structure.
- 2) day\_of\_month\_table structure.
- 3) day\_of\_week\_table structure.
- 4) units\_asc structure.  
    units\_rmx structure.  
    units\_rmx\_tbl of pointers.  
    units\_edit\_tbl of pointers.
- 5) atype\_asc structure.  
    atype\_rmx structure.  
    atype\_rmx\_tbl  
    atype\_edit\_tbl
- 6) total\_asc structure.  
    total\_rmx structure.  
    total\_rmx\_tbl.  
    total\_edit\_tbl.
- 7) color\_asc structure.  
    color\_rmx structure.  
    color\_rmx\_tbl.  
    color\_edit\_tbl.  
    color\_prompt.
- 8) frequency\_asc structure.  
    frequency\_rmx structure.  
    frequency\_rmx\_tbl.  
    frequency\_edit\_tbl.  
    frequency\_prompt.
- 9) interval\_asc structure.  
    interval\_rmx structure.  
    interval\_rmx\_tbl.  
    interval\_edit\_tbl.  
    short\_interval\_edit\_tbl.
- 10) min\_max\_avg\_asc structure.  
    min\_max\_avg\_rmx structure.  
    min\_max\_avg\_rmx\_tbl.  
    min\_max\_avg\_edit\_tbl.
- 11) scale, range and mask variables.
- 12) scan\_stat\_asc structure.  
    scan\_stat\_rmx structure.  
    scan\_stat\_rmx\_tbl.  
    scan\_stat\_edit\_tbl.
- 13) dcin\_loc\_asc structure.  
    dcin\_loc\_rmx structure.  
    dcin\_loc\_rmx\_tbl.  
    dcin\_loc\_edit\_tbl.
- 14) anin\_loc\_asc structure.  
    anin\_loc\_rmx structure.  
    anin\_loc\_rmx\_tbl.  
    anin\_loc\_edit\_tbl.



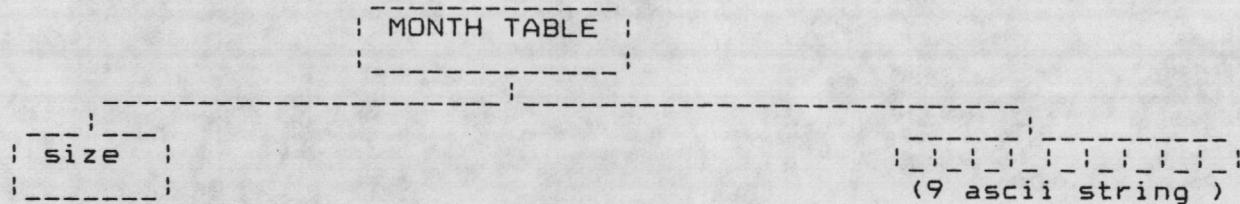
---

15) outp\_loc\_asc structure.  
outp\_loc\_rmx structure.  
outp\_loc\_rmx\_tbl.  
outp\_loc\_edit\_tbl.  
16) nonc\_edit\_asc structure.  
nonc\_rmx\_tbl.  
nonc\_edit\_tbl.  
17) oo\_edit\_asc structure.  
oo\_rmx\_tbl.  
oo\_edit\_tbl.  
18) noyes\_edit\_asc.  
noyes\_rmx\_tbl.  
noyes\_edit\_tbl.  
19) hexbcd\_edit\_asc structure.  
hexbcd\_rmx\_tbl.  
hexbcd\_edit\_tbl.  
20) ae\_edit\_asc structure.  
ae\_rmx\_tbl.  
ae\_edit\_tbl.  
21) inout\_edit\_asc structure.  
inout\_rmx\_tbl.  
inout\_edit\_tbl.  
22) autokey\_edit\_asc structure.  
autokey\_rmx\_tbl.  
autokey\_edit\_tbl.  
23) discrete\_input\_crt\_table.  
24) analog\_input\_crt\_table.  
25) discrete\_output\_crt\_table.  
26) analog\_output\_crt\_table.  
27) construct\_menu structure and table.  
28) param\_crt\_table.  
29) year\_trend\_crt\_table.  
30) control\_stat\_asc structure.  
control\_stat\_rmx structure.  
control\_stat\_rmx\_tbl.  
control\_stat\_edit\_tbl.  
31) control\_flow\_asc structure.  
control\_flow\_rmx structure.  
control\_flow\_rmx\_tbl.  
control\_flow\_edit\_tbl.  
32) control\_dsen\_asc structure.  
control\_dsen\_rmx structure.  
control\_desn\_rmx\_tbl.  
control\_dsen\_edit\_tbl.  
33) hoa\_edit\_asc.  
hoa\_rmx\_tbl.  
hoa\_edit\_tbl.  
34) level\_control\_crt\_table.  
35) control\_crt\_table.  
36) data base static\_asc structures.  
37) df\_edit\_asc structure.  
nulldf\_rmx\_tbl, nulldf\_edit\_tbl  
nullC\_edit\_asc Structure.  
nullC\_rmx\_tbl, nullC\_edit\_tbl.



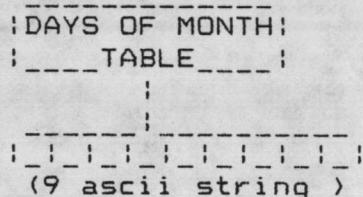
## 1) month\_table STRUCTURE:

This structure holds all the ascii of the month names and the ascii size of each month.



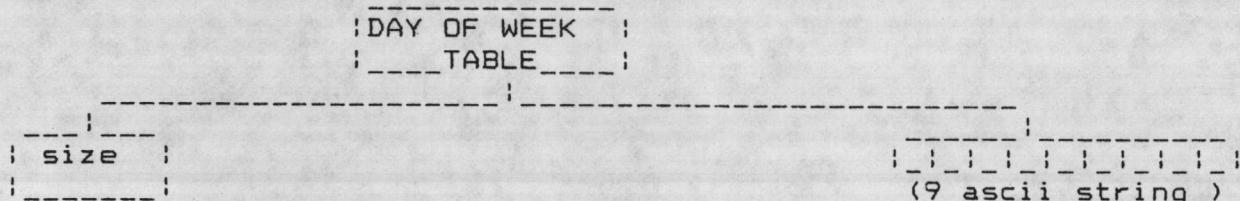
## 2) day\_of\_month\_table STRUCTURE :

This structure holds all the ascii of all the days in the month.



## 3) day\_of\_week\_table STRUCTURE :

This structure holds all the day names and the ascii size of each day



## 4) units\_asc STRUCTURE :

This structure holds all the spac eng. units ascii strings

## units\_rmx STRUCTURE :

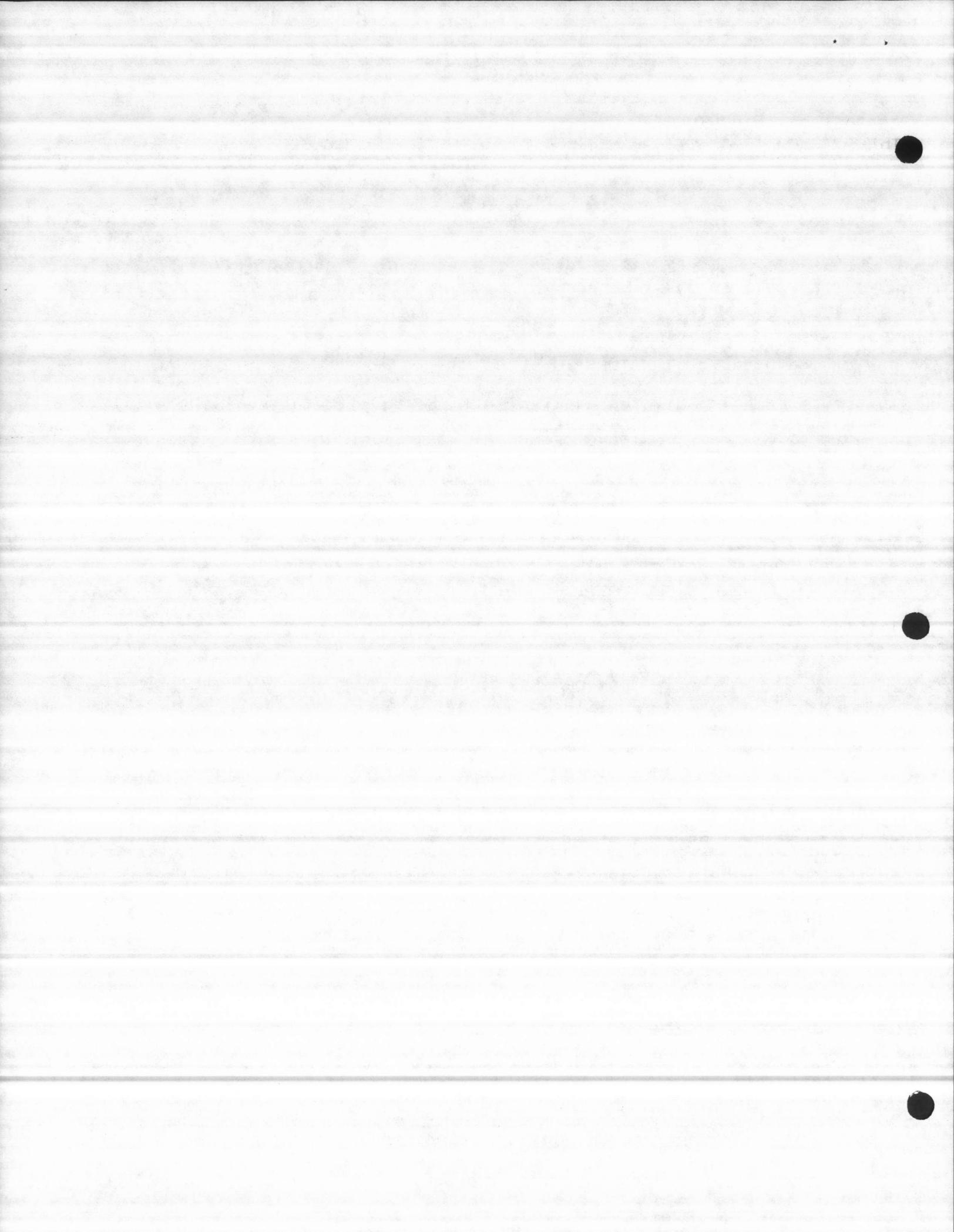
This structure holds all the engineering units ascii strings and the string count.

## units\_rmx\_tbl:

holds pointers to all the engineering units.

## units\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which specifies how many engineering units exist, the second pointer points to the units\_asc structure.



5) atype\_asc STRUCTURE :

This structure holds the alarm type ascii strings

atype\_rmx STRUCTURE :

This structure holds the alarm types ascii strings and the string count.

atype\_rmx\_tbl:

holds pointers to the alarm types

atype\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many alarm types exist, the second pointer points to the atype\_asc structure.

6) total\_asc STRUCTURE :

This structure holds the engineering units ascii strings for totals.

total\_rmx STRUCTURE :

This structure holds the engineering units ascii strings for totals and the string count.

total\_rmx\_tbl:

holds pointers to the engineering units for totals.

total\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many total eng units exist, the second pointer points to the total\_asc structure.

7) color\_asc STRUCTURE :

This structure holds the colors ascii strings

color\_rmx STRUCTURE :

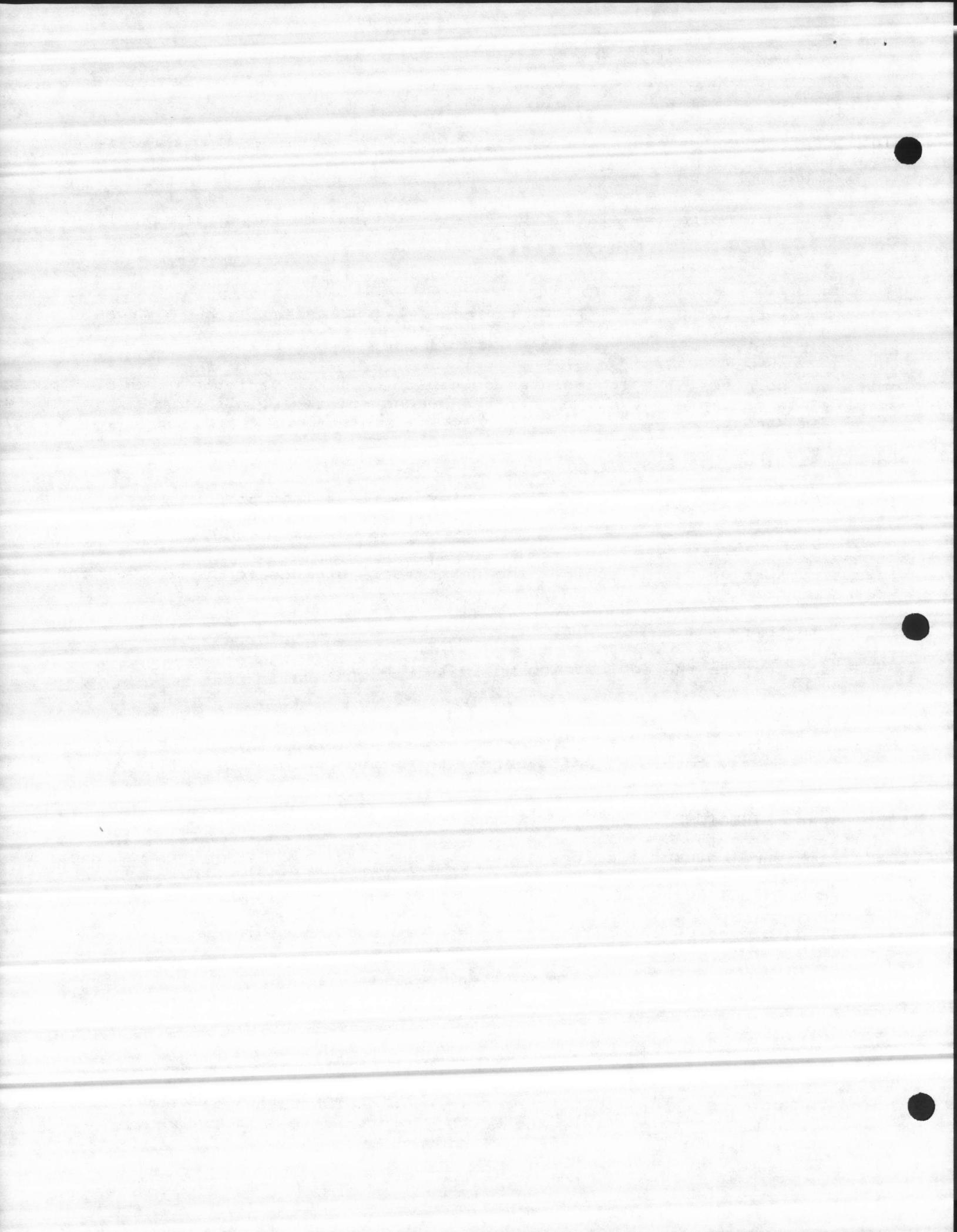
This structure holds the colors ascii strings and the string count.

color\_rmx\_tbl:

holds pointers to the colors

color\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many colors exist, the second pointer points to the color\_asc structure.



color\_prompt STRUCTURE:

This structure similar to the color\_asc and holds the colors menu\_ascii string for menu display.

8) frequency\_asc STRUCTURE :

This structure holds the report frequency ascii strings

frequency\_rmx STRUCTURE :

This structure holds the report frequency ascii strings and the string count.

frequency\_rmx\_tbl:

holds pointers to the report frequencies.

frequency\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many frequencies exist, the second pointer points to the frequency\_asc structure.

frequency\_prompt STRUCTURE:

This structure similar to the frequency\_asc and holds the frequencies menu\_ascii string for menu display.

9) interval\_asc STRUCTURE :

This structure holds the trend interval ascii strings

interval\_rmx STRUCTURE :

This structure holds the trend interval ascii strings and the string count.

interval\_rmx\_tbl:

holds pointers to the trend intervals

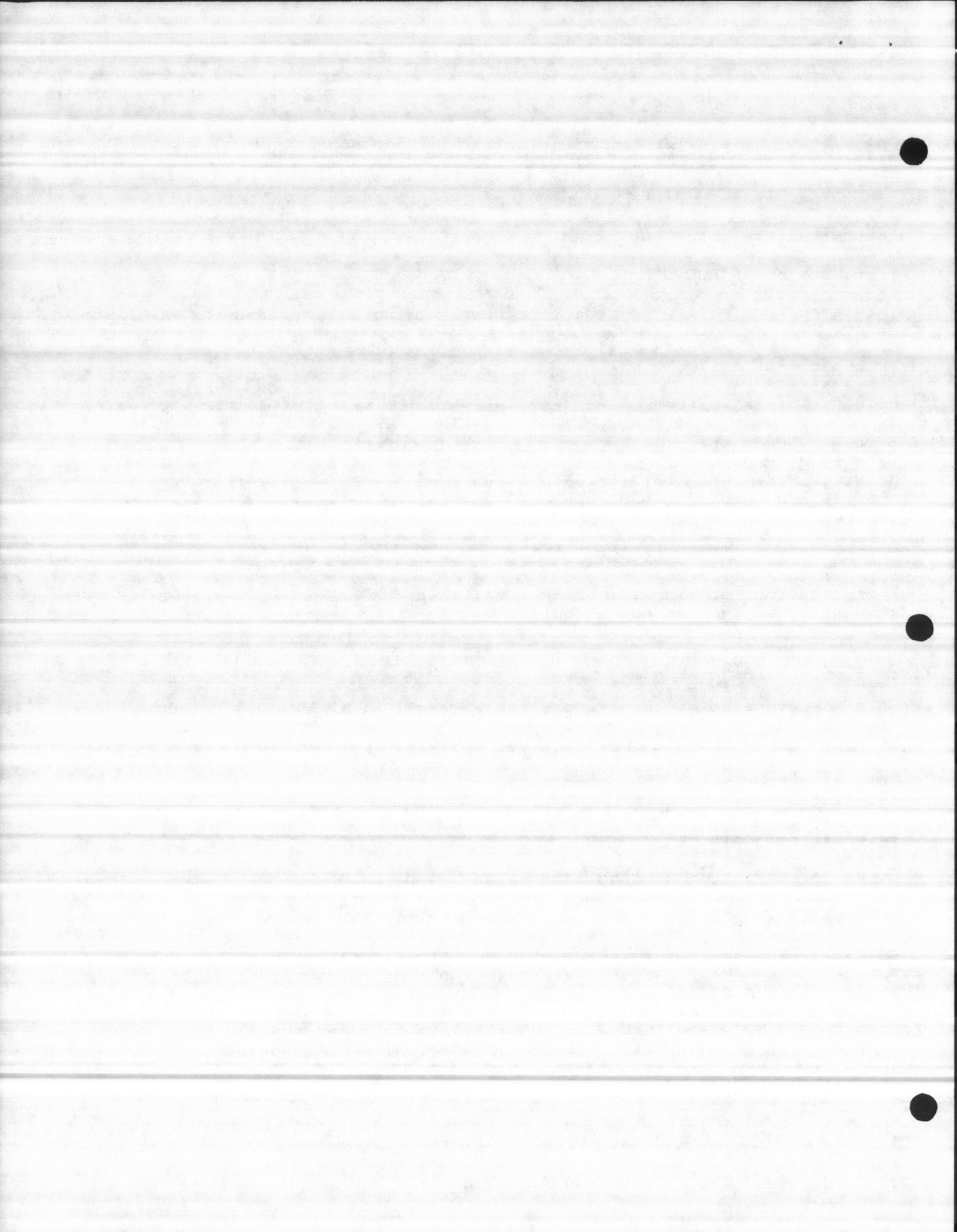
interval\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many trend intervals exist, the second pointer points to the interval\_asc structure.

/\* shortened for use with draw trend task, which cannot handle the \*/  
/\* dword total trends. Used in GGEN to generate trend graphs. \*/

short\_interval\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many trend intervals exist, the second pointer points to the interval\_asc structure.



10) min\_max\_avg\_asc STRUCTURE :

This structure holds the minimum, maximum and average ascii strings

min\_max\_avg\_rmx STRUCTURE :

This structure holds the above ascii strings and the string count.

min\_max\_avg\_rmx\_tbl:

holds pointers to the minimum, maximum and average.

min\_max\_avg\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many string exist, the second pointer points to the min\_max\_avg\_asc structure.

11) scale, range and mask Variables:

rt\_scale => a word holds the rate scale whish is - 2 .

roc\_scale => a word holds the rate of change scale which is - 1.

total\_scale => three words hold the total scales.

scale => three words hold the scales.

alarm\_sec\_min\_max => two words hold the alarm sectin min which is 0 and the alarm section max which is 32.

dailer\_min\_max => two words hold the dailer min which is 0 and the dailer max which is 15.

address\_min\_max => two words hold the word address min which is 0 and the word address max which is 99.

field\_min\_max => two words hold the analog field min which is 0 and the analog field max which is 48.

password\_min\_max => two words hold the password min which is 1 and the password max which is 3. This means we have only three passwords 1th ,2nd and 3rd.

port\_min\_max => two words hold the port # min which is 0 and the port # max which is 99.

scale\_min\_max => two words hold the standard scale min which is - 4 and the standard scale max which is 2.

byte\_min\_max => two words hold the byte min value which is 0 and the byte max value which is 0FFH.

word\_min\_max => two words hold the word min value which is 0 and the word max value which is 0FFFFH.

sword\_min\_max => two words hold the sign word min value which is 8000H and the sign word max value which is 07FFFH.

bit\_min\_max => two words hold the bit min value which is 0 and the bit max value which is 0FFFFH.

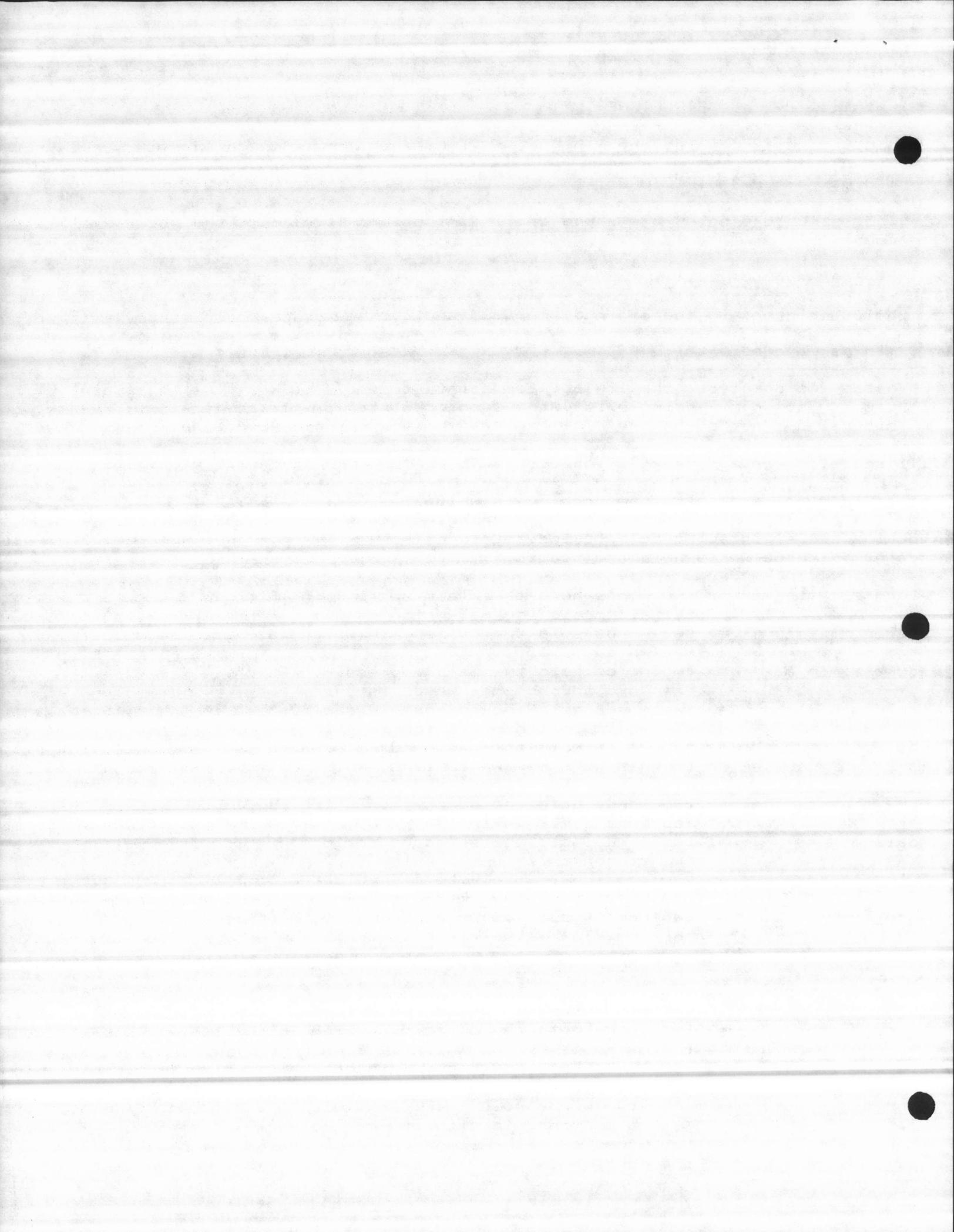
dword\_min\_max => two words hold the dword min value which is 0 and the dword max value which is 0xFFFFFFFF.

sdword\_min\_max => two words hold the sign dword min value which is 80000000H and the sign dword max value which is 07FFFFFFH.

four\_digit\_word\_min\_max=> two words hold the four digit word min value which is 0 and the 4 digit word max value which is 9999.

io\_hex\_min\_max => two words hold the io hex word min value which is 0 and the io hex max value which is 0FFFH.

cond\_str\_table\_size => a word initialized as the NUM\_COND\_STRINGS.



```

neg_one    => a word holds the value of - 1, which is 0FFFFH.
zero       => a word holds the value of 0.
one        => a word holds the value of 1.
two        => a word holds the value of 2.
three      => a word holds the value of 3.
four       => a word holds the value of 4.
five       => a word holds the value of 5.
six        => a word holds the value of 6.
seven      => a word holds the value of 7.
eight      => a word holds the value of 8.
ten        => a word holds the value of 10.
twelve     => a word holds the value of 12.
fourteen   => a word holds the value of 14.
fifteen    => a word holds the value of 15.
sixteen    => a word holds the value of 16.
twenty     => a word holds the value of 20.
twenty_four=> a word holds the value of 24.
thirty     => a word holds the value of 30.
sixty      => a word holds the value of 60.
seventy_one=> a word holds the value of 71.
eighty     => a word holds the value of 80.

```

---

1	2	4	8	10	20	40	80	100	200	400	800	1000	2000	4000	8000
---	---	---	---	----	----	----	----	-----	-----	-----	-----	------	------	------	------

---

```

one_mask      => a word holds the value of 1 to mask on bit 1.
two_mask      => a word holds the value of 2 to mask on bit 2.
four_mask     => a word holds the value of 4 to mask on bit 4.
eight_mask    => a word holds the value of 8 to mask on bit 8.
ten_mask      => a word holds the value of 10H to mask on bit 10.
twenty_mask   => a word holds the value of 20H to mask on bit 20.
fourty_mask   => a word holds the value of 40H to mask on bit 40.
eighty_mask   => a word holds the value of 80H to mask on bit 80.
one_hundred_mask=> a word holds the value of 100H to mask on bit 100.
two_hundred_mask=> a word holds the value of 200H to mask on bit 200.
four_hundred_mask=> a word holds the value of 400H to mask on bit 400.
eight_hundred_mask=> a word holds the value of 800H to mask on bit 800.

```

12) d => 10 bytes hold the string Deactivate.  
a => 10 bytes hold the string Activate.

scan\_stat\_edit\_asc STRUCTURE :

---

This structure holds all the scan status ascii strings.

scan\_stat\_rmx\_tbl:

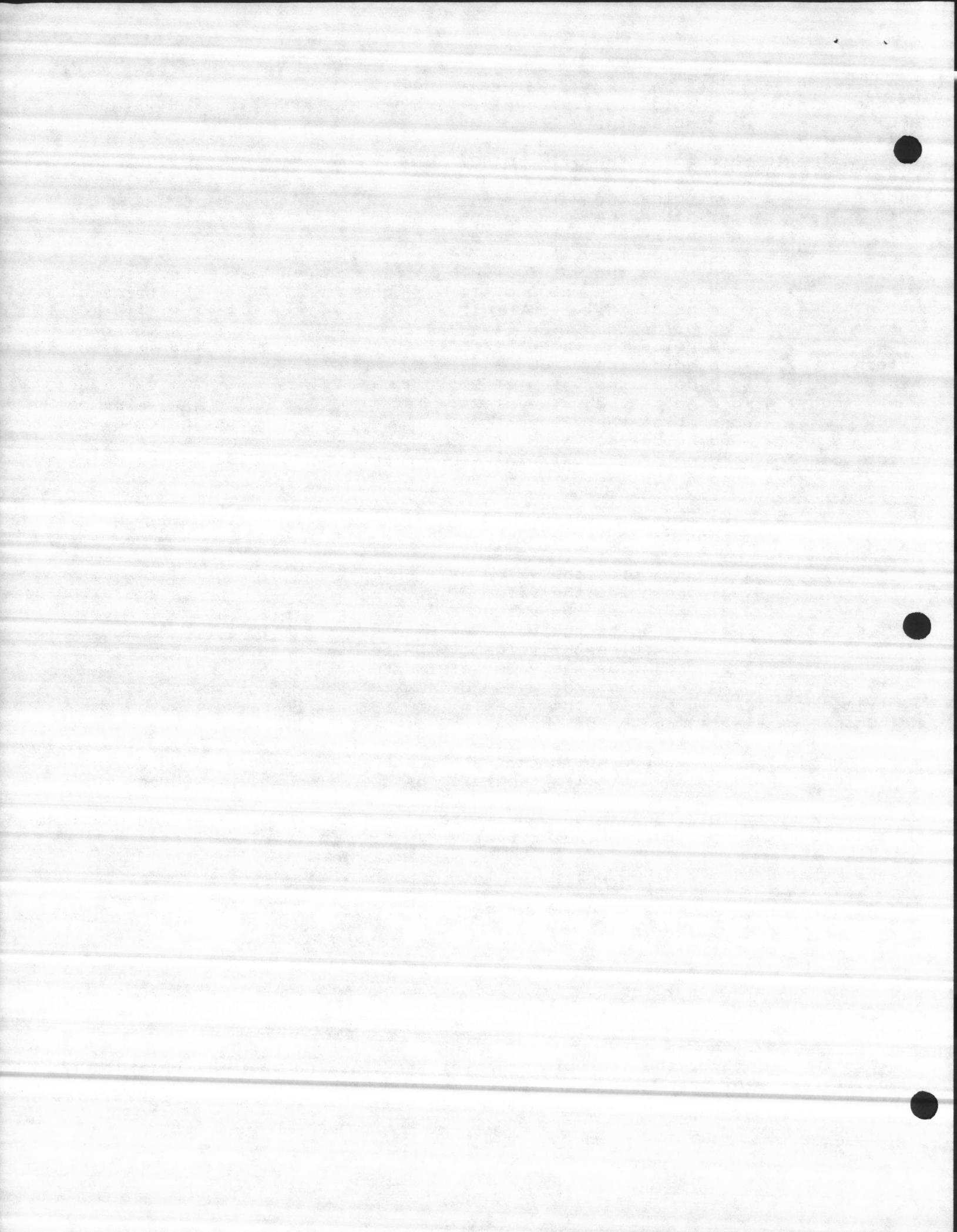
---

holds pointers to all the scan status.

scan\_stat\_edit\_tbl:

---

holds two pointers, the 1th pointer points to declared variable which is specify how many scan status exist, the second pointer points to the scan\_stat\_edit\_asc structure.



13) dcin\_loc\_asc STRUCTURE :

This structure holds all the discrete source ascii strings

dcin\_loc\_rmx STRUCTURE :

This structure holds all the discrete source ascii strings and the string count.

dcin\_loc\_rmx\_tbl:

holds pointers to all the discrete sources.

dcin\_loc\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many sources exist, the second pointer points to the dcin\_loc\_asc structure.

14) anin\_loc\_asc STRUCTURE :

This structure holds all the analog source ascii strings

anin\_loc\_rmx STRUCTURE :

This structure holds all the analog source ascii strings and the string count.

anin\_loc\_rmx\_tbl:

holds pointers to all the analog sources.

anin\_loc\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many sources exist, the second pointer points to the anin\_loc\_asc structure.

15) outp\_loc\_asc STRUCTURE :

This structure holds all the output source ascii strings

outp\_loc\_rmx STRUCTURE :

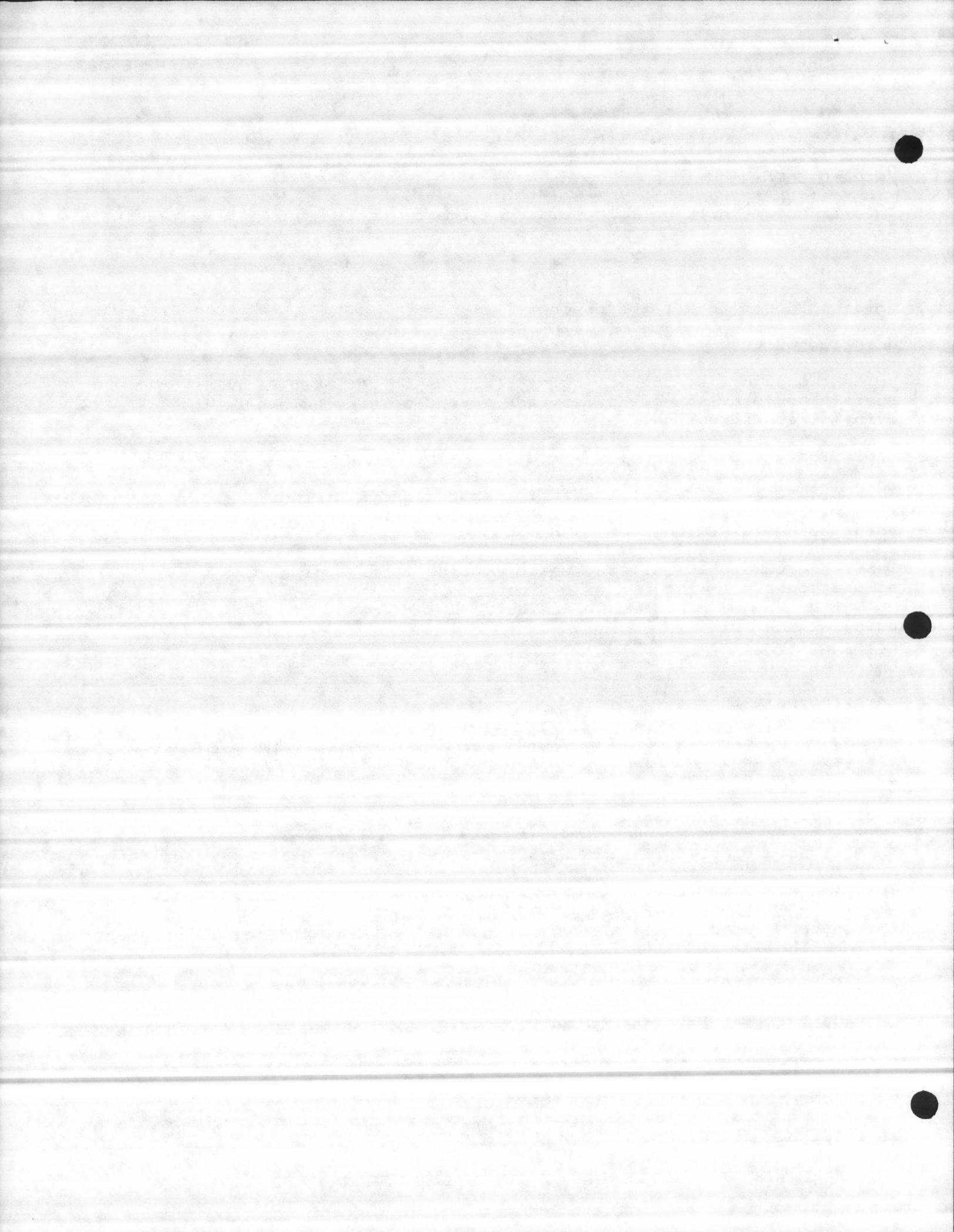
This structure holds all the output source ascii strings and the string count.

outp\_loc\_rmx\_tbl:

holds pointers to all the output sources.

outp\_loc\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many sources exist, the second pointer points to the outp\_loc\_asc structure.



16) open => 4 bytes hold the string Open.  
closed => 6 bytes hold the string Closed.

nonc\_edit\_asc STRUCTURE :

This structure holds all the open and closed status ascii strings

nonc\_rmx\_tbl:

holds pointers to all the open and closed status.

nonc\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is  
specify how many status exist, the second pointer points to the  
nonc\_edit\_asc structure.

17) off => 3 bytes hold the string Off.  
on => 2 bytes hold the string On.

oo\_edit\_asc STRUCTURE :

This structure holds all the off and on status ascii strings

oo\_rmx\_tbl:

holds pointers to all the off and on status.

oo\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is  
specify how many status exist, the second pointer points to the  
oo\_edit\_asc structure.

18) no => 2 bytes hold the string No.  
yes => 3 bytes hold the string Yes.

noyes\_edit\_asc STRUCTURE :

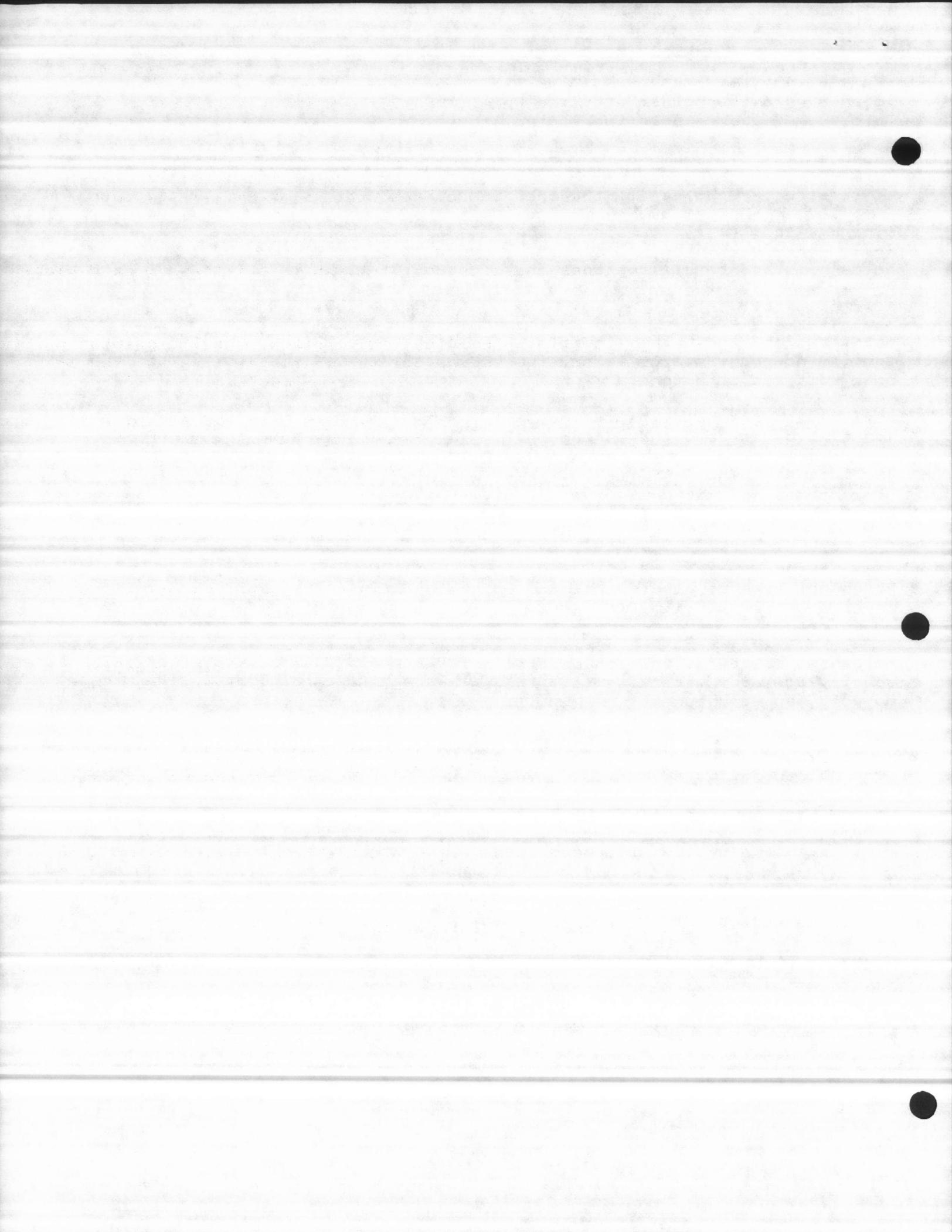
This structure holds all the no and yes conditions ascii strings

noyes\_rmx\_tbl:

holds pointers to the no and yes conditions.

noyes\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is  
specify how many conditions exist, the second pointer points to the  
noyes\_edit\_asc structure.



19) hex => 3 bytes hold the string Hex.  
bcd => 3 bytes hold the string Bcd.

hexbcd\_edit\_asc STRUCTURE :

This structure holds the hex and bcd ascii strings

hexbcd\_rmx\_tbl:

holds pointers to the hex and the bcd.

hexbcd\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many hex and bcd exist, the second pointer points to the hexbcd\_edit\_asc structure.

20) alarm => 5 bytes hold the string Alarm.  
event => 5 bytes hold the string Event.

ae\_edit\_asc STRUCTURE :

This structure holds the alarm and event ascii strings

ae\_rmx\_tbl:

holds pointers to the alarm and the event.

ae\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many alarm and event exist, the second pointer points to the ae\_edit\_asc structure.

21) in => 2 bytes hold the string In.  
Out => 3 bytes hold the string Out.

inout\_edit\_asc STRUCTURE :

This structure holds the in and out ascii strings

inout\_rmx\_tbl:

holds pointers to the in and the out strings.

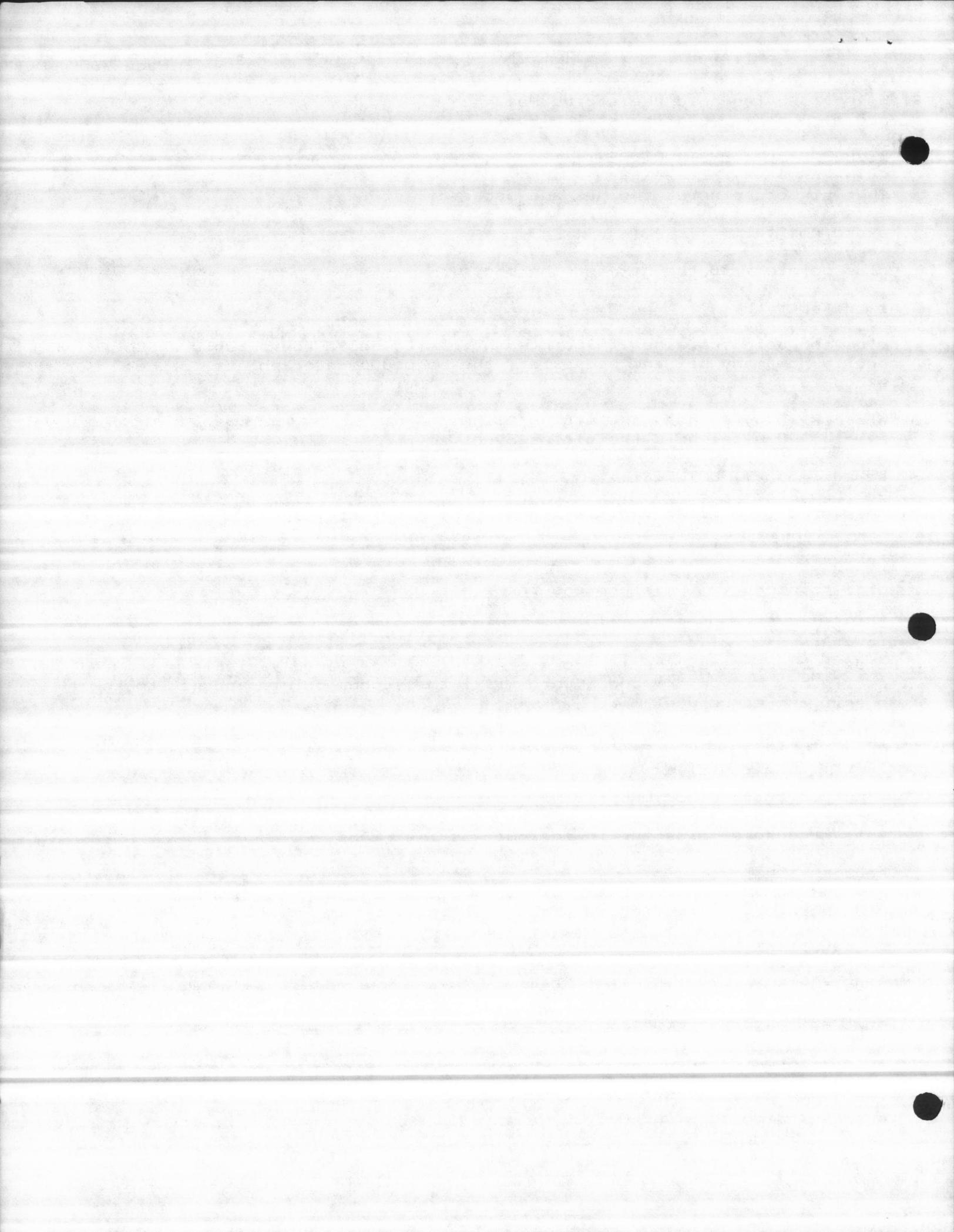
inout\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many in and out exist, the second pointer points to the inout\_edit\_asc structure.

22) auto => 4 bytes hold the string Auto.  
key => 3 bytes hold the string Key.

autokey\_edit\_asc STRUCTURE :

This structure holds the Auto and the Key ascii strings



autokey\_rmx\_tbl:  
-----  
holds pointers to the auto and the key strings

autokey\_edit\_tbl:  
-----  
holds two pointers, the 1th pointer points to declared variable which is specify how many auto and key exist, the second pointer points to the autokey\_edit\_asc structure.

23) discrete\_input\_crt\_table STRUCTURE :  
-----  
This structure is a crt table which holds and displays all the element in each point of the discrete input.(the number of elements = 30)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy

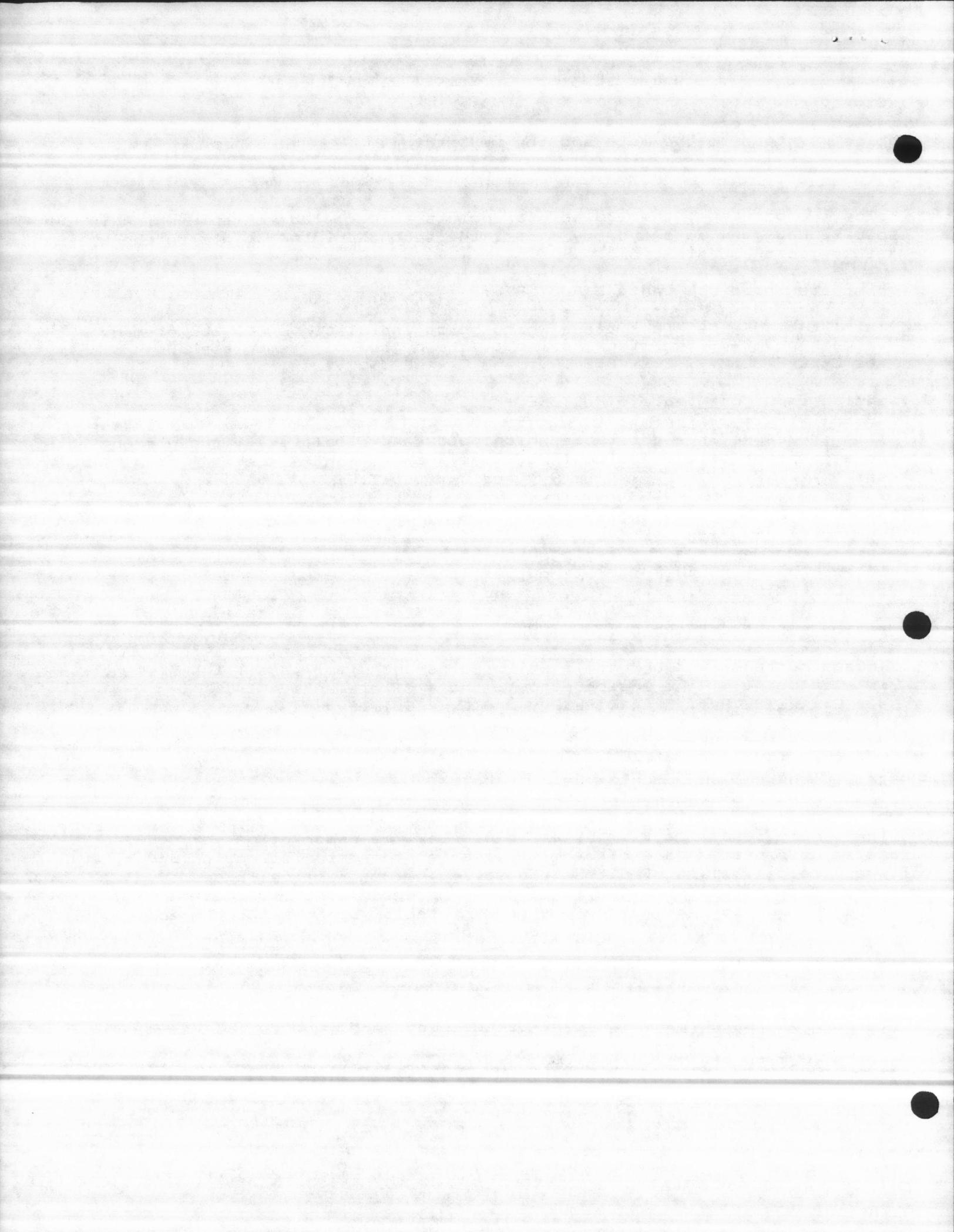
24) analog\_input\_crt\_table STRUCTURE :  
-----  
This structure is a crt table which holds and displays all the element in each point of the analog input.(the # of element = 48)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy

25) discrete\_output\_crt\_table STRUCTURE :  
-----  
This structure is a crt table which holds and displays all the element in each point of the discrete output.(the # of elements = 14)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy

analog\_output\_crt\_table STRUCTURE :  
-----  
This structure is a crt table which holds and displays all the element in each point of the analog output.(the # of elements = 14)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy

27) Construct variables and structure :  
-----  
construct\_menu\_command\_count => a byte holds the maximum # of commands.  
construct\_menu\_prompt\_asc STRUCTURE:  
    holds the menu ascii strings  
configure\_help\_desc => a # of bytes to desplay the description for configure.  
parameter\_help\_desc => a # of bytes to desplay the description for parameter.  
edit\_help\_desc => a # of bytes to desplay the description for edit.  
next\_help\_desc => a # of bytes to desplay the description for next.  
construct\_menu\_help\_desc\_tbl => a 4 pointer table point to the command description.

28) param\_crt\_table STRUCTURE :  
-----  
This structure is a crt table which holds and displays all the element in each parameter page in construct displays and reports.  
(the # of elements = 252)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy



## 29) year\_trend\_crt\_table STRUCTURE :

This structure is a crt table which holds and displays all the element in each point of the yearly trend structure. (the # of elements = 27)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy

## 30) control\_stat\_asc STRUCTURE :

This structure holds all the control status ascii strings

## control\_stat\_rmx STRUCTURE :

This structure holds all the control status ascii strings and the string count.

## control\_stat\_rmx\_tbl:

holds pointers to the two different control status.

## control\_stat\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many control status exist, the second pointer points to the control\_stat\_asc structure.

## 31) control\_flow\_asc STRUCTURE :

This structure holds all the flow directions ascii strings

## control\_flow\_rmx STRUCTURE :

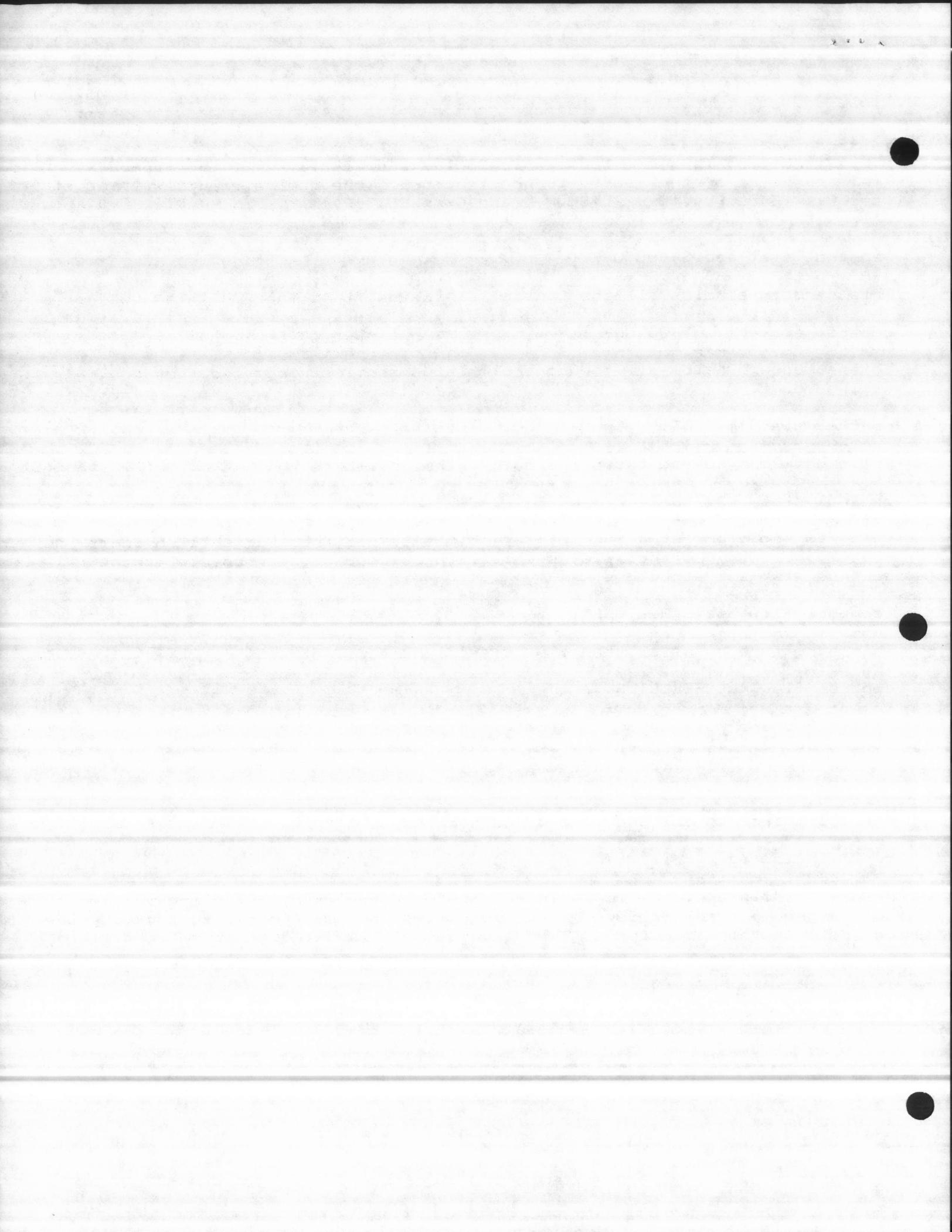
This structure holds all the flow directions ascii strings and the string count.

## control\_flow\_rmx\_tbl:

holds pointers to all the flow directions.

## control\_flow\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many directions exist, the second pointer points to the control\_flow\_asc structure.



32) control\_dsen\_asc STRUCTURE :

This structure holds the disable/enable ascii strings

control\_dsen\_rmx STRUCTURE :

This structure holds all the disable/enable ascii strings and the string count.

control\_dsen\_rmx\_tbl:

holds pointers to the control disable and the control enable.

control\_dsen\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many the control enable or disable, the second pointer points to the control\_dsen\_asc structure.

33) hoa\_edit\_asc :

Holds Handoff and Auto.

hoa\_rmx\_tbl :

Holds pointers to Handoff and Auto strings.

hoa\_edit\_tbl :

holds two pointers, the 1th pointer points to the number of strings (2) the 2nd pointer points to the hoa\_edit\_asc.

34) level\_control\_crt\_table STRUCTURE :

This structure is a crt table which holds and displays all the element in each point of the level points. (the # of elements = 23)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy

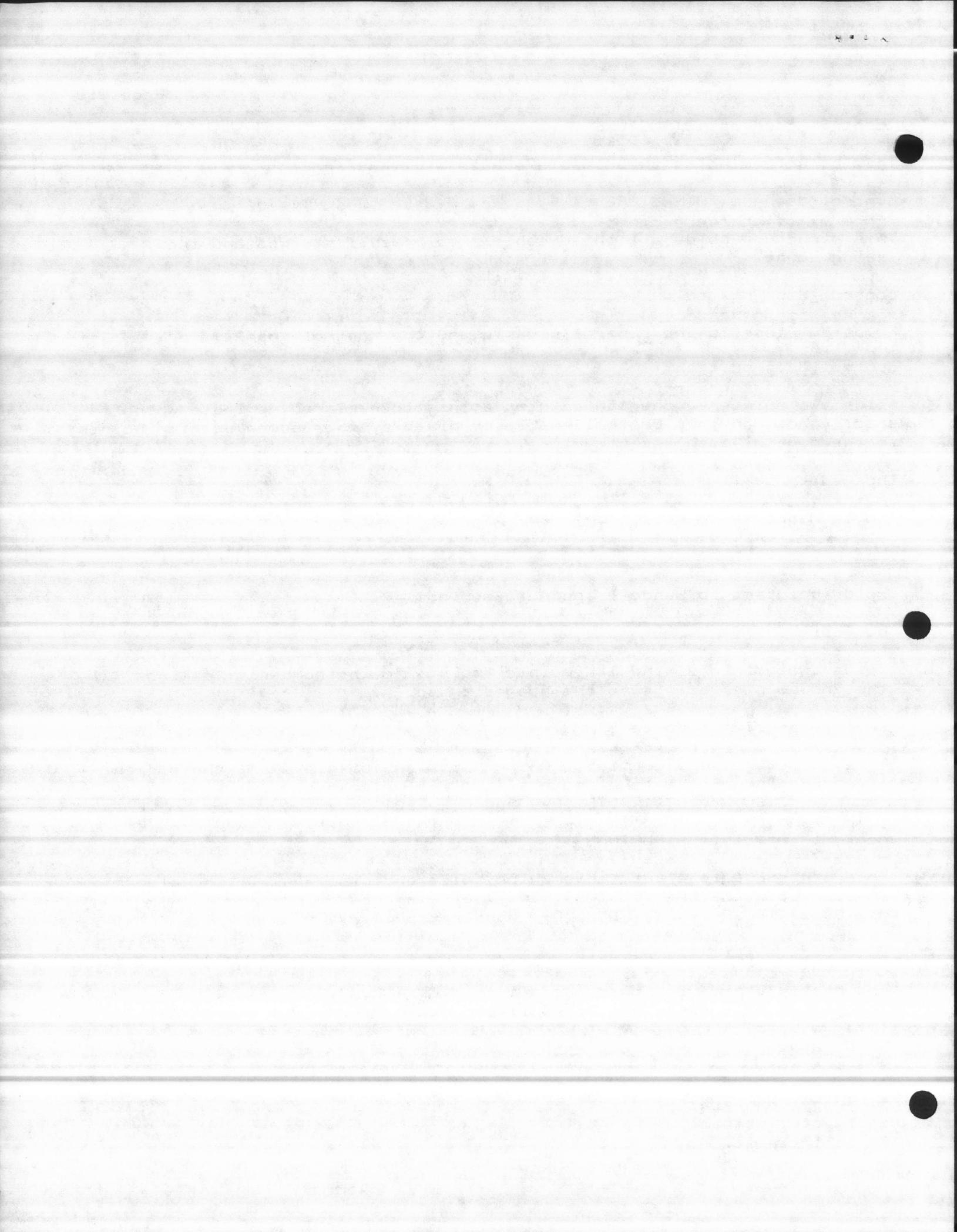
35) control\_crt\_table STRUCTURE :

This structure is a crt table which holds and displays all the element in each point of the control. (the # of elements = 107)  
see CxxxxETBL.Pyy, CxxxxDTBL.Pyy and CxxxxEDIT.Pyy

35) Data base static asc structures:

The static ascii structures used in the CxxxxDUMP.Pyy and CxxxxFILL.Pyy.  
The following are the static ascii structures :

- 1) ai\_static.
- 2) di\_static.
- 3) ao\_static.
- 4) do\_static.
- 5) year\_trend\_static.
- 6) lv\_ctrl\_static.
- 7) ctrl\_static.



36) df\_edit\_asc :

This structure holds the Null & Data Fail ascii strings

nulldf\_rmx\_tbl :

two pointer point to null & data fail strings.

nulldf\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many conditions exist, the second pointer points to the df\_edit\_asc.

nullC\_edit\_asc :

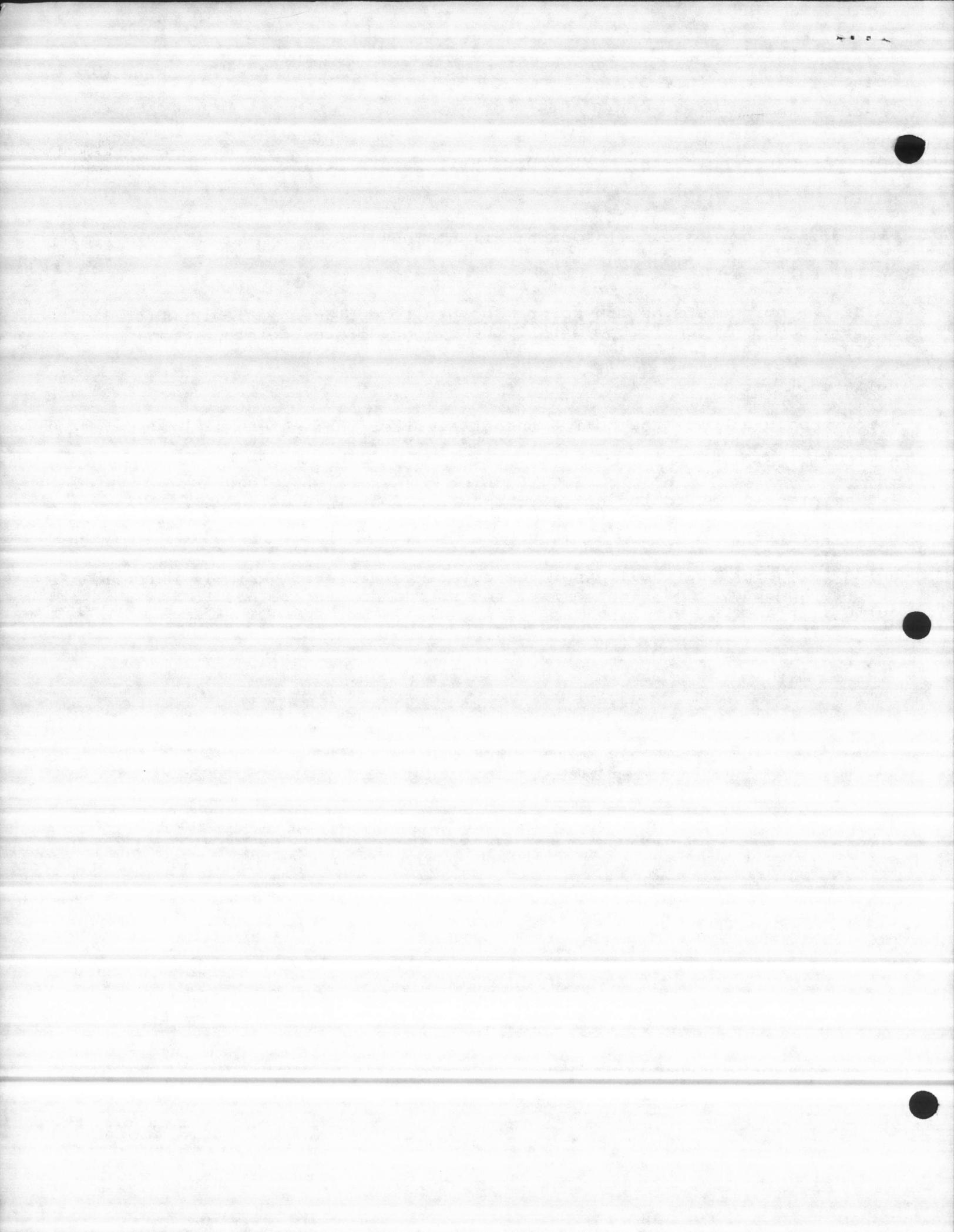
This structure holds the null & change ascii strings

nullC\_rmx\_tbl :

Two pointers point to the null & change ascii strings.

nullC\_edit\_tbl:

holds two pointers, the 1th pointer points to declared variable which is specify how many conditions exist, the second pointer points to the nullC\_edit\_asc.



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 3

HISTORICAL  
GLOBAL

AND

HISTORICAL  
EXTERNAL

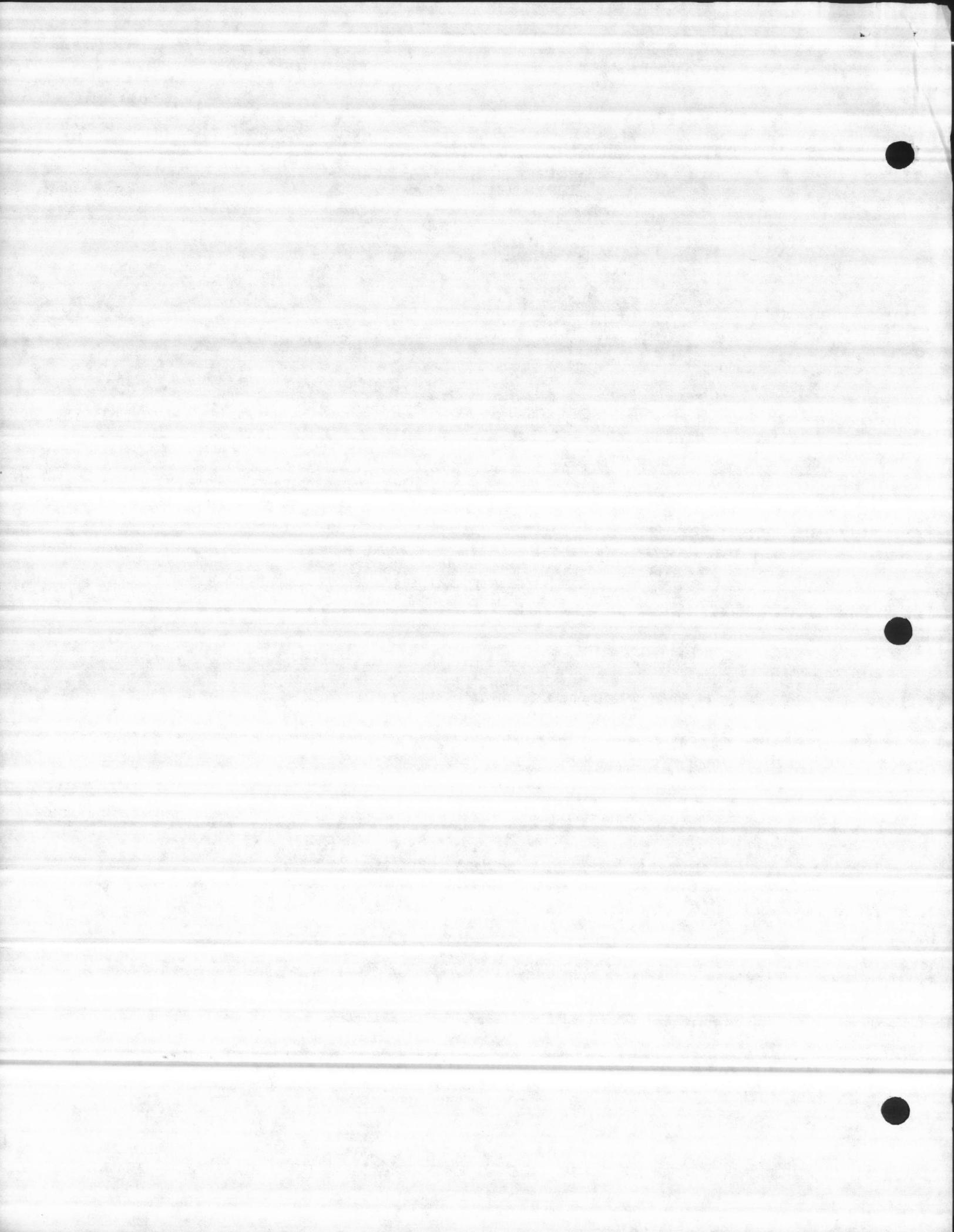
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



INTRODUCTION :

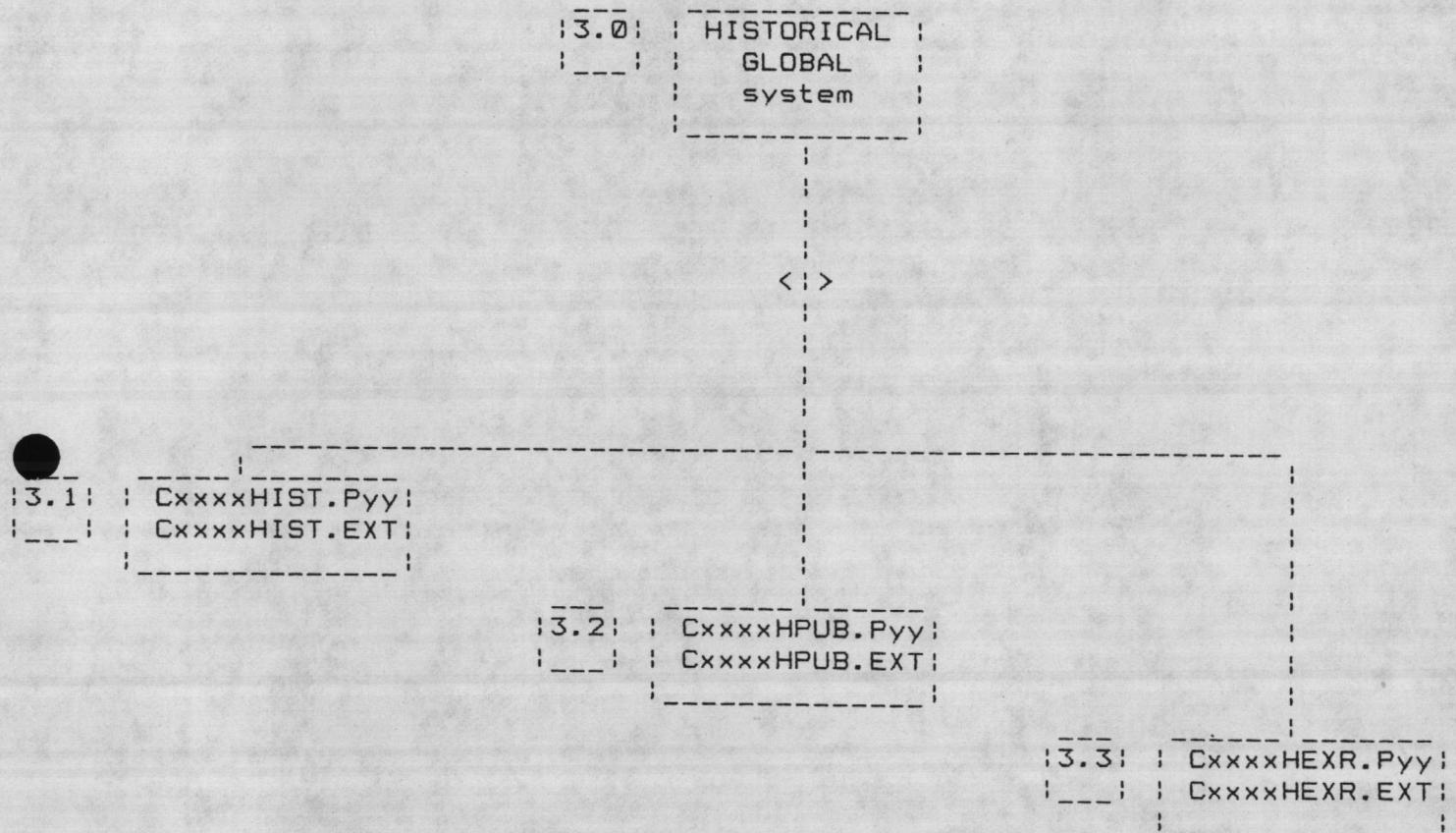
=====  
Historical Global and historical external files hold all the public and external variables and structures

DATE : Oct 10, 1985

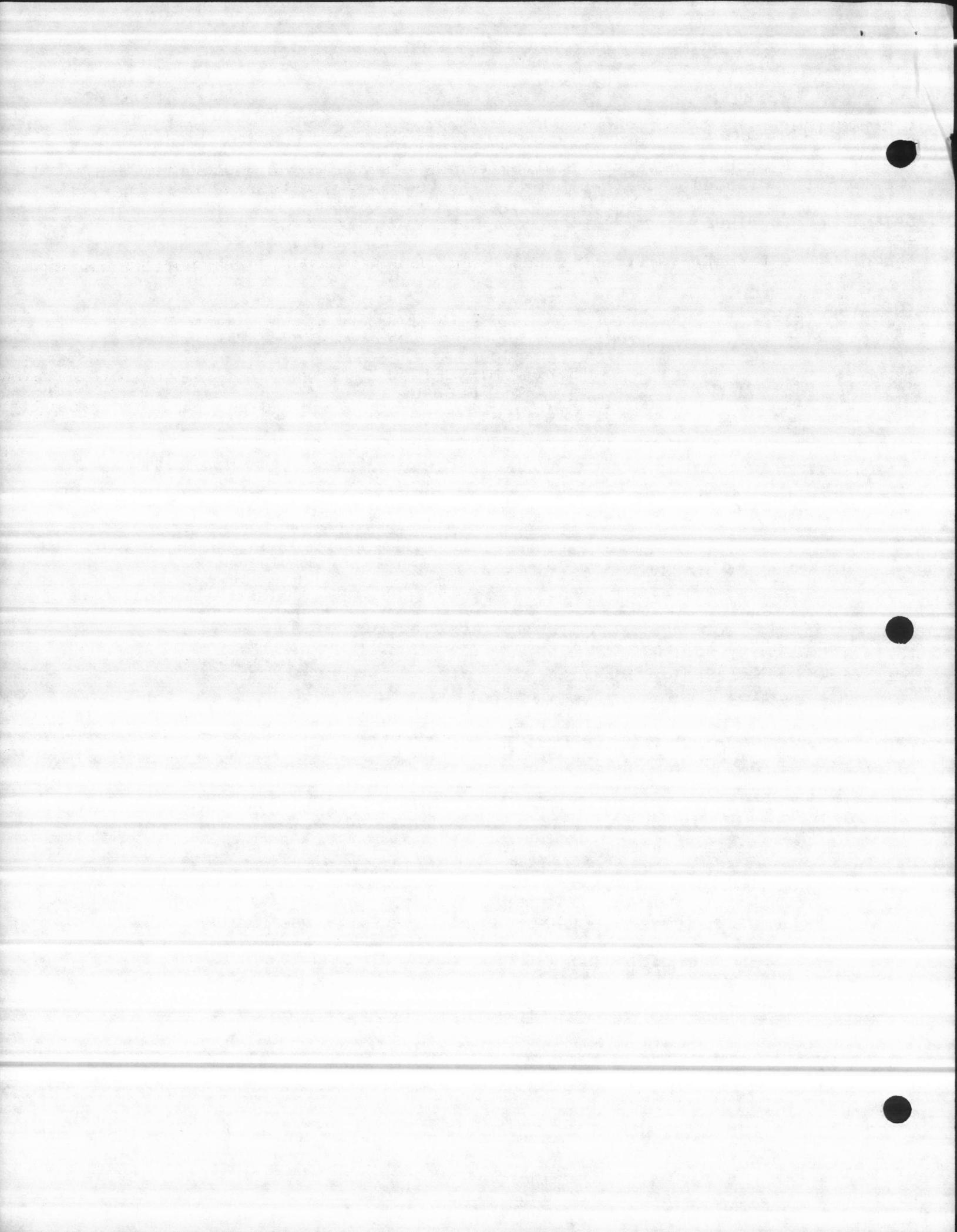
=====  
The glob & external - written by Bob Ryan, Mohamed Fayad, Peter Wollenzien.

HISTORICAL SYSTEM :

=====



Where Cxxxx = C4758 & Pyy = P86



(3.1) MODULE NAME : CxxxxHIST.Pyy & CxxxxHIST.EXT

=====

DESCRIPTION :

=====

The module CxxxxHIST.Pyy holds all the historical public data structures, and variables of the following and the module CxxxxHIST.EXT holds all the external data of the CxxxxHIST.Pyy :

- 1) HISTORICAL DISK HEADER DECLARATION.
- 2) HISTORICAL SAVE TIME.
- 3) DIFFERENT HISTORICAL VARIABLES
- 4) HISTORICAL PASSWORD SYSTEM
- 5) HISTORICAL ALARMS STRUCTURES.
- 6) HISTORICAL OPTIONS
- 7) HISTORICAL GROUP DISPLAY.
- 8) HISTORICAL I O RACK STRUCTURE.
- 9) HISTORICAL REPORT STRUCTURE.
- 10) HISTORICAL SYSTEM CONDITION STRUCTURE.
- 11) HISTORICAL DISCRETE INPUT.
- 12) HISTORICAL DISCRETE OUTPUT.
- 13) HISTORICAL ANALOG INPUT.
- 14) HISTORICAL ANALOG OUTPUT.

=====

= HISTORICAL DISK HEADER DECLARATION =

=====

historical\_header\_info STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

historical\_header\_data STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

= HISTORICAL SAVE TIME =

historical\_save\_time => two dword hold the historical save time.

= DIFFERENT HIST VARIABLES =

hist\_reg\_t => a token used for historical region protection.

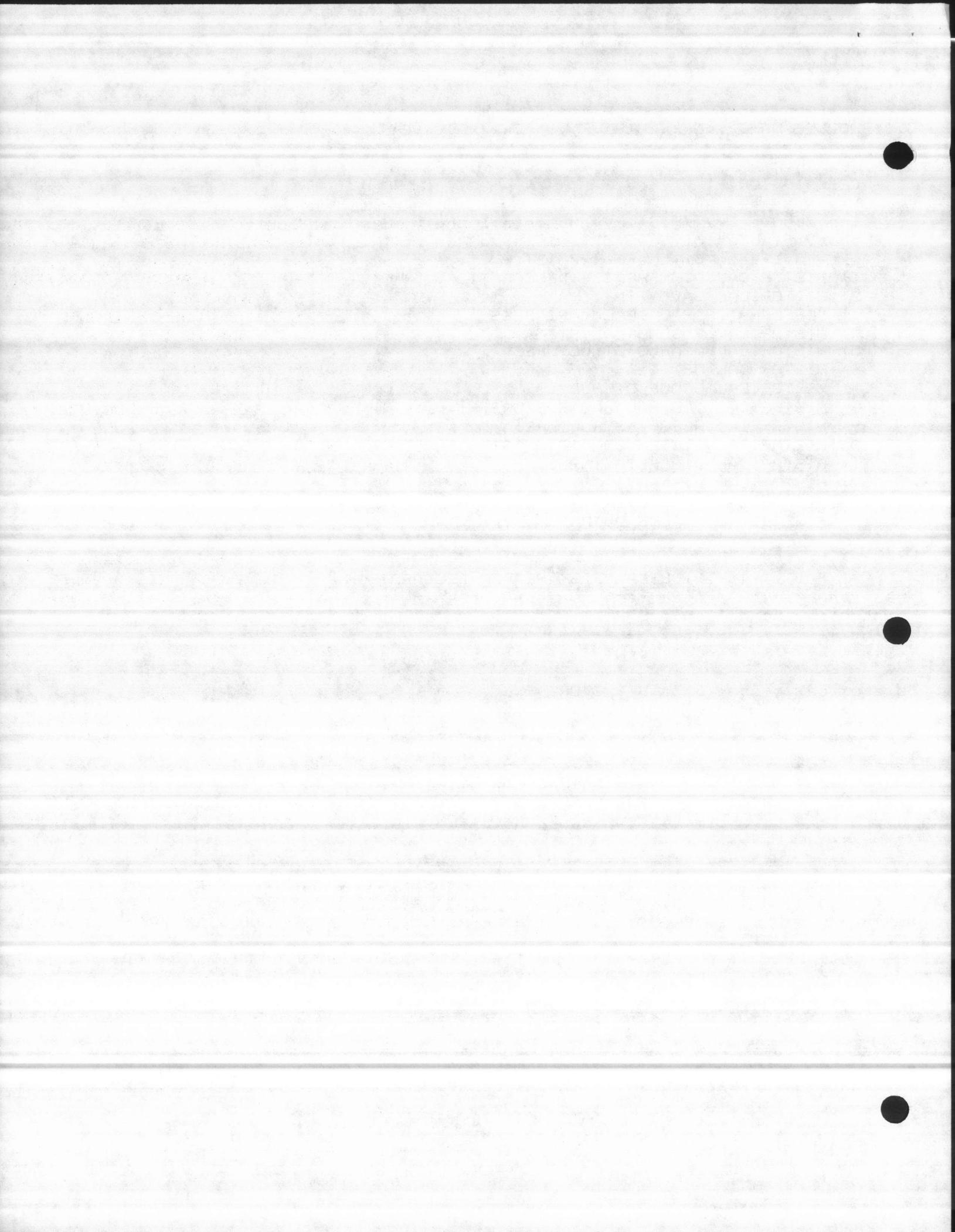
hist\_ram\_busy => a word used as a flag if set that means the ram is busy otherwise ram is not busy.

hist\_file\_name => 40 bytes to hold any historical file name.

= HIST PASSWORD =

password STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2



=====

HIST ALARMS STRUCTURES

=====

h\_alarm\_disp STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

h\_dialers STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

h\_sonalert\_struct STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

=====

HIST SYSTEM OPTIONS

=====

h\_options:

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

=====

HIST GROUP DISPLAY

=====

h\_group\_disp STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

=====

HIST IO PORTS

=====

h\_io\_ports STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

=====

HIST REPORT DISPLAY

=====

h\_report\_disp STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

=====

HIST CONDITION STRING TABLE

=====

h\_cond\_str\_table STRUCTURE:

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

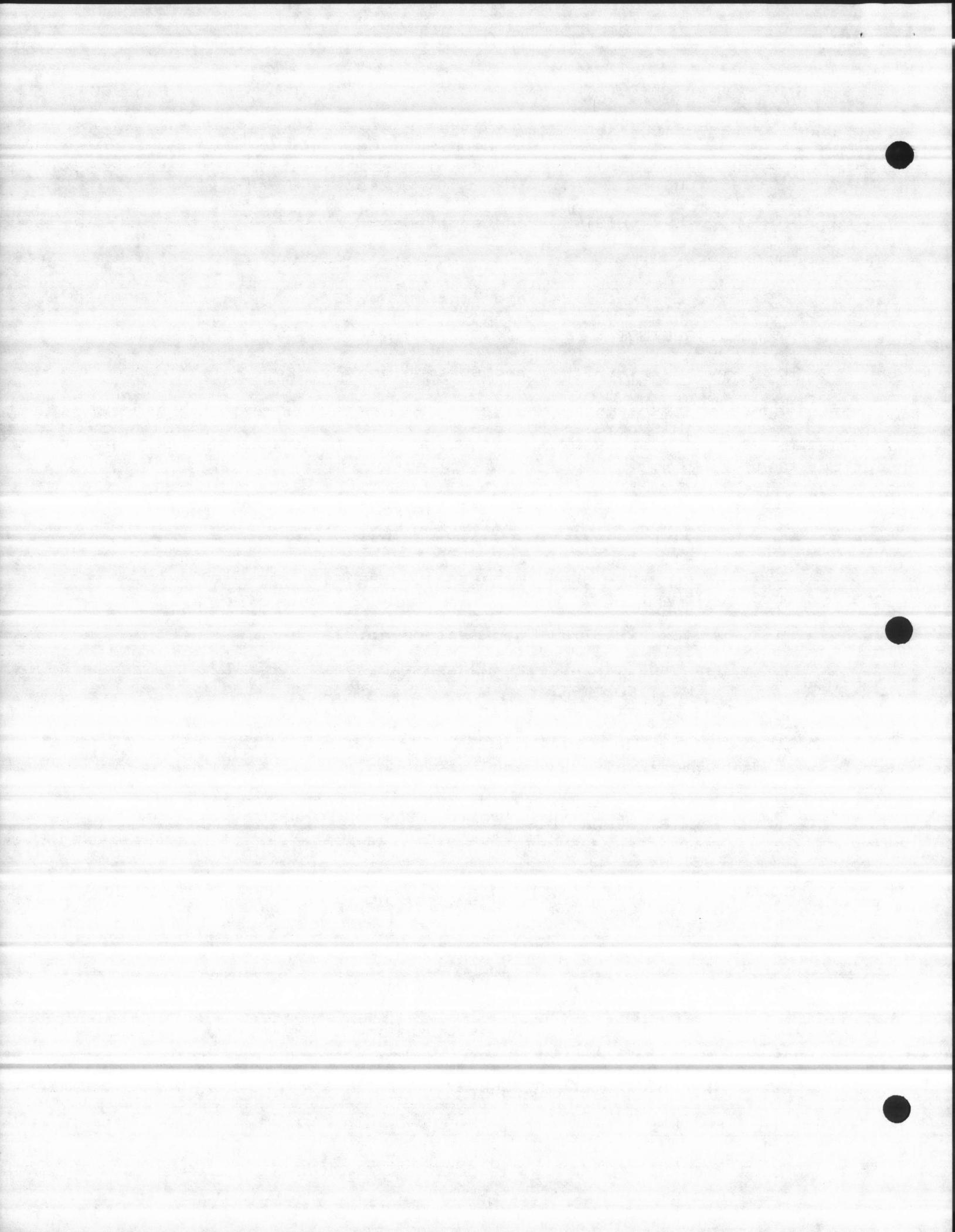
=====

HISTORICAL DISCRETE INPUT

=====

h\_discrete\_in STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2



HISTORICAL DISCRETE OUTPUT

h\_discrete\_out STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

HISTORICAL ANALOG INPUT

hist\_analog\_in STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

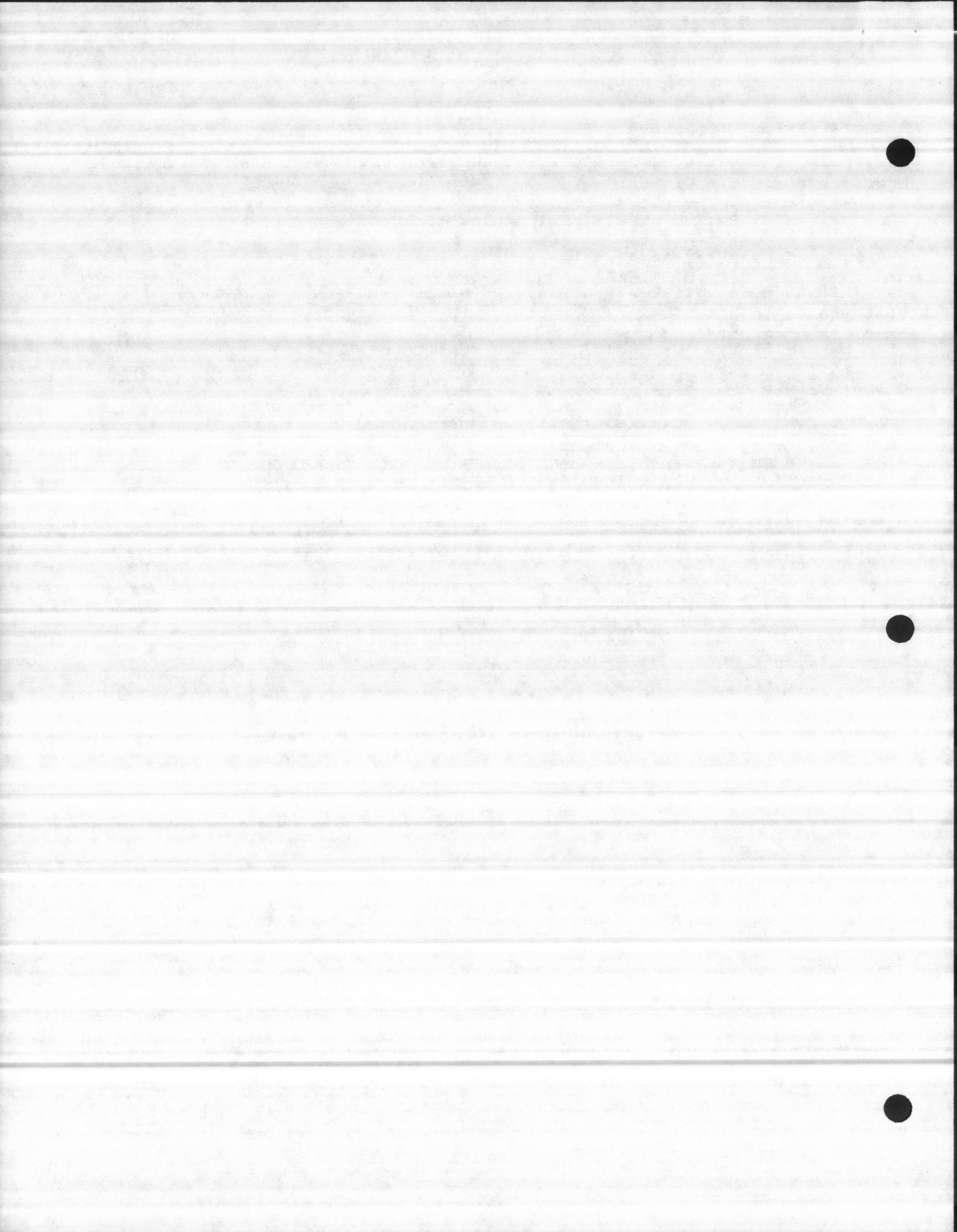
h\_analog\_in STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2

HISTORICAL ANALOG OUTPUT

h\_analog\_out STRUCTURE :

See CxxxxHIST.Pyy, CxxxxHIST.EXT and section no. 2



(3.2) MODULE NAME : CxxxxHPUB.Pyy & CxxxxHPUB.EXT

=====

DESCRIPTION :

=====

The CxxxxHPUB.Pyy module holds all the historical public data structures, and variables & the CxxxxHPUB.EXT holds all the externals for the CxxxxHPUB.Pyy of the following :

TREND DATA STORAGE AND

RETRIEVAL STRUCTURES.

Note that - all the data structure in this section holding the historical trends for analog input only.

=====

= HISTORICAL TREND DATA STORAGE =

=====

h\_two\_min\_trend\_data STRUCTURE :

-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2

h\_one\_hour\_trend\_data STRUCTURE :

-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2

h\_two\_hour\_trend\_data STRUCTURE :

-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2

h\_day\_trend\_buffer :

-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2

h\_week\_trend\_buffer STRUCTURE :

-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2

h\_month\_trend\_buffer STRUCTURE :

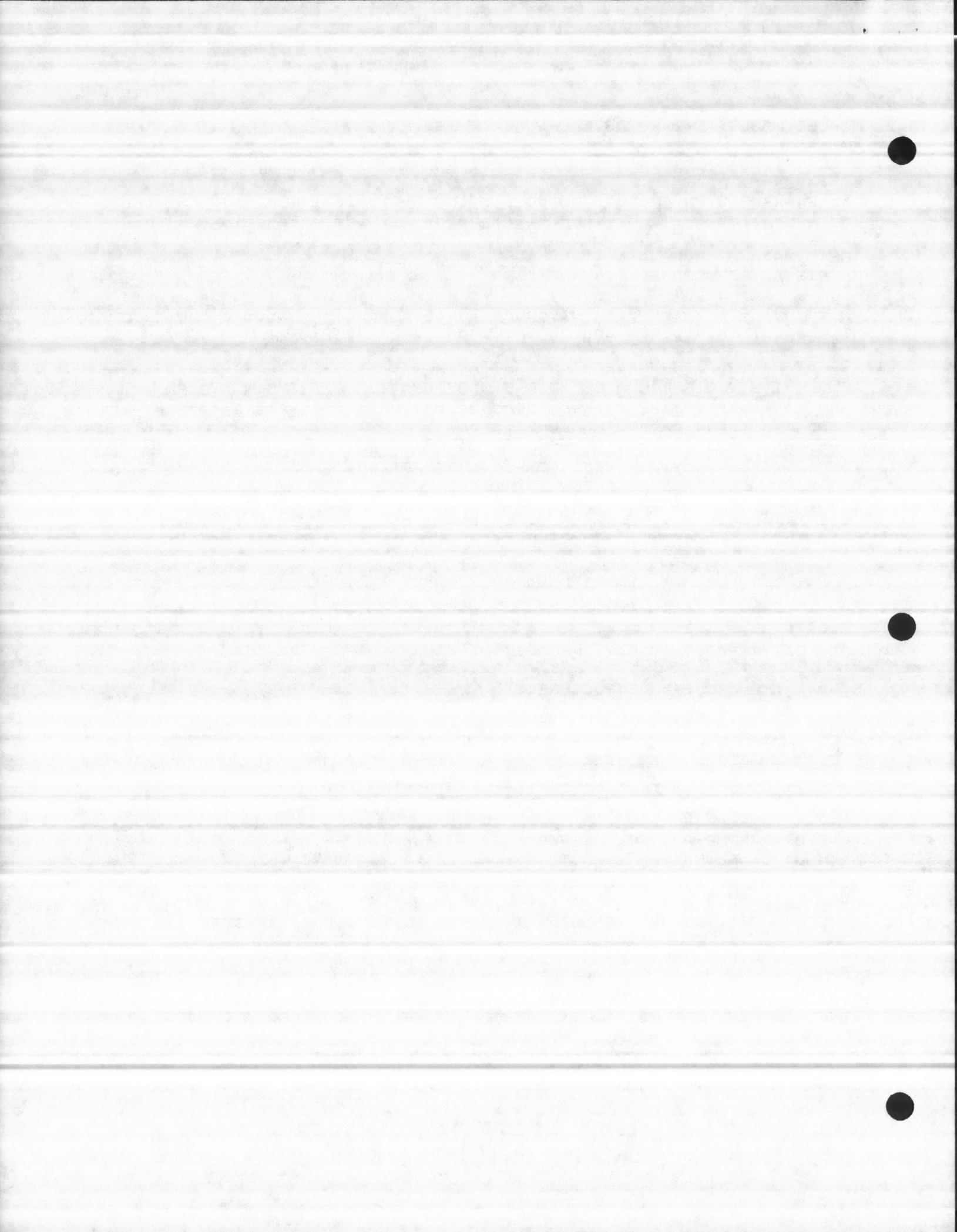
-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2

h\_ahour\_trend\_buffer STRUCTURE :

-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2

h\_fhour\_trend\_buffer STRUCTURE :

-----  
See CxxxxHPUB.Pyy, CxxxxHPUB.EXT and section no. 2



(3.3) MODULE NAME : CxxxxHEXR.Pyy & CxxxxHEXR.EXT

=====

DESCRIPTION :

=====

The CxxxxHEXR.Pyy module holds all the historical public data structures, and variables & the CxxxxHEXR.EXT holds all the externals for the CxxxxHEXR.Pyy of the following :

- 1) HISTORICAL YEARLY ANALOG STRUCTURE.
- 2) HISTORICAL LEVEL CONTROL STRUCTURE.
- 3) HISTORICAL CONTROL STRUCTURE.
- 4) HISTORICAL STEP PUMPS STRUCTURE.

=====

= HISTORICAL YEARLY ANALOG =

=====

h\_yearly\_scale => 30 words for yearly scale.

h\_yearly\_analog STRUCTURE :

See CxxxxHEXR.Pyy, CxxxxHEXR.EXT and section no. 2

=====

= HISTORICAL LEVEL CONTROL =

=====

h\_level\_control STRUCTURE :

See CxxxxHEXR.Pyy, CxxxxHEXR.EXT and section no. 2

=====

= HISTORICAL CONTROL =

=====

h\_control STRUCTURE :

See CxxxxHEXR.Pyy, CxxxxHEXR.EXT and section no. 2

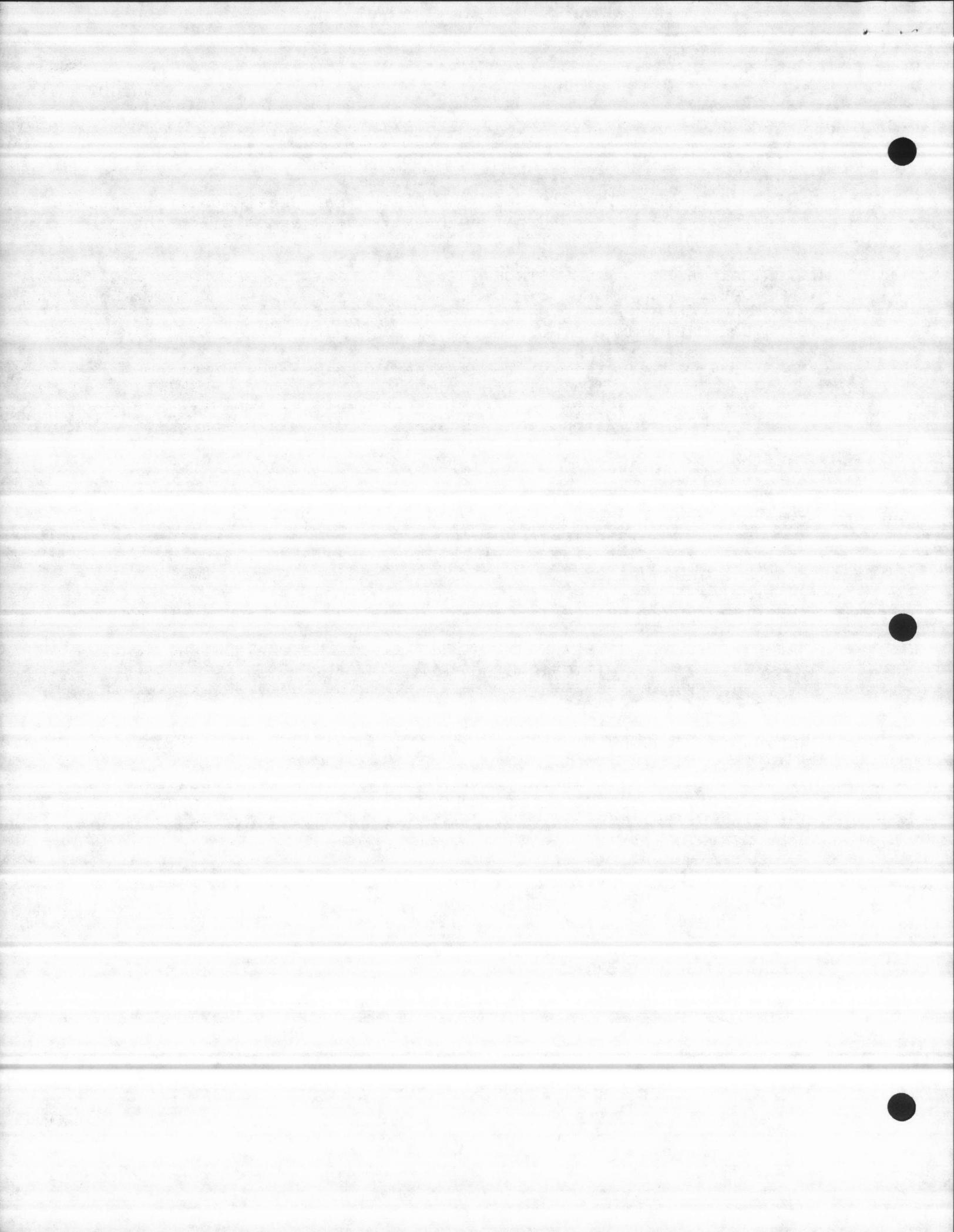
=====

= HISTORICAL STEP PUMPS =

=====

h\_step\_pumps STRUCTURE :

See CxxxxHEXR.Pyy, CxxxxHEXR.EXT and section no. 2



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 4

I N I T I A L I Z A T I O N

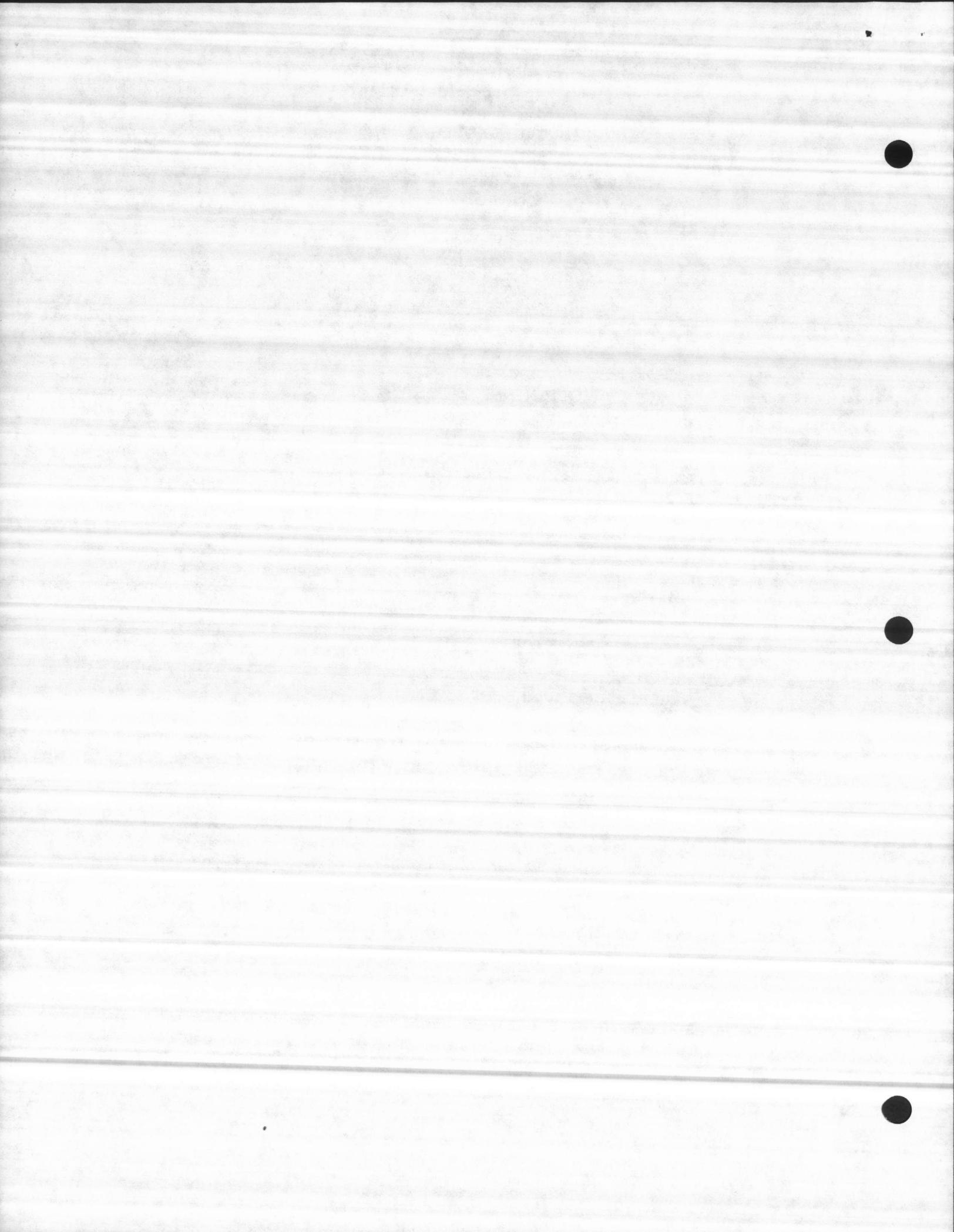
BY

Mohamed E. Fayad

REVIEWED

BY

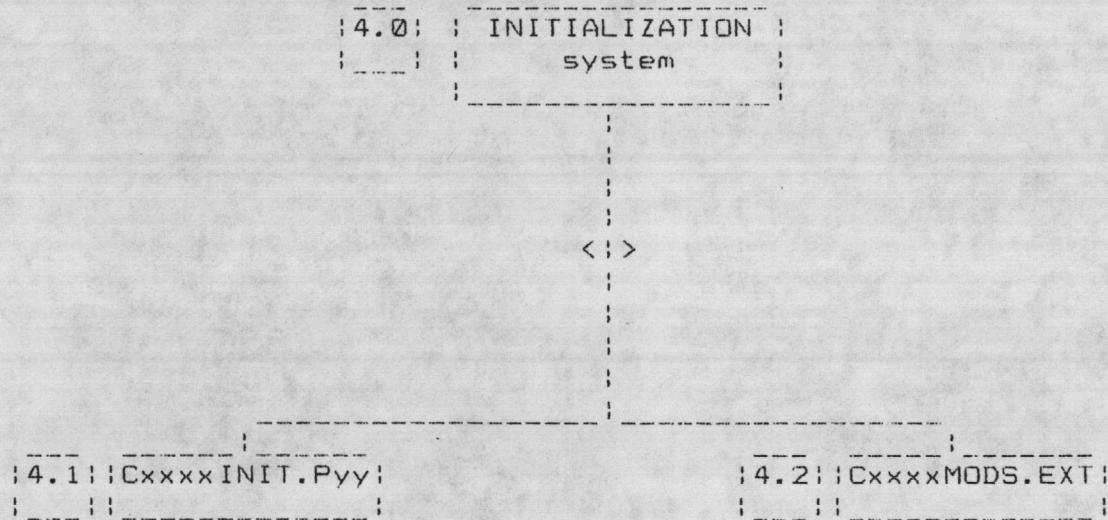
Jerry Knoth



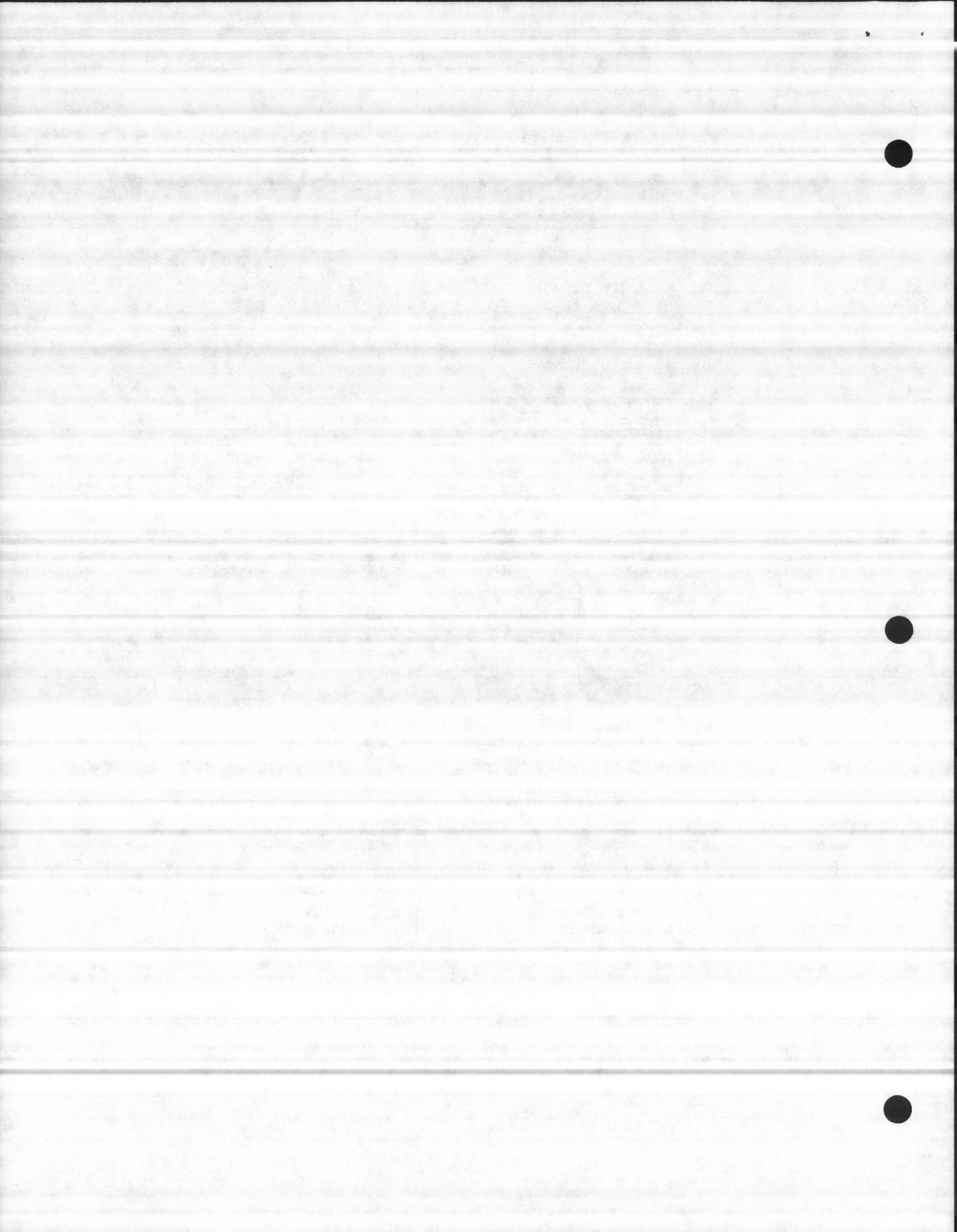
DATE : Oct 10, 1986

----  
T initialization - written by Bob Ryan, Mohamed Fayad.

INITIALIZATION :



Where Cxxxx = C4758  
&  
Pyy = P86



(4.1) MODULE NAME : CxxxxINIT.Pyy

=====

DESCRIPTION : This procedure will startup the application code  
===== initialize variables, and create tasks.

EXTERNAL PROCEDURES :

=====

Init\_Trend: PROCEDURE (trend\_type, name\_type, existence\_check\_flag)  
EXTERNAL;

DECLARE (trend\_type,  
name\_type,  
existence\_check\_flag) BYTE;

END Init\_Trend;

Format\_Disk\_Error: PROCEDURE (buffer\_t, message\_str\_p, disk\_error,  
disk\_operation) EXTERNAL;

DECLARE (disk\_error, disk\_operation) WORD,  
(buffer\_t) TOKEN,  
(message\_str\_p) POINTER;

END Format\_Disk\_Error;

GLOBAL DATA REFERENCE:

=====

See (CxxxxGLOB.Pyy)

MODULE DECLARATIONS :

=====

count - A byte counter.

counter - A word counter.

max\_num\_crt - Maximum number of CRT in the system.

crt1\_macro\_name - File name for the crt 1 macro.

crt1\_device\_name - RMX86 device name for crt 1.

crt\_macros (2) - Array of pointers to the crt macro names.

crt\_devices (2) - Array of pointers to the crt device names.

fail\_task\_t - RMX86 task token for the fail task.

keyboard\_task\_t - RMX86 task token for the CRT 1 keyboard task.

anin\_task\_t - RMX86 task token for the analog input task.

dcin\_task\_t - RMX86 task token for the discrete input task.

anop\_task\_t - RMX86 task token for the analog output task.

dcop\_task\_t - RMX86 task token for the discrete output task.

print\_report\_task\_t - RMX86 task token for the report generator task.

dump\_data\_base\_task\_t - RMX86 task token for the data base dump task.

input\_rack\_task\_t - RMX86 task token for the input local data task.

output\_rack\_task\_t - RMX86 task token for the output local data task.

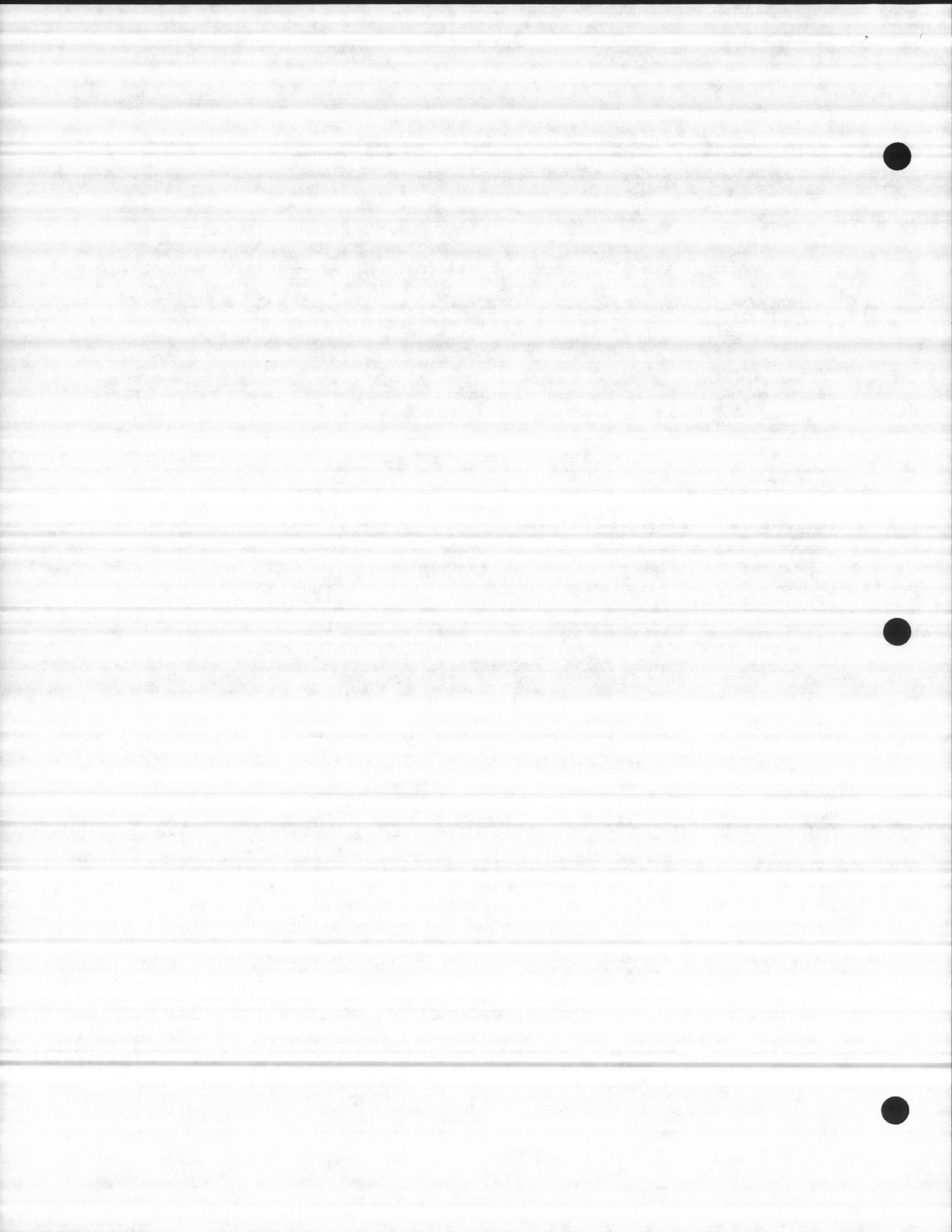
print\_alarmlog\_task\_t - RMX86 task token for the print alarmlog task.

auto\_alarms\_task\_t - RMX86 task token for the auto alarm task.

fifteen\_minute\_task\_t - RMX86 task token for the fifteen minute task.

six\_second\_task\_t - RMX86 task token for the six second task.

thirty\_six\_second\_task\_t - RMX86 task token for the thirty-six second task.



```
two_minute_task_t      - RMX86 task token for the two minute task.
one_hour_task_t        - RMX86 task token for the one hour task.
two_hour_task_t        - RMX86 task token for the two hour task.
end_of_day_task_t     - RMX86 task token for the end of day task.
output_task_t          - RMX86 task token for the general data base
                        output task.
disk_task_t            - RMX86 task token for the disk and tape task.
period_report_task_t   - RMX86 task token for the periodic report task.
control_task_t         - RMX86 task token for the control task.
pump_control_task_t    - RMX86 task token for the pump control task.

temp_word               - Temporary word variable.
exception               - Word variable used for RMX86 exception error
                        handling.

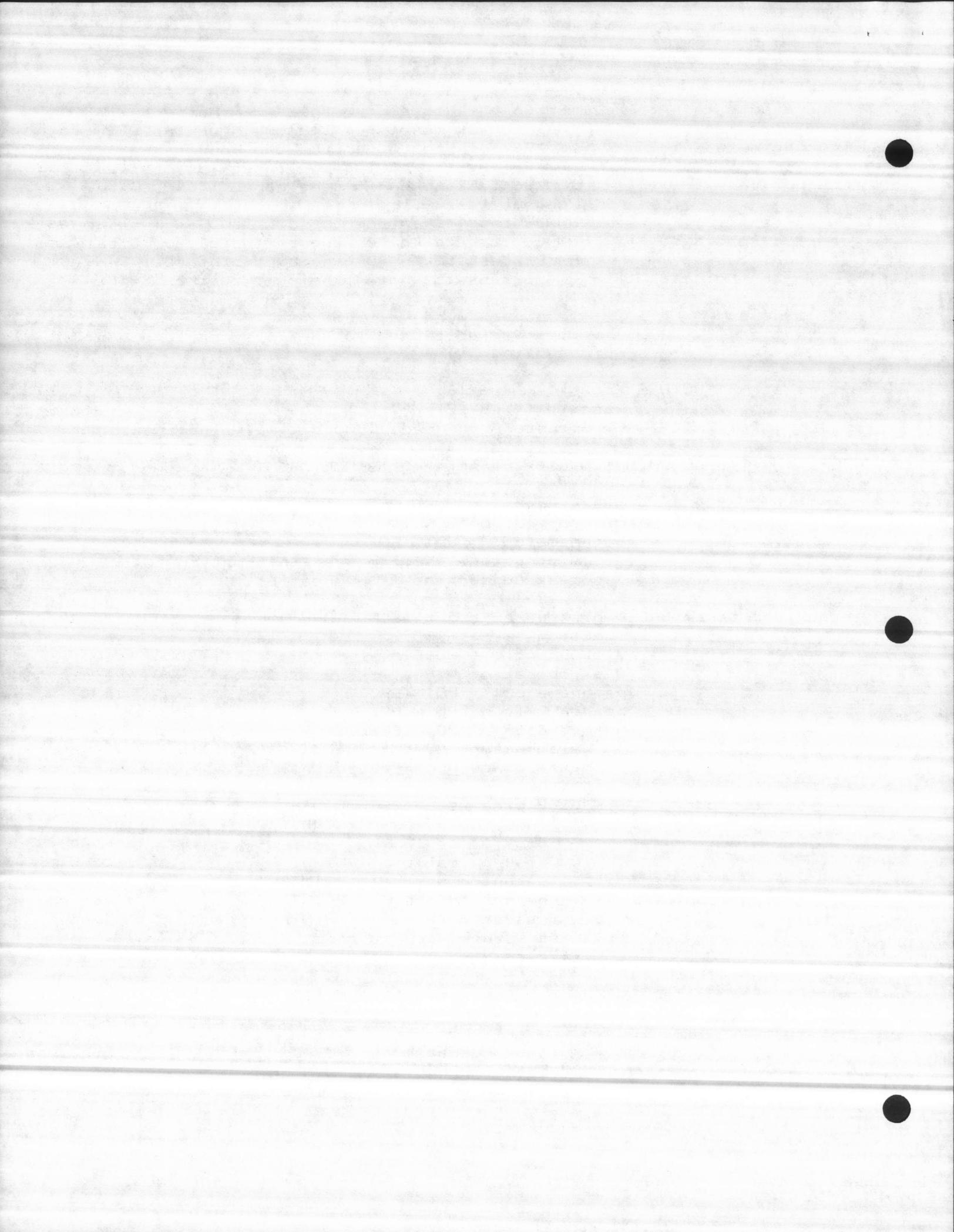
time_str STRUCTURE (count      BYTE,
                     ascii (19) BYTE)
                     - Structure to hold a dword time value in ascii.

create_file_param BASED disk_param_seg_t STRUCTURE (
                     function      WORD,
                     disk_error    WORD,
                     disk_operation WORD,
                     pathname_p   POINTER)
                     - A structure used to send information to the disk
                       task concerning file creation.

read_file_param BASED disk_param_seg_t) STRUCTURE (
                     function      WORD,
                     disk_error    WORD,
                     disk_operation WORD,
                     pathname_p   POINTER,
                     header_info_p POINTER,
                     header_data_p POINTER);
                     - A structure used to send information to the disk
                       task concerning reading a file.

write_file_param BASED disk_param_seg_t STRUCTURE (
                     function      WORD,
                     disk_error    WORD,
                     disk_operation WORD,
                     pathname_p   POINTER,
                     header_info_p POINTER,
                     header_data_p POINTER);
                     - A structure used to send information to the disk
                       task concerning writing a file.

rename_file_param BASED disk_param_seg_t STRUCTURE (
                     function      WORD,
                     disk_error    WORD,
                     disk_operation WORD,
                     pathname_p   POINTER,
                     new_pathname_p POINTER);
                     - A structure used to give a different name to
                       an existing file.
```



```

add_serial_param BASED disk_param_seg_t STRUCTURE (
    function          WORD,
    disk_error        WORD,
    disk_operation    WORD,
    pathname_p        POINTER,
    header_data_p    POINTER,
    data_info_p       POINTER,
    num_elements     WORD);
- A structure used to send information to the disk
task concerning writing an ALOG file.

```

```

alarm_log_header  STRUCTURE
    (file_type      WORD,
     element_size   WORD,
     num_elements   WORD);
- A structure used as a small header by add serial function

```

```

alarm_log_disk_data (2)  STRUCTURE
    (time_dword     DWORD,
     fail_type      BYTE,
     index          WORD,
     condition      WORD,
     analog_flag    BYTE,
     value          WORD,
     limit          WORD,
     units          WORD,
     scale          WORD,
     crit_flag      BYTE,
     alarm_flag     BYTE);
- A structure used in the alarm log & fail procedures

```

#### LOCAL PROCEDURES

---

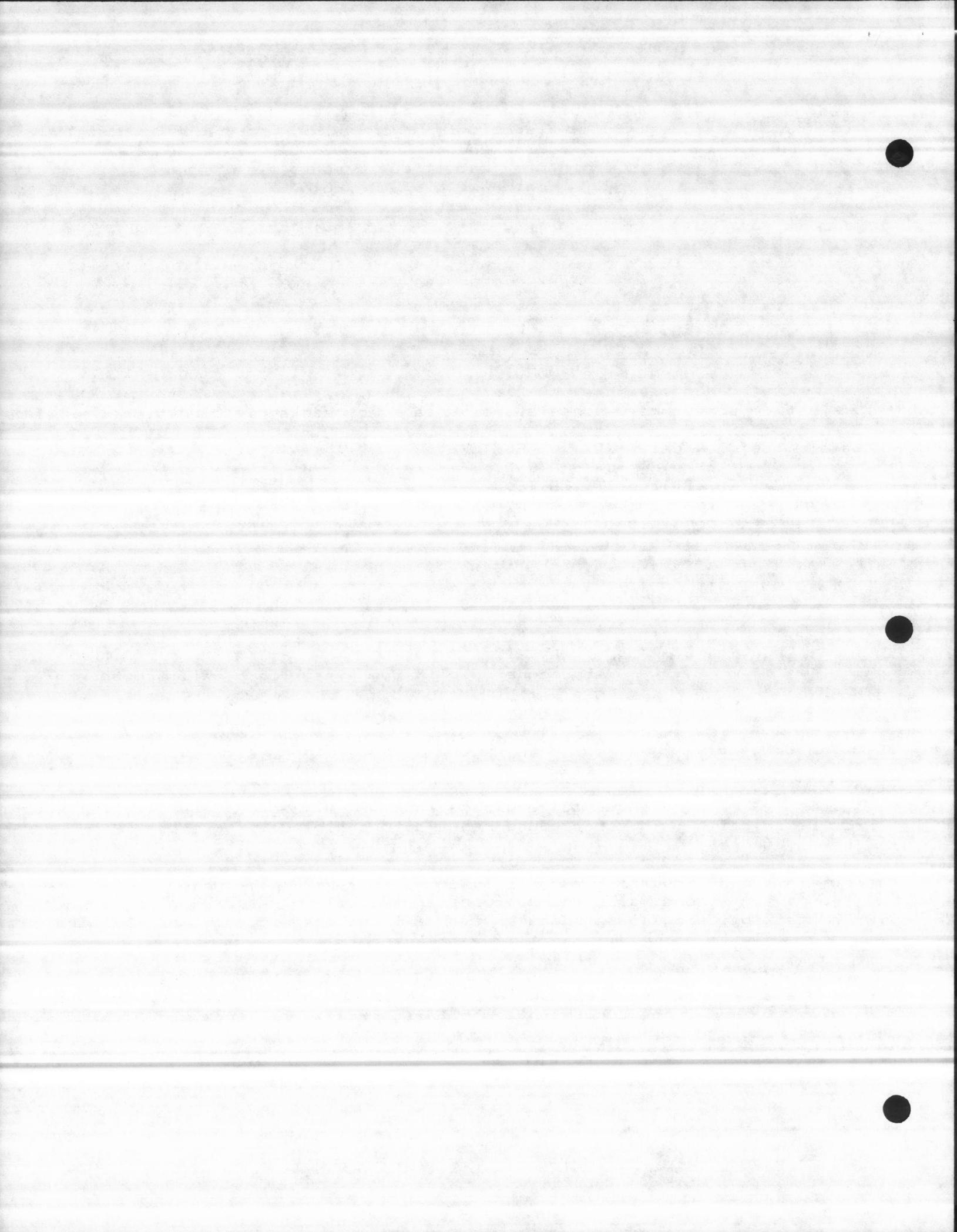
```

Level_Control_init : PROCEDURE;
    count => A word value to keep pump index.
    Proc Description : This procedure reset auto and manual requires.
END Level_Control_Init;

Control_init : PROCEDURE;
    count => A word value to keep control index.
    Proc Description : This procedure reset all the steps time_count & status.
END Control_Init;

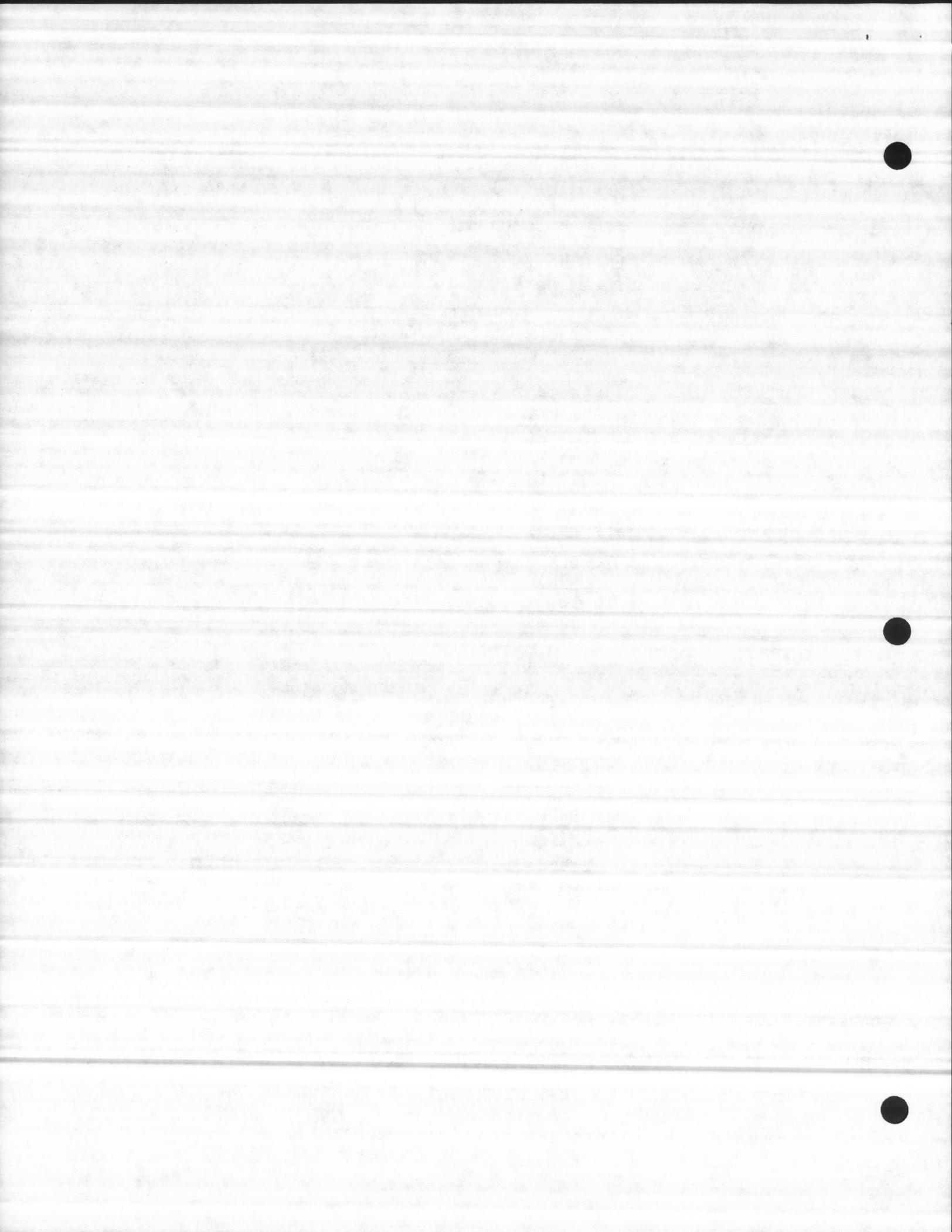
Disk_Header_Proc: PROCEDURE;
    Proc Description: This procedure will initialize the disk headers for
                      the current data and historical data bases.
END Disk_Header_Proc;

```

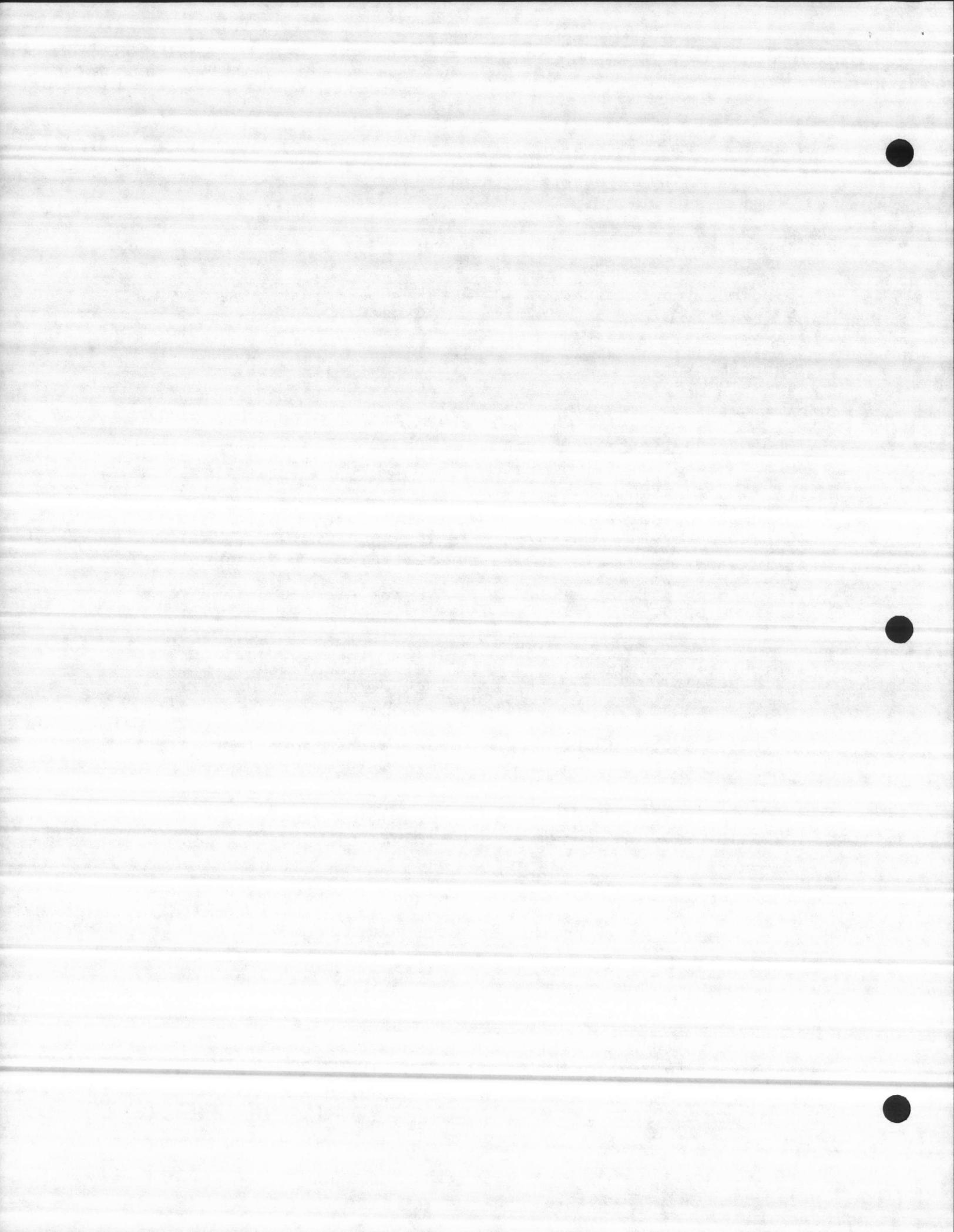


## MODULE FLOW :

```
=====
| Initialize |-----+
|             |   CALL Init$Exception
|             |   CALL Init$W1300
|             |   CALL Init$Clock
|             |   CALL Init$Update
|             |   CALL Init$8087
|             |   CALL Init$Printer
|             |   CALL Init$Console
|
|             |-----+
|             |   Create Regions
|             |
|             |-----+
|             |   Create Buffers
|             |
|             |-----+
|             |   Create Segments
|             |
|             |-----+
|             |   Create Mail Boxes
|             |
|             |-----+
|             |   Create Semaphores
|             |
|             |-----+
|             |   Create Lists
|             |
|             |-----+
|             |   CALL Disk_Header
|             |
|             |-----+
|             |   Create Disk Task
|             |
|             |-----+
|             |   Send Power Up Msg
|             |
|             |-----+
|             |   Create Hist Dir
|             |
|             |-----+
|             |   Read PowerUp Files
|             |
|             |-----+
|             |   Initial Time msg
|             |
|             |-----+
|             |   Init. trend files
|             |
|             |-----+
|             |   Initial ALOG File
|             |
|             |-----+
|             |   Initial lvl & ctrl
|             |
|             |-----+
```



Rest trend data	
Create all the Tasks	
Discrete input Task   CxxxxDCIN.Pyy	Keyboard Task   CxxxxKYBD.Pyy
Input Rack task   CxxxxIORK.Pyy	Output Rack Task   CxxxxIORK.Pyy
Analog InputTask   CxxxxANIN.Pyy	Analog Output Task   CxxxxANOP.Pyy
Discrete Output task   CxxxxDCOP.Pyy	Fail Task   CxxxxFAIL.Pyy
Print Report Task   CxxxxPRTG.Pyy	Print Alarm Log Task   CxxxxALOG.Pyy
Dump Data Base Task   CxxxxPBAS.Pyy	Six Second Task   CxxxxTCTL.Pyy
Thirty Six Sec. Task   CxxxxTCTL.Pyy	Fifteen Minute Task   CxxxxTCTL.Pyy
Two Minute Task   CxxxxTCTL.Pyy	One Hour Task   CxxxxTCTL.Pyy
Two Hour Task   CxxxxTCTL.Pyy	End Of Day Task   CxxxxTCTL.Pyy
Auto Alarm Task   CxxxxFAIL.Pyy	Period Report Task   CxxxxPERD.Pyy
Control Task   CxxxxCTRL.Pyy	Pump Control Task   CxxxxPCON.Pyy



Algorithm :

=====

Initialize the RMX86 exception handler

Initialize the telemetry system.

Initialize the Clock display system.

Initialize the crt update system.

Initialize the 8087 math coprocessor.

Startup the printer macros and attach to the correct devices.

Startup the crt macros and attach to the correct devices.

Create the crt objects, regions and buffers.

Create the printer objects, buffers.

Create the disk system objects, regions, mailboxes, buffers and parameter segments.

Create the report system objects, regions, mailboxes, buffers and parameter segments.

Create The historical region token.

Create the alarm system objects, regions, and the update list.

Create the fail interface objects, fail ring buffers and semaphores.

Create the total objects, update list, totalization region, and trending region.

Create the io lists for the local I/O rack and the regions for data base input control.

Initialize the non-generic disk system. Create the disk task.

Send pwrup message.

Create the historical file directory. Report any errors.

Read power-up files CxxxxPWRUP2, CxxxxPWRUP1 and report any errors.

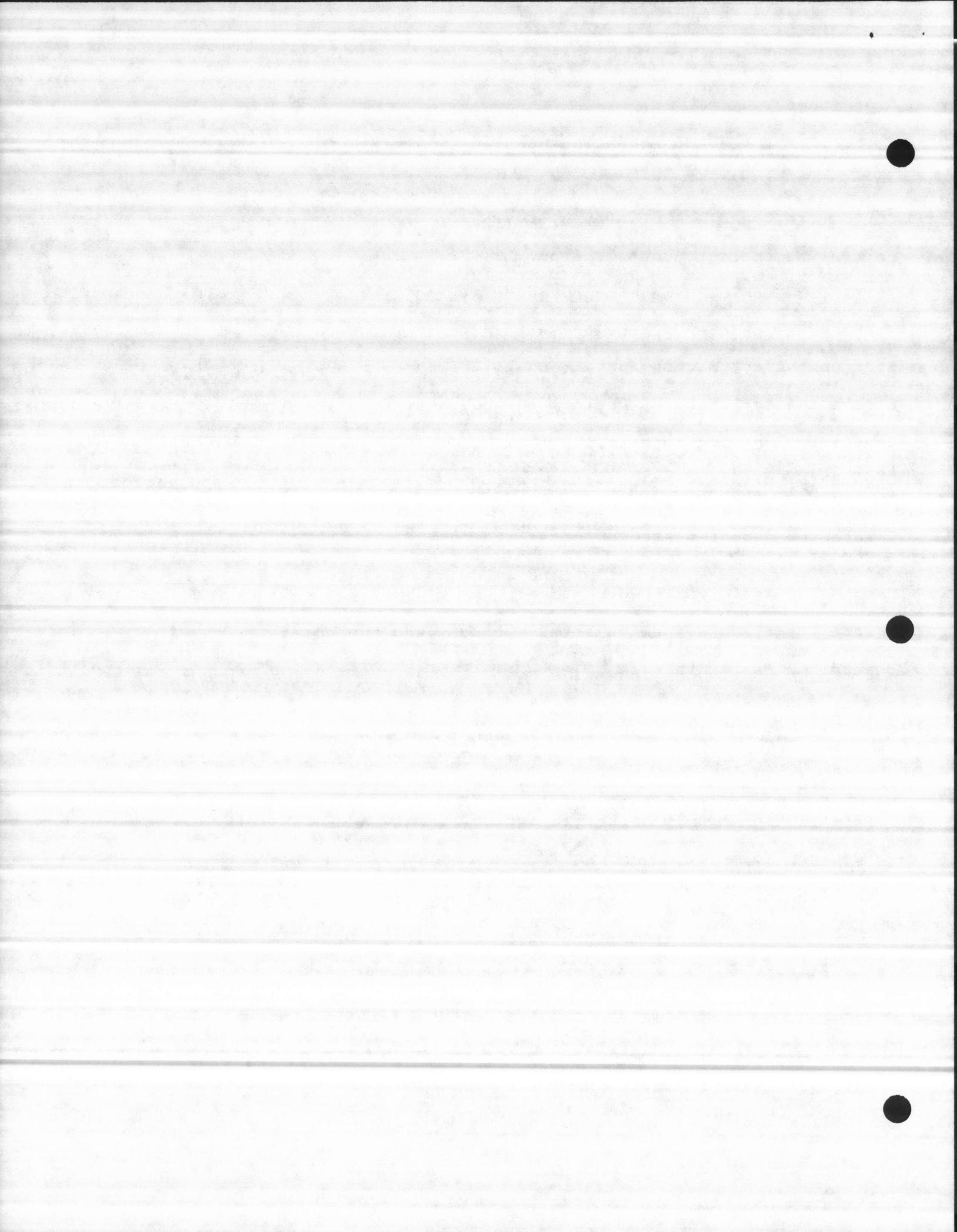
Set initial time as read from the powerup file.

Reset historical save time to signify no historical file loaded.

Initialize the trending files and data structures.

Initialize alarm log file.

Initialize analog\_in status to mark power up state.



Initialize number of condition strings in the system.

Reset all temporary trend data.

Call special procedures to initialize new level control & control usage.

Create discrete input tasks.

Create the keyboard task.

Create the input data from local rack task.

Create the output data base to local rack task.

Create the analog output task.

Create the analog input task.

Create the discrete output task.

Create the fail alarm detection task.

Create the print report generation task.

Create the print alarm log task.

Create the dump data base task.

Create the six second task.

Create the thirty six second task.

Create the fifteen minute task.

Create the two minutes task.

Create the one hour task.

Create the two hour task.

Create the end of day task.

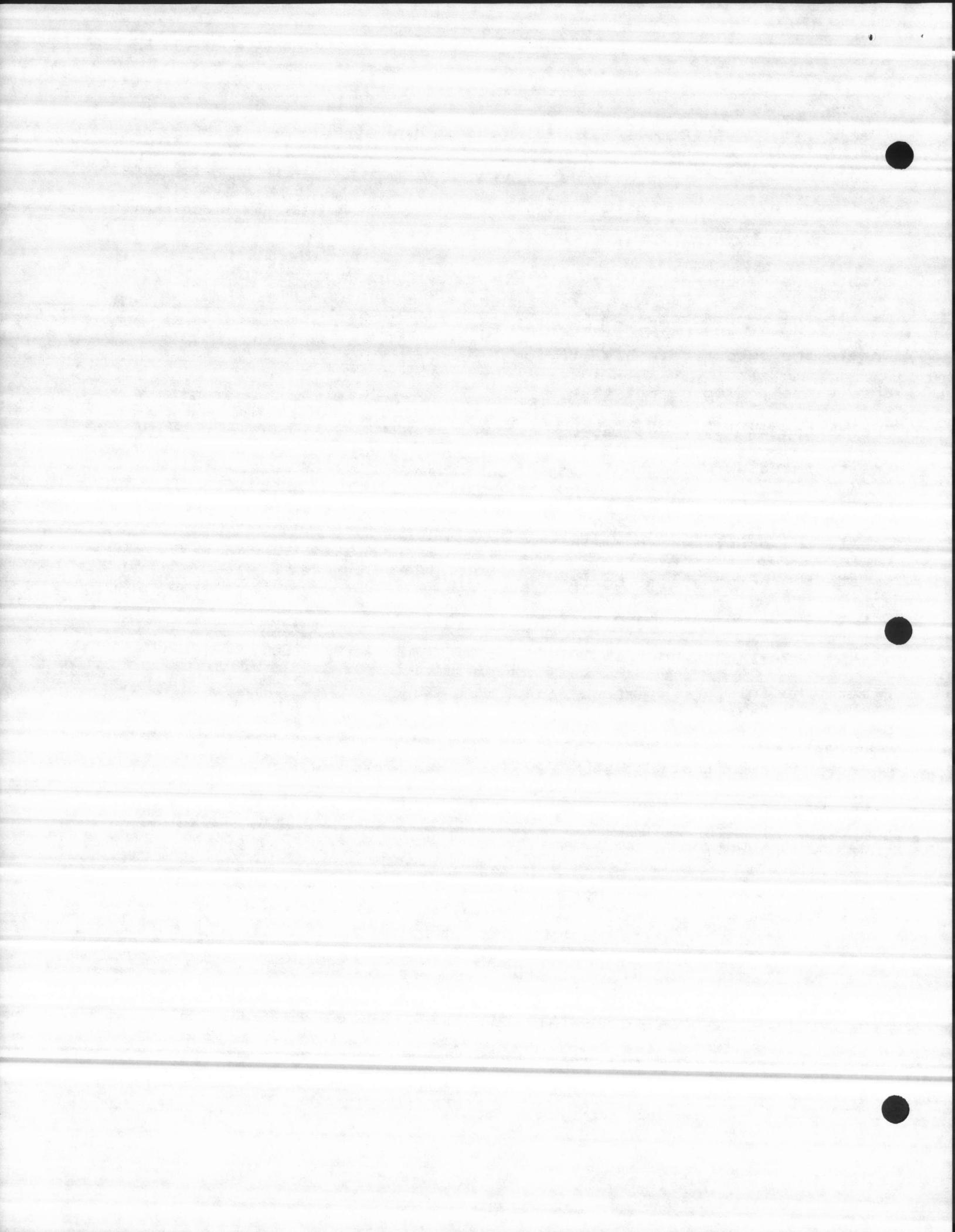
Create the auto alarms task.

Create the periodic report task.

Create the control task.

Create the pump control task.

Delete the initialization task.



(4.2) MODULE NAME : CxxxxMODS.EXT  
=====

DESCRIPTION : This module contains all the external tasks needed  
===== for CxxxxINIT.Pyy.

EXTERNAL PROCEDURES :

=====

Keyboard: PROCEDURE EXTERNAL; /\* CxxxxKYBD \*/  
END Keyboard;

Disk\_Task: PROCEDURE EXTERNAL; /\* CxxxxDISK \*/  
END Disk\_Task;

Fail\_Task: PROCEDURE EXTERNAL; /\* CxxxxFAIL \*/  
END Fail\_Task;

Auto\_Alarms\_Task: PROCEDURE EXTERNAL; /\* CxxxxFAIL \*/  
END Auto\_Alarms\_Task;

Analog\_Input\_Task: PROCEDURE EXTERNAL; /\* CxxxxANIN \*/  
END Analog\_Input\_Task;

Discrete\_Input\_Task: PROCEDURE EXTERNAL; /\* CxxxxDCIN \*/  
END Discrete\_Input\_Task;

Analog\_Output\_Task: PROCEDURE EXTERNAL; /\* CxxxxANOP \*/  
END Analog\_Output\_Task;

Discrete\_Output\_Task: PROCEDURE EXTERNAL; /\* CxxxxDCOP \*/  
END Discrete\_Output\_Task;

Input\_Rack\_Task: PROCEDURE EXTERNAL; /\* CxxxxIORK \*/  
END Input\_Rack\_Task;

Output\_Rack\_Task: PROCEDURE EXTERNAL; /\* CxxxxIORK \*/  
END Output\_Rack\_Task;

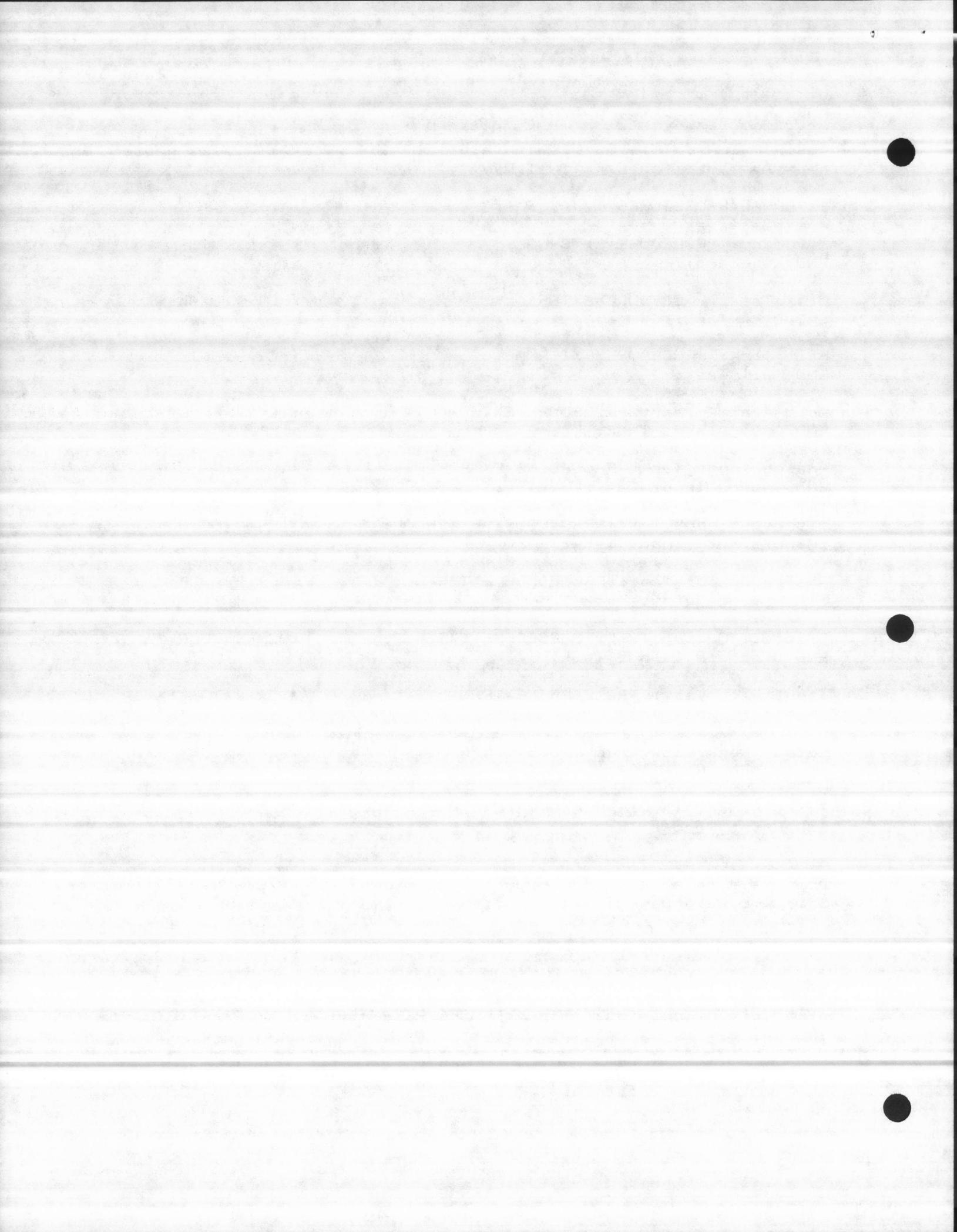
Print\_Report\_Task: PROCEDURE EXTERNAL; /\* CxxxxPRTG \*/  
END Print\_Report\_Task;

Dump\_Data\_Base\_Task: PROCEDURE EXTERNAL; /\* CxxxxDUMP \*/  
END Dump\_Data\_Base\_Task;

Six\_Second\_Total\_Task: PROCEDURE EXTERNAL; /\* CxxxxTOTL \*/  
END Six\_Second\_Total\_Task;

Thirty\_Six\_Second\_Total\_Task: PROCEDURE EXTERNAL; /\* CxxxxTOTL \*/  
END Thirty\_Six\_Second\_Total\_Task;

Two\_Minute\_Total\_Task: PROCEDURE EXTERNAL; /\* CxxxxTOTL \*/  
END Two\_Minute\_Total\_Task;



```
Fifteen_Minute_Total_Task: PROCEDURE EXTERNAL; /* CxxxxTOTL */
END Fifteen_Minute_Total_Task;

One_Hour_Total_Task: PROCEDURE EXTERNAL;           /* CxxxxTOTL */
END One_Hour_Total_Task;

Two_Hour_Total_Task: PROCEDURE EXTERNAL;           /* CxxxxTOTL */
END Two_Hour_Total_Task;

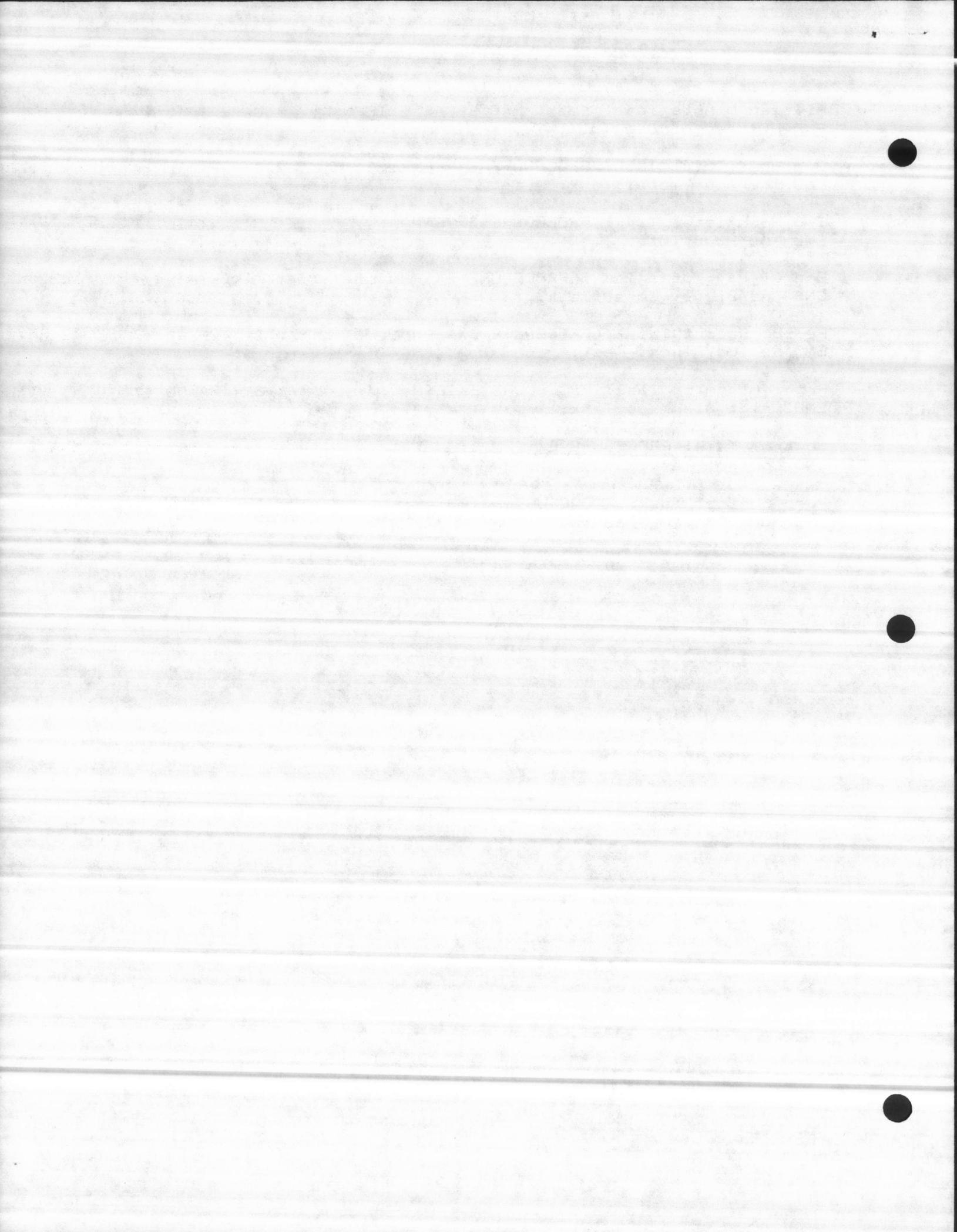
End_Of_Day_Total_Task: PROCEDURE EXTERNAL;         /* CxxxxTOTL */
END End_Of_Day_Total_Task;

Period_Report_Task : PROCEDURE EXTERNAL;           /* CxxxxPERD */
END Period_Report_Task;

Print_Alarmlog_Task: PROCEDURE EXTERNAL;           /* CxxxxALOG */
END Print_Alarmlog_Task;

Control_Task: PROCEDURE EXTERNAL;                  /* CxxxxCTRL */
END Control_Task;

Pump_Control_Task: PROCEDURE EXTERNAL;             /* CxxxxPCON */
END Pump_Control_Task;
```



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 5

K E Y B O A R D

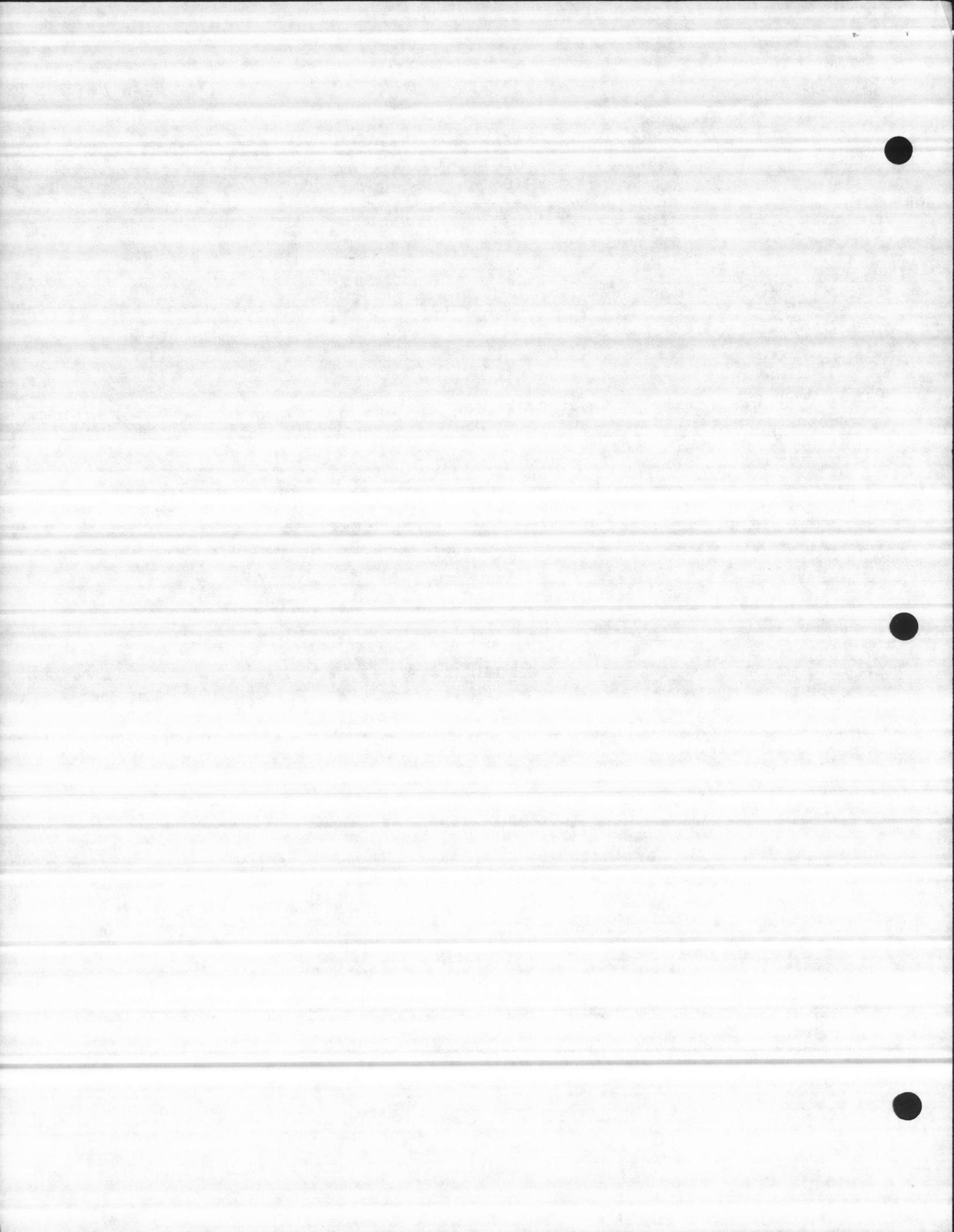
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

The keyboard - written by Bob Ryan, Mohamed Fayad.

(5.0) MODULE NAME : CxxxxKYBD.Pyy

=====

Description : This file contains the major keyboard modules for each of  
===== the three CRTs.

EXTERNAL PROCEDURES :

=====

```
Send_To_Fail: PROCEDURE (fail_type, index)
                EXTERNAL;          /* CxxxxFAIL */
    DECLARE (fail_type) BYTE,
            (index) WORD;
END Send_To_Fail;
```

```
Display_Alarms_Proc: PROCEDURE (crt_num) EXTERNAL;          /* CxxxxALRM */
    DECLARE crt_num BYTE;
END Display_Alarms_Proc;
```

```
Display_Alarmlog_Proc: PROCEDURE (crt_num) EXTERNAL;        /* C4758ALOG */
    DECLARE crt_num BYTE;
END Display_Alarmlog_Proc;
```

```
Keyin_Construct_Sub_Menu: PROCEDURE (crt_num) EXTERNAL; /* CxxxxCRTG */
    DECLARE crt_num BYTE;
END Keyin_Construct_Sub_Menu;
```

```
Keyin_Construct_Report_Sub_Menu: PROCEDURE (crt_num) EXTERNAL;
                                    /* CxxxxREPG */
    DECLARE crt_num BYTE;
END Keyin_Construct_Report_Sub_Menu;
```

```
Keyin_Data_Base_Sub_Menu_Proc: PROCEDURE (crt_num) EXTERNAL;
                                    /* CxxxxDBAS */
    DECLARE crt_num BYTE;
END Keyin_Data_Base_Sub_Menu_Proc;
```

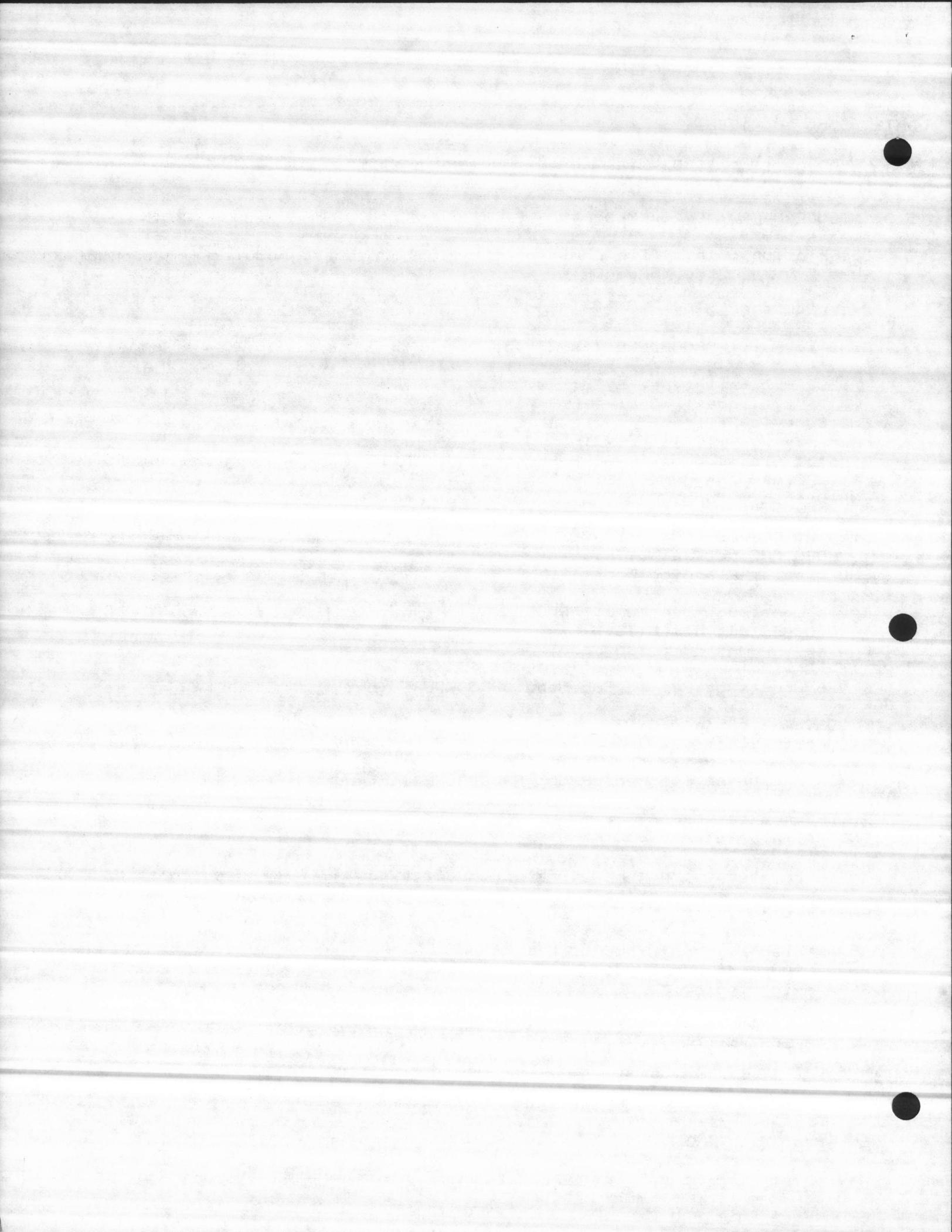
```
Keyin_Print_Data_Base_Proc: PROCEDURE (crt_num) EXTERNAL; /* CxxxxPBAS */
    DECLARE crt_num BYTE;
END Keyin_Print_Data_Base_Proc;
```

```
Keyin_Display_Menu: PROCEDURE (crt_num) EXTERNAL;          /* CxxxxDISG */
    DECLARE crt_num BYTE;
END Keyin_Display_Menu;
```

```
Display: PROCEDURE (crt_num) EXTERNAL;                      /* CxxxxDISG */
    DECLARE crt_num BYTE;
END Display;
```

```
Base_Display: PROCEDURE (crt_num) EXTERNAL;                /* CxxxxDBAS */
    DECLARE crt_num BYTE;
END Base_Display;
```

```
Keyin_Edit_Sub_Menu_Proc: PROCEDURE (crt_num) EXTERNAL; /* CxxxxEDIT */
    DECLARE crt_num BYTE;
END Keyin_Edit_Sub_Menu_Proc;
```



```

Keyin_File_Sub_Menu_Proc: PROCEDURE (crt_num) EXTERNAL; /* CxxxxFILE */
  DECLARE crt_num BYTE;
END Keyin_File_Sub_Menu_Proc;

Keyin_Options_Sub_Menu_Proc: PROCEDURE (crt_num) EXTERNAL; /* CxxxxOPTN */
  DECLARE crt_num BYTE;
END Keyin_Options_Sub_Menu_Proc;

Keyin_Print_Report_Sub_Menu: PROCEDURE (crt_num) EXTERNAL; /* CxxxxPRTG */
  DECLARE crt_num BYTE;
END Keyin_Print_Report_Sub_Menu;

Alarm_System_Display: PROCEDURE (crt_num) EXTERNAL; /* CxxxxASYS */
  DECLARE crt_num BYTE;
END Alarm_System_Display;

Prompt_Entry: PROCEDURE (entry_max_count,crt_num) EXTERNAL; /* CxxxxEDIT */
  DECLARE (crt_num,entry_max_count) BYTE;
END Prompt_Entry;

Graphs_Display_Menu: PROCEDURE (crt_num, hist_select) EXTERNAL;
  DECLARE crt_num           BYTE;                                /* CxxxxLNGR */
  DECLARE hist_select        WORD;
END Graphs_Display_Menu;

Point_Id_Search: PROCEDURE (id_p,struc_type_p,struc_index_p)
  BYTE EXTERNAL; /* CxxxxECAS */
  DECLARE (id_p,struc_type_p, struc_index_p) POINTER;
END Point_Id_Search;

Keyin_Print_Sub_Menu_Proc: PROCEDURE (crt_num) EXTERNAL;
  DECLARE crt_num BYTE;
END Keyin_Print_Sub_Menu_Proc;

Historical_Menu: PROCEDURE (crt_num) EXTERNAL;
  DECLARE crt_num BYTE;
END Historical_Menu;

Keyin_Test_Sub_Menu_Proc: PROCEDURE (crt_num) EXTERNAL; /* CxxxxTEST */
  DECLARE crt_num BYTE;
END Keyin_Test_Sub_Menu_Proc;

Keyin_Start_Stop_Proc: PROCEDURE (crt_num, control_type) EXTERNAL;
  DECLARE crt_num BYTE;                                         /* CxxxxPCON */
  DECLARE control_type WORD;
END Keyin_Start_Stop_Proc;

```

## GLOBAL DATA REFERENCES

=====

See (CxxxxGLOB.Pyy)

## MODULE DECLARATIONS

=====

```

exception          - system calls exception flag.
main_menu_command_count - A byte holds the total number of commands in
                           the menu.

```

main\_menu\_prompt\_asc STRUCTURE :

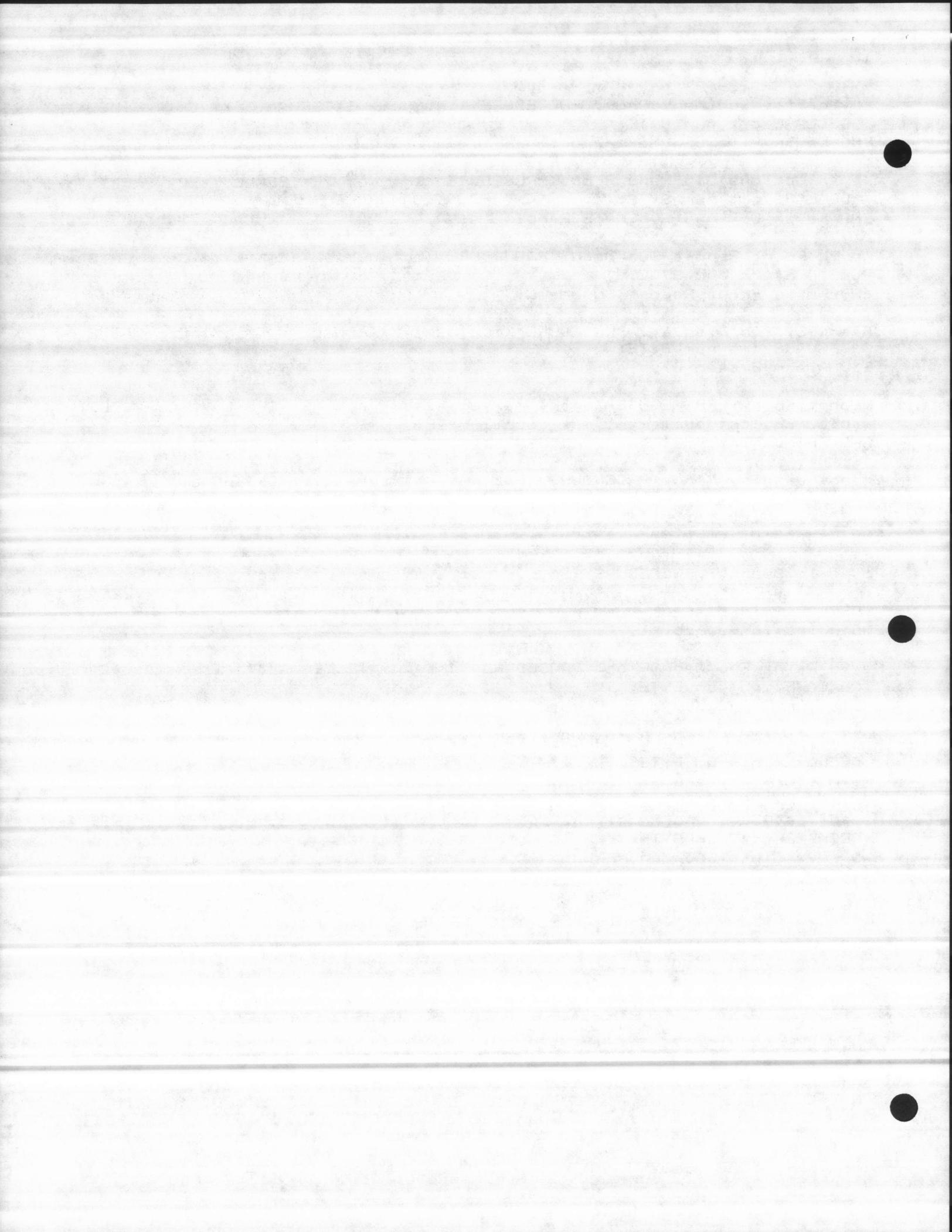
```

-----  

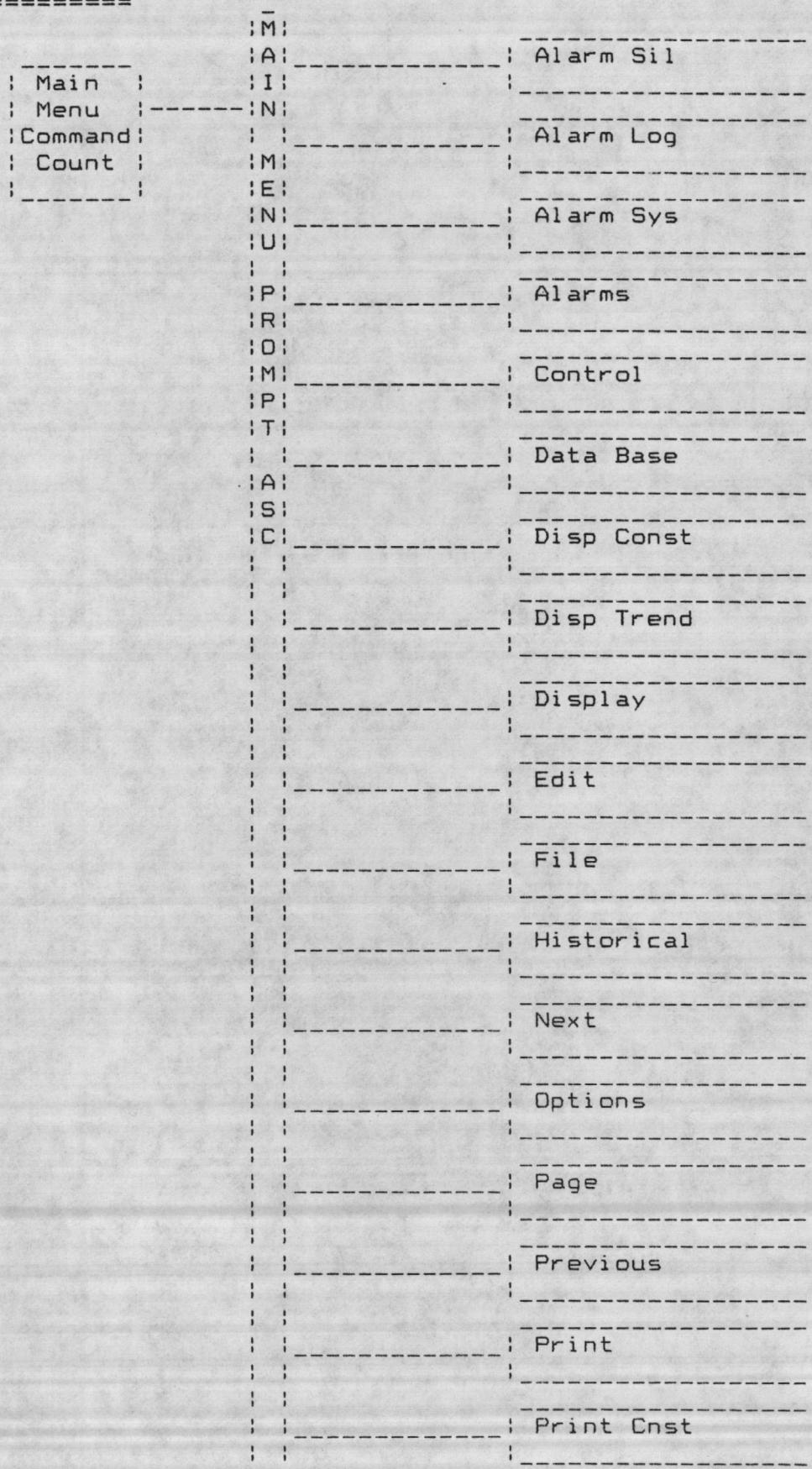
A structure holds 22 commands each command descreption is in menu_asc.  

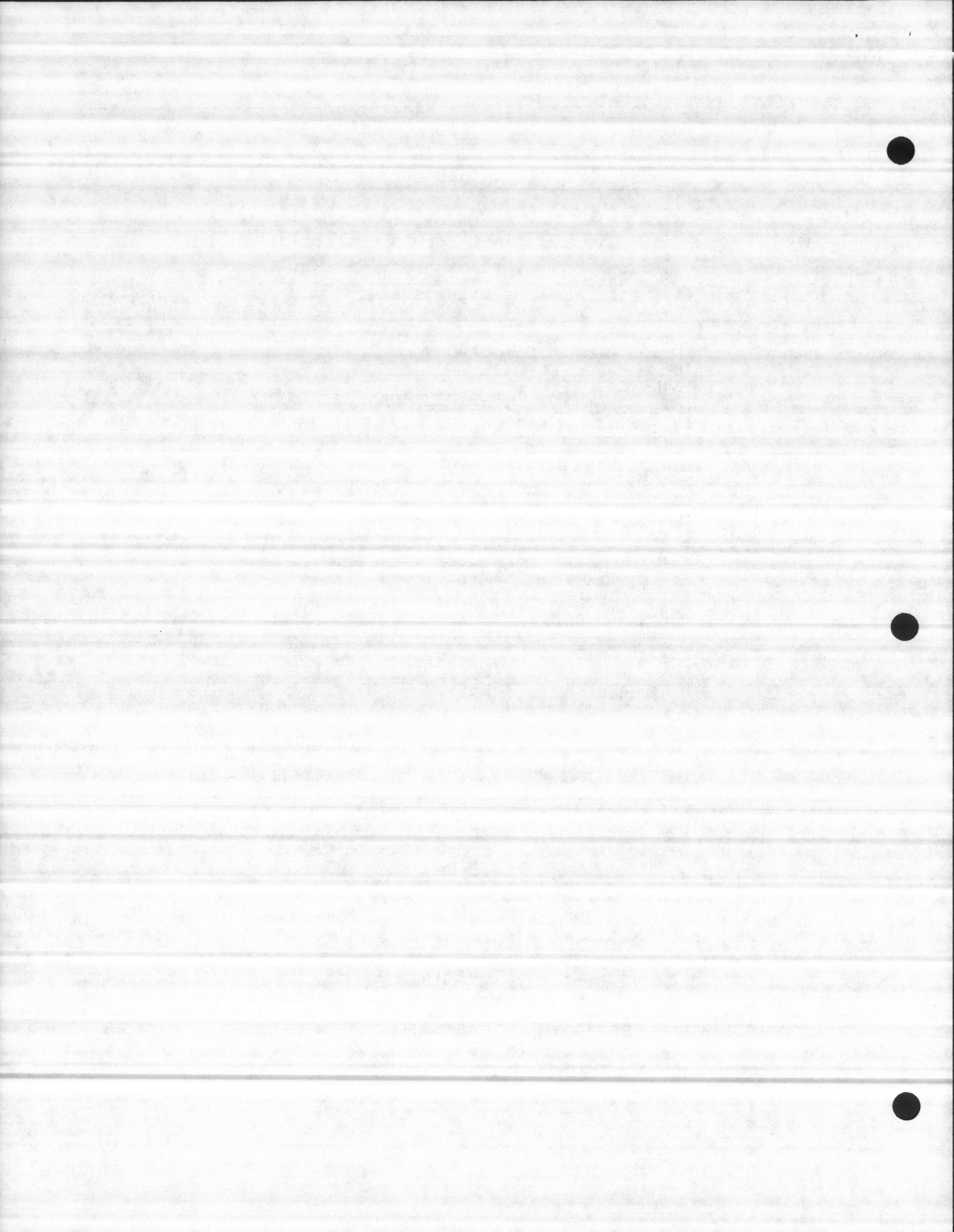
The command descrption is 10 bytes

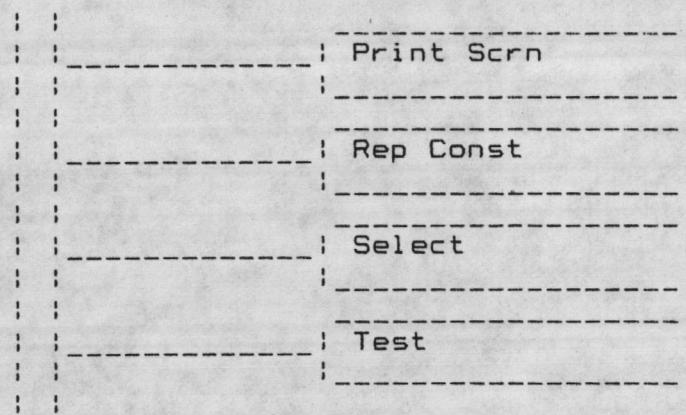
```



## DATA DIAGRAM :



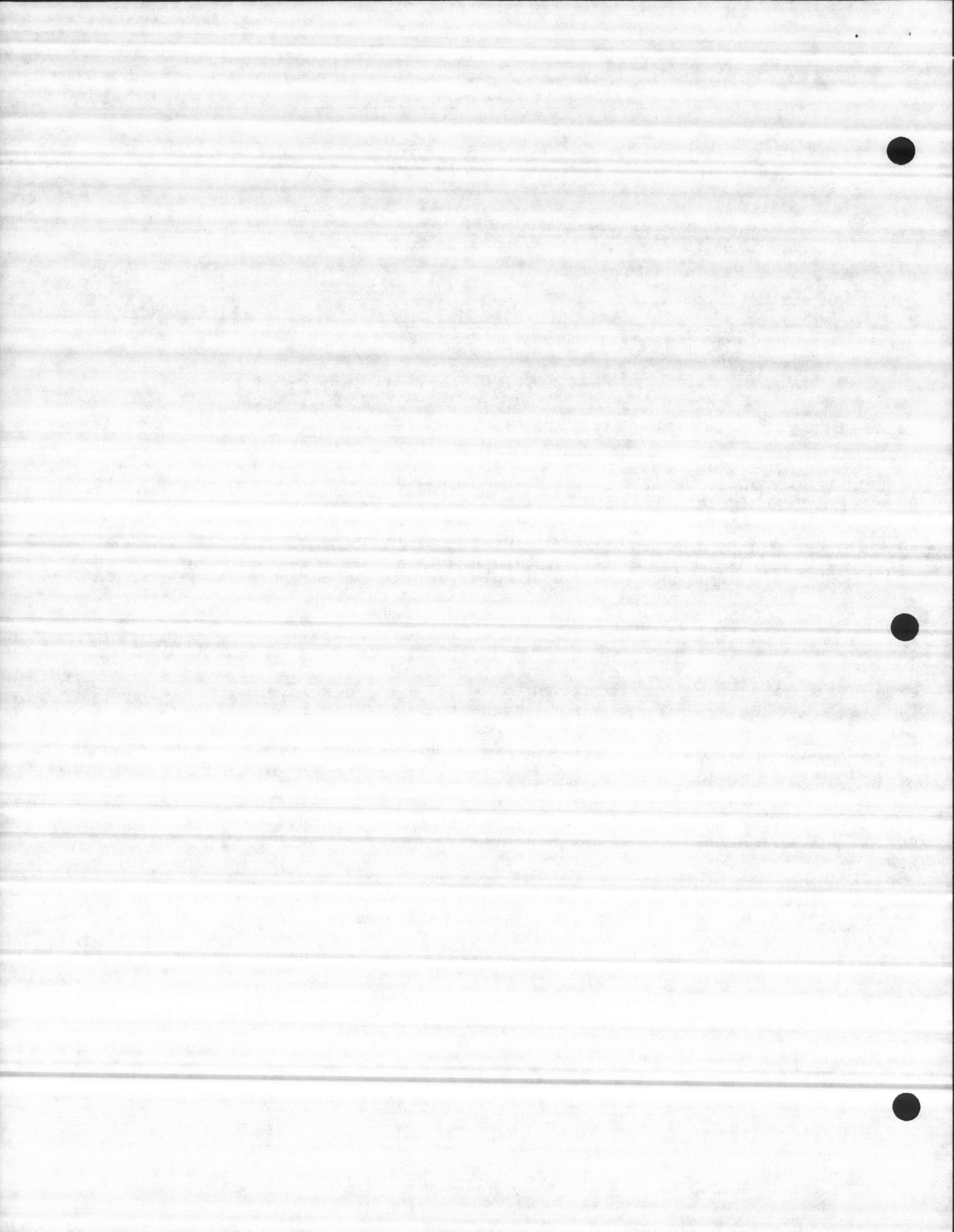


**DESCRIPTION :****=====**

ALARM SIL	=> Acknowledges current alarms.
ALARM LOG	=> Displays the current alarm log.
ALARM SYS	=> Displays the alarm system configuration.
ALARMS	=> Displays current alarm summary.
CONTROL	=> Allows manual control of system pumps.
DATA BASE	=> Enters sub-menu for date base configuration.
DISP CONST	=> Enters sub-menu for display construction.
DISP TREND	=> Enters sub-menu for trend data displays.
DISPLAY	=> Enters sub-menu for display selection.
EDIT	=> Enters sub-menu for editor selection.
FILE	=> Enters sub-menu of disk file commands.
HISTORICAL	=> Enters sub-menu for historical selection.
NEXT	=> Advances to next page on multi-page displays.
OPTIONS	=> Enters sub-menu for system option selection.
PAGE	=> Advances to specified page on multi-page displays.
PREVIOUS	=> Advances to previous page on multi-page displays.
PRINT	=> Enters sub-menu for printer report selection.
PRINT CNST	=> Enters sub-menu for construct report selection.
PRINT SCRN	=> Copies the color CRT screen to the color printer.
REP CONST	=> Enters sub-menu for report construction.
SELECT	=> Advances to a selected display page or data base page.
TEST	=> Enters sub-menu for test io configurations.

`main_menu_help_desc_tbl (22)` - An array of pointers pointing to help menu help descriptions.

`main_menu_proc_tbl (22)` - An array of pointers to procedures called by the menu driver.



**Keyboard Procedure****Keyboard:** PROCEDURE PUBLIC;

result\_word - Status return word from Aquatrol library functions.

Procedure Description: This procedure displays the first level  
help menu at the start of the system.**Algorithmn**

```
=====
set exception handler
set up clock display
Loop FOREVER;
    Call Menu Driver display to setup the main menu and accept operator
    commands.
END;
END Keyboard;
```

**Keyin\_trends\_Display\_menu****=====****Keyin\_Trends\_Display\_Menu:** PROCEDURE (crt\_num) PUBLIC REENTRANT;
crt\_num - Byte containing the number of the calling CRT;

Description : This procedure calls the graphs display menu for trending .
=====

END Keyin\_Trends\_Display\_Menu;

**Keyin\_Alarm\_Proc****=====****Keyin\_Alarm\_Proc:** PROCEDURE (crt\_num) PUBLIC REENTRANT;
crt\_num - Byte containing the number of the calling CRT;

Description : This procedure calls the system wide alarm summary page.
=====

END Keyin\_Alarm\_Proc;

**Keyin\_Alarmlog\_Proc****=====****Keyin\_Alarmlog\_Proc:** PROCEDURE (crt\_num) PUBLIC REENTRANT;
crt\_num - Byte containing the number of the calling CRT;

Description : This procedure calls the alarm log display.
=====

END Keyin\_Alarmlog\_Proc;

**Keyin\_Next\_Proc****=====****Keyin\_Next\_Proc:** PROCEDURE (crt\_num) PUBLIC REENTRANT;

crt\_num - Number of the calling CRT.

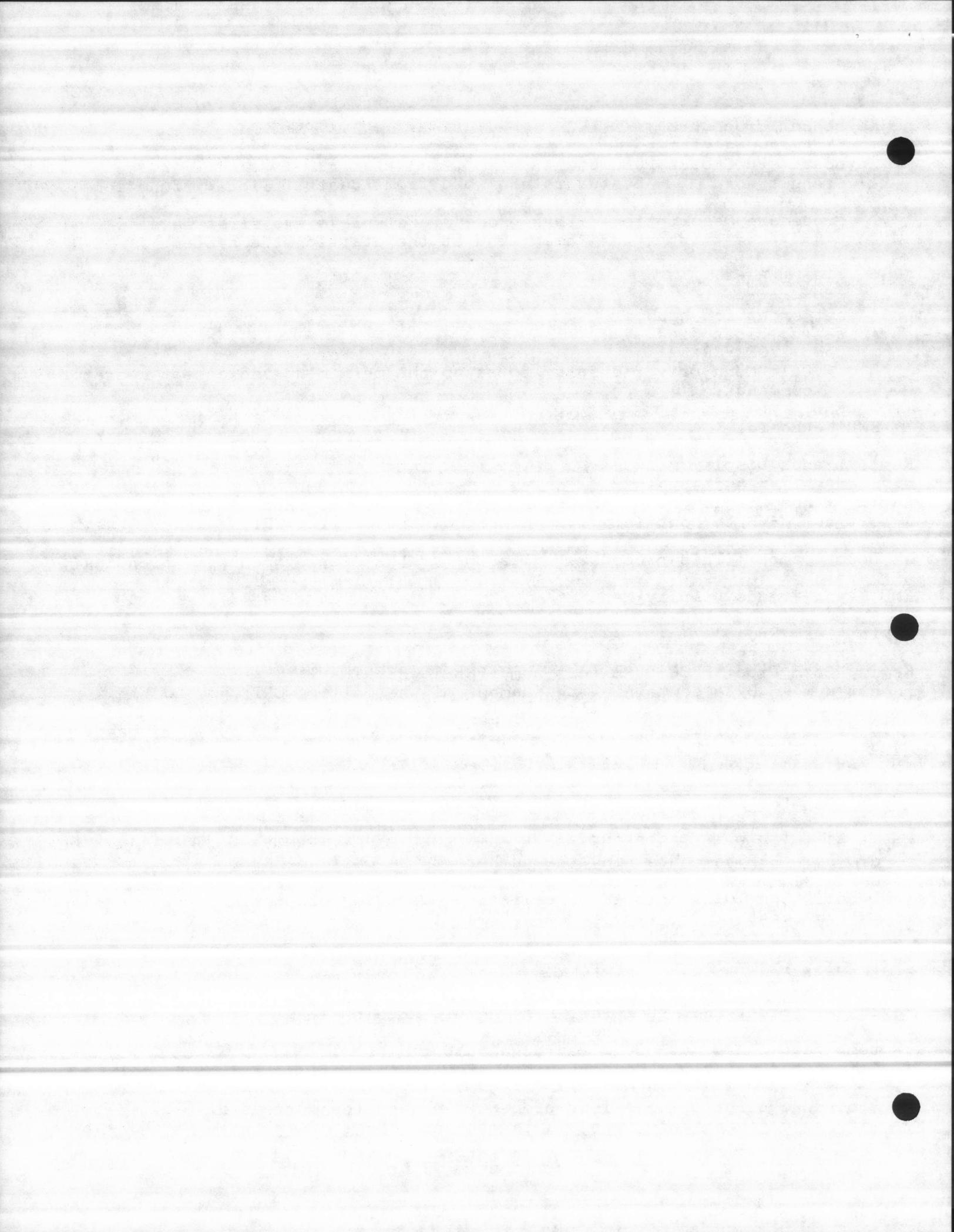
temp\_page - Temporary page number.

temp\_crt\_display\_p - Temporary crt display pointer.

Procedure Description : This procedure causes the current display
=====

to display the next page of multiple page displays.

END Keyin\_Next\_Proc;



**Keyin\_Previous\_Proc**

Keyin\_Previous\_Proc: PROCEDURE (crt\_num) PUBLIC REENTRANT;  
crt\_num - Number of the calling CRT.  
temp\_page - Temporary page number.  
temp\_crt\_display\_p - Temporary crt display pointer.

Procedure Description : This procedure causes the current display  
to display the previous page of multiple page displays.

END Keyin\_Previous\_Proc;

**Keyin\_Page\_Proc**

Keyin\_Page\_Proc: PROCEDURE (crt\_num) PUBLIC REENTRANT;  
crt\_num - Number of the calling CRT.  
temp\_page - Temporary page number.  
temp\_crt\_display\_p - Temporary crt display pointer.

Procedure Description : This procedure causes the current display  
to display the previous page of multiple page displays.

END Keyin\_Page\_Proc;

**Keyin\_Select\_Proc**

Keyin\_Select\_Proc: PROCEDURE (crt\_num) PUBLIC REENTRANT;  
crt\_num - Number of the calling CRT.  
index - Temporary index.  
struc\_index - Subscript of the structure selected.  
struc\_type - Type of structure selected.  
temp\_page - Temporary page number.

Procedure Description : This procedure displays the specific display  
for the point Id that the operator types in.

END Keyin\_Select\_Proc;

**Keyin\_Silence\_Proc**

Keyin\_Silence\_Proc: PROCEDURE (crt\_num) PUBLIC REENTRANT;  
crt\_num - CRT that initiated the silence.

Procedure Description : This procedure sends a message to FAIL  
initiating an alarm silence action.

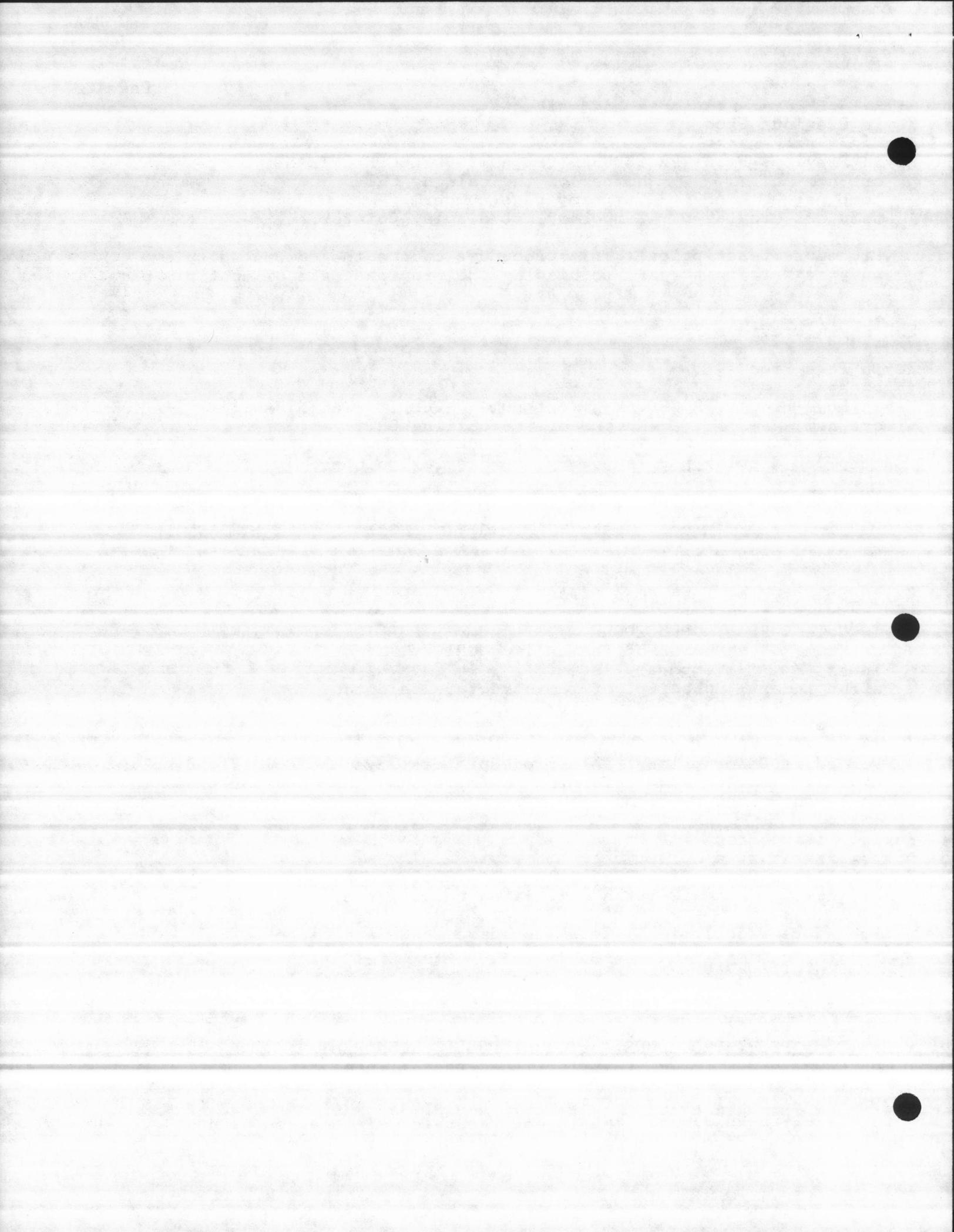
END Keyin\_Silence\_Proc;

**Video\_Copy\_Proc**

Video\_Copy\_Proc: PROCEDURE (crt\_num) PUBLIC REENTRANT;  
crt\_num - CRT that initiated the silence.

Procedure Description : This procedure starts a screen copy from a  
XL-19 colorgraphic CRT to a Jet printer.

END Video\_Copy\_Proc;



Control\_Proc

=====

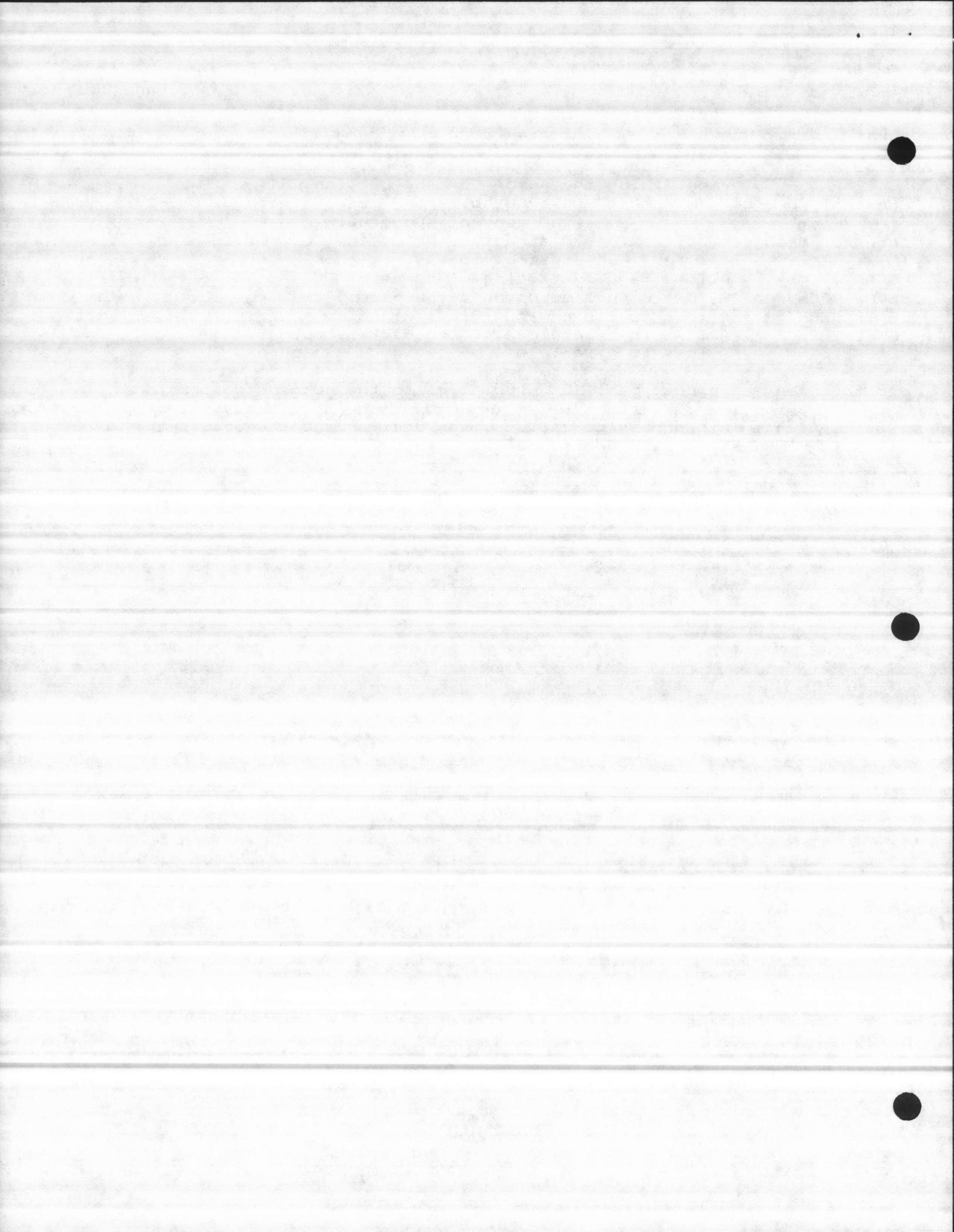
Control\_Proc: PROCEDURE (crt\_num) PUBLIC REENTRANT;  
crt\_num - Byte containing the number of the calling CRT;  
control\_menu\_command\_count - A byte holds the total number of commands in  
the control sub-menu.  
control\_menu\_prompt\_asc - A structure holds the Control command  
descriptions.  
START PUMP => Allows manual starting of pumps.  
STOP PUMP => Allows manual stopping of pumps.  
FAIL RESET => Allows to clear manual fail signal.  
ENABLE BW => Allows to enable one of two backwash pumps.  
DISABLE BW => Allows to disable one of two backwash pumps.

control\_menu\_help\_desc\_tabl (5) - pointers to the command descriptions.

Description : This procedure calls a menu deriver and then calls the keyin  
start\_stop procedure.

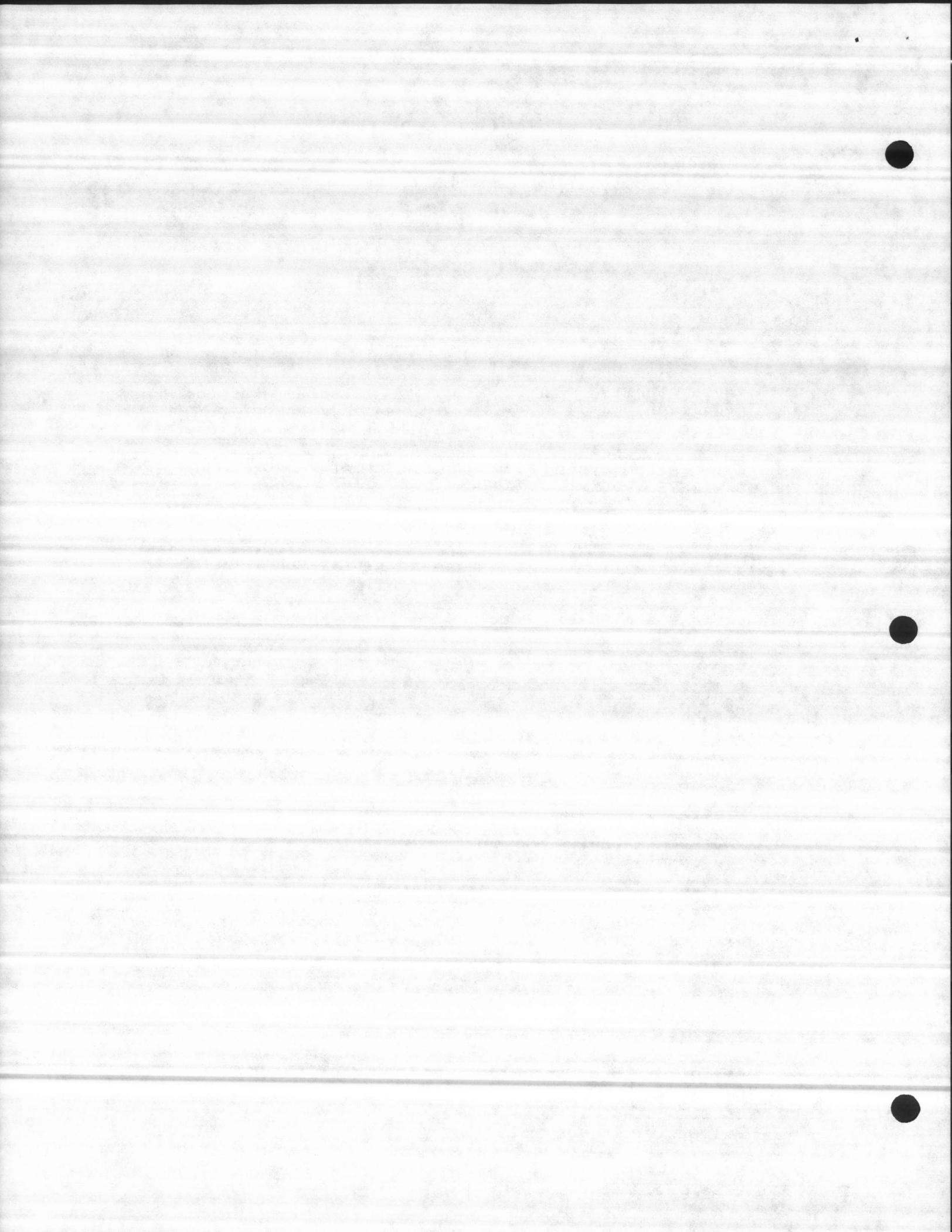
=====

END Control\_Proc;

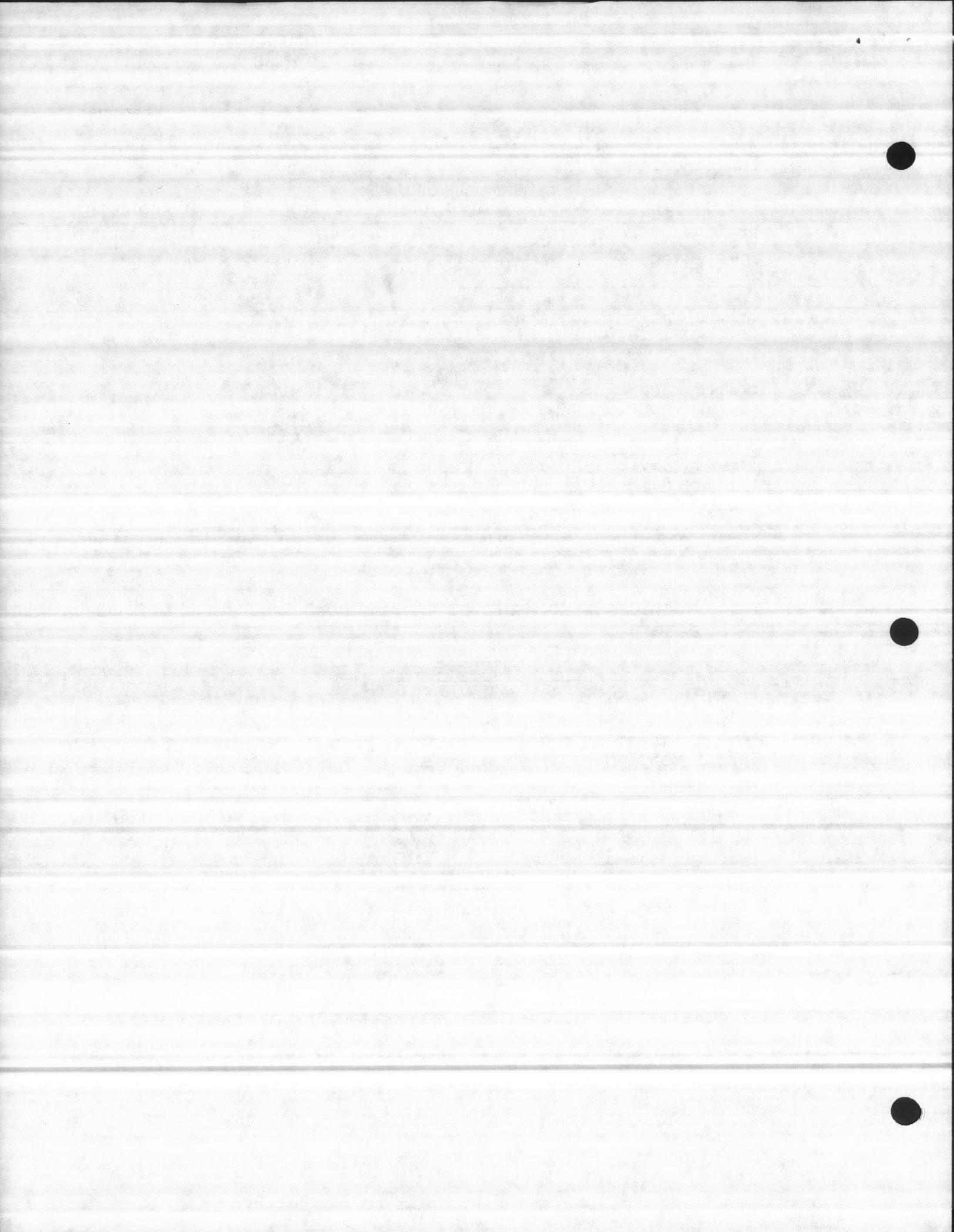


## MODULE FLOW :





```
|-----|
|Historcal_Menu |
|CxxxxHMST.Pyy |
|-----|
|-----|
|Keyin_Next_Proc |
|CxxxxKYBD.Pyy |
|-----|
|-----|
|Keyin_Options_Sub_Menu_Proc |
|CxxxxDPTP.Pyy |
|-----|
|-----|
|Keyin_Page_Proc |
|CxxxxKYBD.Pyy |
|-----|
|-----|
|Keyin_Previous_Proc |
|CxxxxKYBD.Pyy |
|-----|
|-----|
|Keyin_Print_Sub_Menu_Proc |
|CxxxxHMST.Pyy |
|-----|
|-----|
|Keyin_Print_Report_Sub_Menu |
|CxxxxPRTG.Pyy |
|-----|
|-----|
|Video_Copy_Proc |
|CxxxxKYBD.Pyy |
|-----|
|-----|
|Keyin_Construct_Report_Sub_Menu |
|CxxxxREPG.Pyy |
|-----|
|-----|
|Keyin_Select_Proc |
|CxxxxKYBD.Pyy |
|-----|
|-----|
|Keyin_Test_Sub_Menu_Proc |
|CxxxxTEST.Pyy |
|-----|
```



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 6

O P T I O N S

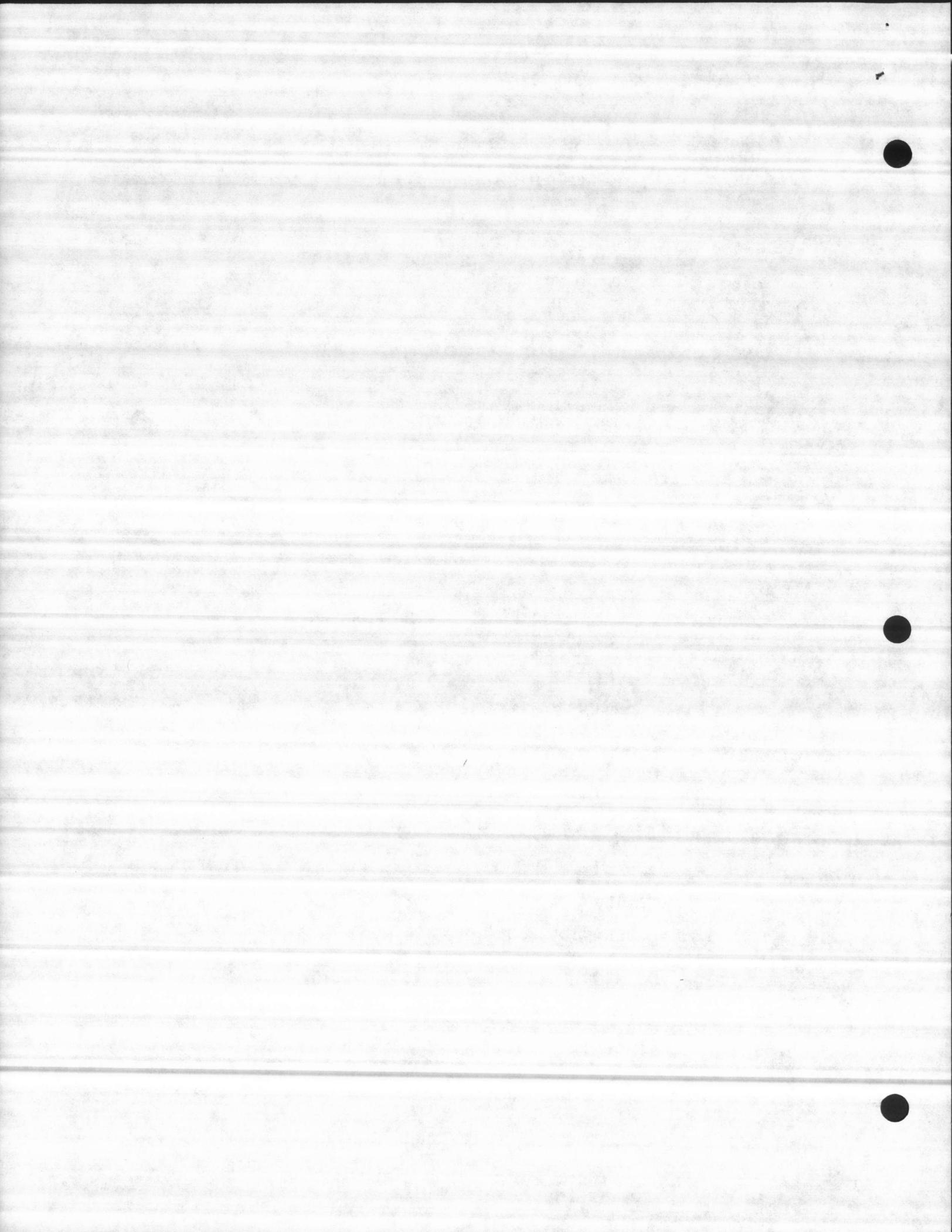
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1986

The system options - written by Bob Ryan, Mohamed Fayad.

(6.0) MODULE NAME : CxxxxOPTP.Pyy

=====

Description : This file contains crt table editable display, which  
===== displays all the options.

EXTERNAL PROCEDURES :

=====

```
Display_Table: PROCEDURE (table_p,
                           param_p,
                           table_reg_t,
                           buf_t,
                           mbx_t) EXTERNAL;                      /* CxxxxEDIT.Pyy */
DECLARE
  (table_reg_t,
   buf_t,
   mbx_t)           TOKEN,
  (table_p,
   param_p)         POINTER;
End Display_Table;
```

GLOBAL DATA REFERENCE:

=====

options See (CxxxxGLOB.Pyy).

MODULE DECLARATIONS :

=====

exception (3) - Three words, one for each crt exception condition.

crt\_update\_sem\_t (3) - Three tokens, used to create a semaphore in update task.

static\_asc - a group of bytes contain the field asc for the lab summary second header, as following :

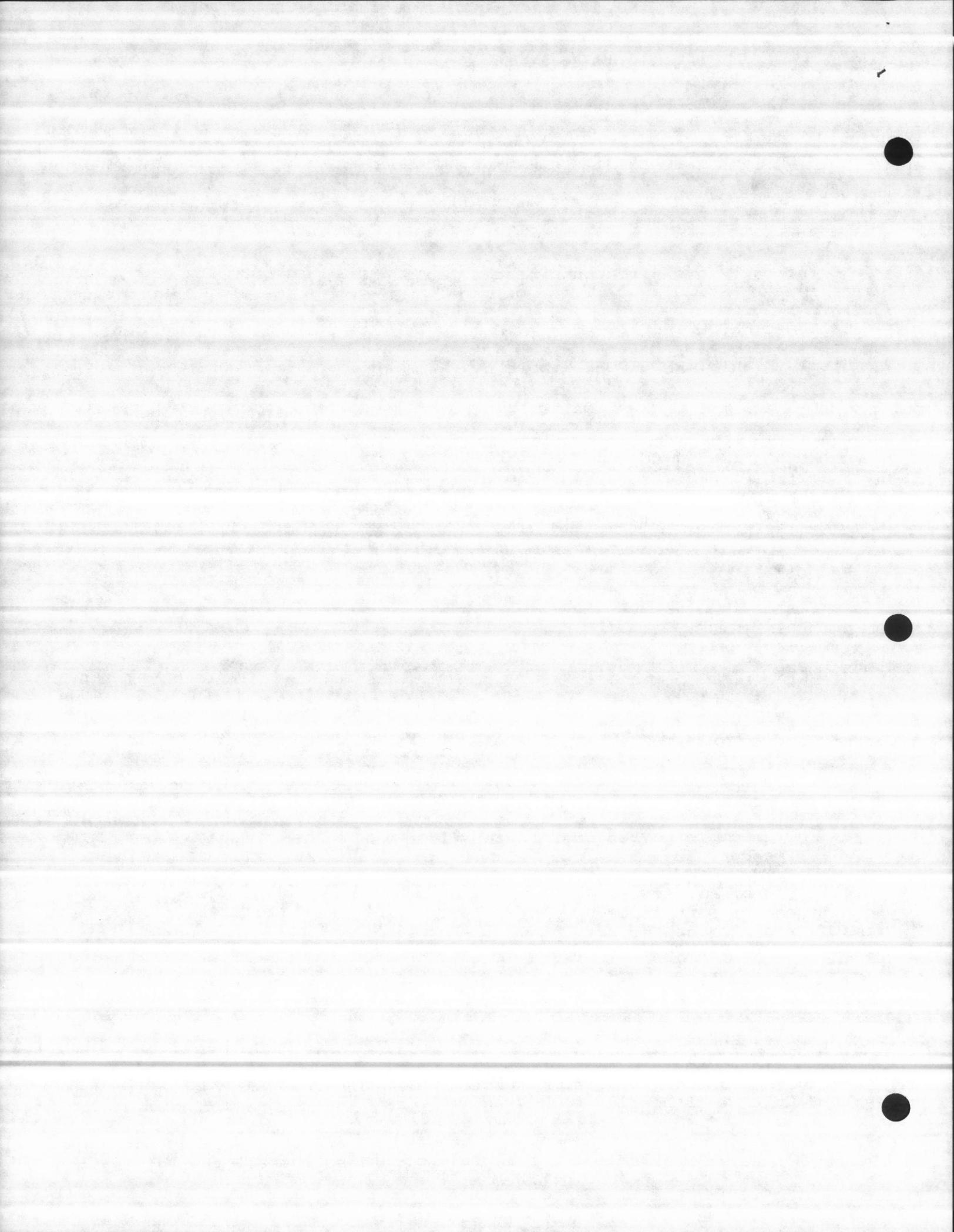
```
Automatic end of day report ?
Automatic end of week report ?
Automatic end of month report ?
Automatic alarm display ?
Automatic end of day alarms report ?
```

option (6) => six words, one for each option.

option\_crt\_table Strucutre:

This structure holds the crt display table, which displays all the option in the system. Edit is allowed.

crt\_table\_1\_seg - A segment token, which is used to create a segment for displaying the crt table1(which displays all the information for an lab point). -



crt\_table\_2\_seg - The same as crt\_table\_1\_seg;

crt\_table\_1 - A structure contains all the display and edit parameters for all the fields in the option page(BASED crt\_table\_1\_seg).

crt\_table\_2 - The same as crt\_table\_1(BASED crt\_table\_2\_seg).

option\_Update\_table - A table of Option\_Update pointer.

option\_delete\_table - A table of Option\_Delete pointer.

LOCAL PROCEDURES :

=====

keyin\_Options\_Sub\_Menu\_Proc: PROCEDURE (crt\_num) PUBLIC REENTRANT;

crt\_num - A byte, the number of the crt to display on.

count - A byte, used as a counter.

Proc description:

----- This procedure displays the option page header and its static\_asc.

End Keyin\_Options\_Sub\_Menu\_Proc;

Option\_Update: PROCEDURE PUBLIC REENTRANT;

crt\_num - A byte, the number of the crt to display on.

count - A word, used as a count.

remaining\_units - A word, used as an index.

result - A word BASED result\_p used in Create\$Clock\$Entry

index - A word , used as an index.

table\_p - A pointer used to point to the crt table structure.

crt\_table - A structure contains all the information available for the current page display (BASED table\_p).

Proc description:

----- This procedure fills the crt table for the desired display options. The crt table every clock entry interval(every 10 sec) by creating a semaphore and clock entry.

END Option\_Update;

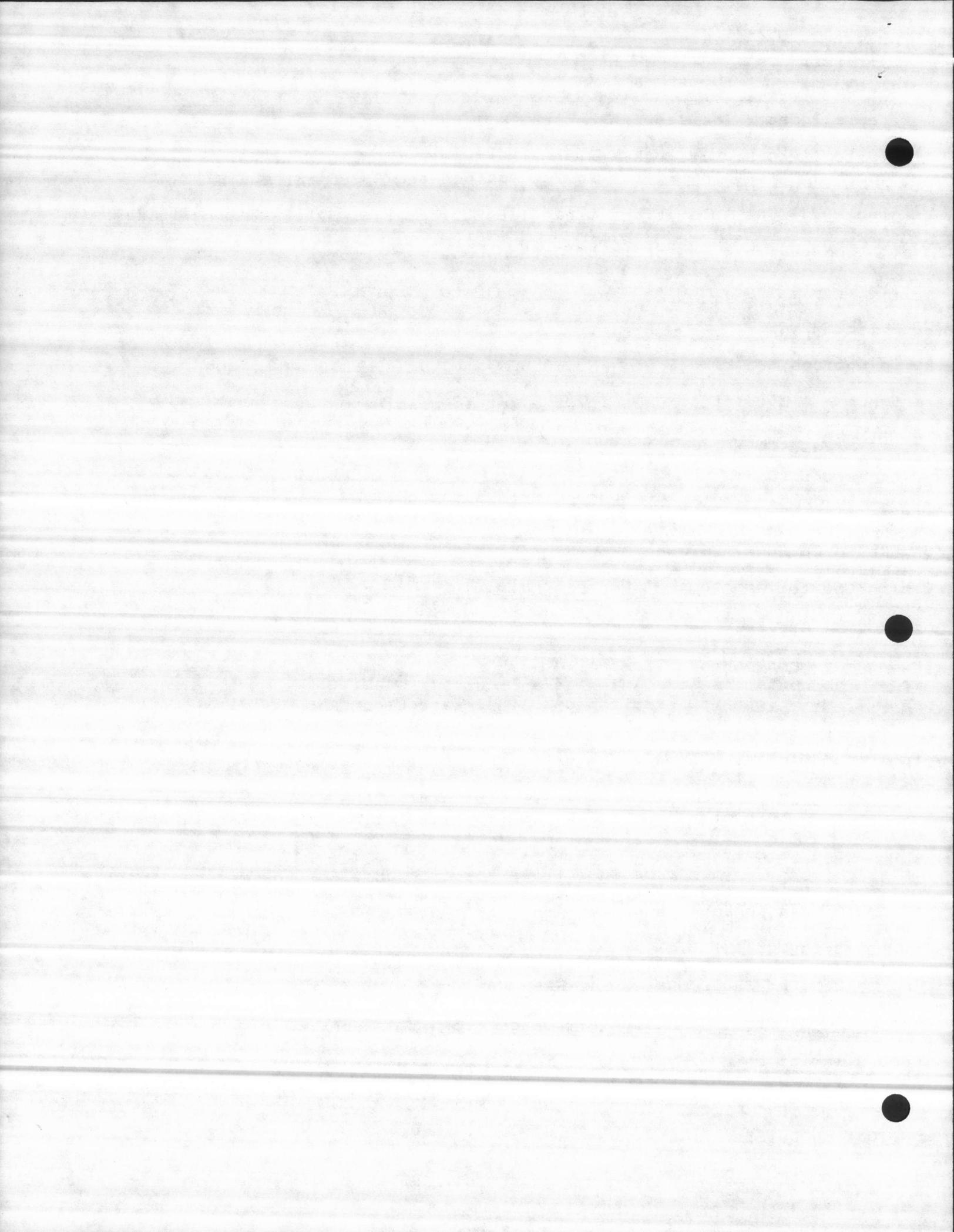
Option\_Delete: PROCEDURE PUBLIC REENTRANT;

crt\_num - A byte, the number of the crt to display on.

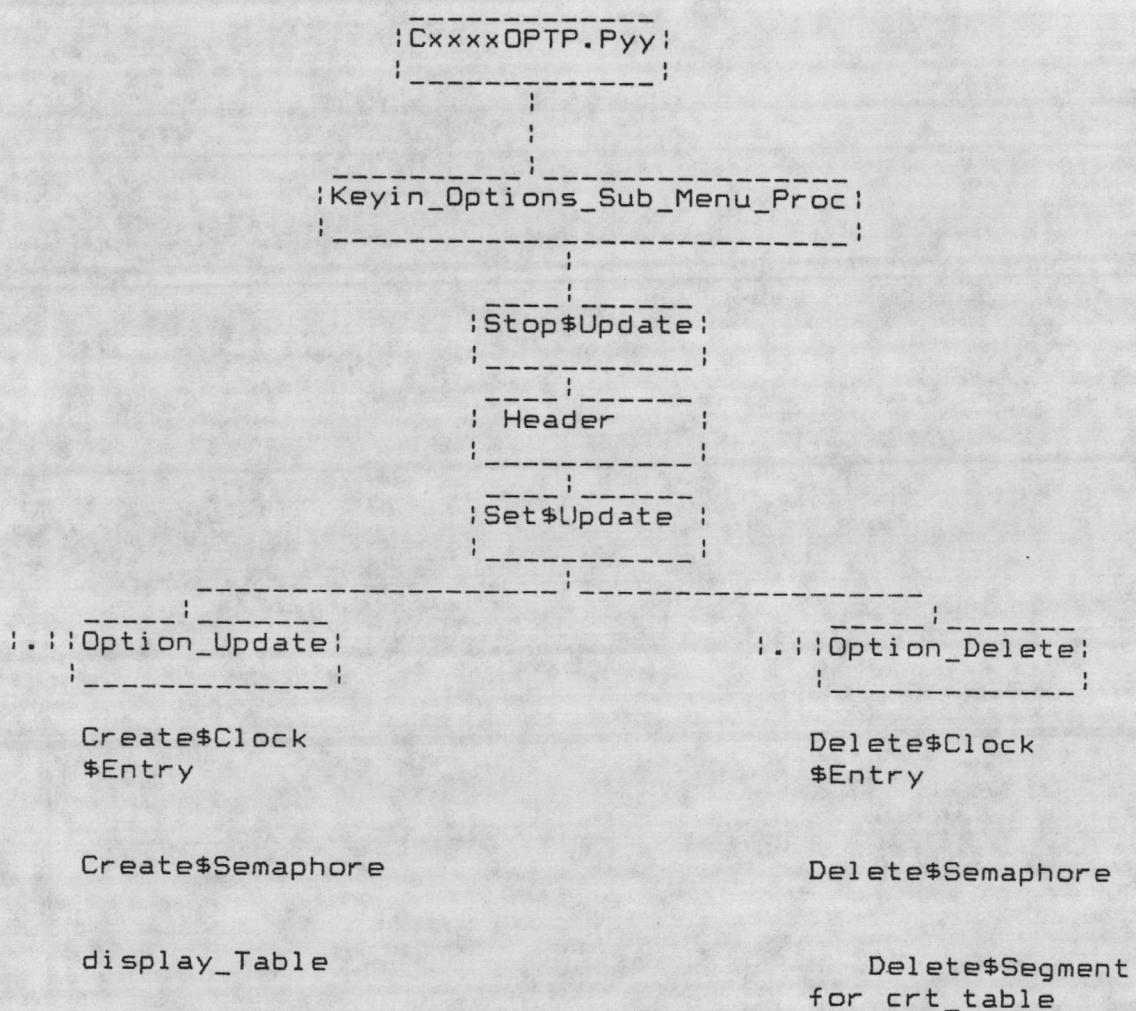
Proc description:

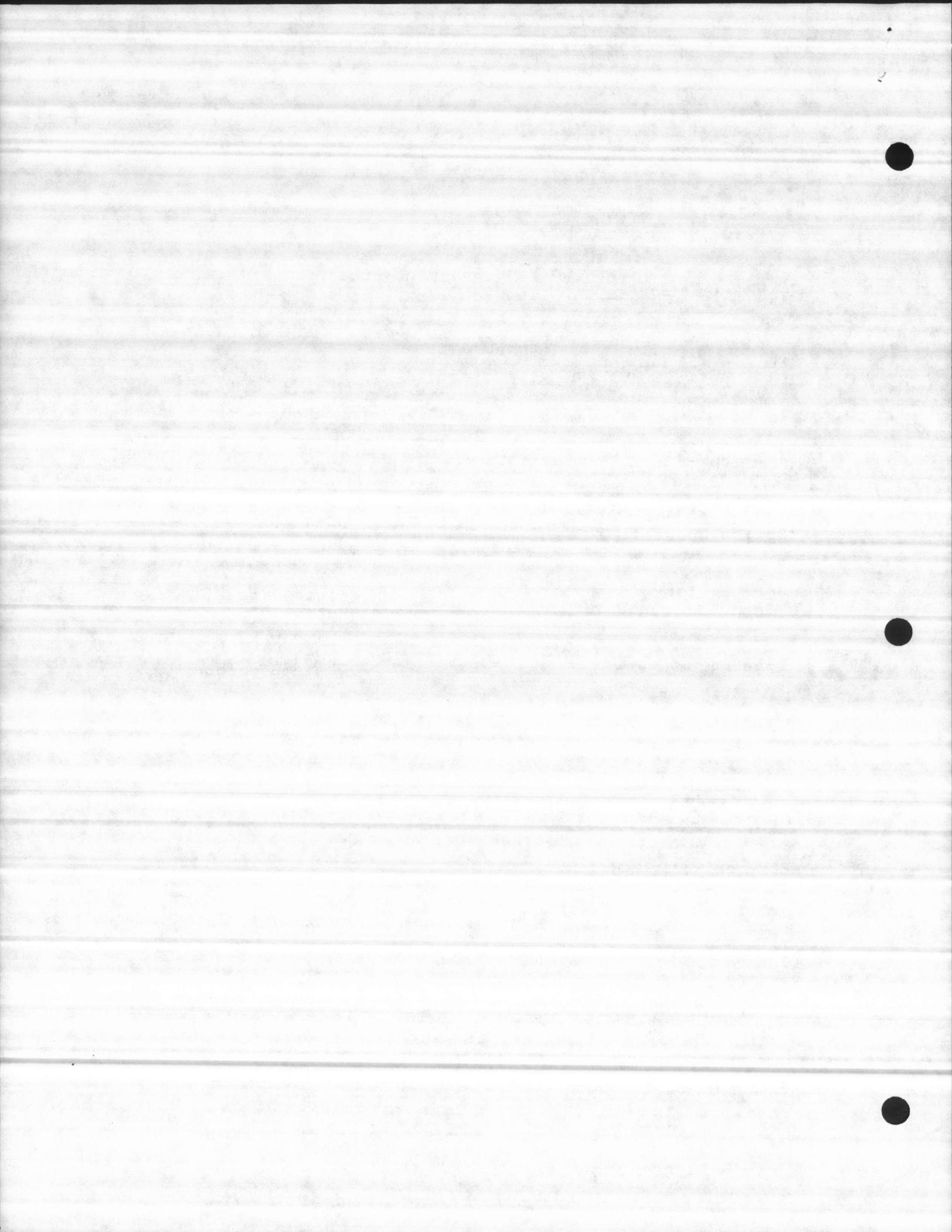
----- This procedure deletes the clock entry, the semaphore and the segments associate with the crt tables.

END Option\_Delete;



MODULE FLOW :





AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

---

SECTION No. 7

S C R E E N

---

E D I T O R

---

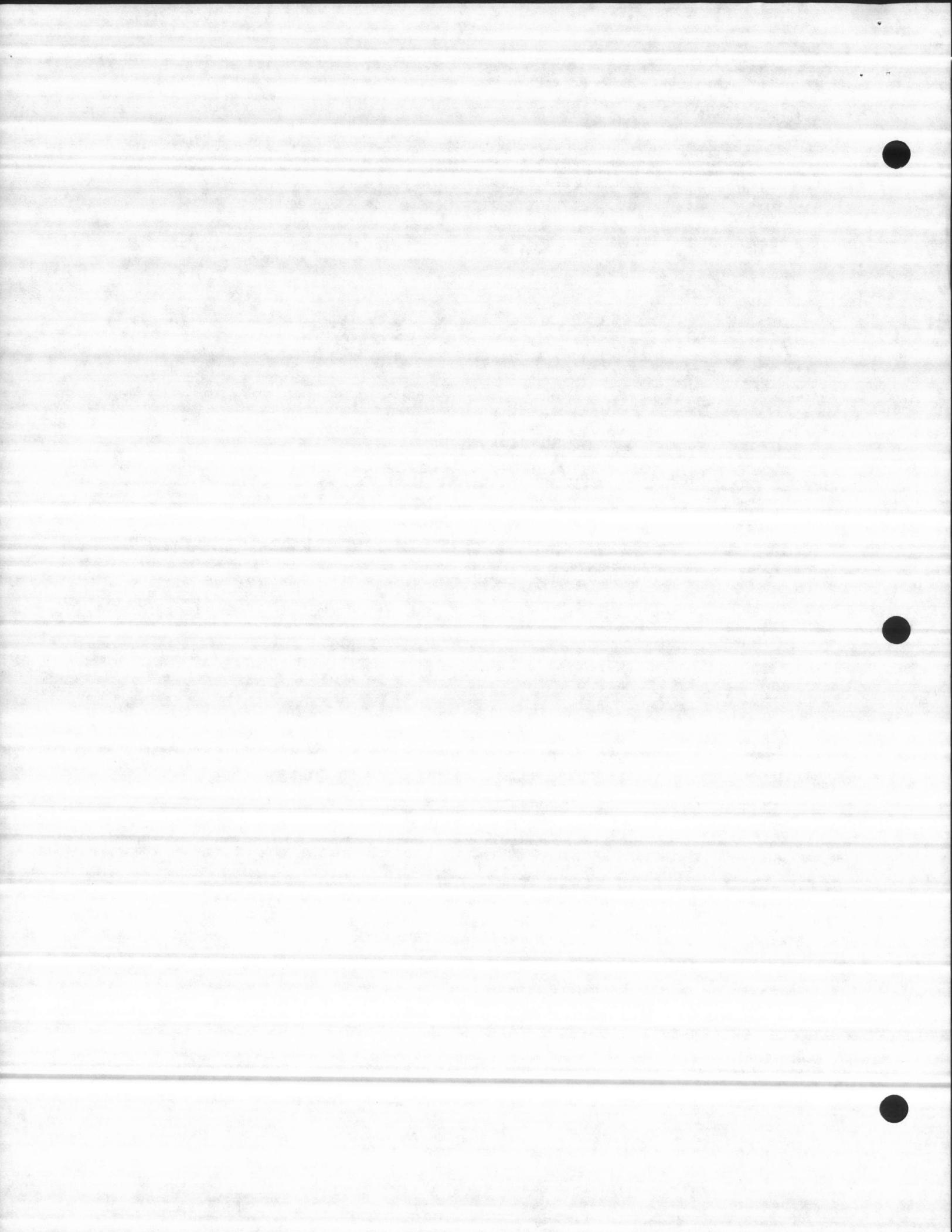
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

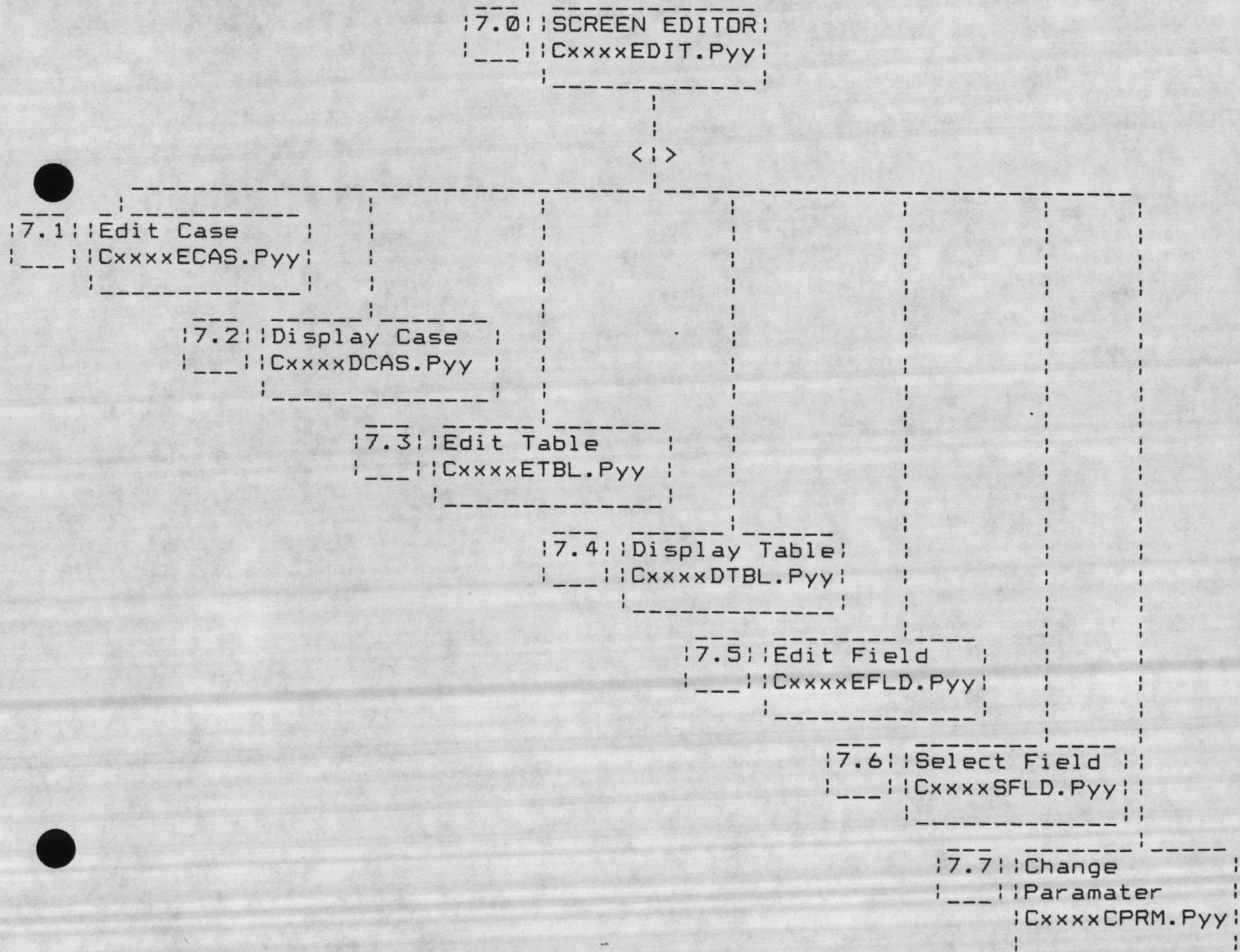
T screen editor - Written by Peter Wollenzien, Bob Ryan.

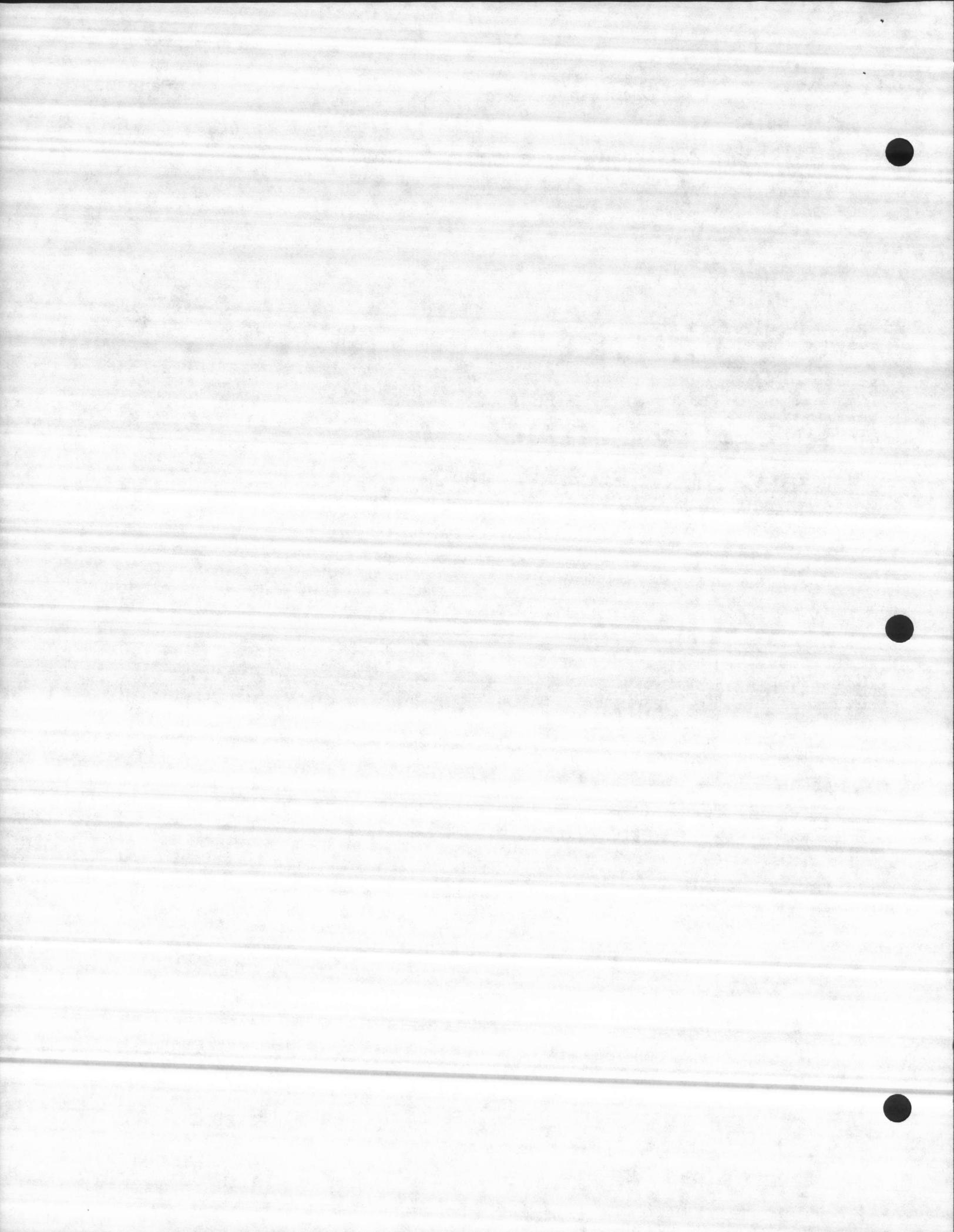
INTRODUCTION :

This section describes the screen editor or  
the 'Non-Generic Crt Display & Field Editor'.

Advantages :

- 1) Easy to use.
- 2) The ability to edit any (numeric or ascii string) field in the screen at any time.
- 3) The maximum of 10 level of security.
- 4) The ability to edit time & date.
- 5) Can be used on multiple crts.





---

7.0 MODULE NAME : CxxxxEDIT.Pyy

---

=====

DESCRIPTION :

=====

1 => Screen Edit Module.

2 => Contains the following standard procedures. These procedures should and could be made reentrant to allow multiple crt tasks. Additional parameters would need to be added passing such variables as display\_buf\_t, crt\_out\_mbx\_t, etc.

Prompt\_Entry,

Enter\_Password\_And\_Verify,

Enter\_New\_Password,

Edit\_Time\_Procedure,

Keyin\_Edit\_Sub\_Menu\_Proc.

Contains the following reentrant utility procedures used for table displaying and editing :

Prompt\_Display (display\_buf\_t, crt\_out\_mbx\_t, row, column,  
display\_str\_p); This procedure will display the passed rmx  
string constants to the designated displayed position.

Format\_Cursor\_And\_Color (display\_buf\_t, row, column, color\_str\_p);  
This procedure will format into the passed buffer the cursor  
position and a color string. If either row or column is zero  
the cursor position will not be formatted. If a zero is passed  
for the color\_str\_p then no color will be formatted. It is  
assumed that the buffer's region has bee gained through the  
Clear\$Buffer call.

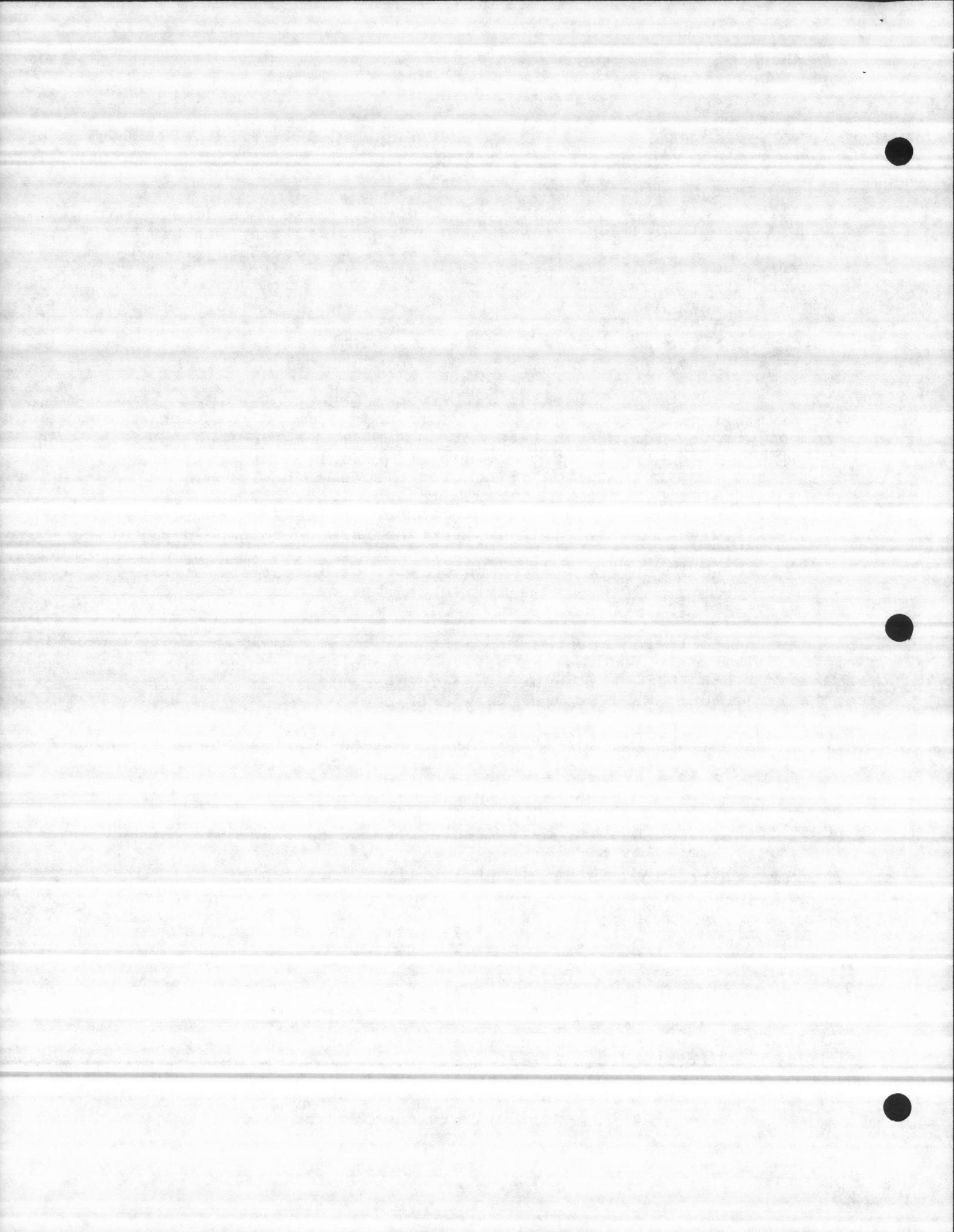
Includes the following reentrant procedures. These procedures are written as separate modules (mainly for readability). All have parameter lists that will enable them to be called from multiple crt tasks (to get away from multiple crt jobs).

C4376DTBL : CALL Display\_Table (table\_p, param\_p, table\_reg\_t,  
----- display\_buf\_t, crt\_out\_mbx\_t);  
This procedure will format all entries and display all entries in a table.

C4376DCAS : CALL Display\_Case (table\_element\_p, param\_element\_p,  
----- display\_buf\_t);  
Called by Display\_Table to individually format table elements.

C4376ETBL : CALL Edit\_Table (crt\_num, table\_p, param\_p,  
----- table\_reg\_t, display\_buf\_t, crt\_out\_mbx\_t, edit\_buf\_t,  
ci\_conn\_t, ci\_mbx\_t).  
This procedure will allow cursor selection of field on the crt, will call Edit case for actual field editting.

C4376ECAS : Edit\_Case (crt\_num, table\_element\_p, param\_element\_p,  
----- table\_reg\_t, display\_buf\_t,  
crt\_out\_mbx, edit\_buf\_t, ci\_conn\_t, ci\_mbx\_t);  
This procedure will set up the necessary range display and call either Edit\_Field or Select\_Field.



```
C4376EFLD : enter_flag =
----- Edit_Field (crt_num, table_element_p,display_buf_t,
display_mbx_t, edit_buf_t, ci_conn_t, ci_mbx_t,
return_dword_p);
This procedure will replace characters within a buffer
and echo these onto the screen. It will also provide
cursor positioning within a field.

C4376SFLD : enter_flag =
----- Select_Field (crt_num, table_element_p, param_element_p,
display_buf_t, display_mbx_t, ci_conn_t, ci_mbx_t);
Uses converted Menu$Driver to select an index to a table
entry.

C4376CPRM : return_byte = Check_Param_Change (table_element_p,
param_element_p);
Used to check the change in any parameter in the crt
table.
```

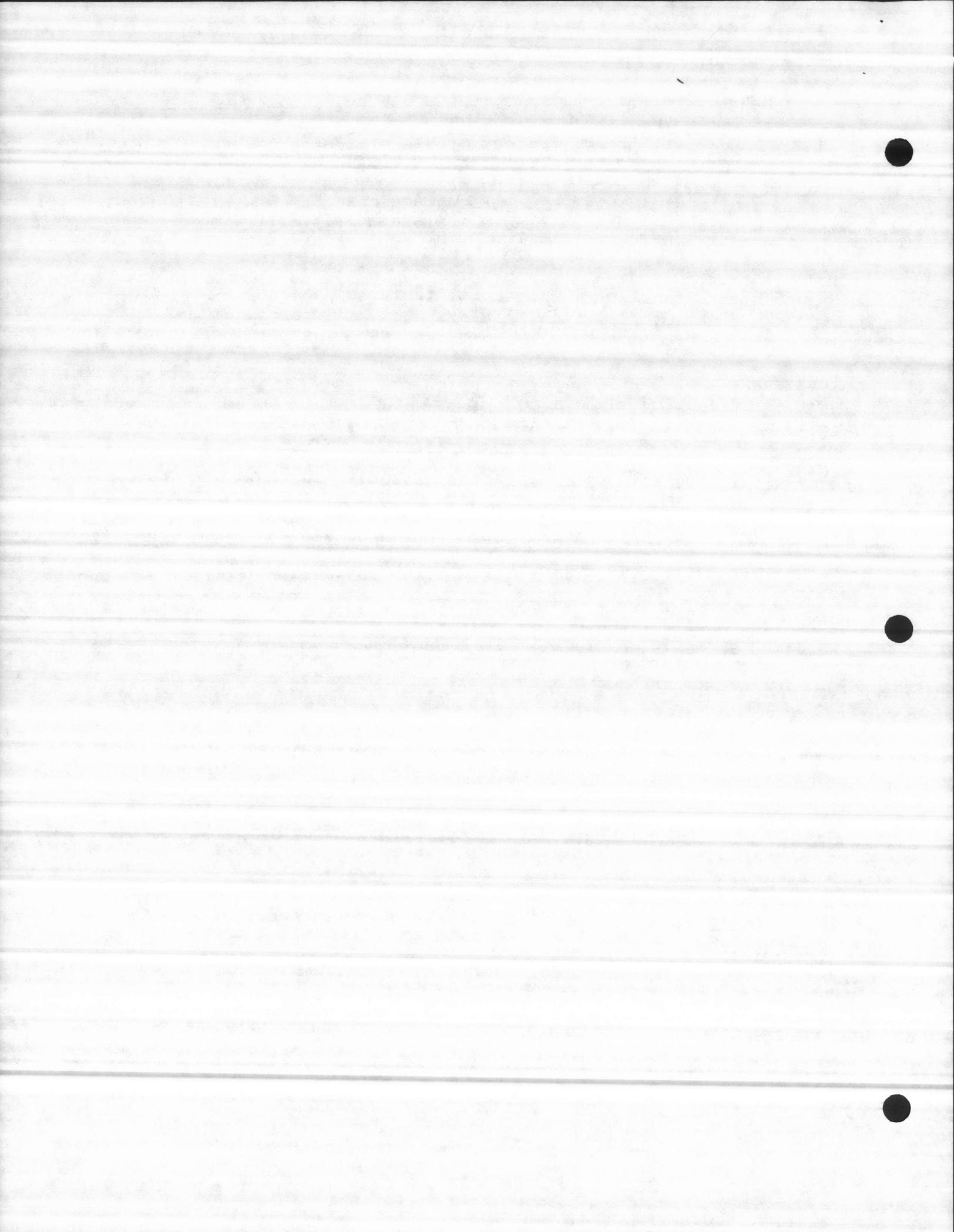
A crt display page will consist of the following sections :

- A static display string. This static ascii string will be sent directly to the crt through the macro and is not updated.
- A display/edit table. This table will be covered in the next few pages.

#### THE DISPLAY / EDIT CRT TABLE

```
crt_table(NUM_ELEMENTS_IN_THE_TABLE_PLUS_ONE) STRUCTURE
{
    display_type     BYTE,
    edit_flag        BYTE,
    row              BYTE,
    column           BYTE,
    field_length     BYTE,
    reg_t_p          POINTER,
    field_color      BYTE,
    pointer_1         POINTER,
    pointer_2         POINTER,
    pointer_3         POINTER,
    pointer_4         POINTER};
```

The table used for the numeric and string display will be of the above structure. This table will be one referenced with the zeroth element containing information about the table.



crt\_table structure elements :

-----  
display\_type =>

a byte - holds the case number to be displayed. This is a primary element entry specifying what type of data is to be displayed. It defines the usage of the pointer entries in the table element. The following will be the standard numerical meaning for common data types :

0 - null  
1 - signed byte  
2 - hex byte  
3 - unsigned byte  
4 - signed word (integer)  
5 - hex word  
6 - unsigned word  
7 - signed dword  
8 - hex dword  
9 - unsigned dword  
10 - real  
11 - indexed rmx string table entry  
12 - indexed asc string table element  
13 - signed word trend entry  
14 - unsigned word trend entry  
15 - signed dword trend entry  
16 - unsigned dword trend entry  
17 - clock time entry , dword  
18 - clock time and date entry, dword  
19 - single ascii string  
20 - clock time entry, word value  
21 - rmx string loc table using bit mask  
22 - alarm condition string table  
23 - point id string  
24 - report id string

A set of standard display and edit functions will be built using the display type as an index into a CASE statement.

The zero element entry will contain the number of elements in the table. This will be used in looping through the table for display and edit purposes and will limit the number of elements in the the table to 255.

edit\_flag =>

-----  
A byte, this entry is used to indicate whether the field is editable and at what password level the field is accessed.

Where 0 = is not editable.

1 to 9 the field is editable and assign to a certain security level.

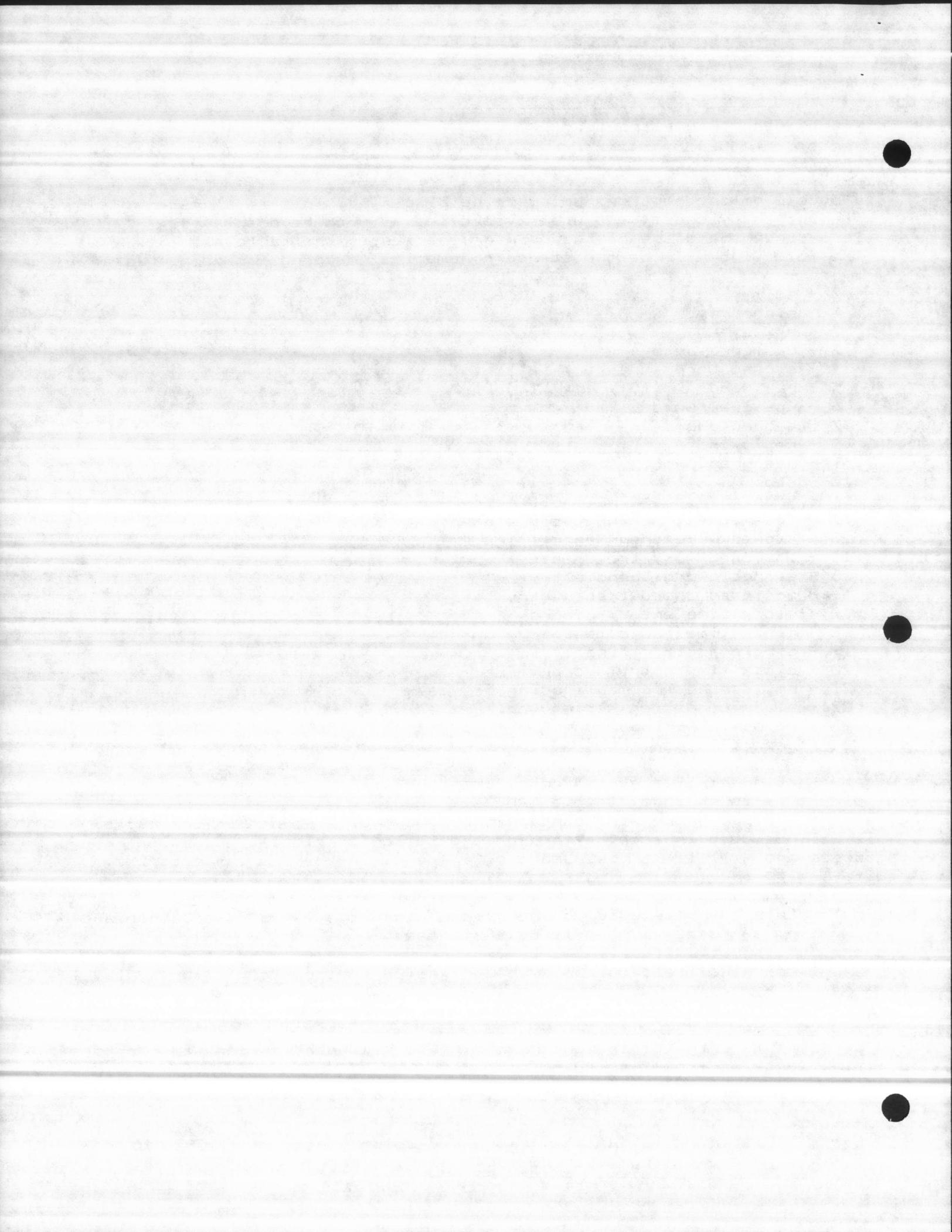
row =>

---

A byte - this entry is the one referenced to row entry at the start of the field. range is from 1 to 48.

column =>

-----  
A byte - this entry is the one referenced column entry of the start of the field. Range is from 1 to 80.



field\_length =>

A byte - holds the number of characters allowed in a field.

reg\_t\_p =>

A pointer points to a token that will used as the region token protecting the element's data. If the pointer is NIL or 0 then it is assumed that no region associated with the data.

field\_color =>

A byte - This entry is an index to a lead in string for foreground/background/attribute selection.

If this entry is set to 0 then it is assumed that color/attribute combination of single size, black background, white foreground will be used for the field display.

1 - black

2 - red

3 - green

4 - yellow

5 - blue

6 - magenta

7 - cyan

8 - white.

pointer\_1, pointer\_2, pointer\_3, pointer\_4 =>

Pointers. These four pointers will assume different meaning dependent upon the display type

For numeric data the following will be assumed :

+pointer1 : data location pointer points to byte, word, etc in RAM.

+pointer2 : scale\_loc\_p points to words scale.

+pointer3 : min\_max\_range\_p points to a RAM segment that will contain two variables the same type as indicated by the display type variable. This two variable array will be used in the edit procedure for the range evaluations.

+pointer4 : not used.

For rmx string data the following could be assumed :

+pointer1 : an index to a pointer points to an RMX string.

+pointer2 : rmx\_str\_p\_tbl points to an array of pointers.

+pointer3 : points to an edit prompt table.

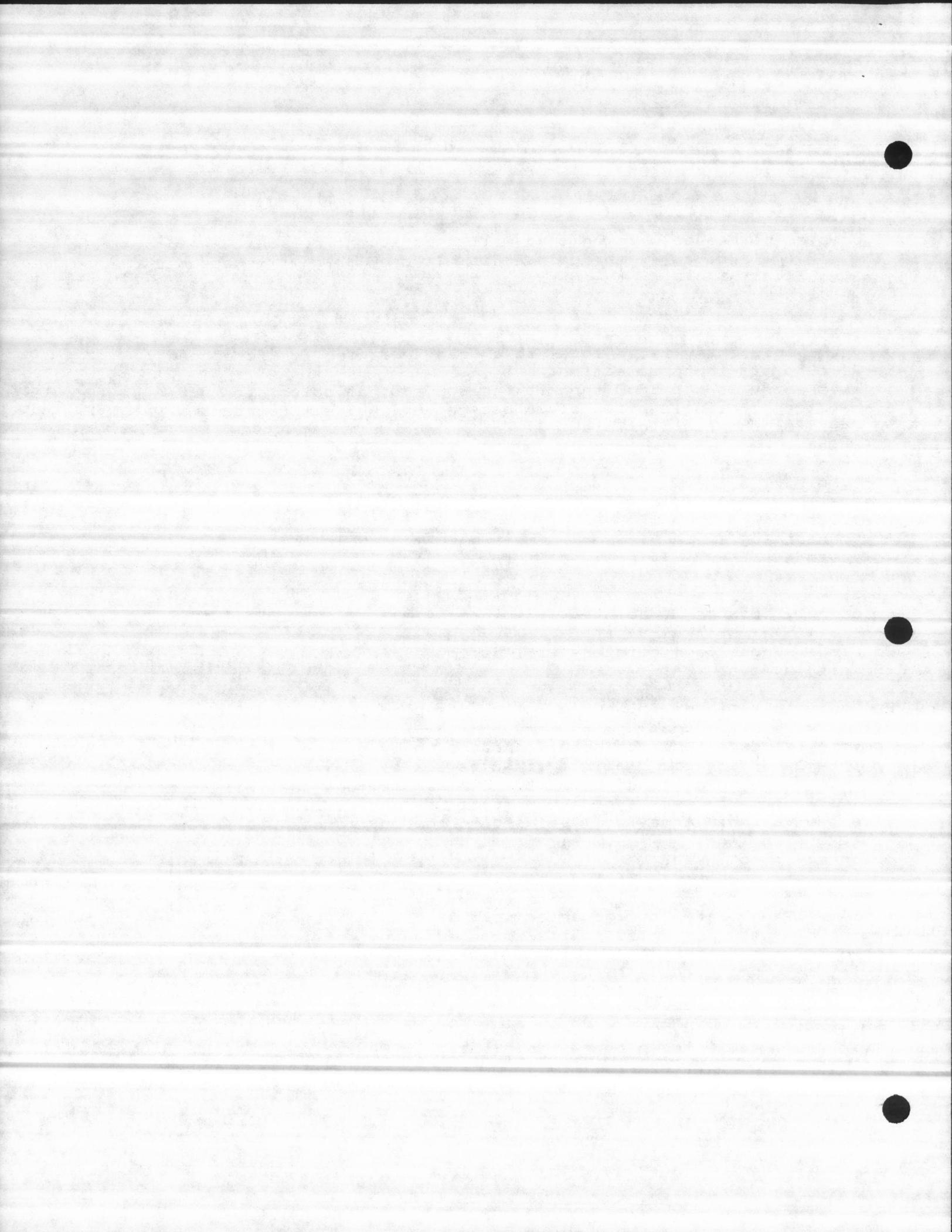
+pointer4 : not used.

For indexed string data following could be assumed :

+pointer1 : index\_p points to a word index.

+pointer2 : table\_loc\_p points to a table in RAM.

+pointer3 : unused.



NOTE THAT :

It must be noted that a table must be built with the row and column in ascending, that is, the elements must be ordered with the first element being the field closest to top left corner of the screen and the last element being the bottom left corner of the screen, or have an exref to each entry.

MODULE VARIABLES :

=====

```
DECLARE exception WORD EXTERNAL; /* declared in C4376KYBD */
```

LOCAL PROCEDURES :

=====

1) Prompt\_Display:

-----

```
PROCEDURE (crt_num, display_buf_t, crt_out_mbx_t,
           row, column, display_str_p) PUBLIC REENTRANT;
```

row => a byte holds the row number.

column => a byte holds the column number.

crt\_num => a byte holds the crt number, which is used to display on.

display\_buf\_t => a display buffer token.

crt\_out\_mbx\_t => a crt output mail box token.

display\_str\_p => a pointer points to a string to be displayed, called display\_str structure.

row\_column\_asc (4) => bytes hold the row and column ascii

Proc description :

This procedure will clear, format the passed rmx string, and send (via the passed crt\_mbx\_t to row 48 or 24, column 1 of the display.

END Prompt\_Display;

2) Format\_Cursor\_And\_Color:

-----

```
PROCEDURE (display_buf_t, row, column, color_index) PUBLIC REENTRANT;
```

row => a byte holds the row number.

column => a byte holds the column number.

display\_buf\_t => a display buffer token.

color\_index => a byte holds the color index.

color\_str\_p => a pointer points to the color string.

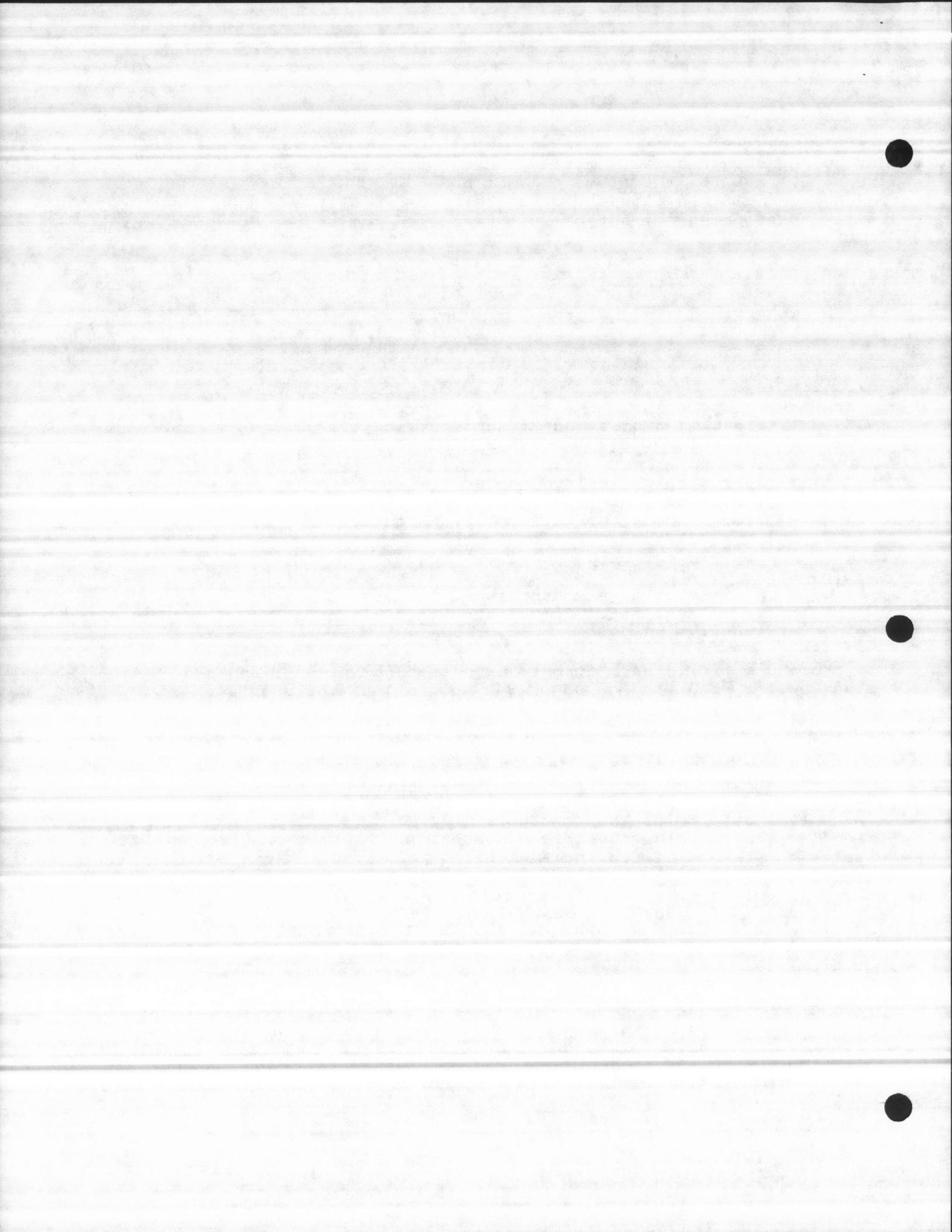
color\_count => a byte holds the color count.

row\_column\_asc (4) => bytes hold the row and column ascii

Proc description :

This procedure will format into the passed buffer the cursor position and a color string. If either row or column is zero the cursor position will not be formatted. If a zero is passed for the color\_str\_p then no color will be formatted. It is assumed that the buffer's region has been gained through the Clear\$Buffer call.

END Format\_Cursor\_And\_Color;



3) Prompt\_Entry: PROCEDURE (entry\_max\_count,crt\_num) PUBLIC REENTRANT;

crt\_num => a byte holds the crt number, whish is used to display on.  
entry\_max\_count=> a byte holds the entry max count.  
prompt\_entry\_buf\_t - formatted standard buffer,  
prompt\_entry\_count - independently kept buffer char count (any  
format call must also add number of chars  
to this variable),  
entry\_str - rmx string filled with keyed input.

Proc description :

This procedure will prompt the input entry.

END Prompt\_Entry;

Save\_Prompt\_Entry: PROCEDURE (save\_entry\_max\_count,crt\_num) PUBLIC REENTRANT;

crt\_num => a byte holds the crt number, whish is used to display on.  
entry\_max\_count=> a byte holds the entry max count.  
prompt\_entry\_buf\_t - formatted standard buffer,  
prompt\_entry\_count - independently kept buffer char count (any  
format call must also add number of chars  
to this variable),  
save\_entry\_str - rmx string filled with keyed input.

Proc description :

This procedure will save prompted entry.

END Save\_Prompt\_Entry;

Enter\_Password\_And\_Verify: PROCEDURE(crt\_num) WORD REENTRANT PUBLIC;

returns :

0 - password entered is incorrect or display invoked,  
1 to 3 : password entered has matched level  
(equal passwords will return highest level).  
display invoked by entering PATIODOOR as password  
default password is PASSWORD

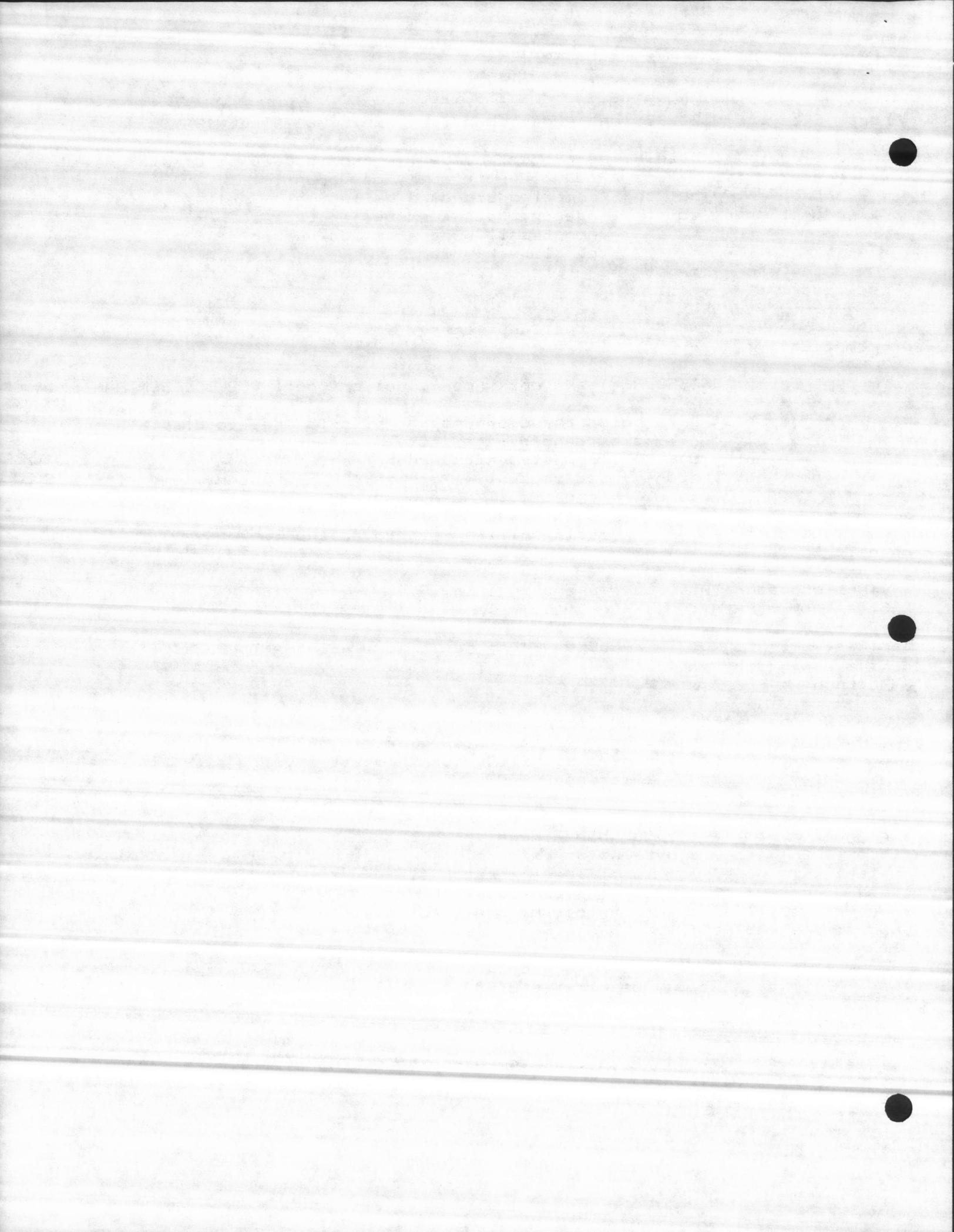
crt\_num => a byte holds the display crt number.  
index => a byte holds an index  
match => a flag, holds the match status.  
temp\_asc => a byte - as temp ascii  
password\_level => a word holds the password level  
backdoor\_str STRUCTURE :

count = a byte holds the string count.  
asc (9) = bytes hold the ascii string which is PATIODOOR.

Proc description :

This procedur will ask the user to enter the password, and make sure  
the the password is legal.

END Enter\_Password\_And\_Verify;



Get\_Password:

```
PROCEDURE (crt_num, table_p, table_reg_t,
          display_buf_t, display_mbx_t,
          ci_conn_t, ci_mbx_t) BYTE PUBLIC REENTRANT;

count      => a byte holds a count.
crt_num    => a byte holds the display crt number.
pass_level => a byte holds the password level.
table_reg_t => a token, used for the display table region protection.
display_buf_t => a token, used as a display buffer token.
display_mbx_t => a token, used as a display mail box token.
ci_conn_t   => a token used as a CI connection token.
ci_mbx_t    => a token used as a CI mail box token.
table_p     => a pointer points to the crt display table.
```

Proc description :

This procedure do the following :

- 1) get password
  - 2) get region of table RMX286 FIX
  - 3) check for table with higer pass editable fields or table being edited
  - 4) release control of region RMX286 FIX
  - 5) display error on bottom line when the screen editor not used correctly.
- for example:  
 IF you try to edit a command page this procedure will send the following message

Edit not possible : displayed screen  
 does not have any editable fields.

END Get\_Password;

Enter\_New\_Password: PROCEDURE (crt\_num) PUBLIC REENTRANT;

```
count      => a byte holds a count.
crt_num    => a byte holds the crt display number.
password_level => a word contains the password level.
result      => a word contains the result value.
```

Proc description :

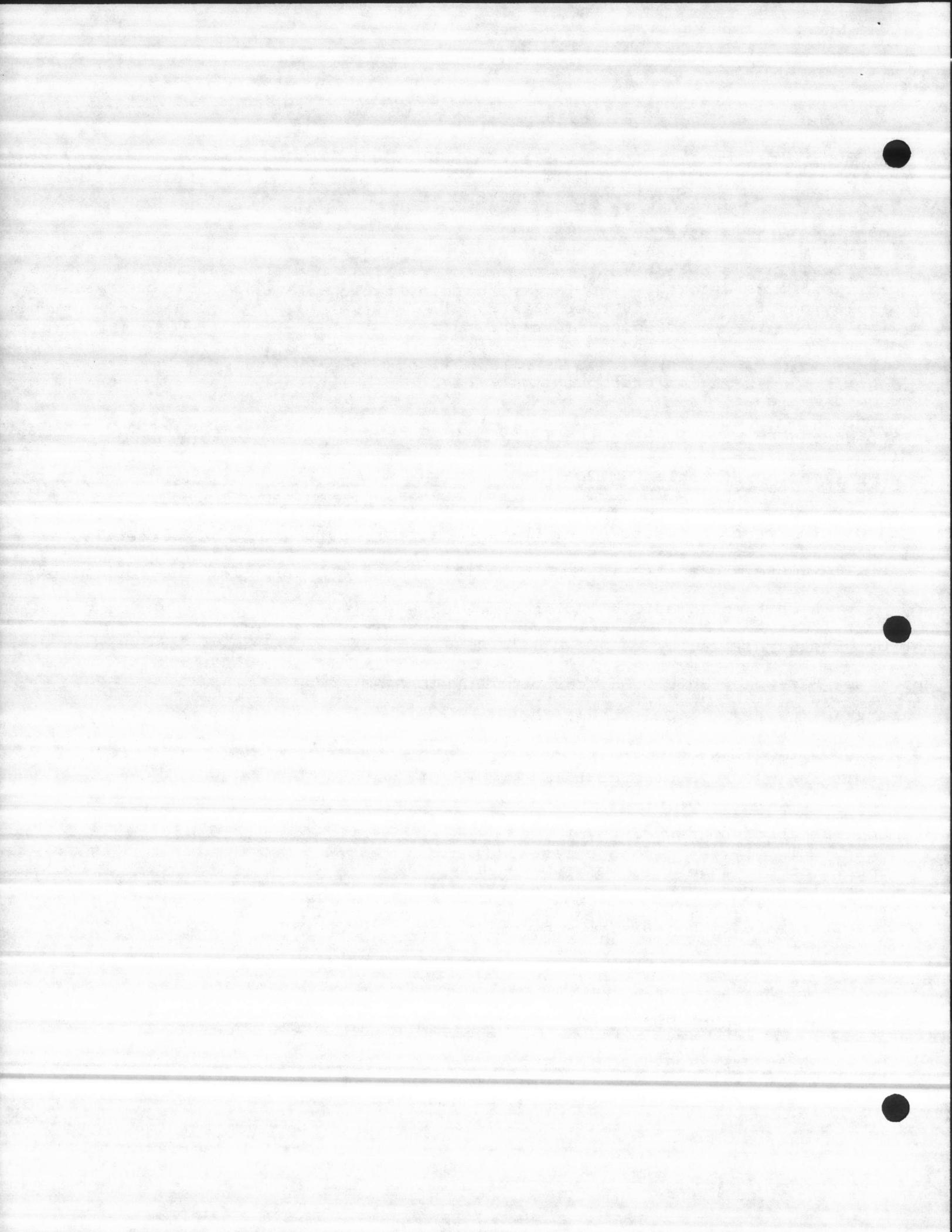
```
get operator's password level
query which level to change password
check out of range ( 1 <= password <= 3)
lower level cannot redefine upper password
enter new password
transfer entered password (allow only caps)
```

END Enter\_New\_Password;

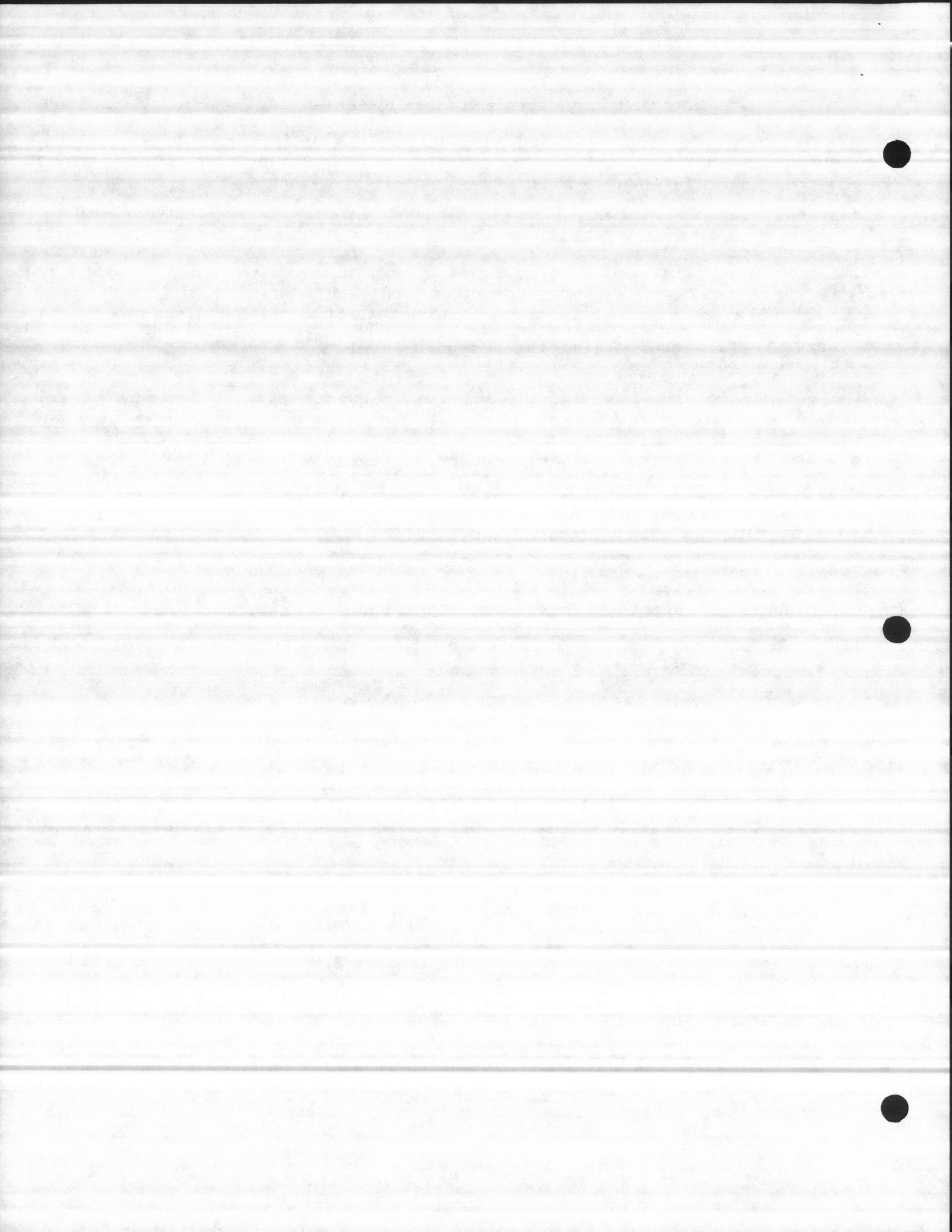
Edit\_Time\_Proc:

PROCEDURE (crt\_num) PUBLIC REENTRANT;

```
crt_num => a byte contains the crt number.
old_time_str (21) => bytes contain the previous time string.
new_time_str (21) => bytes contain the new time string.
```



```
-----  
Proc description :  
-----  
    get and store previous time  
    enter time  
    enter date  
    format message for printer  
  
END Edit_Time_Proc;  
  
Keyin>Edit_Sub_Menu_Proc:PROCEDURE (crt_num) PUBLIC REENTRANT;  
-----  
    crt_num      => a byte holds the crt number.  
    pass_level   => a byte contains the password level.  
    result_word  => a word value holds the index to the desired command.  
  
    edit_menu_command_count => a byte holds the number of command available.  
  
    edit_menu_prompt_asc (3) STRUCTURE  
        This structure will display the menu prompt ascii strings.  
  
    New Pass  => Allows changing of system password.  
    Screen    => Allows editing of screen entries.  
    Time      => Allows change of time and date.  
  
Proc description :  
-----  
    This procedure will display the screen editor command page.  
  
END Keyin>Edit_Sub_Menu_Proc;
```



7.1 MODULE NAME : CxxxxECAS.Pyy

=====

DESCRIPTION :

=====

This procedure will set up the nessary range display and call either  
Edit\_Field or Select\_Field.

LOCAL PROCEDURE :

=====

Report\_Id\_Search:

-----

This procedure searches for the report point ID.

Point\_Id\_Search:

-----

This procedure will search for the tag number.

Convert\_Date\_Time\_String:

-----

This procedure will convert the date and time string.

Write\_And\_Print\_Old\_String:

-----

This procedure will write and send the old string to the line printer.

Write\_And\_Print\_New\_String:

-----

This procedure will write and send the new string to the line printer.

Write\_And\_Print\_Data:

-----

This procedure will write and print the old and new data.

Write\_Data:

-----

This procedure will write the changes in numeric data.

Display\_Range:

-----

This procedure will display any field range if exist when the field is in edit mode.

Display\_Clock\_Range:

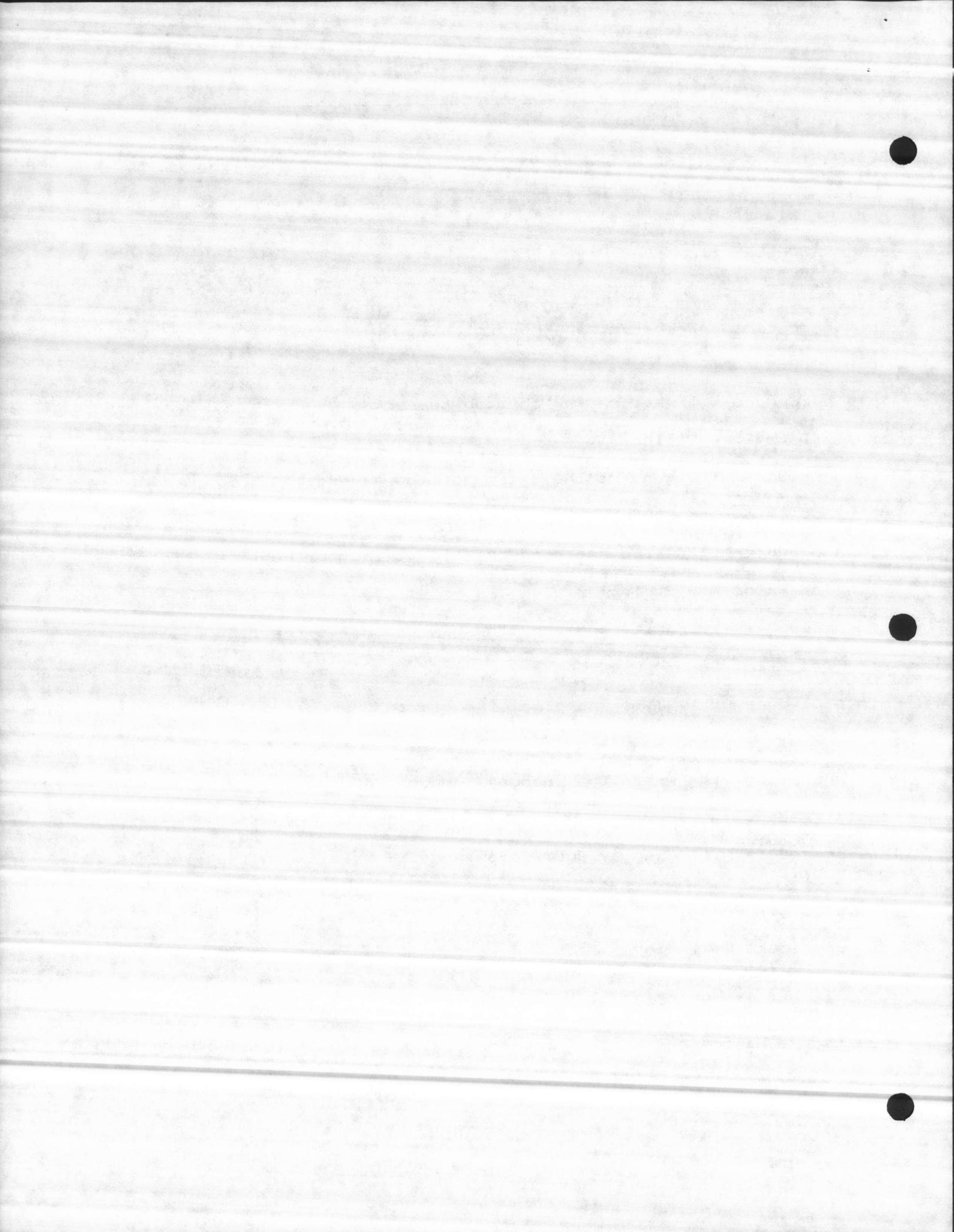
-----

This procedure will display the date and time ranges, when date, time fields in the edit mode.

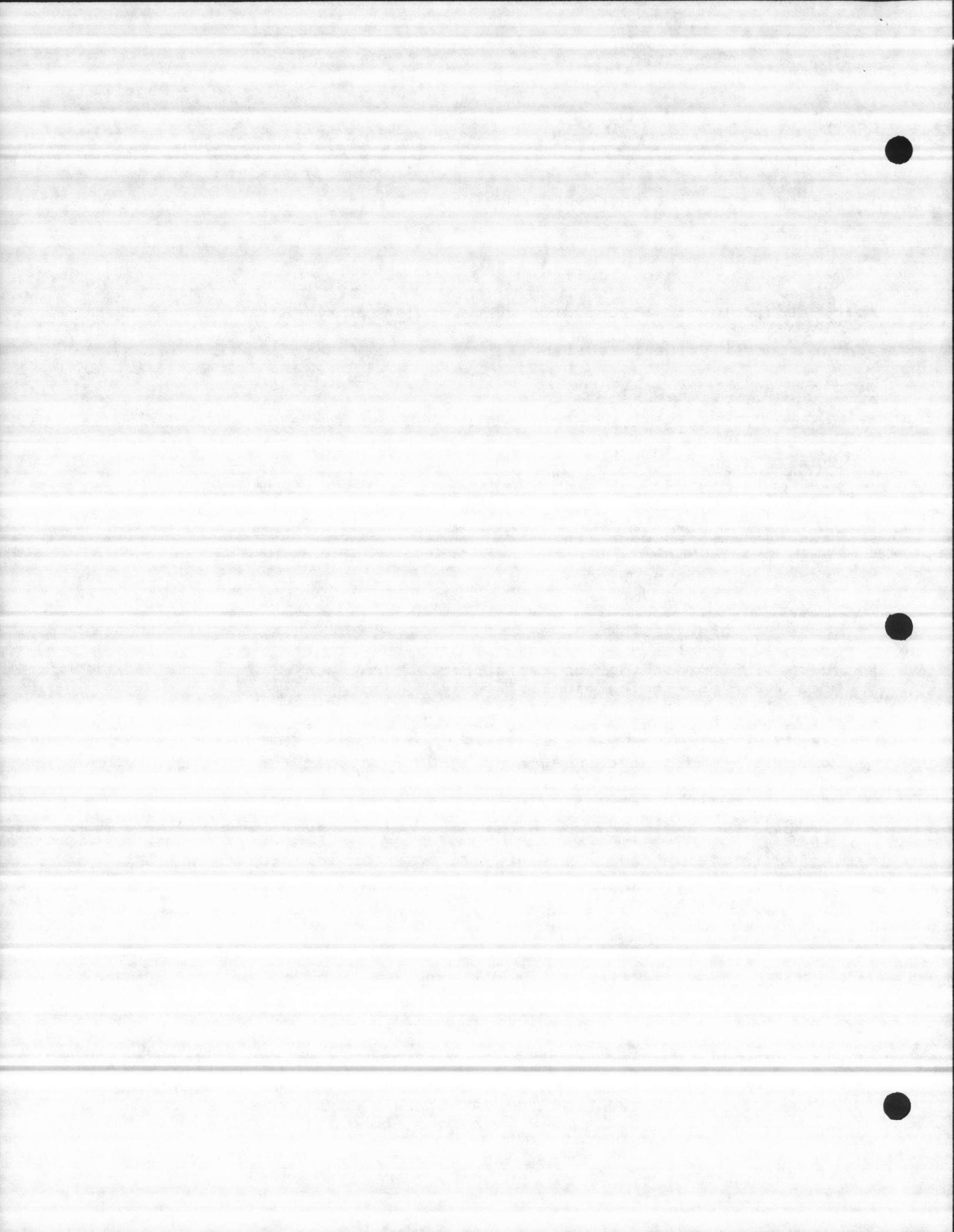
Edit\_Case:

-----

This is the major procedure in this module.



```
Algorithm:  
=====  
Edit_Case:  
-----  
main display and value entered check loop  
  
Loop FOREVER  
  
    format string into edit buffer  
  
    reformat display to insure displayed value hasn't changed  
  
    format current value in edit buffer  
  
    display rev video  
  
    DO CASE (display_type)  
  
        case 0 - null  
        case 1 - signed byte  
        case 2 - hex byte  
        case 3 - unsigned byte  
        case 4 - signed word (integer)  
        case 5 - hex word  
        case 6 - unsigned word  
        case 7 - signed dword  
        case 8 - hex dword  
        case 9 - unsigned dword  
        case 10 - real - null case  
        case 11 - rmx string loc table - select rmx string  
        case 12 - asc string std size table  
        case 13 - signed word trend entry  
        case 14 - unsigned word trend entry  
        case 15 - signed dword trend entry  
        case 16 - unsigned dword trend entry  
        case 17 - clock time entry  
        case 18 - clock date entry  
        case 19 - asc string  
        case 20 - clock time entry  
        case 21 - rmx string loc table select rmx string-using mask  
        case 22 - rmx string loc table select rmx string-using mask  
        case 23 - point id search  
        case 24 - report id search  
  
    END display_type CASE  
  
    main edit filed call : returns  
        0FFH for esc,  
        1 for default (trend edit cases 12 - 15)  
        0 for cr  
  
    check if esc entered  
    do neccessary range checking for each display_type  
    for checking, edit_result <> 0 means err found  
  
END forever  
  
END Edit_Case;
```



7.2 MODULE NAME : CxxxxDCAS.Pyy

=====

DESCRIPTION :

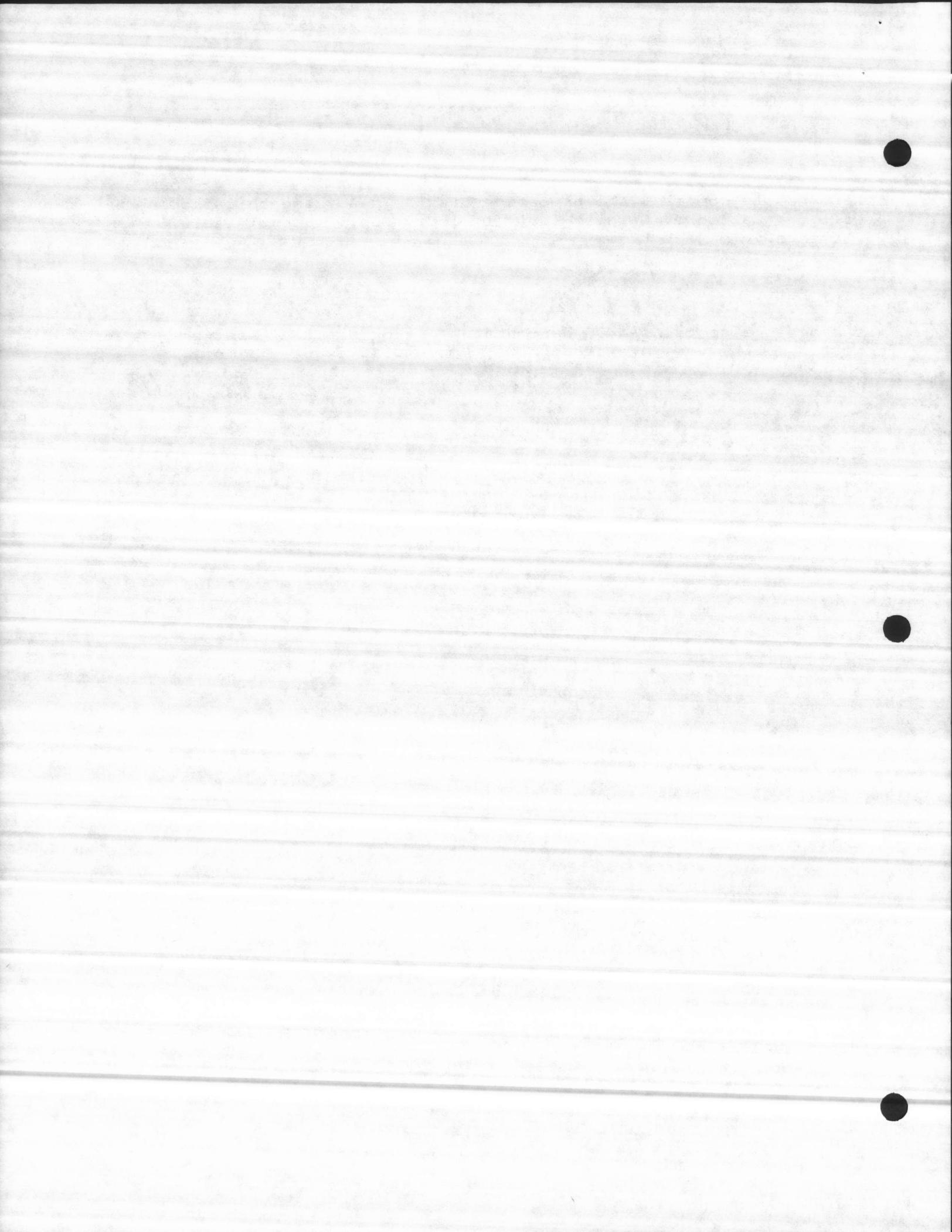
=====

This Module contains the procedure Display\_case, and this procedure will be Called by Display\_Table to individually format table elements.

Display\_Case:

-----

This procedure contains a DO CASE statement for display\_type for formatting each type in a certain way to be displayed.



7.3 MODULE NAME : CxxxxETBL.Pyy

=====

DESCRIPTION :

=====

This procedure is given the current table index and returns with the index of the editable field below or above it in the table. Wrap around is done when the bottom or the top of the table is encountered. These procedure uses the first\_table\_index and last\_table\_index variables that are initialized when Screen>Edit\_Proc is called. Caution must be used with these procedures because they may return the same index as the entered index if only one editable field exists. Also, if multiple calls are done to these procedures one may end up back at the original index.

LOCAL PROCEDURES :

=====

Find\_Next\_Field :

-----

increment index (allowing roll over to beginning of table)  
continue incrementing table index until editable field found

Find\_Prior\_Field :

-----

decrement table index  
continue decrementing table index until editable field found

edit\_Table:

-----

This is the major procedure in this module.

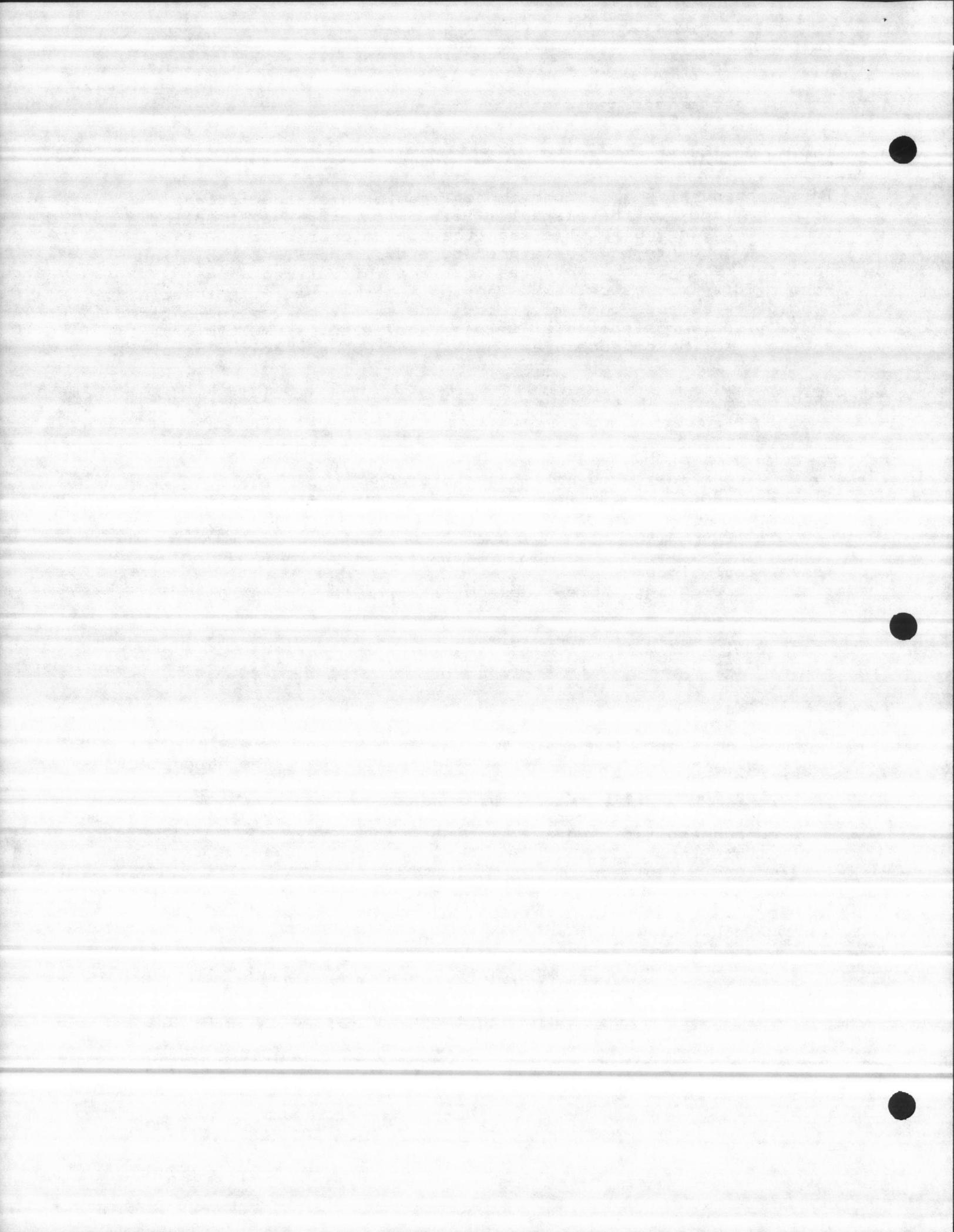
Algorithm :

=====

get region of table.  
set 80H bit of table(0).edit\_flag to mark that table is being edited.  
release control of region.  
init variables for usage.  
find first editable table entry  
    assign current table entry (will be HOME field)  
    first\_table\_index = count.  
find last editable table entry .

DO FOREVER;     start of main loop.

    get region of table.  
    mask 80H off previous index's edit\_flag  
    display selected field normal colors  
    mask 80H on current index's edit\_flag



```
release region
display selected field in rev video
get input character
check for esc
    display last selected field normal colors

get region of table

mask 80H off current index's and table's edit_flag

release region

act upon input character

DO CASE char_case;

case 0 : invalid input, sound bell

case 1 : cursor forward

    get next editable field index
    set display flag

case 2 : cursor backward

    get prior field index
    set display flag

case 3 : cursor up

    count variable used as error (drop through) flag
    find first editable field in previous row
    determine which field column closest to previous position
    calculate distances between columns
    select count's field if shorter distance
    distance calculations
    column determination
    set display flag

case 4 : cursor down

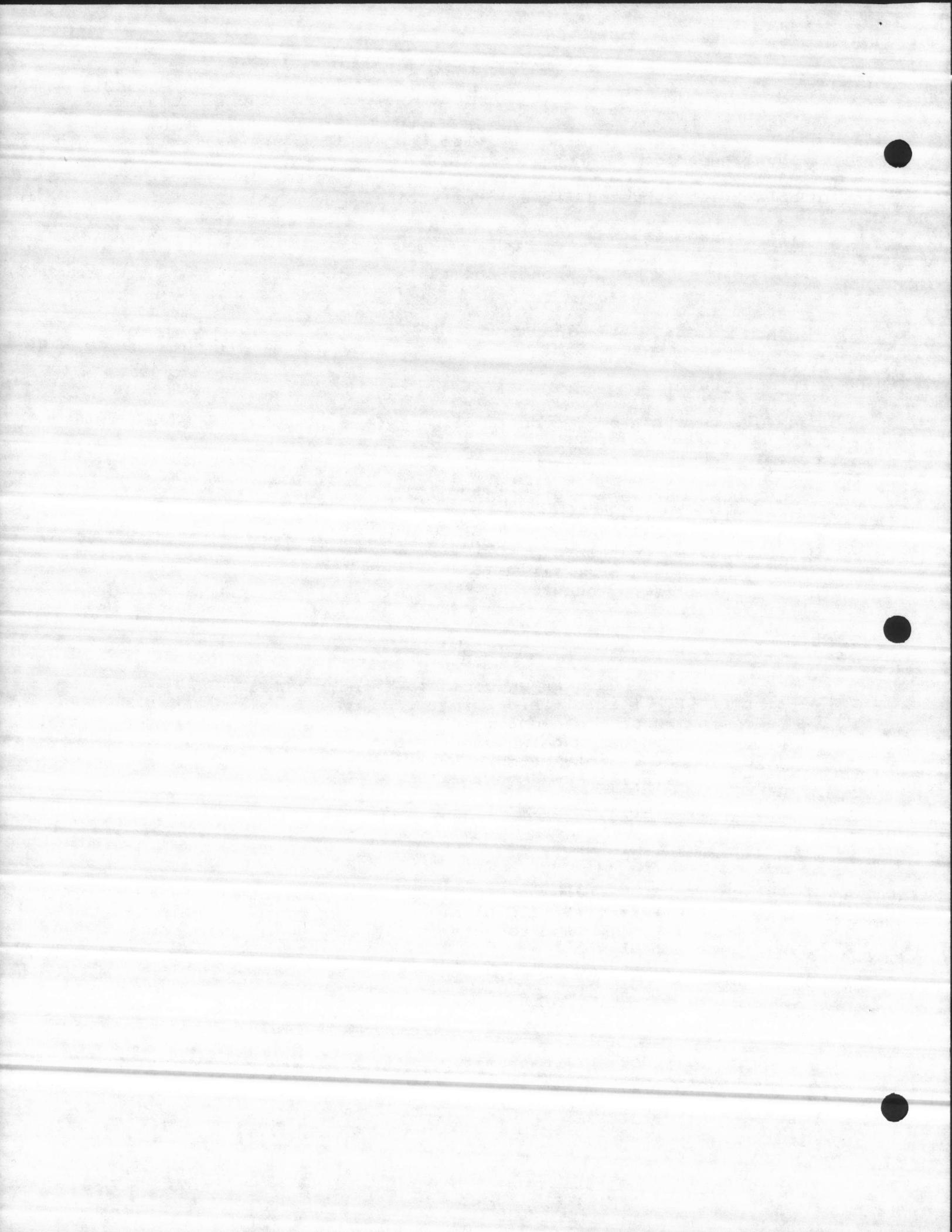
    determine which field column closest to previous position
    find prior editable field within row
    select count's field if shorter distance
    distance calculations
    column determination
    set display flag

case 5 : home

    set table index to first editable field
    set display flag

case 6 : enter edit "E"

END char_case
clear bottom line messages
```



7.4 MODULE NAME : CxxxxDTBL.Pyy

=====

DESCRIPTION :

=====

Displays (Single or Multiple CRT TABLE)

LOCAL PROCEDURES :

=====

Display\_Table:

-----

This is the major procedure in this module.

ALGORITHMS:

=====

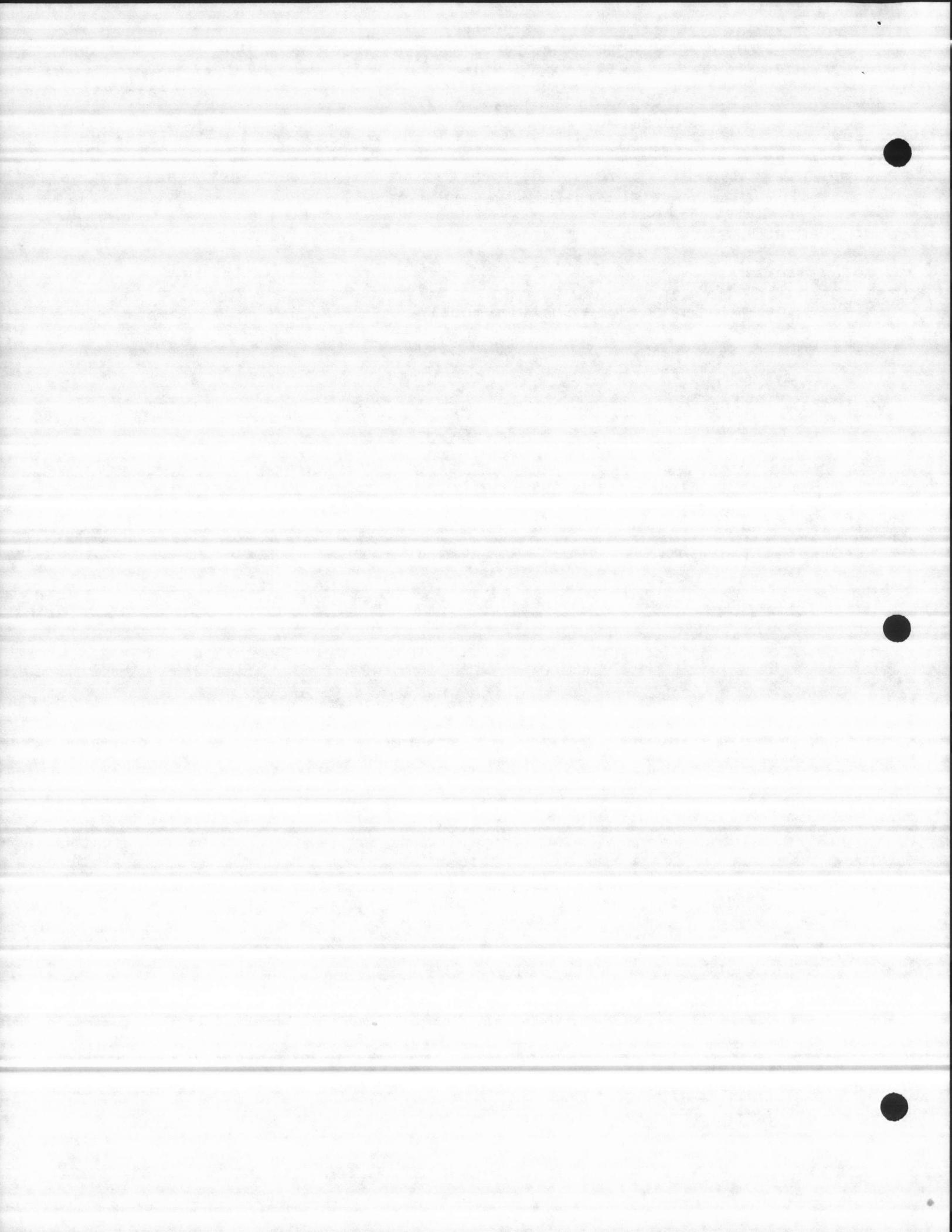
go thru all elements in table and format

get table region\_t

transfer values for basing

release table region

END Display\_Table.



7.5 MODULE NAME : CxxxxEFLD.Pyy

=====

DESCRIPTION :

=====

Edit Field .(Multiple Crts)

LOCAL PROCEDURES:

=====

Edit\_Field:

=====

This is the major procedure in this module.

ALGORITHMS :

=====

Basing for edit buffer access

Declare field characters and an xref to thier values.

Declare the special characters used by the screen editor.

Base pointer for numeric scale

Initialize char\_limit variable : check to limit certain chars :

0 = numerical only;  
1 = numerical only and sign;  
2 = numerical and hex;  
3 = numerical and multiple minus signs  
4 = all alpha

Initialize min, max buffer index

If decimal at beginning or end of field alter min, max

DO FOREVER;

IF beep\_flag = true THEN

    BEEP, BEEP, BEEP, BEEP

IF display flag is true

    Do the following

        display previous character on white background.

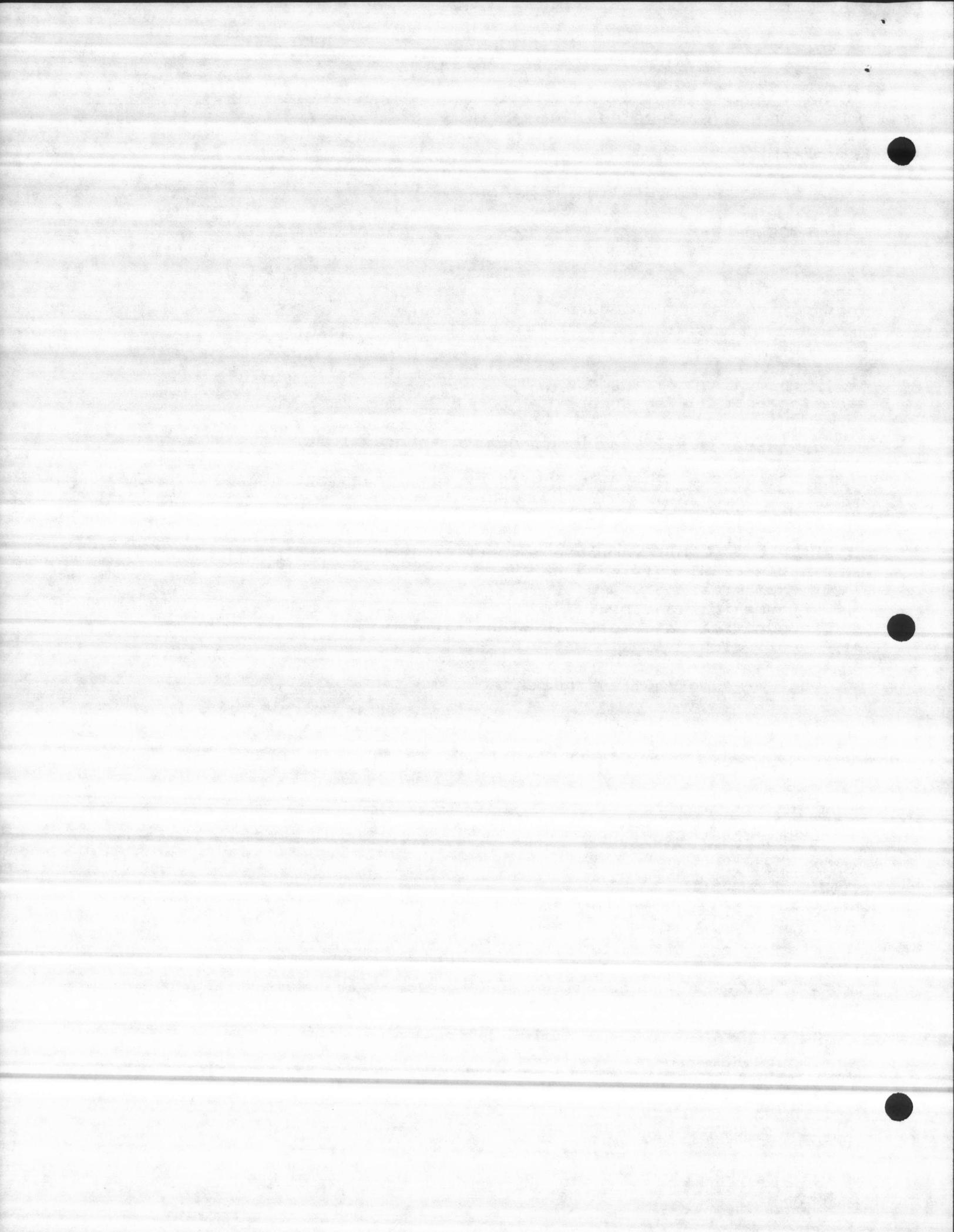
        display current character in rev video.

End.

get input character

check for esc character

control char entered



```
*** space entered ***
all characters allowed.

allow space only in front of number
allow entry only at beginning of string
other space exist, allow only at end or replacement

*** minus sign entered '-' ***

DO case;
    0 = numerical only (allows no '-')
    1 = numerical only and sign
    2 = numerical and hex (hex allows no '-')
    3 = numerical and multiple minus signs
    4 = all alpha (alpha allows all)

IF (char > 20H) AND (char <> 2DH) THEN

    DO CASE (char_limit);
        case 0 = numerical only
        case 1 = numerical and sign
        case 2 = numerical and hex
        case 3 = numerical and multiple minus signs
        case 4 = all alpha

perform function defined by char case

DO CASE char_case;

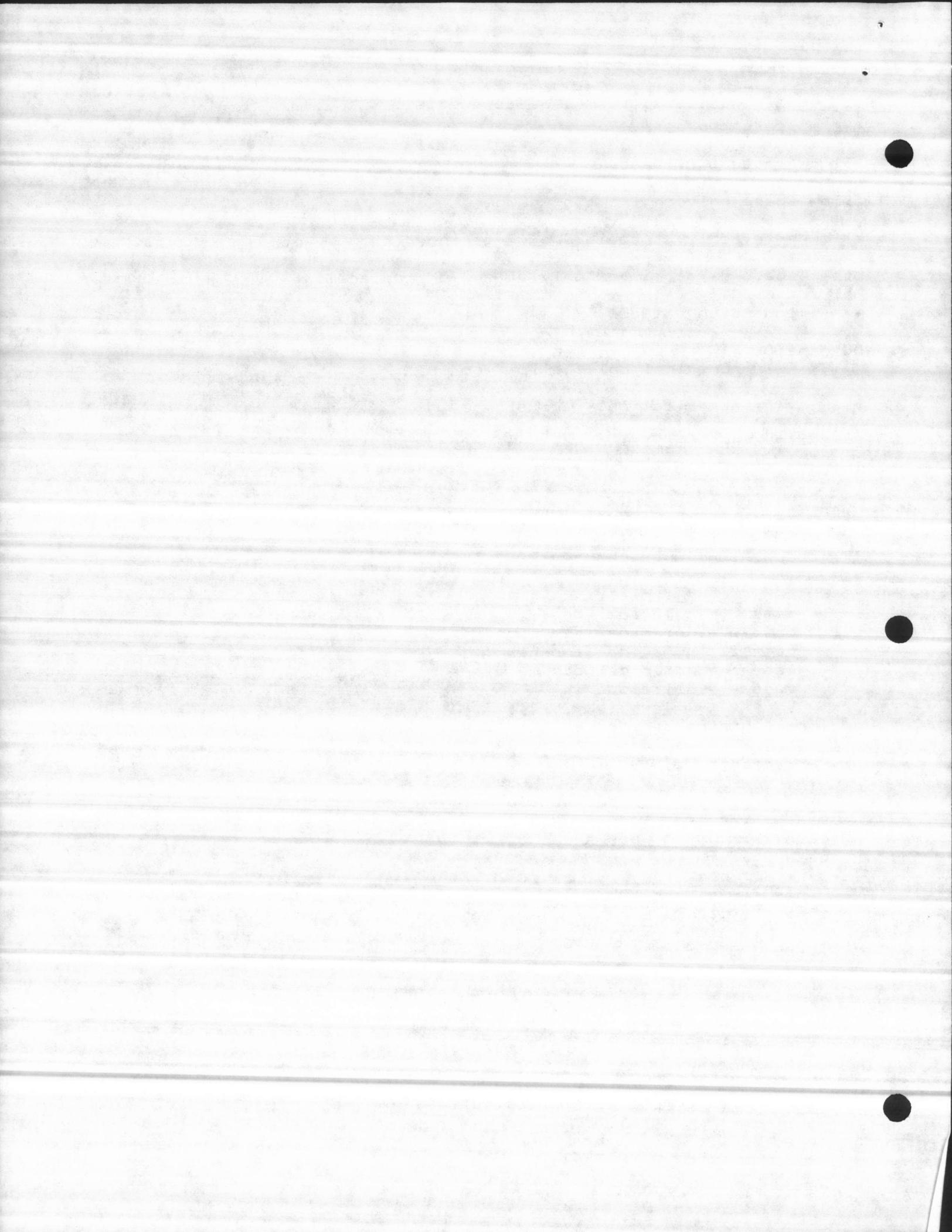
    case 0 : invalid input, sound bell
    case 1 : cursor forward
    case 2 : cursor backward
    case 3 : home
    case 4 : back tab
    case 5 : forward tab
    case 6 : carriage return (evaluate entry)

    DO CASE char_limit;

        case 0 : numerics (no check for sign)
        case 1 : numerics (check for sign)
        case 2 : hex
        case 3 : numerics and multiple signs
        case 4 : alpha

        case 7 : enter character into buffer

End forever loop
```



7.6 MODULE NAME : CxxxxSFLD.Pyy

=====

DESCRIPTION : Uses converted Menu\$Driver to select an index to a table  
===== entry.

ALGORITHM :

=====

Select\_Field:

initialize variables;

transfer values for basing;

Loop FOREVER;

    display selected index in field;

    format cursor and color;

    base rmx.string from address in table;

    string will be concatenated if longer than field length;

    If So

        loop;

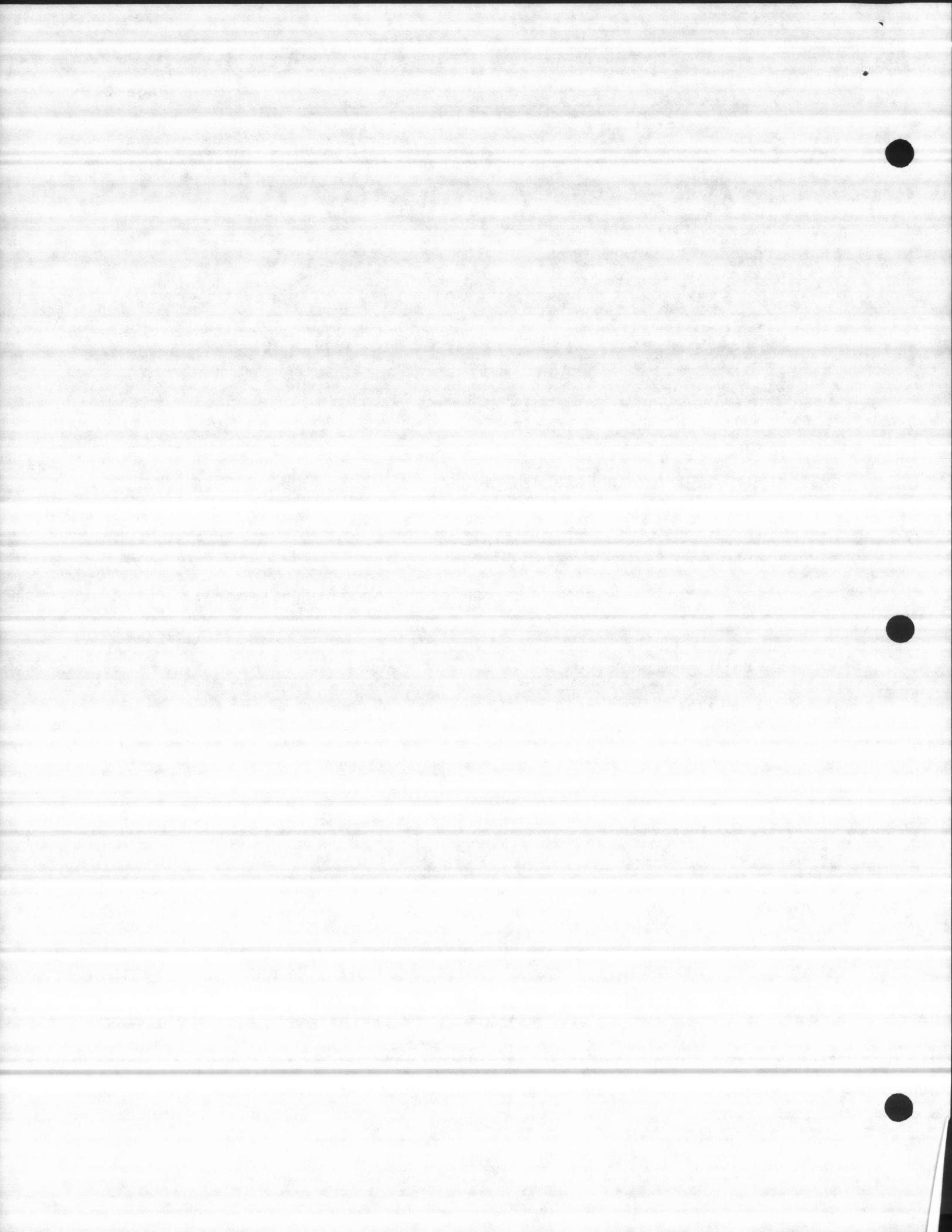
            trail space fill when string shorter than field length;

        call Menu\$Driver;

        get data region token;

        release data region;

END Loop Forever;



7.7 MODULE NAME : CxxxxCPRM.Pyy

=====

DESCRIPTION : Used to check the change in any parameter in the CRT table.

=====

ALGORITHM :

=====

Check\_Param\_Change:

```
basing for region  
basing for string tables  
basing for table index masks  
basing for time formating  
    start of procedure  
transfer values for basing  
return_byte = 1;
```

```
IF (table.display_type <= 24) THEN  
    gain control of region if pointer to token non-zero
```

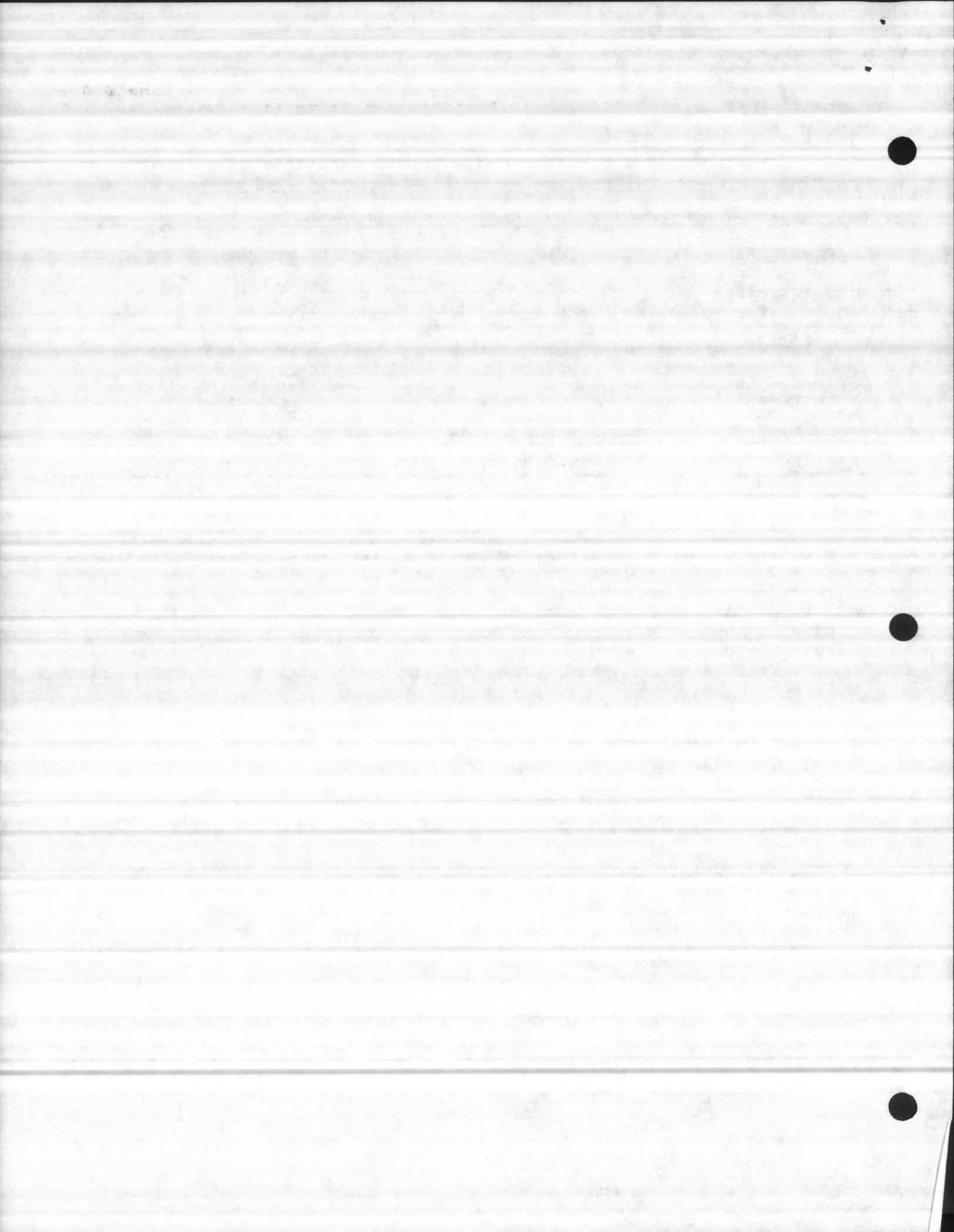
```
DO CASE (table.display_type);      main case
```

```
case 0 - null  
case 1 - signed byte  
case 2 - hex byte  
case 3 - unsigned byte  
case 4 - signed word (integer)  
case 5 - hex word  
case 6 - unsigned word  
case 7 - signed dword  
case 8 - hex dword  
case 9 - unsigned dword  
case 10 - real  
case 11 - rmx string loc table  
case 12 - asc string std size table  
case 13 - signed word trend entry  
case 14 - unsigned word trend entry  
case 15 - signed dword trend entry  
case 16 - unsigned dword trend entry  
case 17 - clock time entry  
case 18 - clock date entry  
case 19 - single ascii string  
case 20 - clock time entry  
case 21 - rmx string loc table using bit masks  
case 22 - asc string std size table  
case 23 - point id string  
case 24 - report id string
```

```
END;  main table.display_type CASE
```

```
    release region if has been gained  
END;  range check on table.display_type
```

```
RETURN  return_byte;  
END Check_Param_Change;
```



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 8

A L A R M S Y S T E M

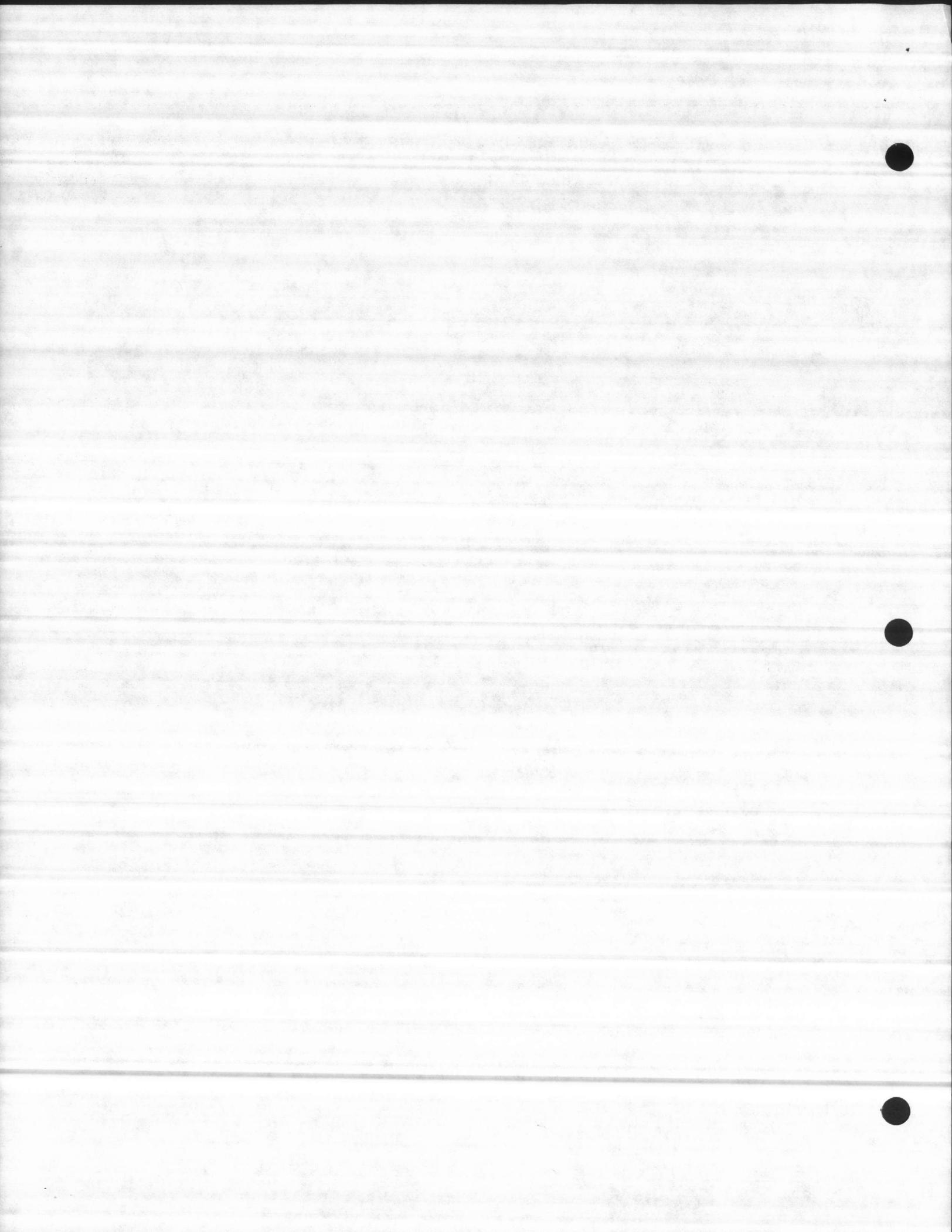
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

The Alarm system - written by Bob Ryan, Peter Wollenzien

(8.1) MODULE NAME : CxxxxALRM.Pyy

=====

DESCRIPTION :

=====

To display the active alarms in the system.

EXTERNAL MODULES :

=====

```
Format_Event_Entry: PROCEDURE (buffer_t, time_dword,
                                fail_type, index, condition) EXTERNAL;
DECLARE fail_type     BYTE,
        (index,
         condition) WORD,
        buffer_t      TOKEN,
        time_dword    DWORD;
END Format_Event_Entry;
```

GLOBAL DATA REFERENCE :

=====

See (CxxxxGLOB.Pyy & CxxxxHIST.Pyy)

MODULE DECLARATIONS :

=====

```
alarm_page          => 3 bytes one for each CRT. Each holds alarm page
                           number.
exception           => 3 words, each holds an exception condition.
alarms_update_sem_t=> 3 tokens, each for alarm update semaphore token.

alarm_head          = A string of bytes - 'ACTIVE ALARM SUMMARY'

alarm_sub_heading   = A string of bytes hold the alarm page sub heading.

alarm_update_table  => A pointer points to the Alarm_Update_Task.
alarm_delete_table  => A pointer points to the Alarm_Delete_Task.
```

MODULES PROC'S :

=====

Display\_Alarms\_Proc:

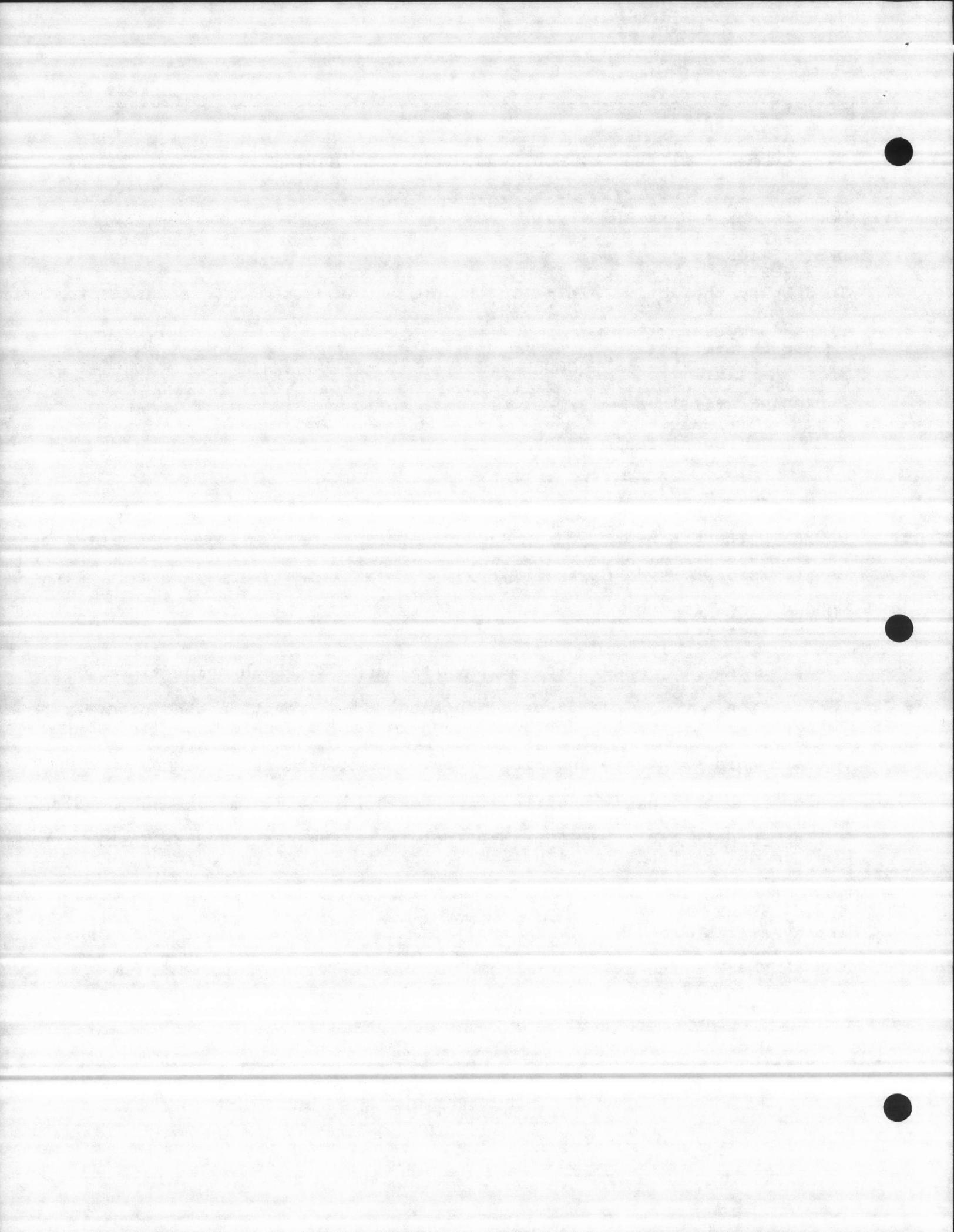
=====

Proc Description :

A public reentrant procedure to display the active alarm page.

Proc Declaration :

crt\_num => A byte, the number of the crt to display on.



Proc Algorithm :

-----

recieve control of crt update region.

stop previous display update.

set help\_display\_flag(crt\_num) to zero.

set crt\_table\_p(crt\_num) to NIL.

clear the screen before displaying the alarm page

set alarm\_page(crt\_num) = alarms\_display\_page(crt\_num).

set alarms\_display\_flag(crt\_num) to 0FFH.

set alarm\_display\_type(crt\_num) to zero.

call set\$update.

release control of crt update region.

end of Display\_Alarms\_Proc.

Alarm\_Update\_Task:

-----

Task description :

-----

updates the active alarm page(s).

Task Algorithm :

-----

set crt\_num = update\_crt\_control.

set alarm\_type = alarm\_display\_type(crt\_num).

set alarm\_page\_size = 19 lines.

set color\_len = 4 bytes.

assign the following pointers :

crit\_alarm\_color\_p points to BLINK\_ON,RED.

crit\_silence\_color\_p points to BLINK\_OFF,RED.

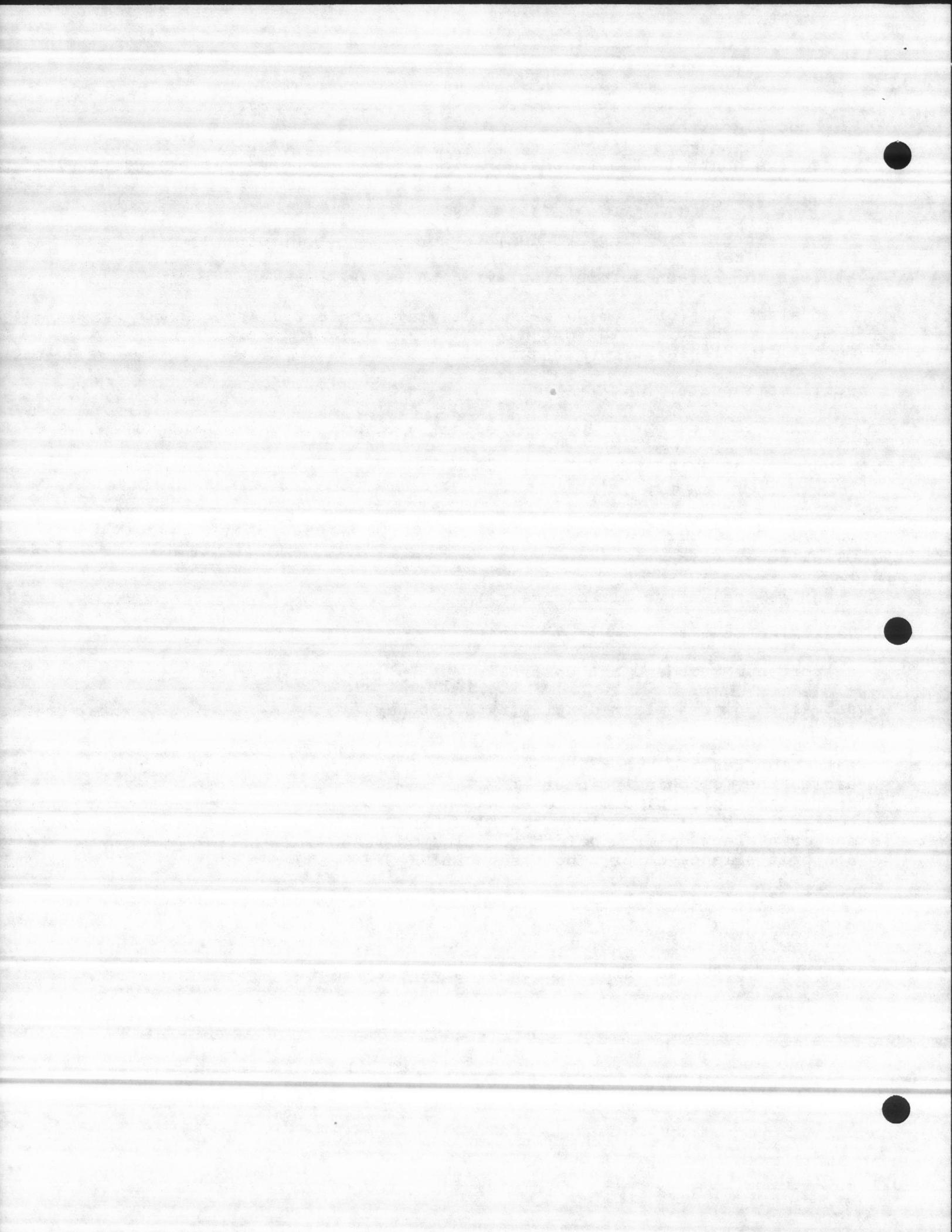
non\_crit\_alarm\_color\_p points to BLINK\_ON,YELLOW.

non\_crit\_silence\_color\_p points to BLINK\_OFF,YELLOW.

create alarm update semaphore.

create alarm update list.

CALL Rq\$Send\$Units using update\_receive\_crt\_sem\_t.



```
DO FOREVER;

    set remaining_units = 1;
    DO WHILE (remaining_units > 0);
        remaining_units = Rq$Receive$Units using alarms_update_sem_t(crt_num)
    END;

    get the control of the crt update region.
    get control of the alarm region.

    set up display parameters.

    send control of the alarm region.

    CALL Heading to display active alarm page heading.

    display alarm page sub heading.

    DO count = 0 TO (alarm_page_size - 1);
        decreement alarm_count_temp.
        decreement last_ack_alarm_temp.

        determine alarm type
        IF alarm_type = Non Critical THEN
            display the non critical alarm line with non_critical alarm color.
        IF alarm_type = Critical THEN
            display the critical alarm with critical alarm color.

        CALL Format_Event_Entry.
    END;    do count.

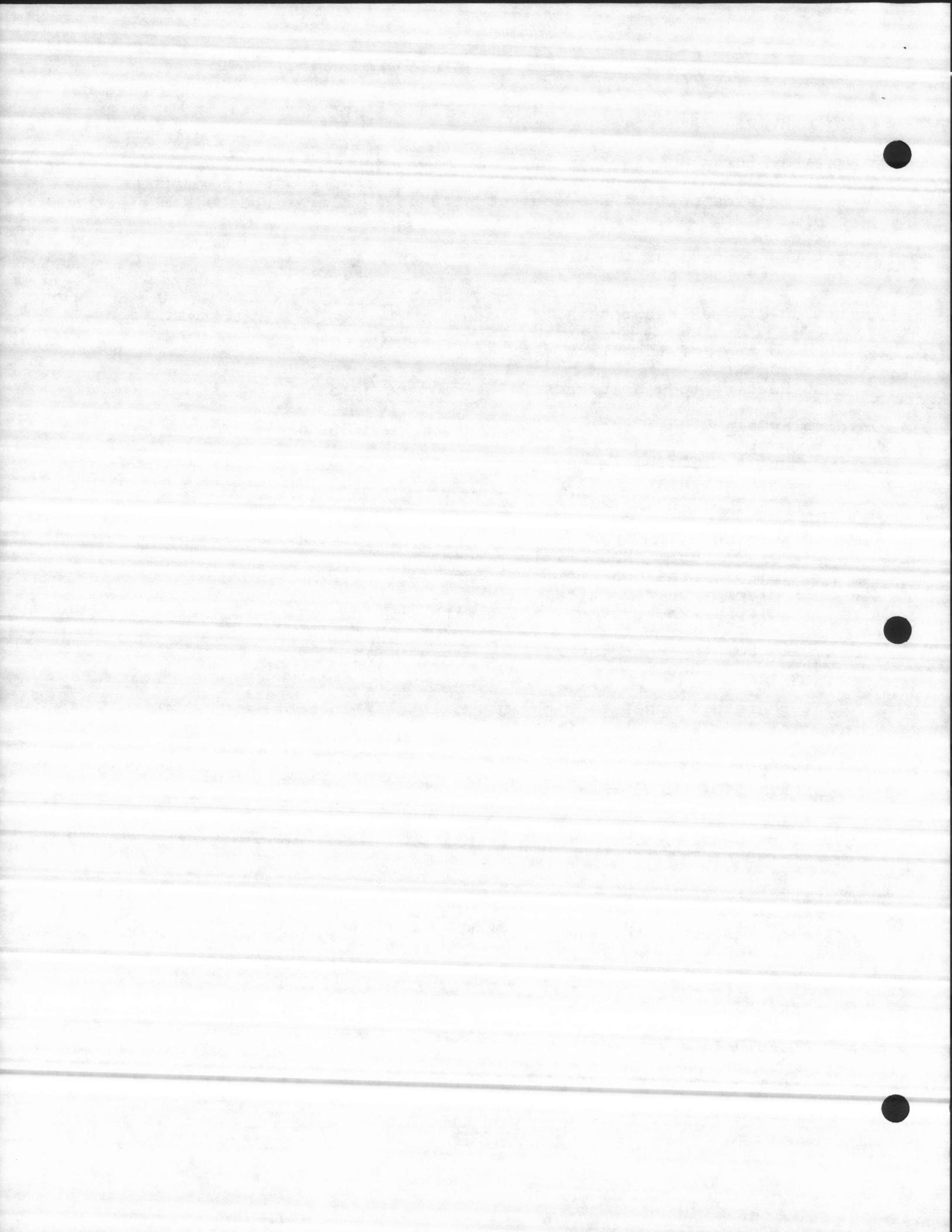
    Enter graphic mode, start blink, exit graphic mode,
            set blink rate to 5.

    release the crt update region

END;    do forever.

END Alarm_Update_Task.

-----  
Alarm_Delete_Task:  
-----  
crt_num => A byte, the number of the crt to display on.  
Task description :  
-----  
This task deletes the clock entry & the alarm update semaphore.  
END Alarm_Delete_Task;
```



(8.2) MODULE NAME : CxxxxASYS.Pyy

=====

DESCRIPTION :

=====

This module displays all the setup of the alarm system.

EXTERNAL MODULES :

=====

```
Display_Table: PROCEDURE (table_p,
                           param_p,
                           table_reg_t,
                           buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p)      POINTER;
End Display_Table;
```

GLOBAL DATA REFERENCE :

=====

See (CxxxxGLOB.Pyy)

MODULE DECLARATION :

=====

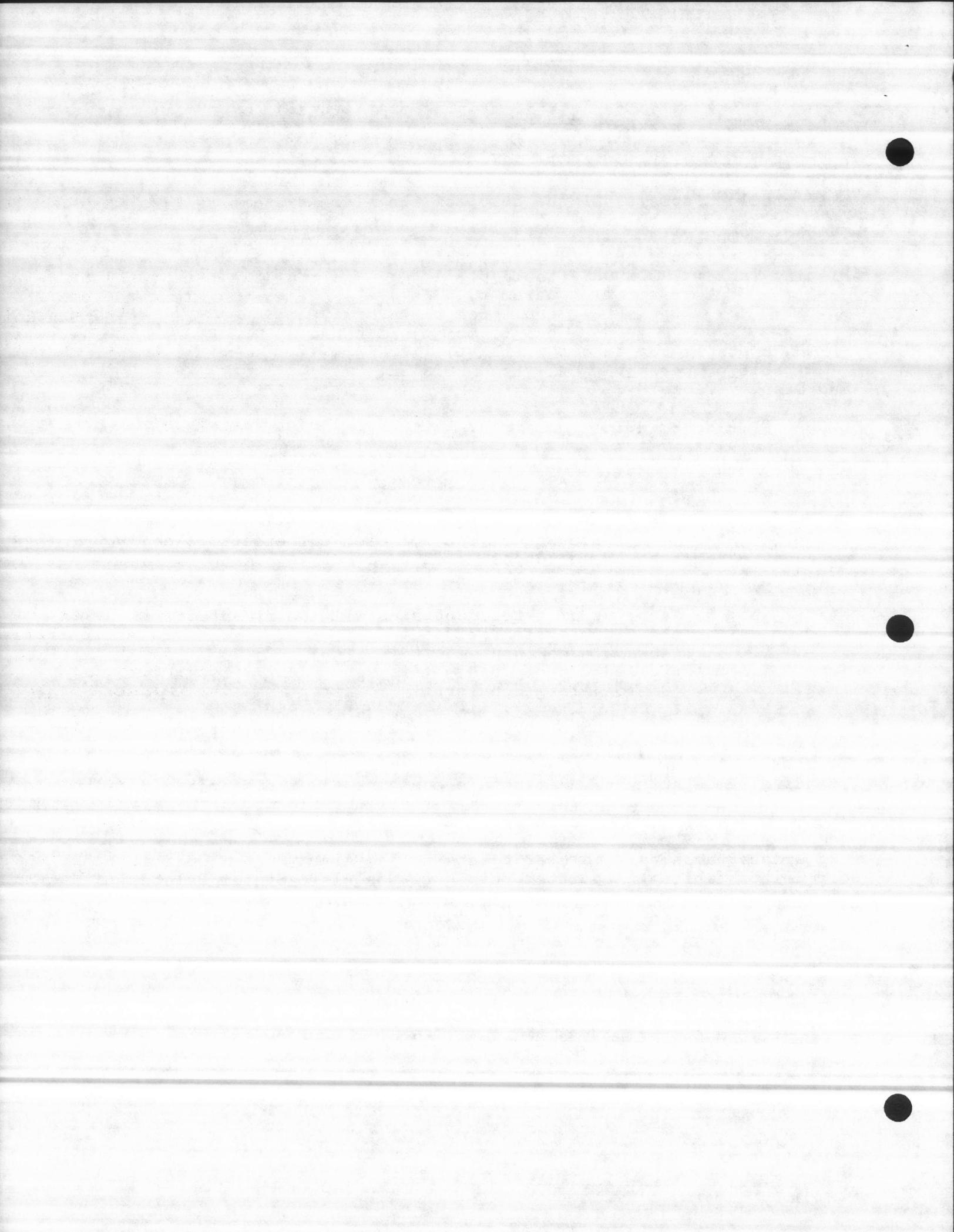
exception => Three words, one for each crt exception condition.  
crt\_update\_sem\_t => Three tokens, used to create a semaphore in the  
 update task.

static\_asc => A string of bytes, contains all the field asc for  
 the alarm system setup.  
'Acknowledge Input Id :'  
'Pulsed Output Id :'  
'Common Alarm Sonalert:'

dialer\_labels => A group of bytes contains the sub header of dialer  
 labels.

standard\_table\_entry => A structure contains all the display and edit  
 parameters for all the fields in the alarm system  
 display.

crt\_update\_table => A pointer points to the Alarm\_System\_Update.  
crt\_delete\_table => A pointer points to the Alarm\_System\_Delete.



-----  
MODULE PROC'S :

=====

Alarm\_System\_Display:

-----  
crt\_num => A byte, the number of the crt to display on.

Proc description :

-----  
This procedure displays the alarm setup.

End Alarm\_System\_Display;

-----  
Alarm\_System\_Update:

crt\_num => A byte, the number of the crt to display on.

remaining\_units => A word, used as an index.

result => A word BASED on result\_p used in Create\$Clock\$Entry.

Task description :

-----  
This task fills the crt table for the desired setup display.

END Alarm\_System\_Update;

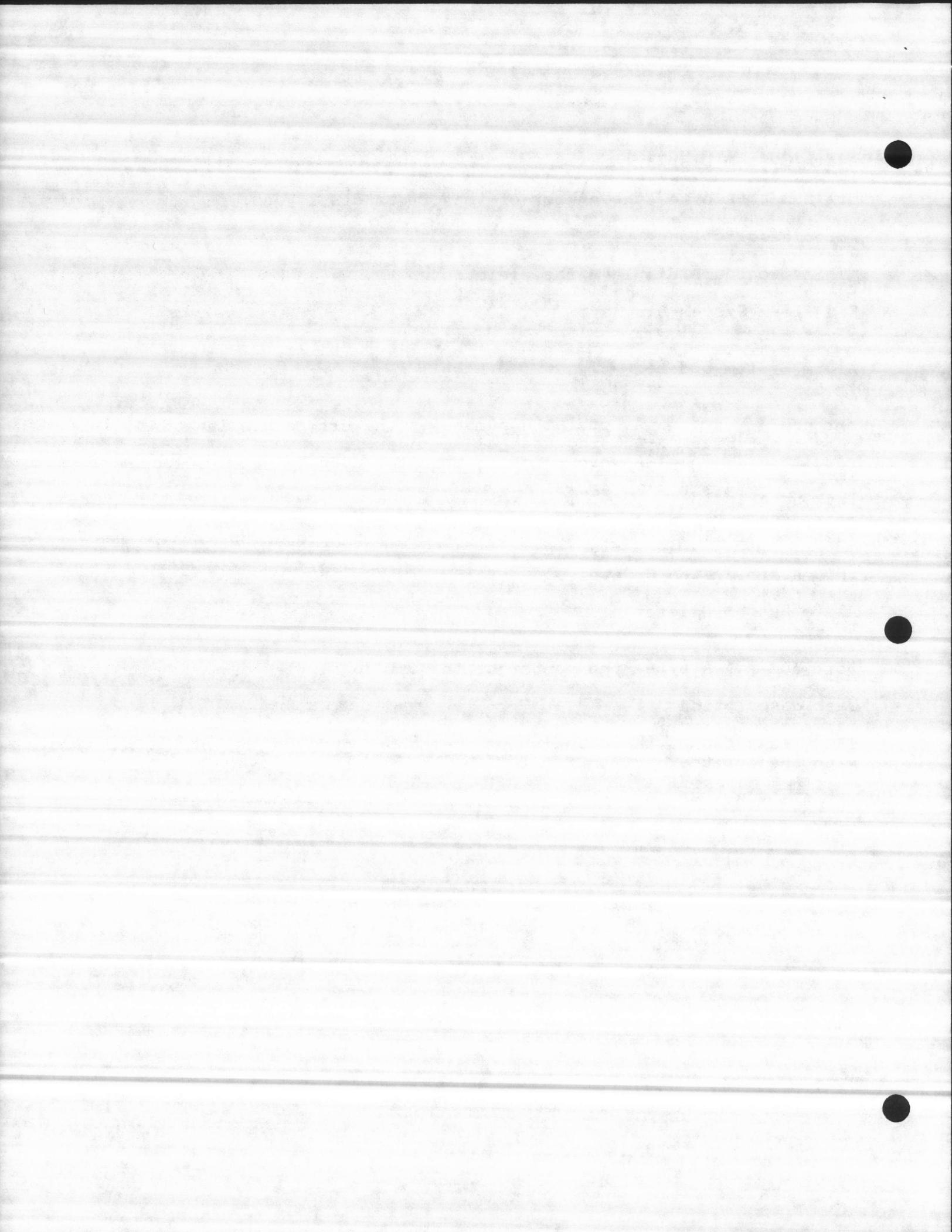
-----  
Alarm\_System\_Delete:

crt\_num => A byte, the number of the crt to display on.

Task description:

-----  
This task deletes the clock entry, semaphore and the segments associate with the crt tables.

END Alarm\_System\_Delete;



(8.3) MODULE NAME : CxxxxALOG.Pyy

=====

DESCRIPTION :

=====

This module displays all the alarms in the alarm log for the current day on demand or historical on demand & prints the alarms in the alarm log current file or historical file.

EXTERNAL MODULES :

=====

```
Format_Disk_Error: PROCEDURE (buffer_t, message_str_p, disk_error,
                               disk_operation) EXTERNAL;
  DECLARE (disk_error, disk_operation) WORD,
          buffer_t TOKEN,
          message_str_p POINTER;
END Format_Disk_Error;
```

```
Format_Disk_Event_Entry: PROCEDURE (buffer_t,
                                     alarm_log_entry_p) EXTERNAL;
  DECLARE (buffer_t) TOKEN,
          (alarm_log_entry_p) POINTER;
END Format_Disk_Event_Entry;
```

```
Format_Historical_Entry: PROCEDURE (buffer_t,
                                     alarm_log_entry_p) EXTERNAL;
  DECLARE (buffer_t) TOKEN,
          (alarm_log_entry_p) POINTER;
END Format_Historical_Entry;
```

```
Clear_Screen: PROCEDURE (crt_num) EXTERNAL;
  DECLARE (crt_num) BYTE;
END Clear_Screen;
```

```
Menu_Display: PROCEDURE (crt_num, result, off_set, start_id_p,
                         bytes_per_element, num_elements) WORD EXTERNAL;
  DECLARE (crt_num) BYTE,
          (result, off_set) WORD,
          (bytes_per_element, num_elements) WORD,
          (start_id_p) POINTER;
END Menu_Display;
```

MODULE DECLARATION :

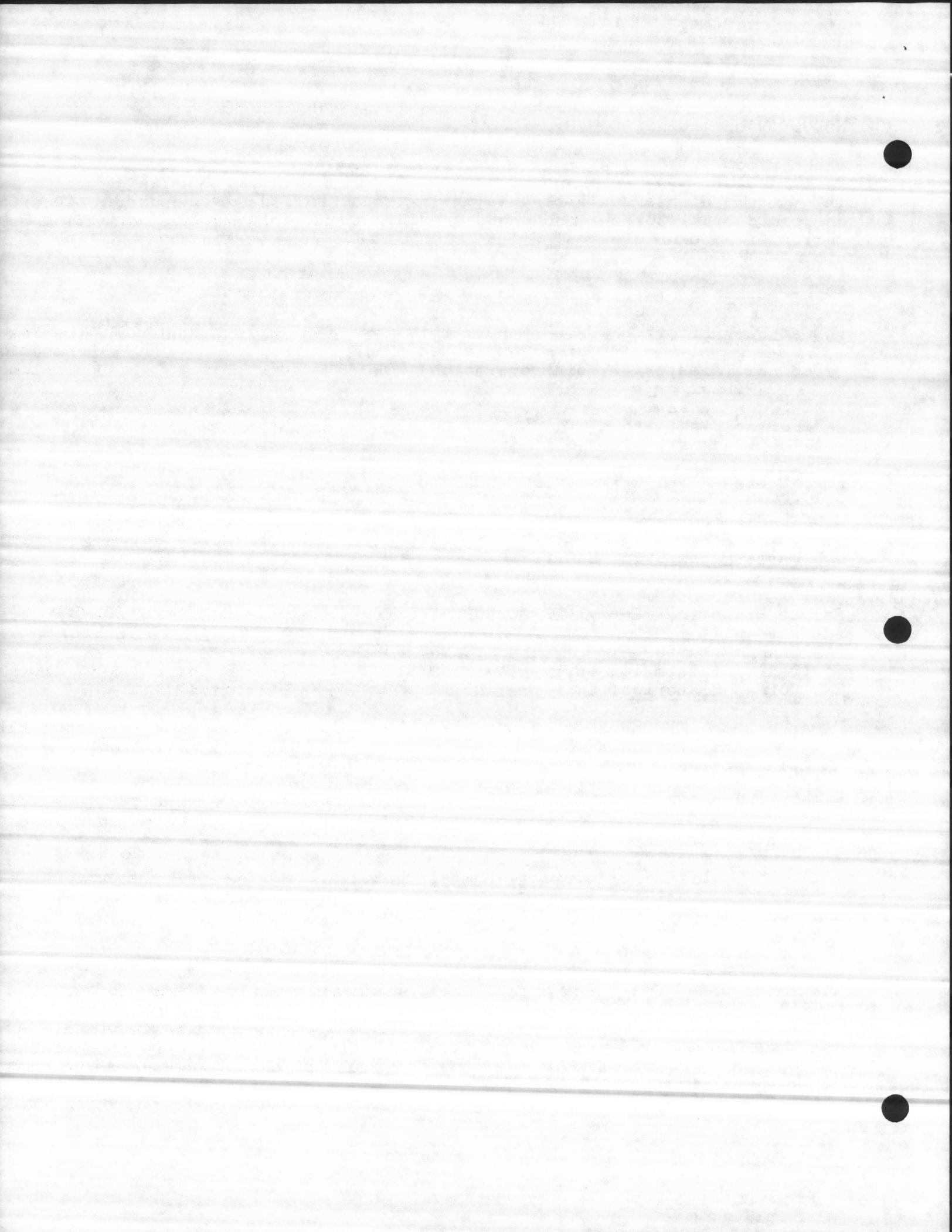
=====

alarmlog\_page => Three bytes, one for each crt, holds the alarmlog page number.

alarmlog\_type => Three words, one for each crt, holds the alarmlog type (non\_critical, critical).

exception => Three words, one for each crt exception condition.

alarmlog\_update\_sem\_t => Three tokens, used to create a semaphore in the alarmlog update task.



crt\_alog\_filename (3) :

This structure alarm log file name.

crt\_alog\_heading\_p => Three pointers, each points to the alarm log header.

crt\_alarmlog\_head => A string of bytes represent the alarm log header.  
crt\_his\_alarmlog\_head => A string of bytes represent the historical alarm log header.

crt\_alarmlog\_sub\_heading => A string of bytes for alarm log sub-heading.

crit\_alarm\_set\_color\_p => A pointer points to the critical alarm color.  
crit\_alarm\_reset\_color\_p => A pointers points to the critical alarm reset color.

non\_crit\_alarm\_set\_color\_p => A pointer points to the common alarm color.  
non\_crit\_alarm\_reset\_color\_p => A pointer points to the common reset color.

event\_color\_p => A pointer points to the event color.

ack\_color\_p => A pointer points to the ACK color.

alarm\_log\_header STRUCTURE :

-----  
file\_type => A word, historical or current.

element\_size => A word holds the element size which is initially 20 BYTES.

num\_elements => A word holds the number of elements of the alarm log header.

alarm\_log\_disk\_data (2) STRUCTURE:

-----  
time\_dword => A dword holds the alarm time, which indicates when the alarm occurs.

fail\_type => A byte value holds the fail\_type.

index => A word value holds an index for how many alarms occurred.

condition => A word value holds the alarm condition.

analog\_flag => A byte value holds an analog flag, which indicates if the alarm is an analog or not.

value => A word value holds the current analog value.

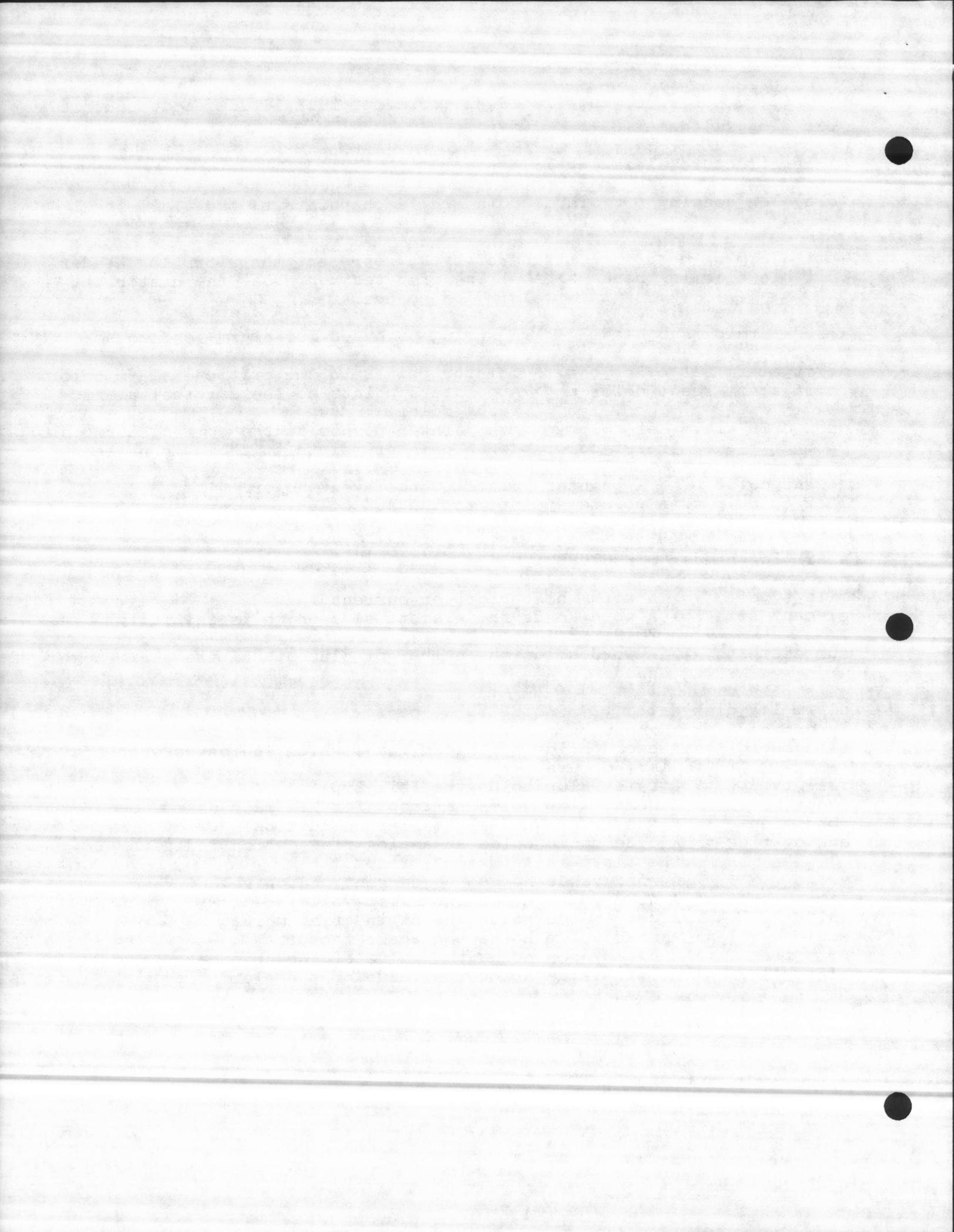
limit => A word value holds the limit value.

units => A word value holds the Engineering unit.

scale => A word value holds the scale factor.

crit\_flag => A byte value holds the value of the critical flag.

alarm\_flag => A byte value holds the value of the alarm flag.



read\_serial\_param BASED disk\_param\_seg\_t STRUCTURE:

This structure used to read the alarm log file.

alarmlog\_update\_table => A pointer points to Alarmlog\_Update\_Task.  
alarmlog\_delete\_table => A pointer points to Alarmlog\_Delete\_Task.

MODULE PROC'S :

=====

Read\_Num\_Disk\_Entries:

-----  
file\_name\_p => A pointer points to the alarm log file name.  
num\_entries => A word value holds the number of entries in the alarm  
log file.

Warrning :

-----  
MUST have control of disk\_param\_reg\_t before calling this procedure  
This is to guard against a condition at end of day where the alarm-  
log is reset, i.e., num\_entries would not be correct.

Proc description :

-----  
This procedure returns the number of entries in the alarm log file.

END Read\_Num\_Disk\_Entries;

Read\_Alarmlog\_File:

-----  
num\_elements\_offset => A word value holds the number of elements offset.  
file\_name\_p => A pointer points to the alarm log file name.  
alog\_disk\_data\_p => A pointer points to the alarm log disk data.  
num\_elements\_read\_p => A pointer points to the number of elements read.  
print\_error\_p => A pointer points to the print error.

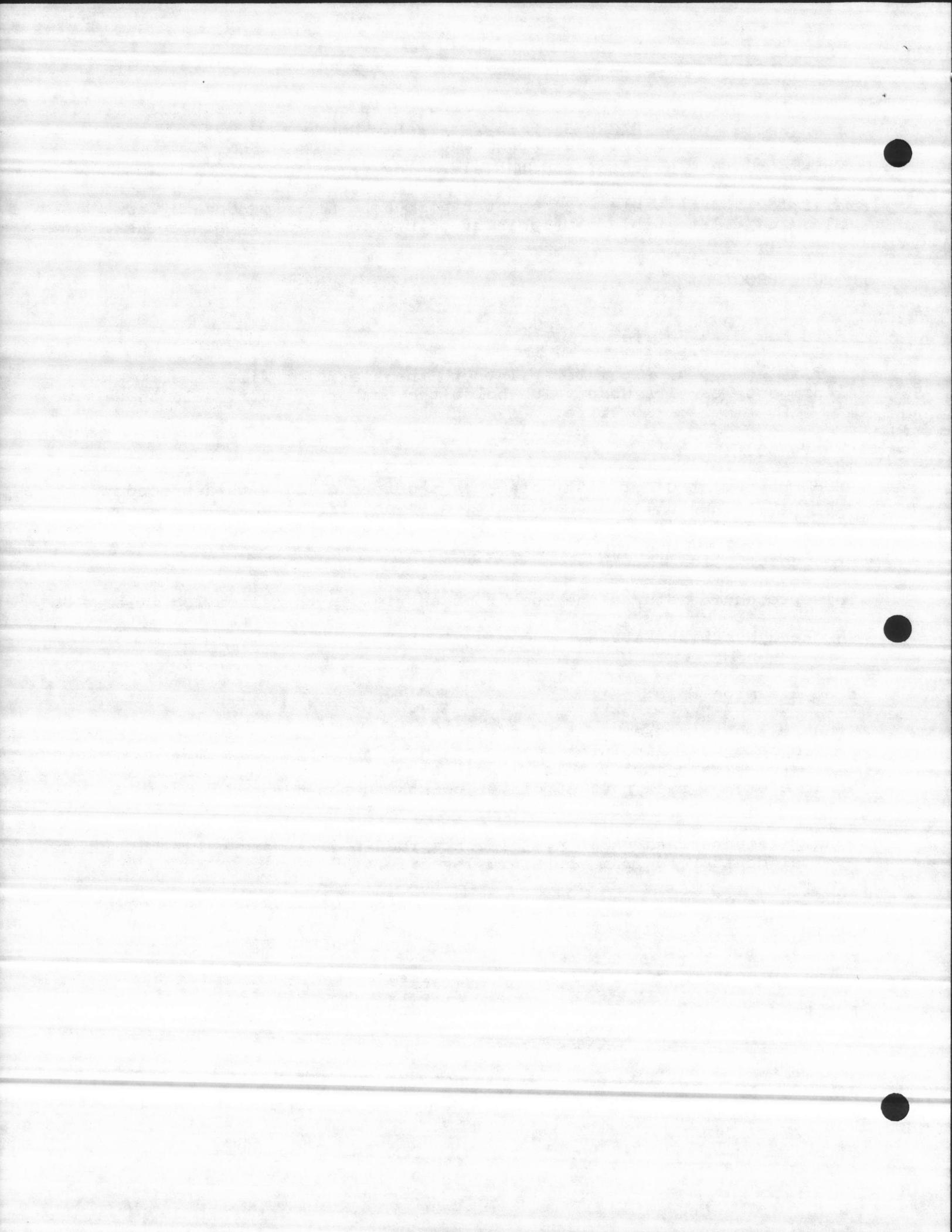
num\_elements\_read BASED num\_elements\_read\_p => A word holds the number  
of elements read from the alarm log file.

MUST have control of disk\_param\_reg\_t before calling this procedure.

Proc description :

-----  
This procedure reads the alarm log file.

END Read\_Alarmlog\_File;



**Display\_Alarmlog\_Proc:**

crt\_num => A byte value holds the number of the crt to display on.

**Proc description :**

This procedure displays the alarm log from the alarm log file.

END Display\_Alarmlog\_Proc;

**Hist\_Display\_Alog\_Proc:**

crt\_num => A byte value holds the number of the crt to display on.

**Proc description :**

This procedure displays the alarm log from the historical alarm log file.

END Hist\_Display\_Alog\_Proc;

**Format\_Crt\_Alarmlog\_Entry:**

buf\_t => A token, used as a buffer token.

alarmlog\_element\_p => A pointer points to alarm log element.

hist\_flag => A byte holds the historical flag value.

**Proc description :**

This procedure formats alarm line on the crt.

END Format\_Crt\_Alarmlog\_Entry;

**Alarmlog\_Update\_Task:**

count => A byte holds a count value.

crt\_num => A byte value holds the number of the crt to display on.

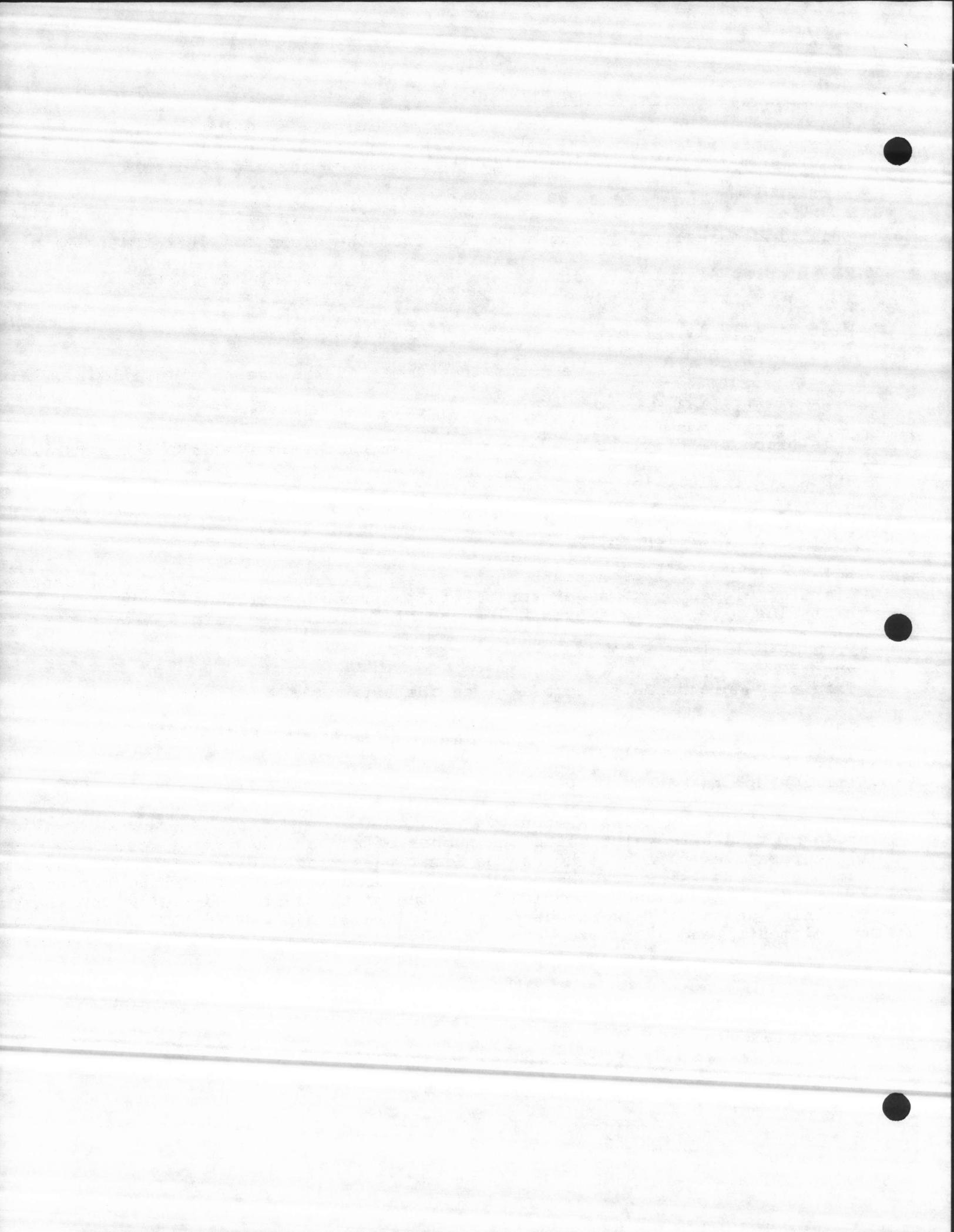
first\_pass\_flag => A byte holds the first\_pass\_flag value.

num\_entries => A word holds the number of entries in the alarm log.

last\_num\_entries => A word holds the value of the last number of entries.

remaining\_units => A word, used as an index.

num\_elements\_read => A word holds the number of elements read from the alarm log file.



-----  
found\_flag => A bytes holds the value of found\_flag.  
num\_read => A word holds the number read.  
skip\_num => A word holds the value of skipping number.  
skip\_num\_count => A word value holds the skip number count.  
index => A word used as an index.

Task description :

-----  
This task updates the alarm log displays.

END Alarmlog\_Update\_Task;

Alarmlog\_Delete\_Task:

-----  
crt\_num => A byte value holds the number of the crt to display on.

Proc description :

-----  
This procedures deletes the list\$entry and the alarmlog update semaphore.

END Alarmlog\_Delete\_Task;

Print\_Alarmlog:

-----  
crt\_num => A byte value holds the number of the crt to display on.  
exception => A word used for exception condition.  
plog\_param\_t => A token used for print log param region.

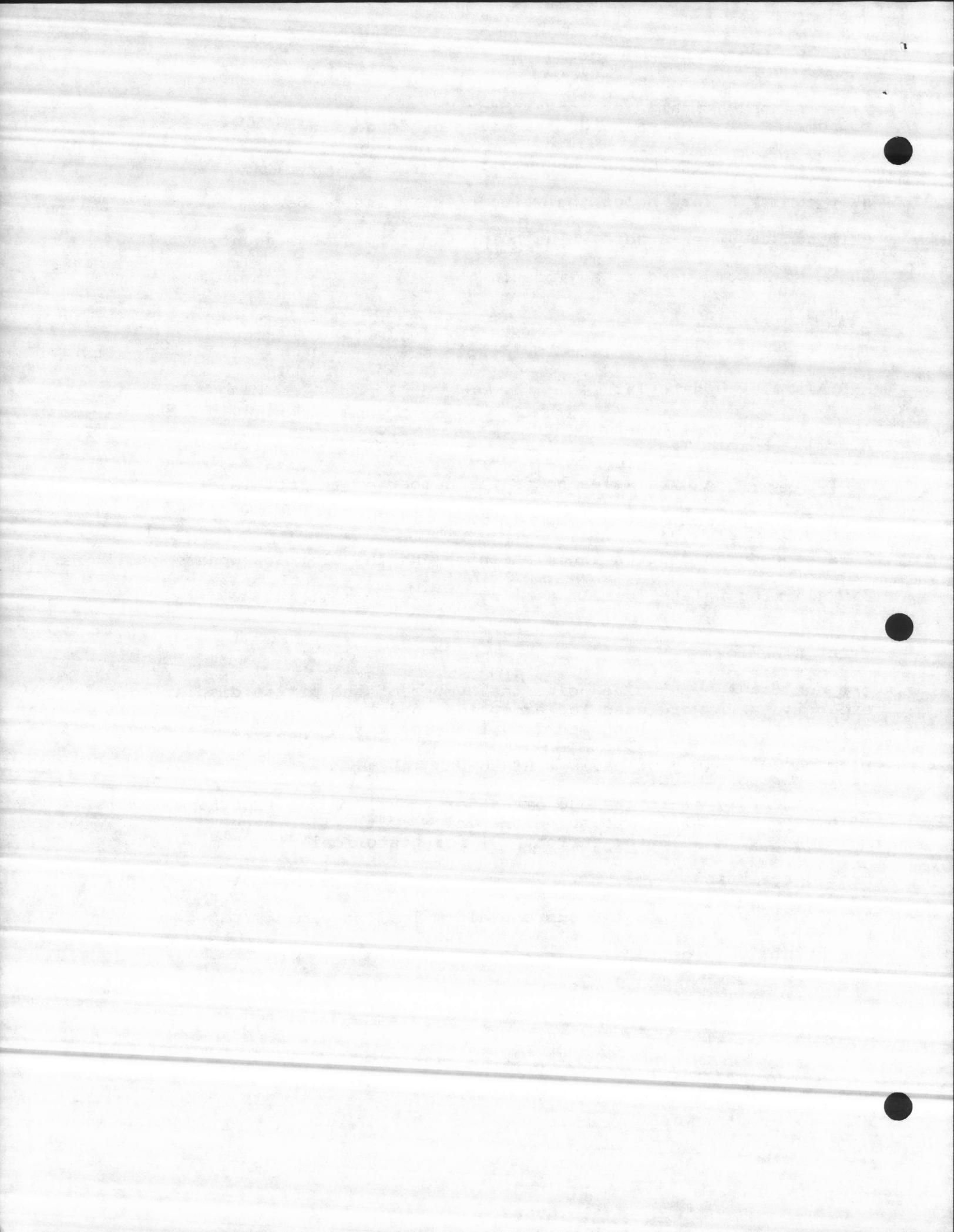
plog\_param BASED plog\_param\_t STRUCTURE :

-----  
file\_name (31) => Thirty one bytes for the alarm log file name.  
file\_type => A byte holds the value of the file type.  
0 for current , 1 for historical file name.

Proc description :

-----  
This procedure prints the current alarm log (on demand).

END Print\_Alarmlog;



**Hist\_Print\_Alog:**

crt\_num => A byte value holds the number of the crt to display on.  
exception => A word used for exception condition.  
plog\_param\_t => A token used for print log param region.

plog\_param BASED plog\_param\_t STRUCTURE :

file\_name (31) => Thirty one bytes for the alarm log file name.  
file\_type => A byte holds the value of the file type.  
0 for current , 1 for historical file name.

Proc description :

This procedure prints the historical alarm log (on demand).

END Hist\_Print\_Alog;

**Print\_Alarmlog\_Heading:**

heading\_type => A byte holds the value of the heading type  
page\_num => A word holds the page number.  
log\_time => A dword holds the log time.

prt\_alarmlog\_sub\_heading => A string of characters represent the alarm log sub heading.

Proc description :

This procedure will print the subheading for each page of the current alarm log or for historical alarm log.

END Print\_Alarmlog\_Heading;

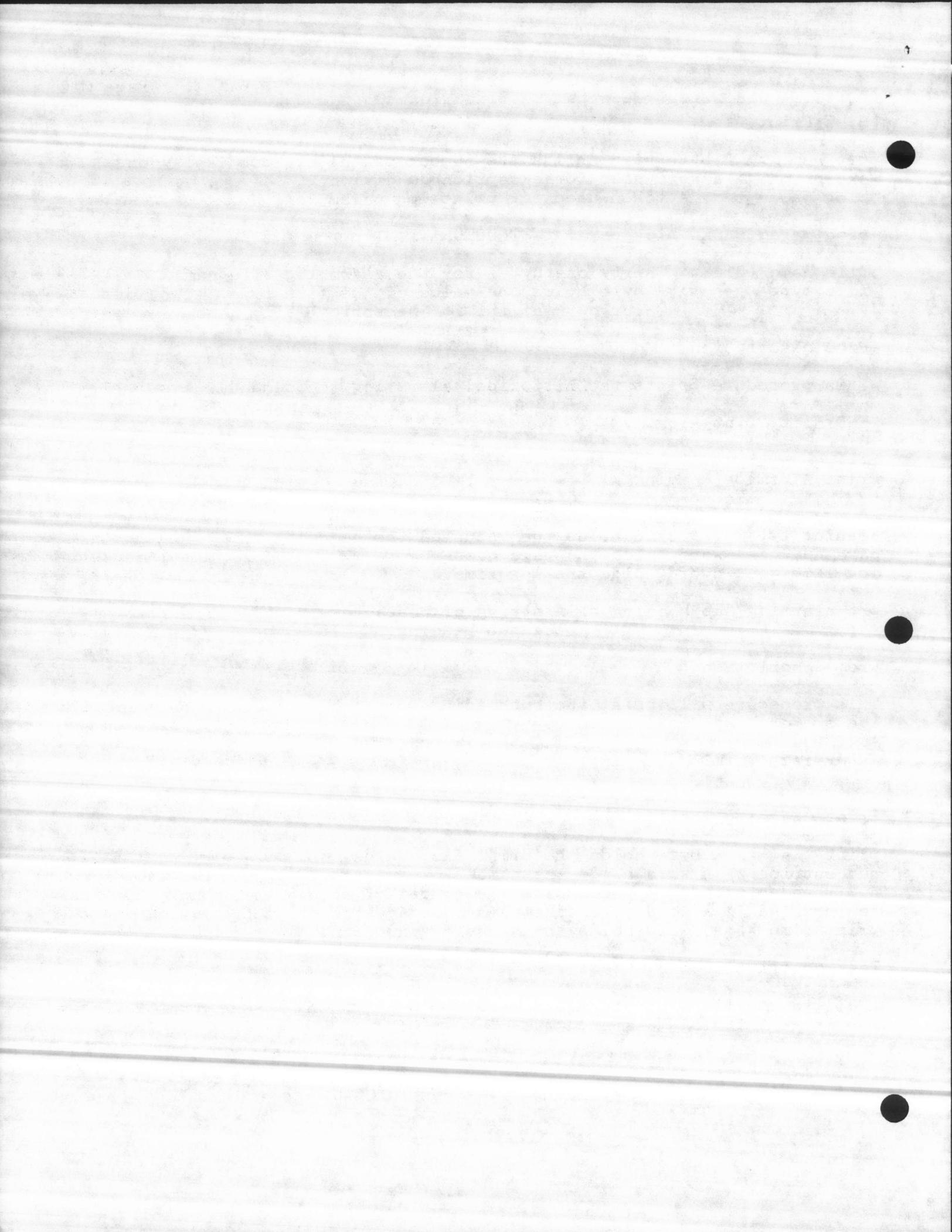
**Print\_Alarmlog\_Task:**

print\_flag => A byte holds the print flag value.  
num\_entries => A word holds the number of entries.  
num\_read\_entries => A word holds the number of entries are read.  
num\_printed => A word holds the number of entries printed.  
print\_time => A dword holds the value of the print time.

Task description :

This task will print the alarm log.

END Print\_Alarmlog\_Task;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 9

T O T A L I Z A T I O N

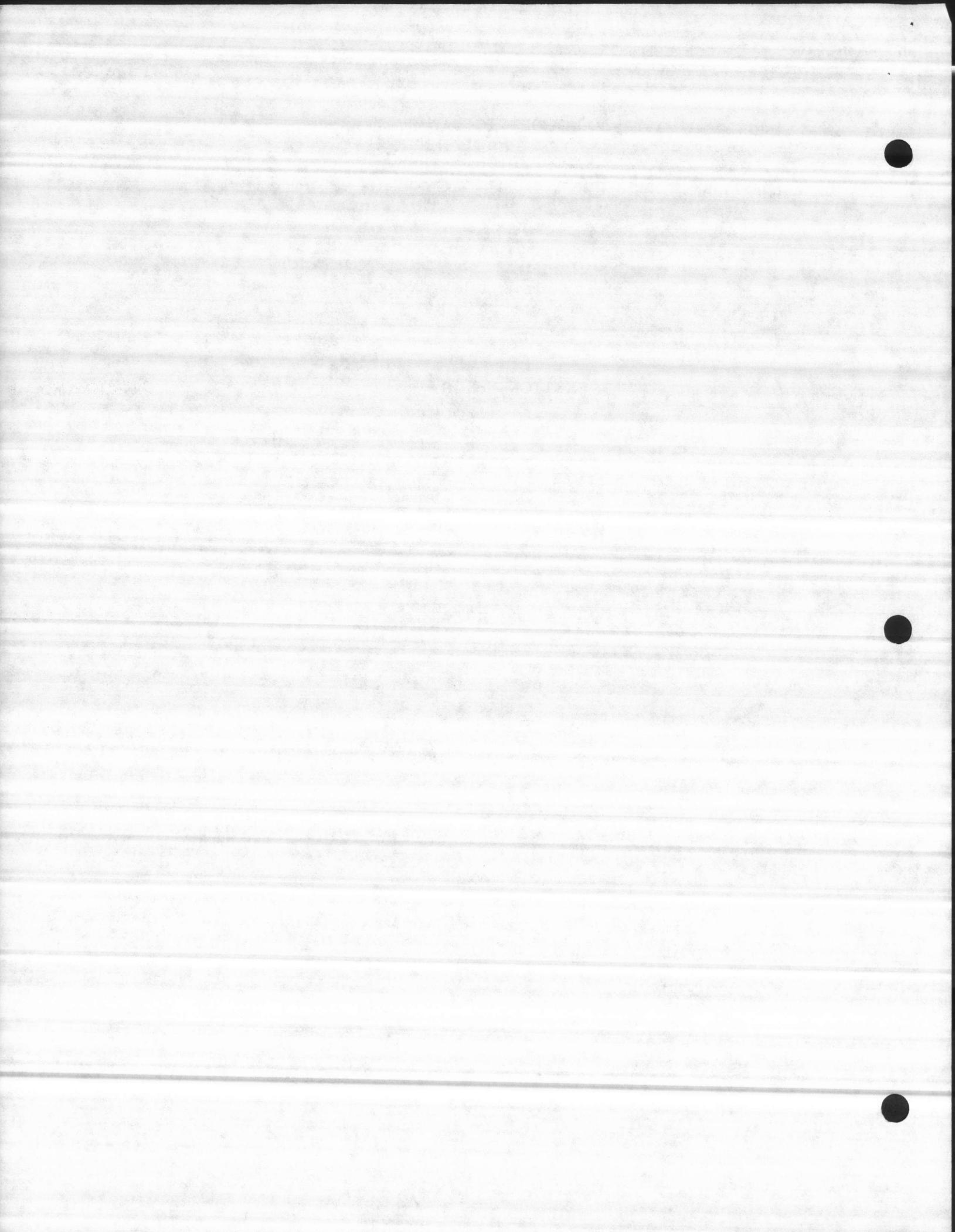
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1986

The Totalization - written by Bob Ryan, Peter Wollenzien, Mohamed Fayad.

(6.0) MODULE NAME : CxxxxTOTL.Pyy

=====

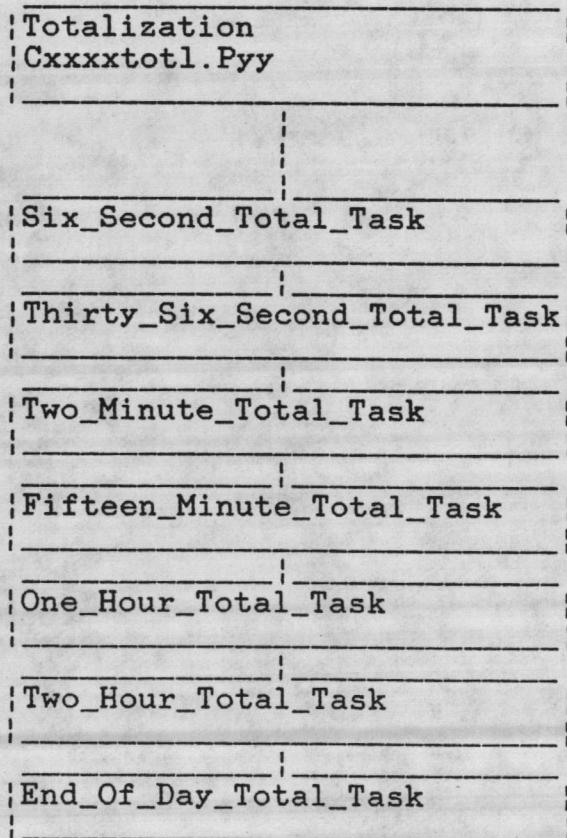
Description :

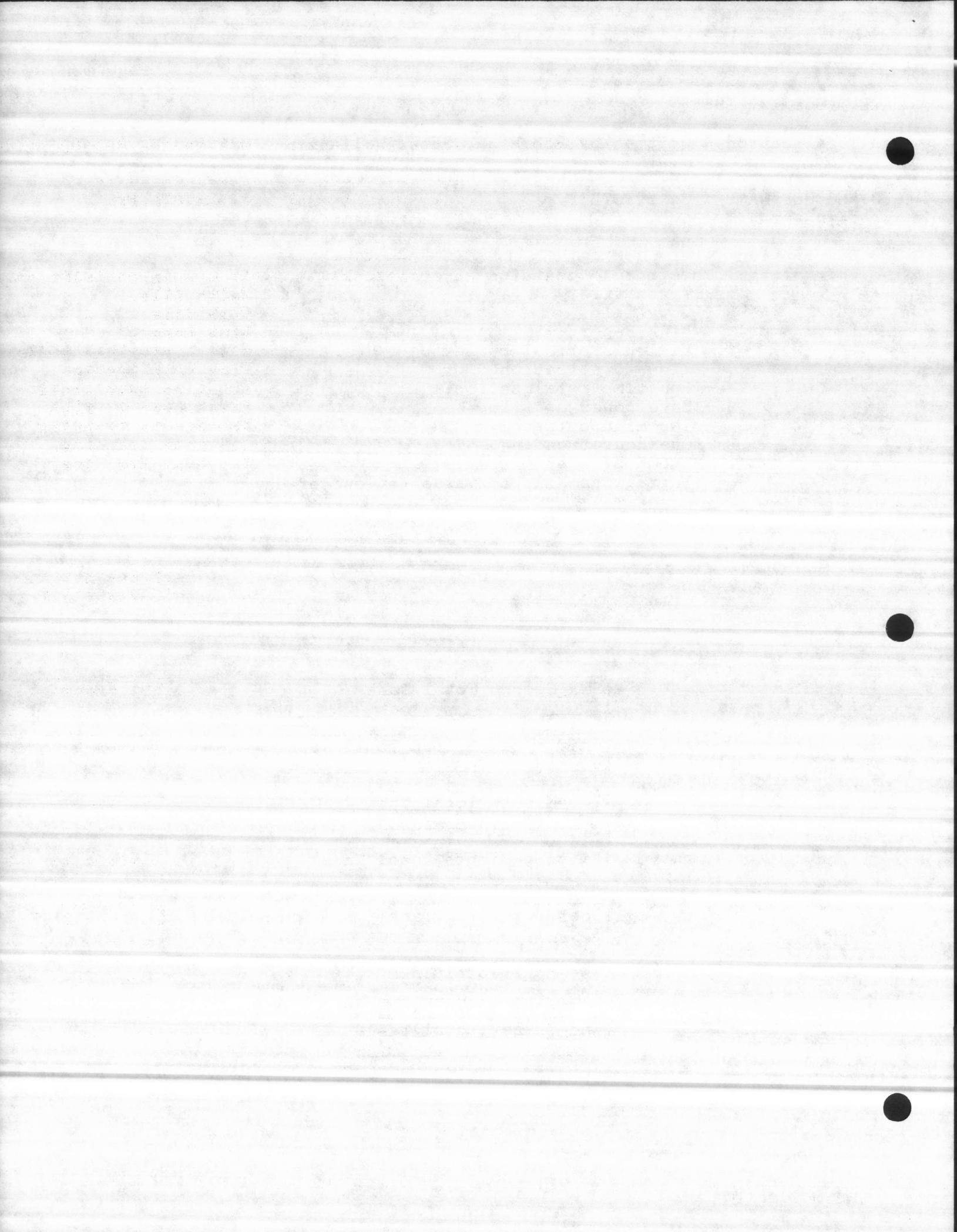
===== This module contains several tasks

- Write file.
- Copy file.
- Link Report
- Fill historical buffer.
- Flow totalization.
- Run time totalization.
- Save power up file.
- Two minute trending.
- One hour trending.
- Two hour trending.
- Initializes all the values at end of day.

MODULE FLOW :

=====





## EXTERNAL PROCEDURES :

```
=====
Format_Disk_Error: PROCEDURE (buffer_t,
                               message_str_p,
                               disk_error,
                               disk_operation) EXTERNAL;
DECLARE (disk_error,
         disk_operation) WORD,
         buffer_t      TOKEN,
         message_str_p  POINTER;
END Format_Disk_Error;
```

```
Collect_Trend: PROCEDURE (trend_type, time_dword) EXTERNAL; C4758TRND
```

```
DECLARE (trend_type)   BYTE,
        (time_dword)    DWORD;
END Collect_Trend;
```

```
Init_Trend: PROCEDURE (trend_type, name_type, existence_check_flag)
                      EXTERNAL;
DECLARE (trend_type,
         name_type,
         existence_check_flag)           BYTE;
END Init_Trend;
```

## GLOBAL DATA REFERENCE:

```
=====
analog_in, discrete_in                                See (CxxxxGLOB.Pyy).
two_minute_trend, one_hour_trend, two_hour_trend
day_trend_buffer, week_trend, month_trend             See (CxxxxTPUB.Pyy).
yearly_analog                                         See (CxxxxGEXR.Pyy).
```

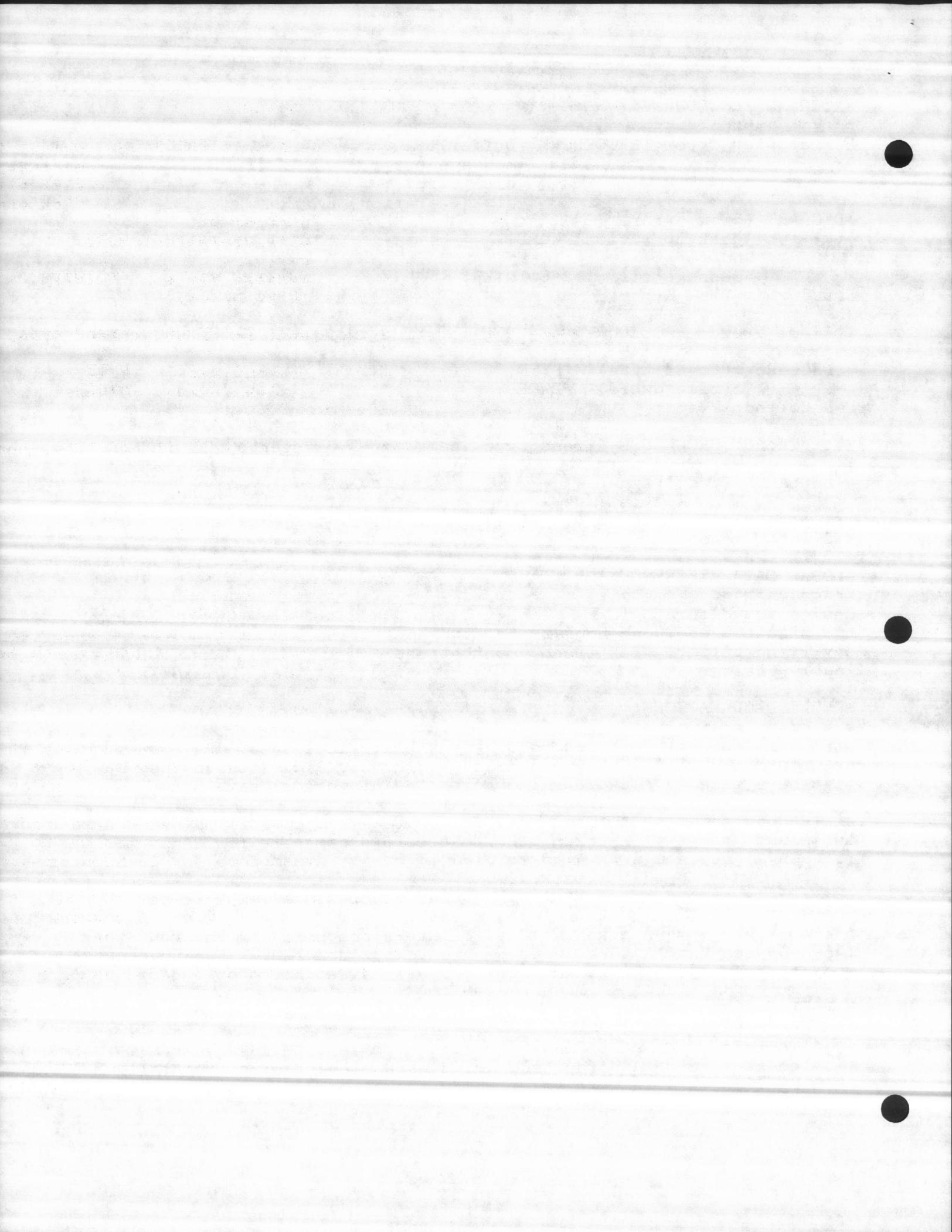
## MODULE DECLARATIONS :

```
=====
hist_name_date => 20 bytes holding the historical date.
bytes_written => A word value holds the number of written bytes.
hist_save_time => A dword value holds the historical save time.

six_second_sem_t => A token used to create a semphore for 6 seconds task.
two_minute_sem_t => A token used to create a semphore for 2 minutes task.
fifteen_minute_sem_t => A token used to create a semphore for 15 minutes
                        task.
thirty_six_second_sem_t => A token used to create a semphore for 36
                           seconds task.
one_hour_sem_t => A token used to create a semphore for one hour task.
two_hour_sem_t => A token used to create a semphore for two hours task.
end_of_day_sem_t => A token used to create a semphore for end of day task.
```

## alarm\_log STRUCTURE :

```
=====
file_type      => A word value holds the file type.
element_size   => A word value holds each element size (initially 20)
num_elements  => A word value holds the number of elements in the log.
```



```
alarm_log_disk_data STRUCTURE :  
-----  
    time_dword => A dword holds the time, which indicates when the alarm  
                  occurs.  
    fail_type   => A byte value holds the fail type.  
    index       => A word value holds an index for how many alarms occurred.  
    condition   => A word value holds the alarm condition.  
    analog_flag=> A byte value holds an analog flag, which indicates if  
                  alarm is analog alarm or not.  
    value       => A word value holds the current value of the analog.  
    limit       => A word value holds the limit value.  
    units       => A word value holds the engineering unit.  
    scale       => A word value holds the scale factor.  
    crit_flag   => A byte value holds a critical flag.  
    alarm_flag  => A byte value holds an alarm flag.  
temp_flow STRUCTURE :  
-----  
    flow_count => A byte value holds the flow counter.  
    total_flow  => A dword value holds the flow total.  
    remainder   => A byte value holds the the remainder after unit  
                  conversion.
```

## LOCAL PROCEDURES :

```
=====
```

## Start\_Report\_Link:

```
-----  
    count      => A word value holds a counter.  
    report_type => A word value holds the report type.
```

## Proc Description :

```
-----  
This procedure goes through all the available reports, and finds out  
which report to be printed on auto mode.
```

```
End Start_Report_Link;
```

## Hist\_Disk\_Buffer\_Fill:

```
-----  
    count      => A word value holds a counter.  
    src_header_data_p => A pointer points to source header data  
    src_header_info_p => A pointer points to source header information.  
    dst_header_data_p => A pointer points to destination header data.  
    dst_header_info_p => A pointer points to destination header information.
```

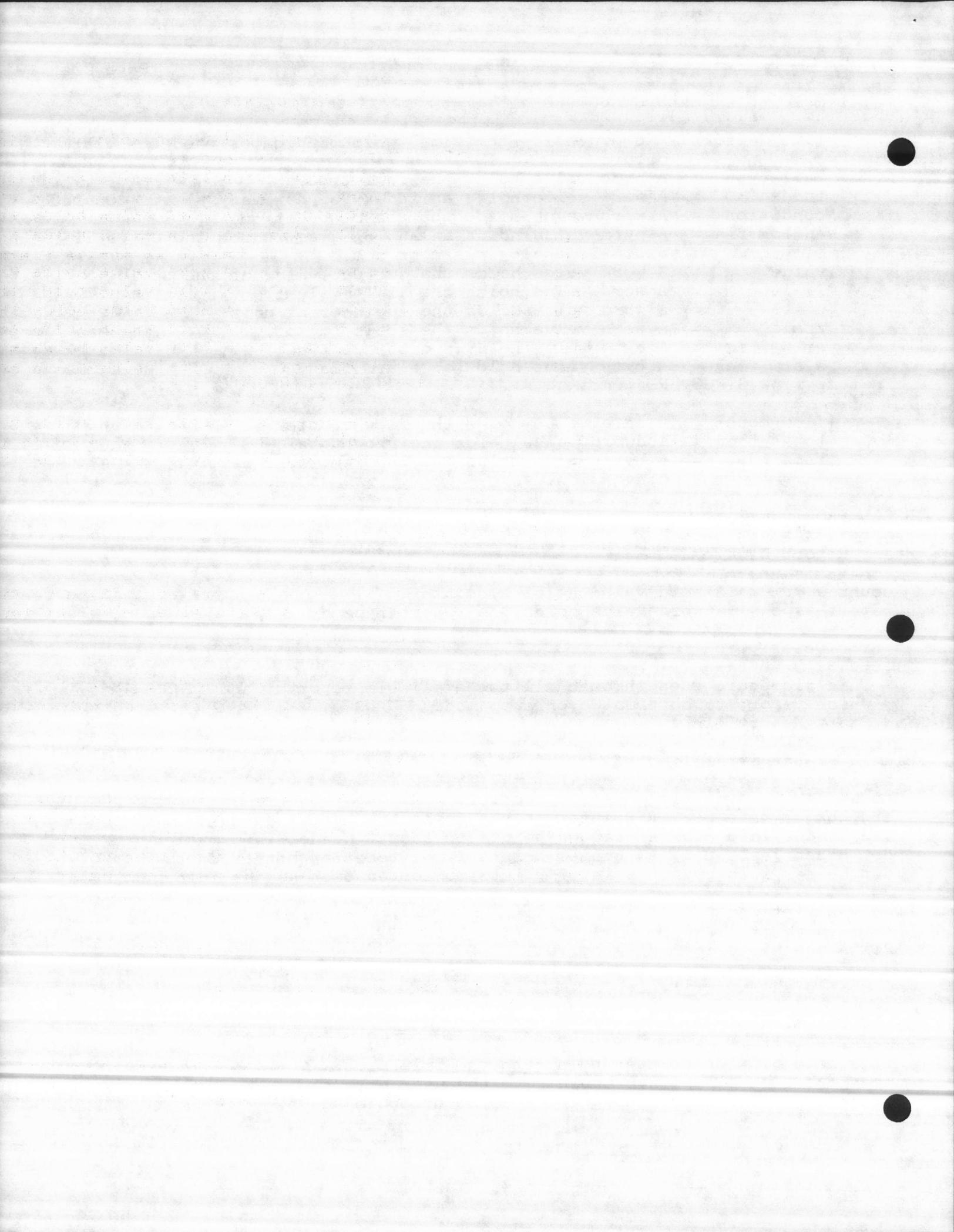
## Proc Description :

```
-----  
fills the historical disk buffer.
```

```
End Hist_Disk_Buffer_Fill;
```

## Write\_File:

```
-----  
    file_name_p  => A pointer points to the file name.  
    user_error_p  => A pointer points to the user error  
    exception_p  => A pointer points to the exception code.
```



```
-----  
temp_word    => A word holds a temprarly value.  
exception => A word holds the exception condition.  
  
user_error BASED user_error_p STRUCTURE :  
-----  
count => A byte holds a counter.  
asc   => 30 bytes holds the user error string ascii.  
error_msg  STRUCTURE :  
-----  
count      => A byte holds a counter.  
asc       => 60 bytes holds the message error string ascii
```

```
write_file_param BASED disk_param_seg_t STRUCTURE :
```

```
-----  
See disk module.
```

```
Proc Discription :
```

```
-----  
Writes a file to the disk.
```

```
END Write_File;
```

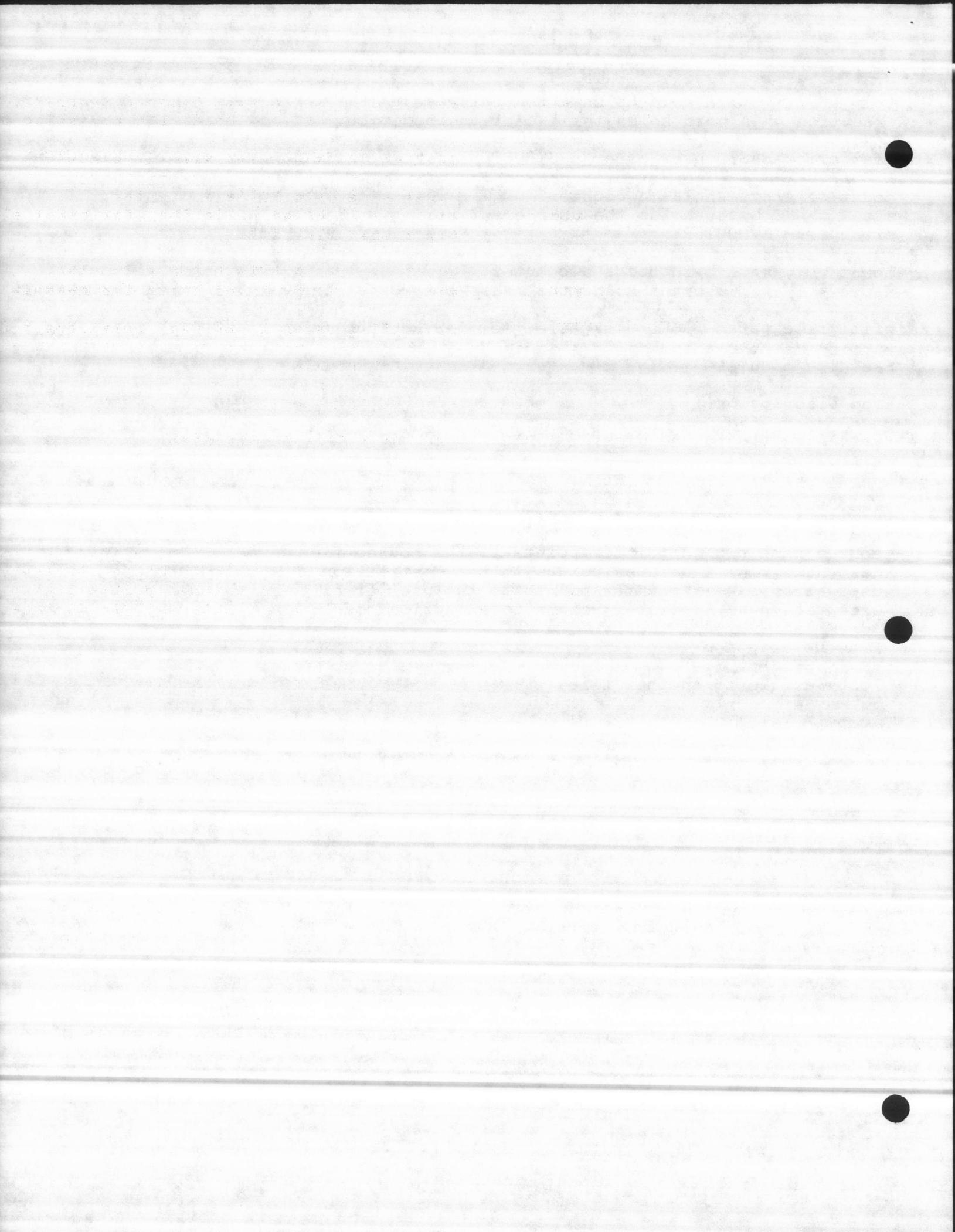
```
Copy_File:
```

```
-----  
source_name_p => A pointer points to the source file name.  
dest_name_p  => A pointer points to the destination file name.  
user_error_p  => A pointer points to the user error.  
exception_p  => A pointer points to exception code.
```

```
Proc Description :
```

```
-----  
copy a file to another file.
```

```
END Copy_File;
```



```
+-----+
+          SIX SECOND TASK
+
Six_Second_Total_Task:
=====
TASK DESCRIPTION : This task runs once every 6 seconds.
=====      Task assignment : Totalizes flow.

local task procedures :
-----
Flow_Totalization:
=====
flow      => A word holds the current flow
units     => A word holds the total engineering unit.
remain_p  => A pointer points to the remainder.
flow_total_p => A pointer points to the flow total.

Proc Description :
-----      engineering unit conversion & totalizes the flow.
END Flow_Totalization;

TASK ALGORITHM :
=====
Set exception handler.

Initialize temp accumulators
Create a semaphore using six_second_sem_t.

Go through all the analogs and assign the current value to the
last_roc_sample(last rate of change sample).

Create a clock entry (6 seconds).

Loop FOREVER;

    wait for six second clock

    get control of total region

    check ROC alarm

    totalize all flows

        totalize temporary samples for averages and calculate mininmums
        maximums for trends(1 minute, 1 hour, 2 hours).
        fill each temp buffer prior to setup for write

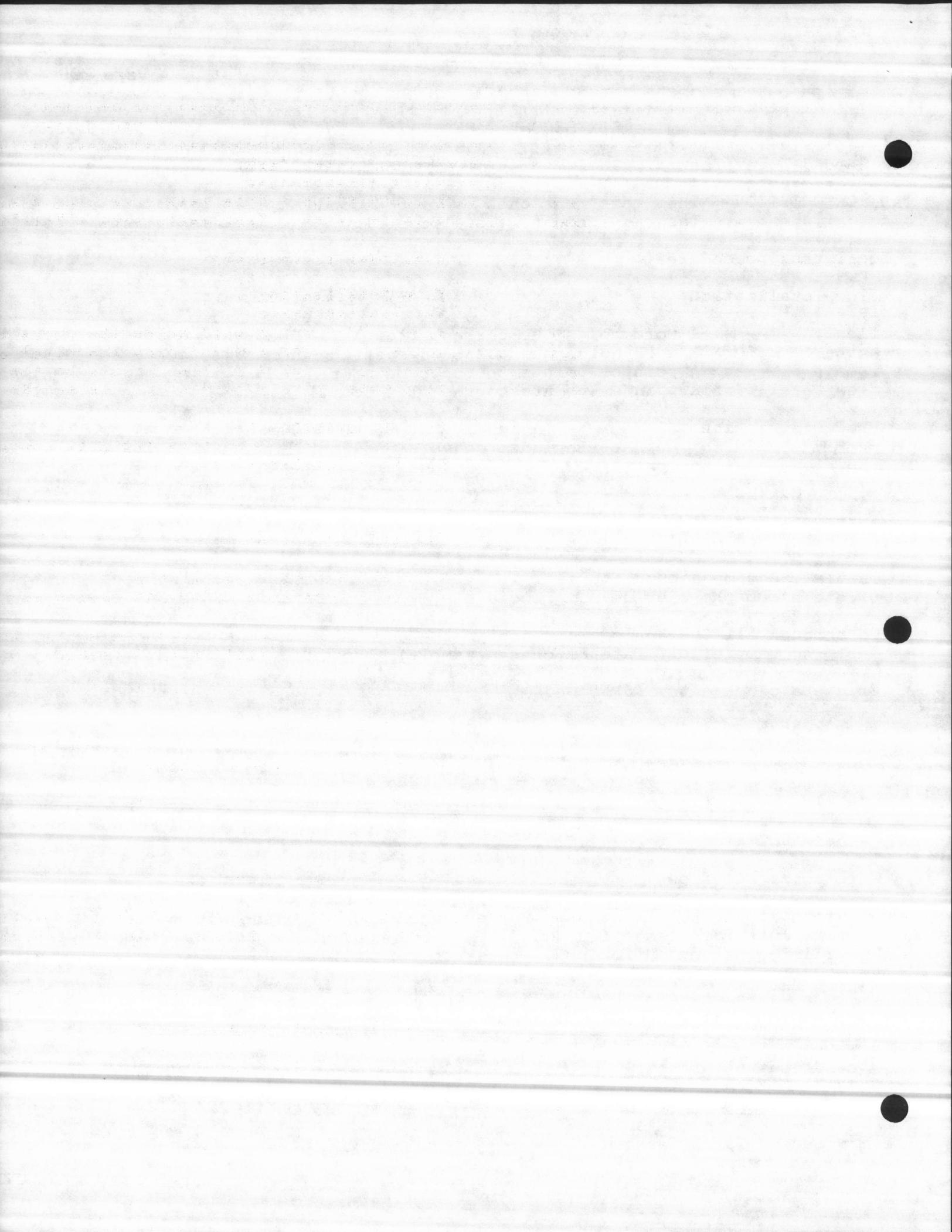
    get control of trend region

    send control of trend region

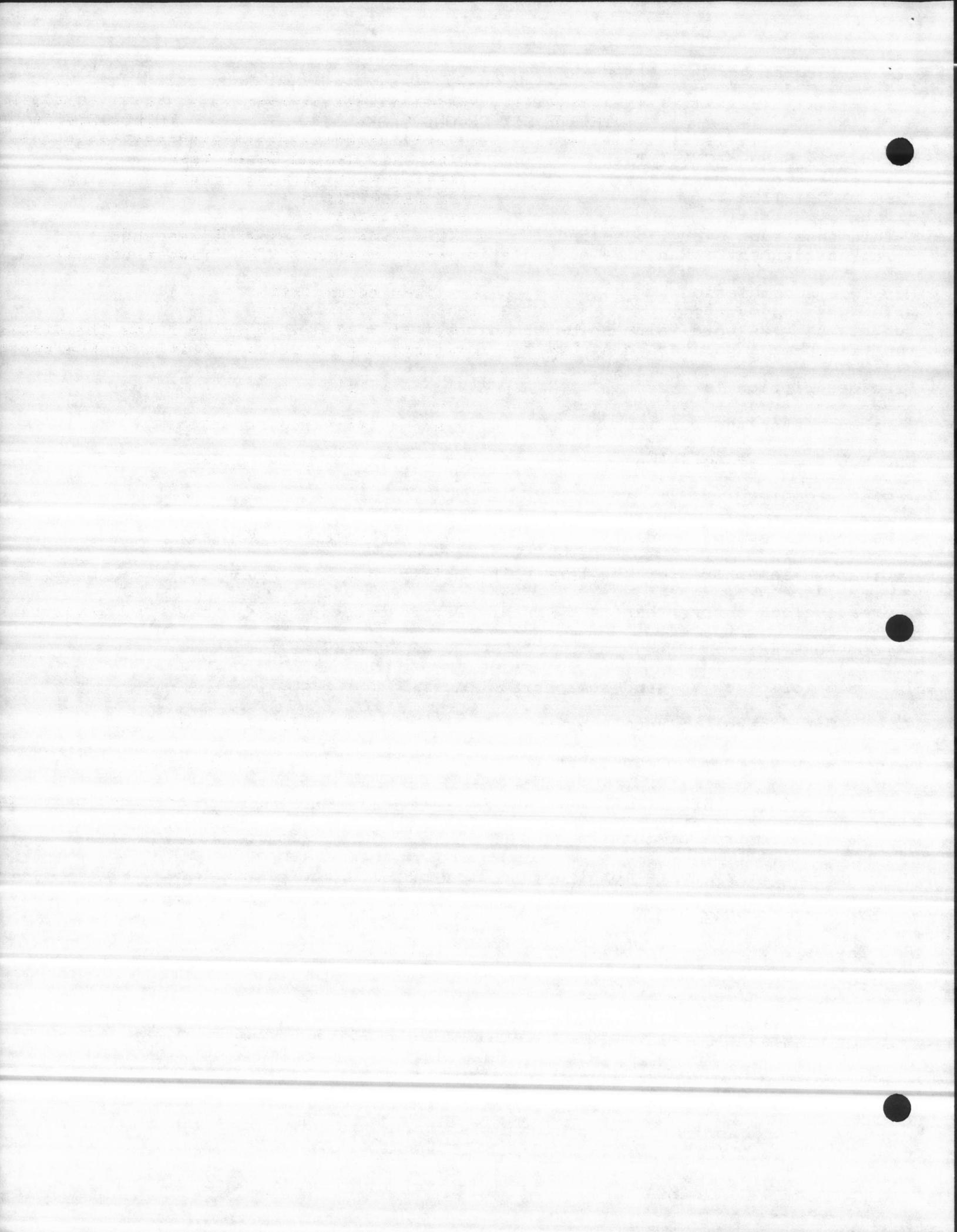
    send control of total region

END forever Loop

END Six_Second_Total_Task;
```



```
-----  
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +  
+                                                                                            +  
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +  
Thirty_Six_Second_Task:  
=====  
TASK DESCRIPTION :  
=====  
This task runs every 36 seconds.  
Task assignment : Run time accumulation.  
  
Run_Time_Accumulation:  
=====  
run_status => A word value holds the discrete run status.  
run_time_p => A pointer points to the run time.  
  
Proc description :  
=====  
    accumulates the run time.  
  
END Run_Time_Accumulation;  
  
TASK ALGORITHM :  
=====  
    Set the exception handler.  
  
    Create a semaphore for the task using thirty_six_second_sem_t.  
  
    Create clock entry : base = 0, interval = 36 seconds  
  
    Loop FOREVER;  
  
        wait for clock to send semaphore  
  
        Get control of total region  
  
        accumulate pump run times  
  
        totalize temporary samples for averages  
  
        send control of total region  
  
        Send list by using total_update_list_t.  
  
    END loop forever  
  
END Thirty_Six_Second_Task;
```



```
-----+
+++++TWO MINUTE TASK+
+
Two_Minute_Total_Task:
=====
TASK DESCRIPTION :
=====
    This task runs once every two minutes.
    Task assignment : Collects trends every two minutes.

TASK ALGORITHM :
=====
Set the exception handler.

Create a semaphore for the task using two_minute_sem_t.

Create clock entry : base = Ø, interval = 120 seconds

Loop FOREVER;

    wait for clock to send semaphore

    Get time.

    get control of total region

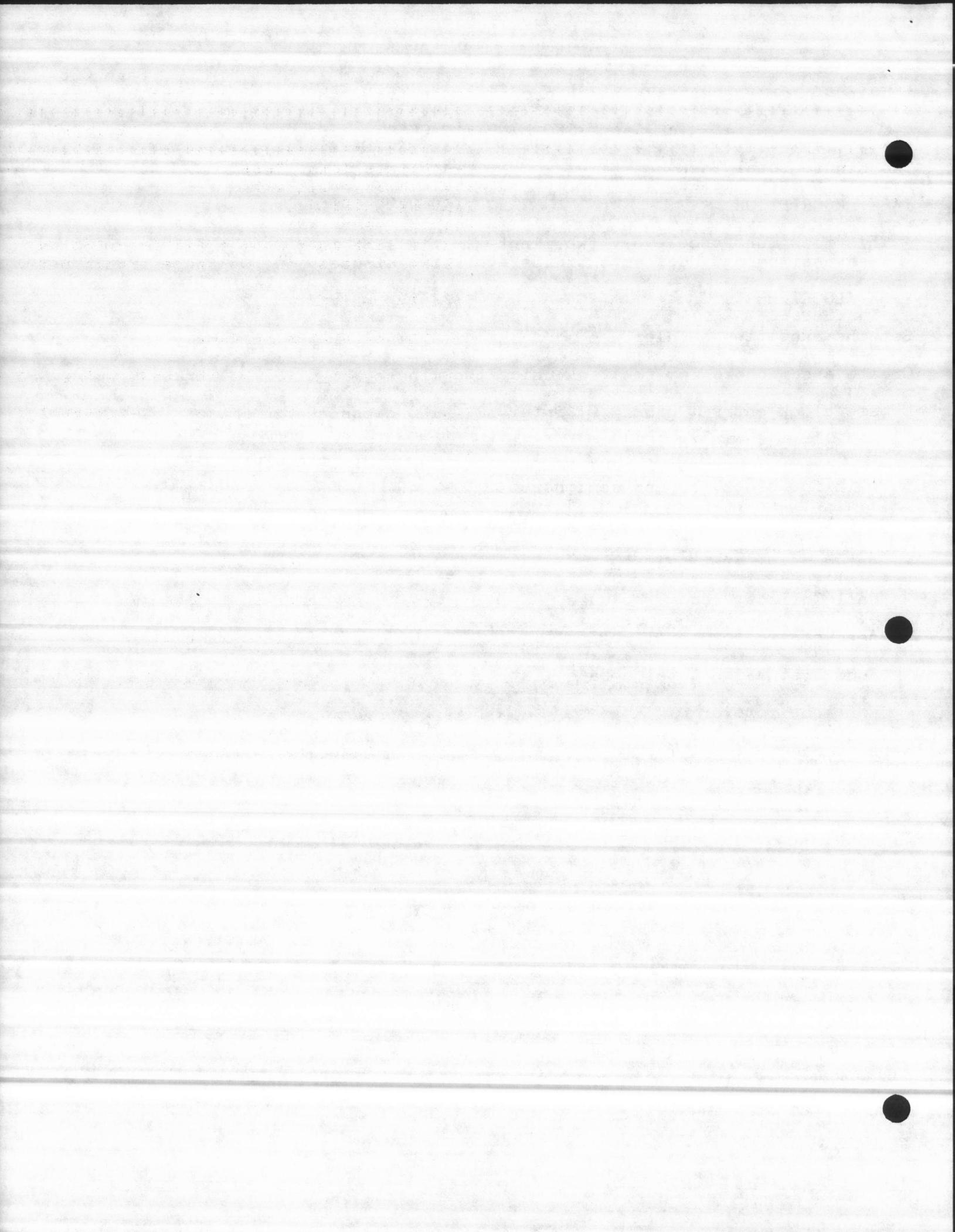
    Call Collect_Trend Procedure.

    send control of total region

    Send List using total_update_list_t.

    END loop forever

END Two_Minute_Total_Task;
```



```
-----+
+ FIFTEEN MINUTE TASK +
+-----+
Fifteen_Minute_Total_Task:
=====
TASK DESCRIPTION :
=====
This task runs every fifteen minutes.
Task assignment :
    Writes the power up files every 15 minutes.

TASK ALGORITHM :
=====
Set the exception handler.

Create a semaphore using fifteen_minute_sem_t.

Create a clock entry 15 minute.

Loop FOREVER;

    wait for clock to send semaphore

    WRITE THE POWER UP FILE

    get time and assign it to power_up_save_time.

    get control of disk region.

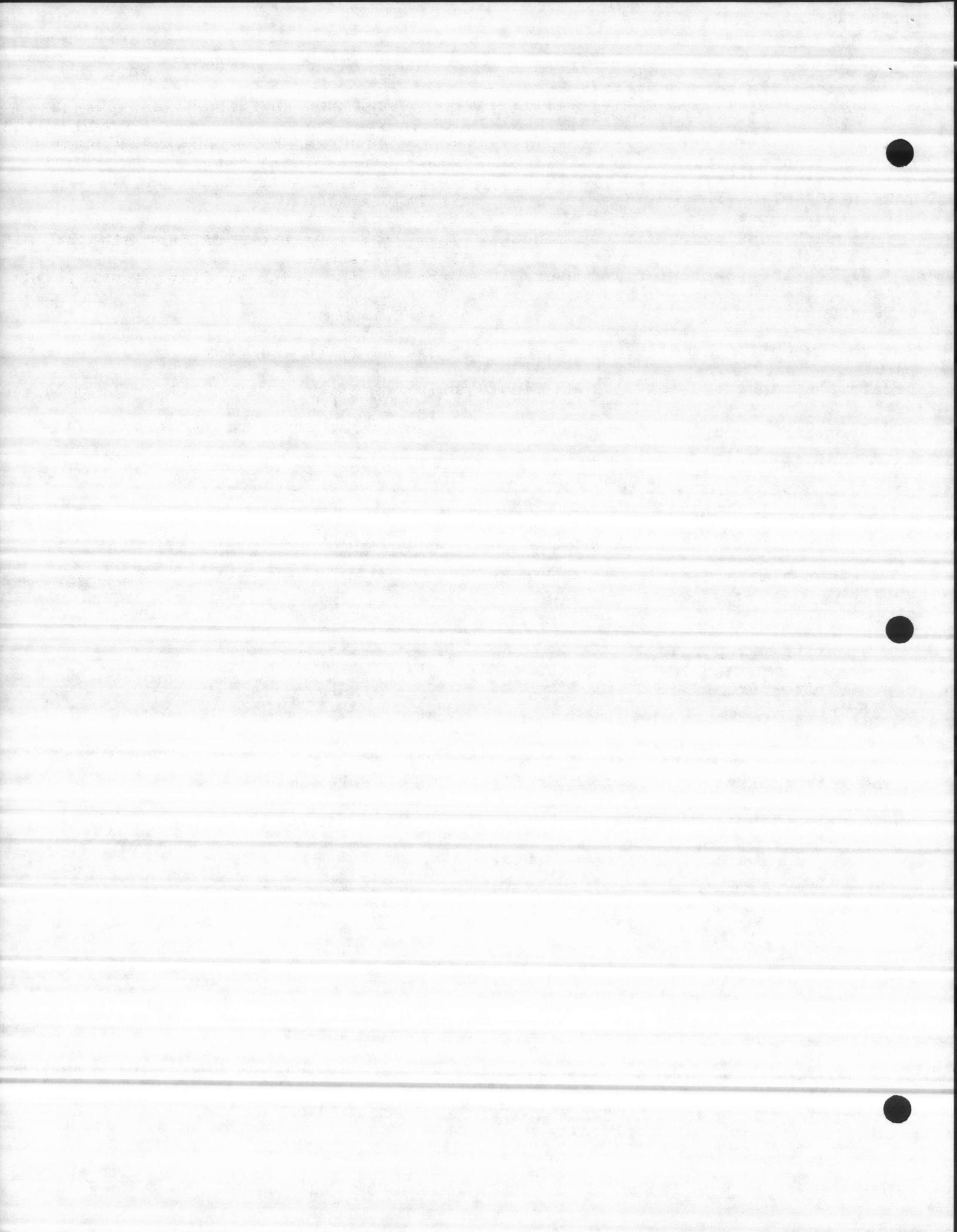
    Write pwrup file 1.

    Write pwrup file 2.

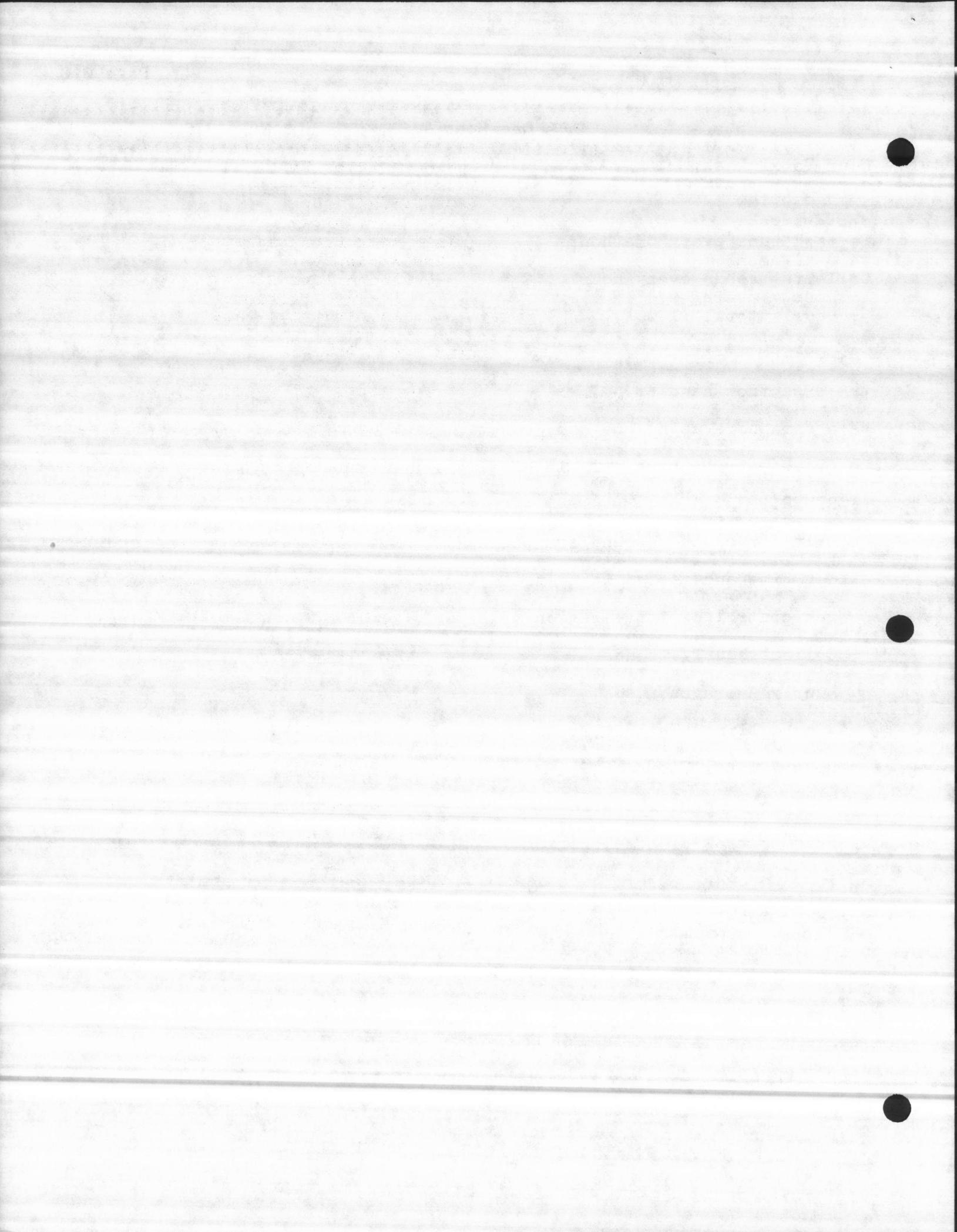
    give up control of disk param region

END Loop forever

END Fifteen_Minute_Total_Task;
```



```
-----  
+++++  
+ One Hour TASK +  
+  
One_Hour_Total_Task:  
=====  
TASK DESCRIPTION :  
=====  
This task runs every one hour.  
Task assignment :  
    Collects trends every hour.  
        - Hourly averages.  
        - Hourly total flow.  
  
TASK ALGORITHM :  
=====  
Set the exception handler.  
  
Create a semaphore using one_hour_sem_t.  
  
Create clock entry : base = 0, interval = 1 Hour  
  
Loop FOREVER;  
  
    wait for clock to send semaphore  
  
    Get time and assign it to trend_time.  
  
    get control of total region  
  
    Collect hourly trends.  
  
    Hourly Averages  
  
        by calling Collect_Trend.  
  
    Hourly Total Flows  
  
        by calling Collect_Trend.  
  
    send control of total region  
  
END loop forever  
  
END One_Hour_Total_Task;
```



```
+-----+
+          TWO HOUR TASK
+
Two_Hour_Total_Task:
=====
TASK DESCRIPTION :
=====
    This task runs every two hours.
    task assignment : Collects weekly trends.

TASK ALGORITHM :
=====
Set the exception handler.

Create a semaphore by using two_hour_sem_t.

create clock entry : base = Ø, interval = 2 Hours

Loop FOREVER;

    Wait for clock to send semaphore

    Get time and assign it to trend_time.

    Get control of total region

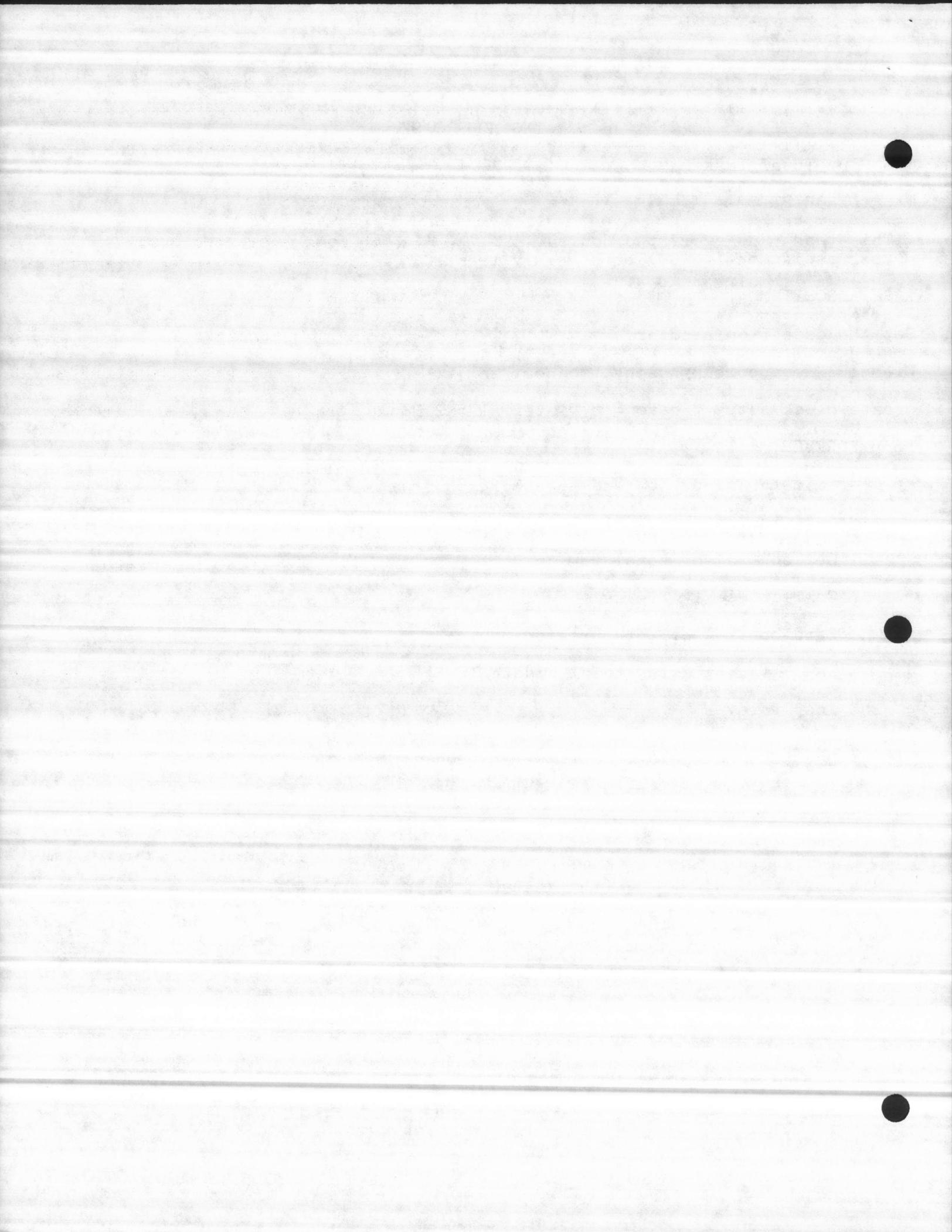
    Collect Weekly Trends

    Send control of total region

    Send list by using total_update_list_t

END loop forever

END Two_Hour_Total_Task;
```



+++++  
+ END OF DAY TASK +  
+  
EndOfDay\_Task:

=====

TASK DESCRIPTION :

=====

This task runs once every day.

Task assignments :

- Get historical save time.
- Format historical file name.
- Yearly data collection.
- Collect monthly trends.
- Move historical data to the historical buffer.
- Reset daily data.
- Check for end of week & reset weekly data.
- Check for the end of month & reset monthly data.
- Check for the end of year.
- Copy daily ALARM LOG file to historical file.
- Delete & reinitialize current alarm log.
- Save end of day trend files.
- Reinitialize disk trend files.
- Save historical files on the hard disk.
- Check for daily, weekly and monthly reports.

TASK ALGORITHM :

=====

Set the exception handler.

Create a semaphore for end of day task using end\_of\_day\_sem\_t.  
Create a clock entry base = 0 interval = 24 hours.

Loop FOREVER;

    Wait for clock to signal end of day

    Get time minus 10 seconds and assign it to hist\_save\_time.

    Call convert time to hist\_name\_date.

    Format the historical file name.

    Get day of the month.

    Grap control of the trend region.

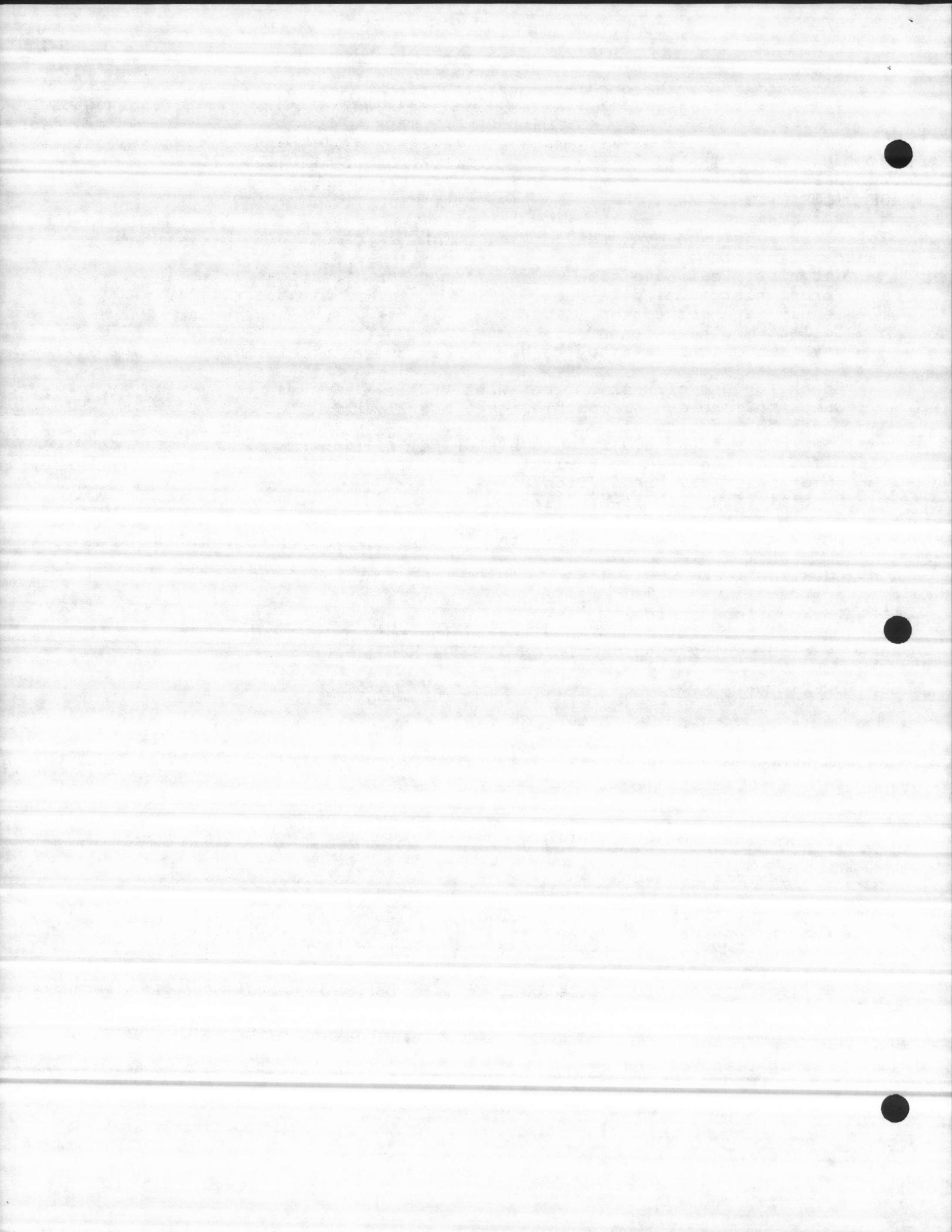
    Yearly data collection

    Release control of disk region

    Collect Monthly Trends, a call is made to Receive control trend reg

    Set up flag for end of week

    Set up flag for end of month



Get control of the DISK region  
Move Historical Data to Historical Buffer  
Release control of DISK region

Reset daily data

Check for end of week  
Reset weekly data

Check for end of month  
Reset monthly data

Check for end of year

Get control of the DISK region

Copy daily ALARM LOG file to historical file  
Formulate disk name

Copy current file into previous file

Delete and reinitialize current alarm log

Re-initialize alarm log file

Release control of DISK region

Save end of day trend files

Get control of the DISK region

Formulate disk name

Copy current file into previous file  
Copy current file into historical file

Release control of disk region

Re-initialize disk trend file (cannot have disk\_reg\_t when called)

Get control of the DISK region

Formulate disk name for AHR

Copy current file into previous file CURR\_AHR to PREV\_AHR

Copy current file into historical file PREV\_AHR to MmmDddYyyAHR

Formulate disk name FHR

Copy current file into previous file CURR\_FHR to PREV\_FHR

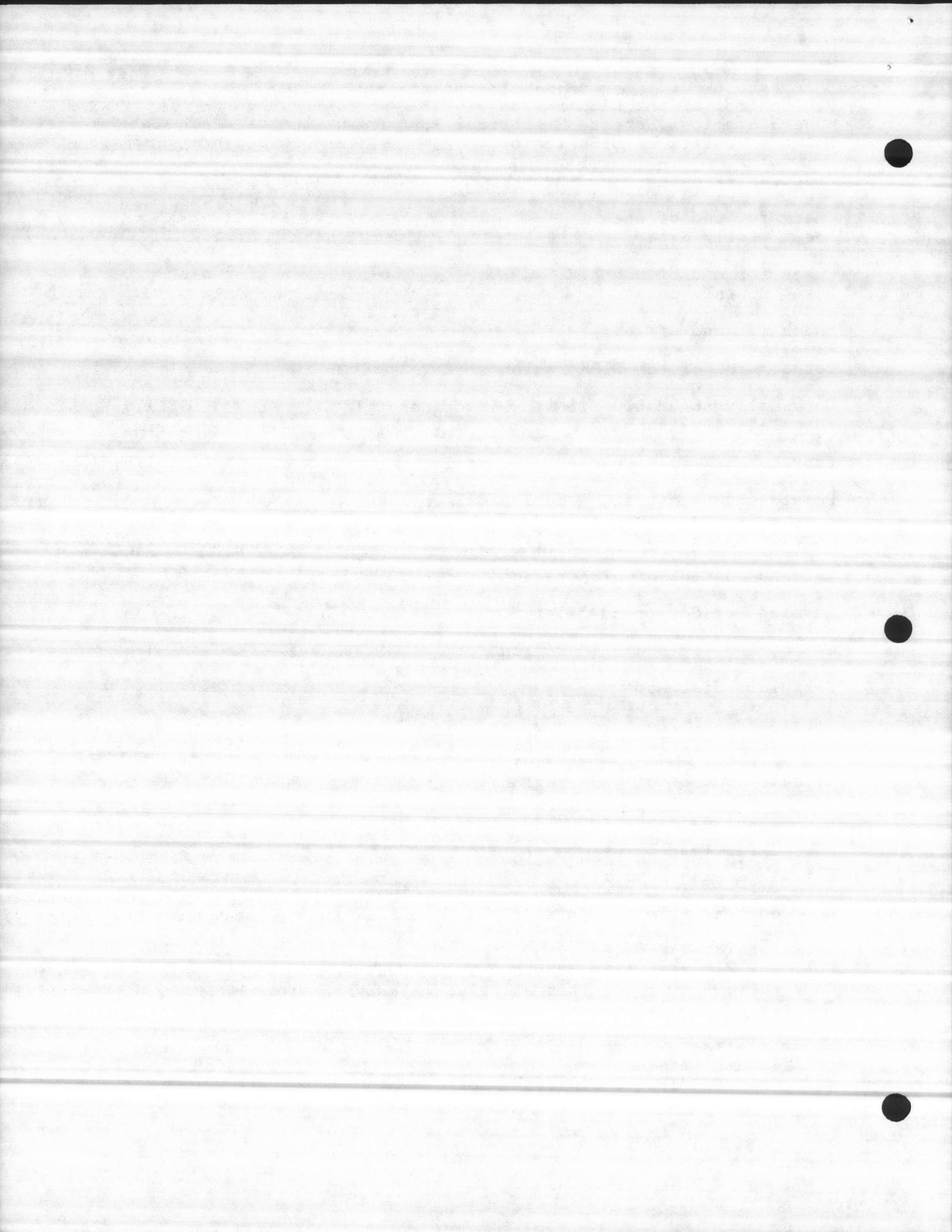
Copy current file into historical file PREV\_FHR to MmmDddYyyFHR

Release control of disk region

Re-initialize disk trend file (cannot have disk\_reg\_t when called)

Daily hourly minmax trend

Daily hourly flow total trend



Save weekly files every day  
Get control of the DISK region

Formulate disk name ex. WEEK  
Copy current file into previous file at end of week CURR\_WEEK to PREV\_WEEK  
Copy current file into historical file every day MmmDddYyyWEEK

Release control of disk region

Re-initialize disk trend file (cannot have disk\_reg\_t when called)  
Weekly trend

Save monthly files  
Get control of the DISK region

Formulate disk name MNTH  
Copy current file into previous file at end of month CURR\_MNTH to PREV\_MNTH  
Copy current file into historical file every day MmmDddYyyMNTH

Release control of disk region

Re-initialize disk trend file (cannot have disk\_reg\_t when called)

Monthly trend

Save historical file on the hard disk (will be under C4758ARCH)  
Historical data already moved into historical buffer

Create a segment using report\_param\_seg\_t for printing auto reports.

Check for daily report  
IF auto\_daily\_report\_option = yes THEN  
Print daily reports.

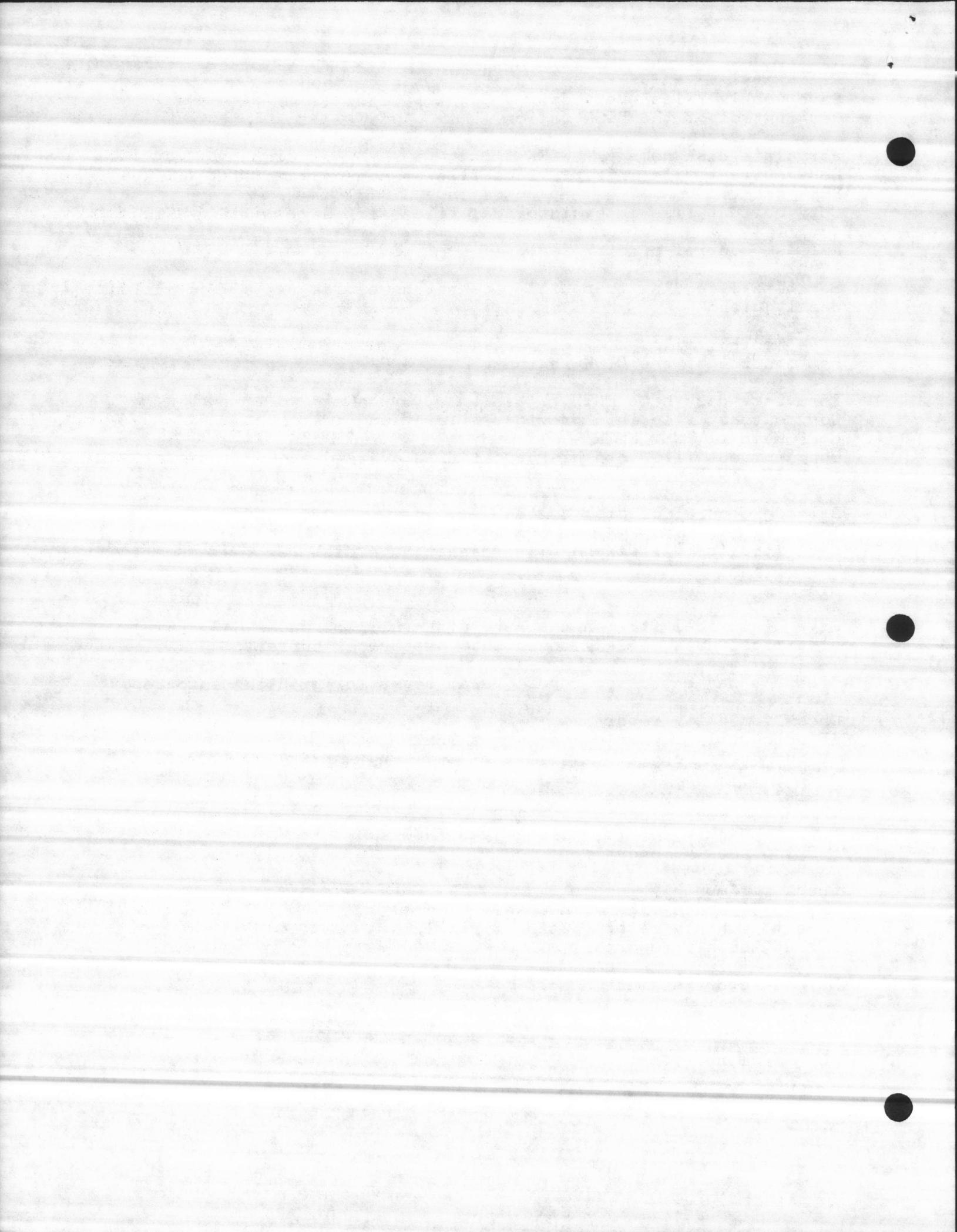
Check for weekly report  
IF end\_of\_week\_flag is true AND  
auto\_weekly\_report\_option = yes THEN  
Print weekly reports.

Check for monthly report  
IF end\_of\_month\_flag = TRUE AND  
auto\_monthly\_report\_option = yes THEN  
Print monthly reports.

Delete report segment.

END loop forever

END EndOfDayTotalTask;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 10

F A I L

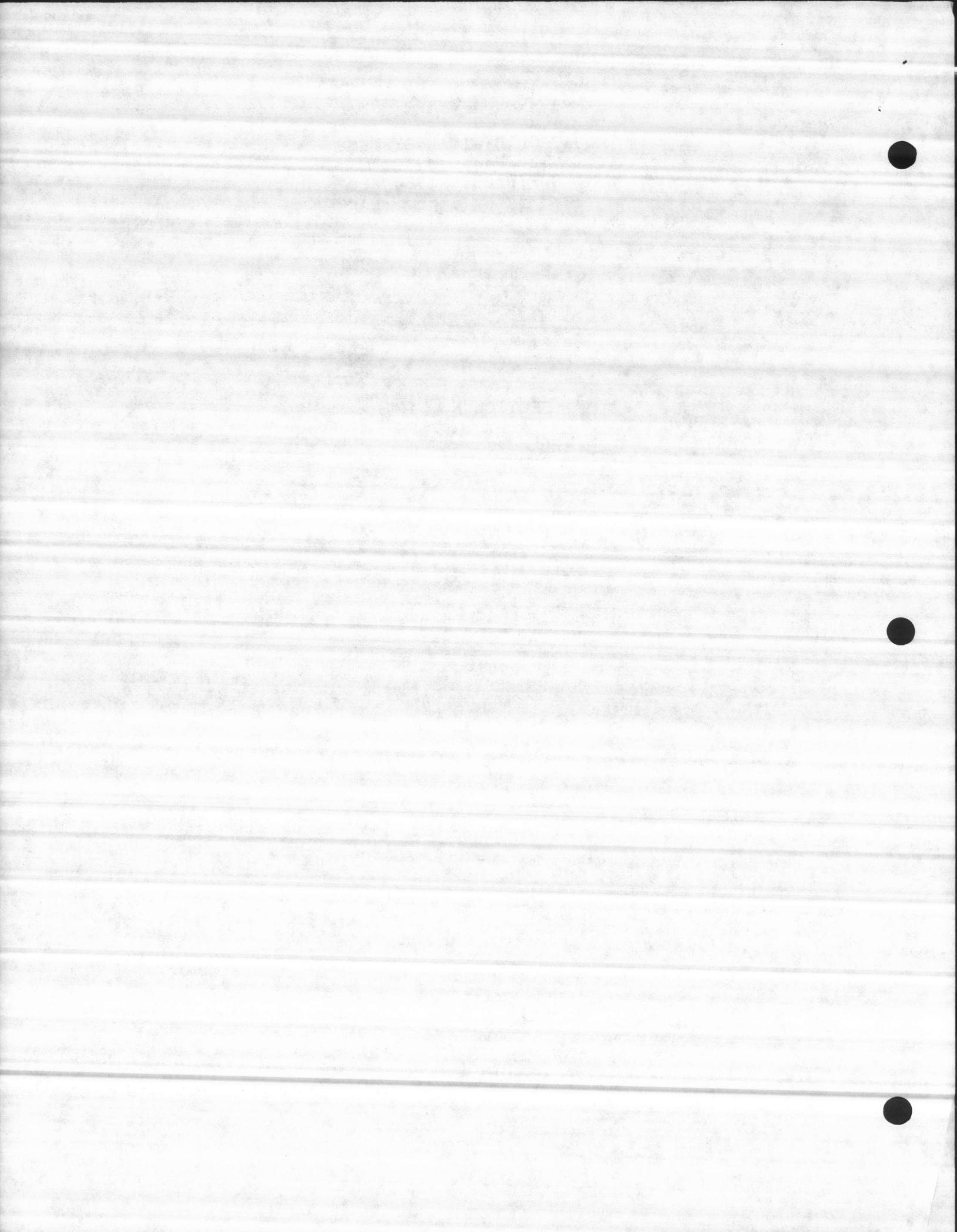
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

ail Module - written by Bob Ryan, Peter Wollenzien, Greg Jensen, M Fayad

(8.1) MODULE NAME : CxxxxFAIL.Pyy

=====

DESCRIPTION :

=====

This procedure prints alarm and event messages, and maintains the alarms structure and associated variables. It is written as a task which must be created in the main init module. After being created, this task waits at the global fail\_sem\_t semaphore for a unit to signal an alarm or event change, and processes this change from the fail\_ring\_buf using recv\_fail\_ring\_buf\_count as an index into this ring buffer.

EXTERNAL MODULES :

=====

```
Display_Alarms_Proc: PROCEDURE (crt_num) EXTERNAL;
  DECLARE crt_num BYTE;
END Display_Alarms_Proc;
```

```
Format_Disk_Error: PROCEDURE (buffer_t,
                                message_str_p,
                                disk_error,
                                disk_operation) EXTERNAL;
  DECLARE (disk_error,
           disk_operation) WORD,
           buffer_t      TOKEN,
           message_str_p  POINTER;
END Format_Disk_Error;
```

MODULE DECLARATION :

=====

exception => A word used for exception condition.

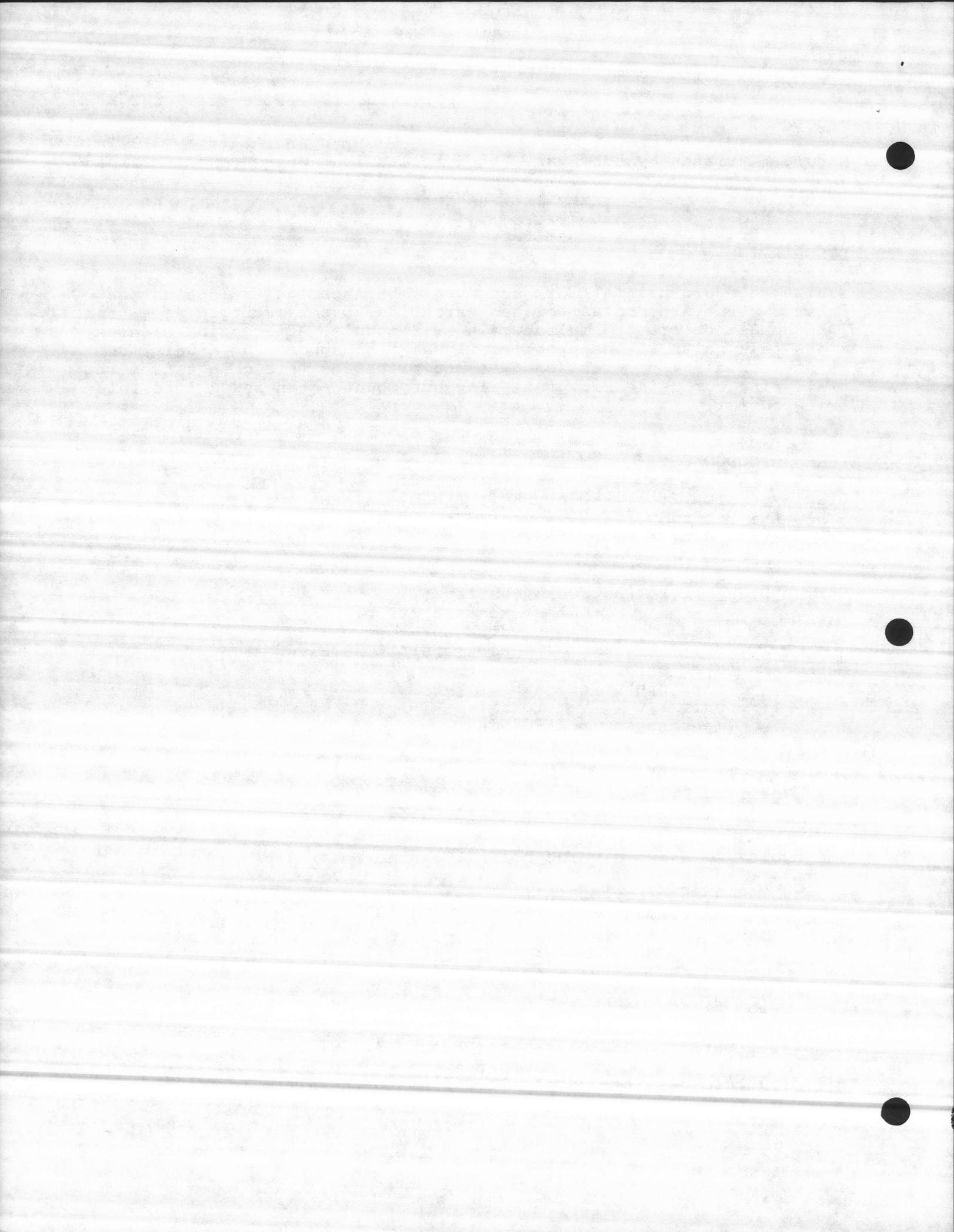
time\_dword => A dword value holds the time.

```
format (4) => Four bytes initially
  format (0) = '>'
  format (1) = 5
  format (2) = 1
  format (3) = 0FFH.
```

spaces (15) => Fifteen bytes hold blanks.

add\_serial\_param BASED disk\_param\_seg\_t STRUCTURE:

-----  
This structure used to add an alarm line to the alarm log file.

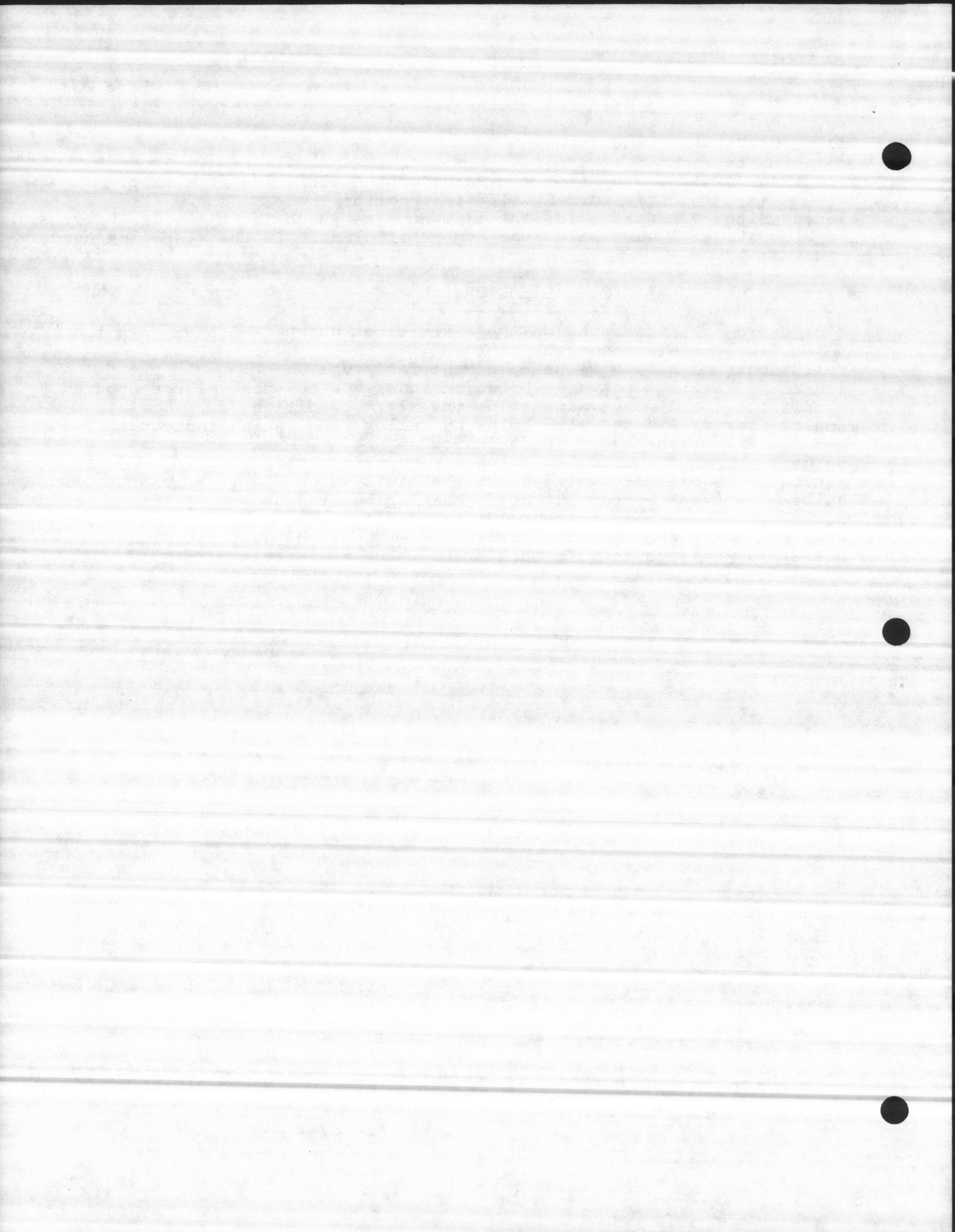


alarm\_log\_header STRUCTURE :

-----  
file\_type -> A word holds the value of the alarm log file type.  
    Ø for current, 1 for historical.  
element\_size -> A word holds the element size which is initially 20 bytes.  
num\_elements -> A word holds the number of element in the alarm log header

alarm\_log\_disk\_data (2) STRUCTURE :

-----  
time\_dword => A dword holds the alarm time, which indicates when the  
alarm occurs.  
fail\_type => A byte value holds the fail\_type.  
index => A word value holds an index for how many alarms occurred.  
condition => A word value holds the alarm condition.  
analog\_flag => A byte value holds an analog flag, which indicates if  
the alarm is an analog or not.  
value => A word value holds the current analog value.  
limit => A word value holds the limit value.  
units => A word value holds the Engineering unit.  
scale => A word value holds the scale factor.  
crit\_flag => A byte value holds the value of the critical flag.  
alarm\_flag => A byte value holds the value of the alarm flag.



---

MODULE PROCEDURES :

---

Send\_To\_Fail:

---

++++++

fail\_type -&gt; A byte holds the value of the fail type.

next\_count-&gt; A byte holds the next count.

index -&gt; A word holds an index.

condition -&gt; A word holds the value of the condition.

## Proc Description :

---

This Procedure used to fill the ring buffer and send a unit to the waiting fail task.

END Send\_To\_Fail;

## Format\_Event\_Entry:

---

++++++

fail\_type -&gt; A byte holds the value of the fail type.

index -&gt; A word holds an index.

analog\_index -&gt; A word holds an index to analog input.

condition -&gt; A word holds the condition value.

buffer\_t -&gt; A token to the buffer.

time\_dword -&gt; A dword holds the time.

## log STRUCTURE :

---

point\_id\_p -> A pointer points to the point Id.

point\_desc\_p -&gt; A pointer points to the point description.

value\_p -&gt; A pointer points to the point value.

condition\_p -&gt; A pointer points to the point condition.

limit\_p -&gt; A pointer points to the limit.

units\_p -&gt; A pointer points to the engineering unit.

scale -&gt; A word holds the scale factor of the analog point.

## Proc description:

---

This procedure used to format an alarm or event entry into a buffer that has been previously cleared and must be sent outside this procedure.

END Format\_Event\_Entry;

## Format\_Disk\_Event\_Entry:

---

++++++

buffer\_t) TOKEN,

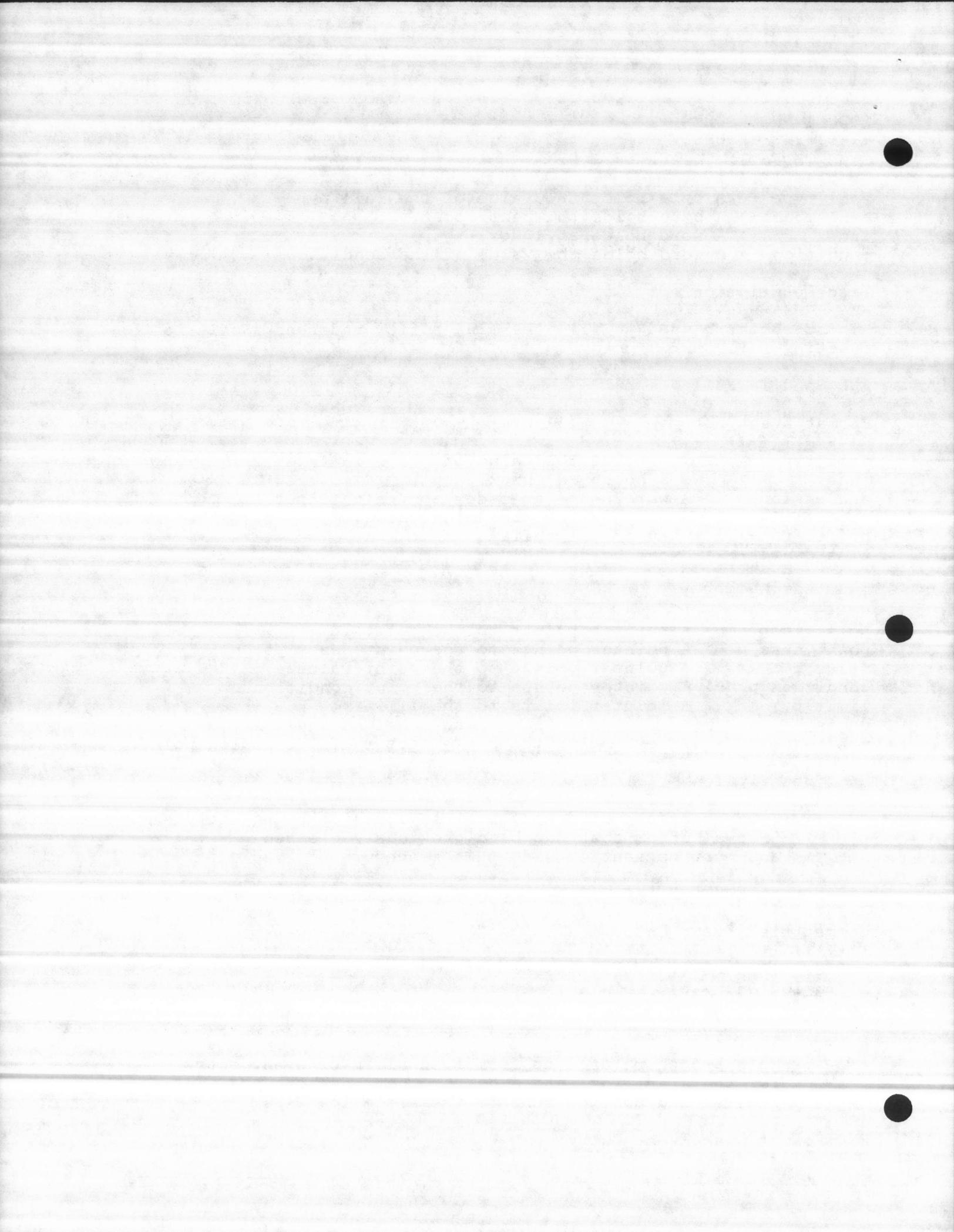
alarm\_log\_entry\_p) POINTER;

## Proc description:

---

This procedure used to format an alarm or event entry for the alarm log into a buffer that has been previously cleared and must be sent to the alarm log file.

END Format\_Disk\_Event\_Entry;



```
Format_Historical_Entry:  
+++++  
buffer_t) TOKEN,  
alarm_log_entry_p) POINTER;
```

Proc description:

-----  
This procedure used to format an alarm or event entry  
for the hist alarm log into a buffer that has been previously  
cleared and must be sent to the outside.

```
END Format_Historical_Entry;
```

```
Format_And_Save_Event_Entry:  
+++++  
Proc description:
```

-----  
This procedure used  
1. Initialize disk save entry.  
2. Check the fail type.  
3. Call format\_disk\_event\_entry.  
4. Add entry to the disk.

```
END Format_And_Save_Event_Entry;
```

```
Print_Event:  
+++++
```

Proc description:

-----  
This procedure prints an alarm or an event when it occurs.

```
END Print_Event;
```

```
Look_Up_Dialer:  
+++++
```

Proc description:

-----  
This procedure looks up the dialer and return the dialer condition.

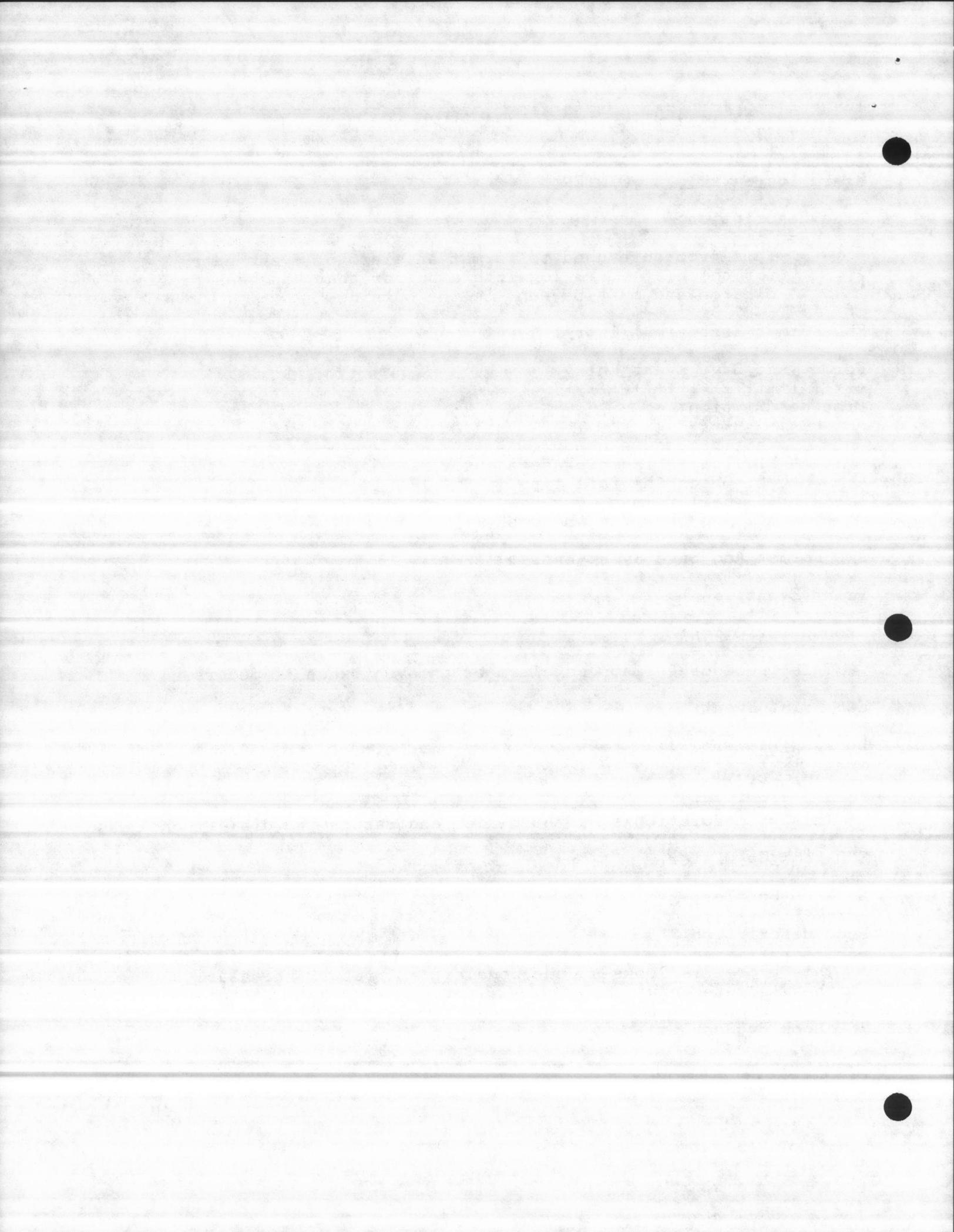
```
END Look_Up_Dialer;
```

```
Look_Up_Sonalert:  
+++++
```

Proc description:

-----  
This procedure looks up the Sonalert and return the alarm\_type of  
the discrete input.

```
END Look_Up_Sonalert;
```



Ack\_Alarms:

++++++

Proc description:

This procedure

1. turns off the sonalert and dialers.
2. clears any locked out alarms.
3. ACK an alarm.
4. resets the last ACK alarm
5. prints an ack message.

END Ack\_Alarms;

Add\_Alarm:

++++++

Proc description:

This procedure

1. Shifts current alarms to make room at the top.
2. Sets the new alarm.
3. Turns On the sonalert and assigned dialer
4. Prints an alarm message.

END Add\_Alarm;

Clear\_Alarm:

++++++

Proc description:

This procedure

1. Removes an alarm.
2. Check lockout option.
3. Clear an alarm entry.
4. Decrement dialer count.
5. Silences sonalert if the last alarm cleared.
6. Prints Clear message.

END Clear\_Alarm;

++++++

FAIL\_TASK

++++++

Fail\_Task:

++++++

units -> A word holds the units.

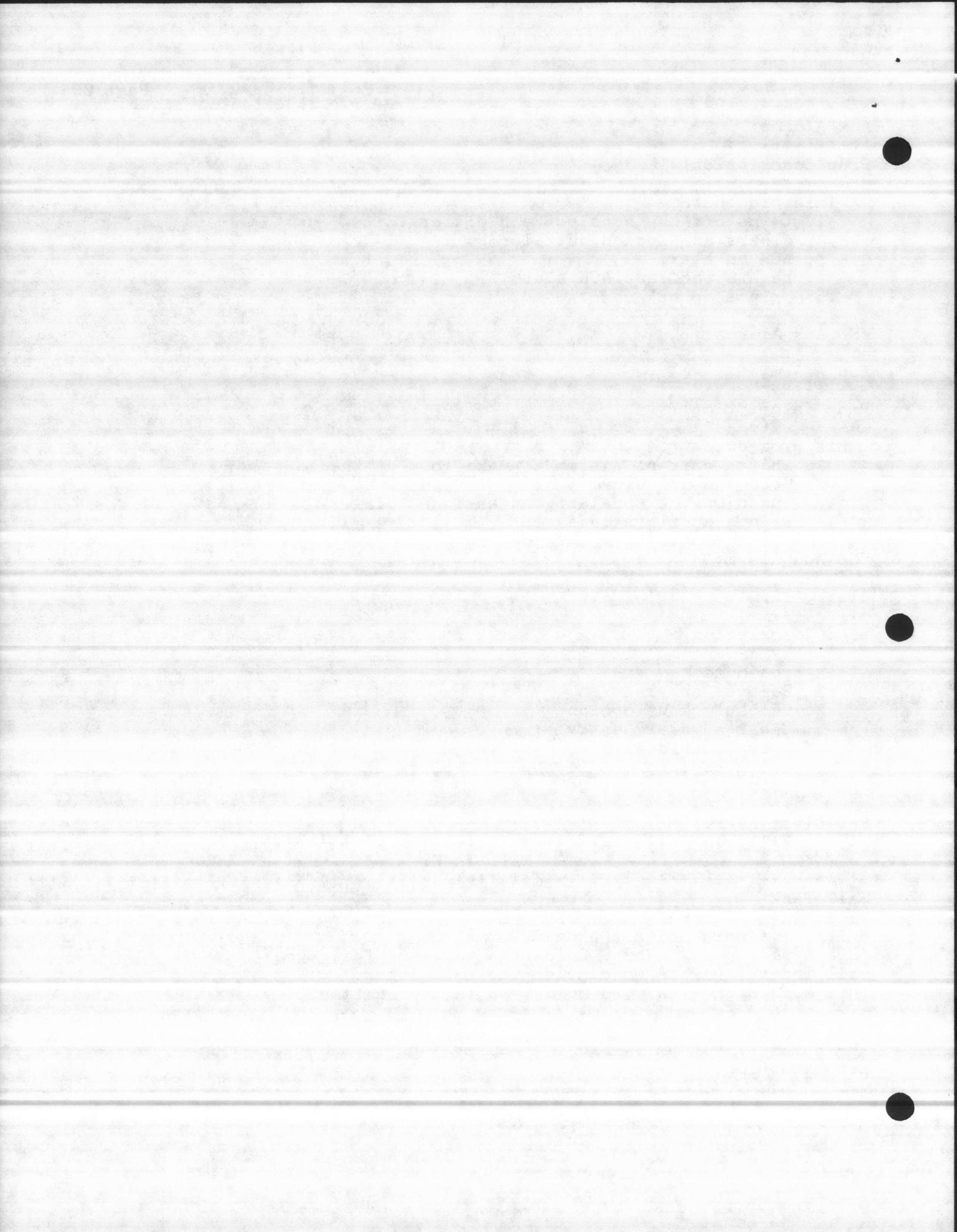
fail\_type -> A byte holds the value of fail\_type.

fail\_index -> A word holds the fail index.

fail\_condition -> A word holds the fail condition.

Task Algorithm:

-----  
set exception handler.



```
DO FOREVER;

    wait for a unit.
    get the current time.
    get control of send_fail_ring_buffer region.
    assign fail_ring_buf structures elements to temp locations
    fail_type , fail_index, fail_condition.

    increment ring buffer pointer.
    release the send_fail_ring_buffer region.

    check fail_type.
        Fail_type = 0, CALL Ack_Alarms;
        Fail_type = 1, check discrete input points & make the proper calls
        Fail_type = 2, null.
        Fail_type = 3, null.

    send list to alarms displays.
END; forever

END Fail_Task;
+++++
+           AUTO_ALARMS_TASK
+++++
Auto_Alarms_Task:
+++++
exception      -> A word holds the exception condition number.
remaining_units-> A word value holds an index.
auto_alarms_sem_t -> A token, used to create an auto alarm semaphore.
display_p         -> A pointer points to the crt display.

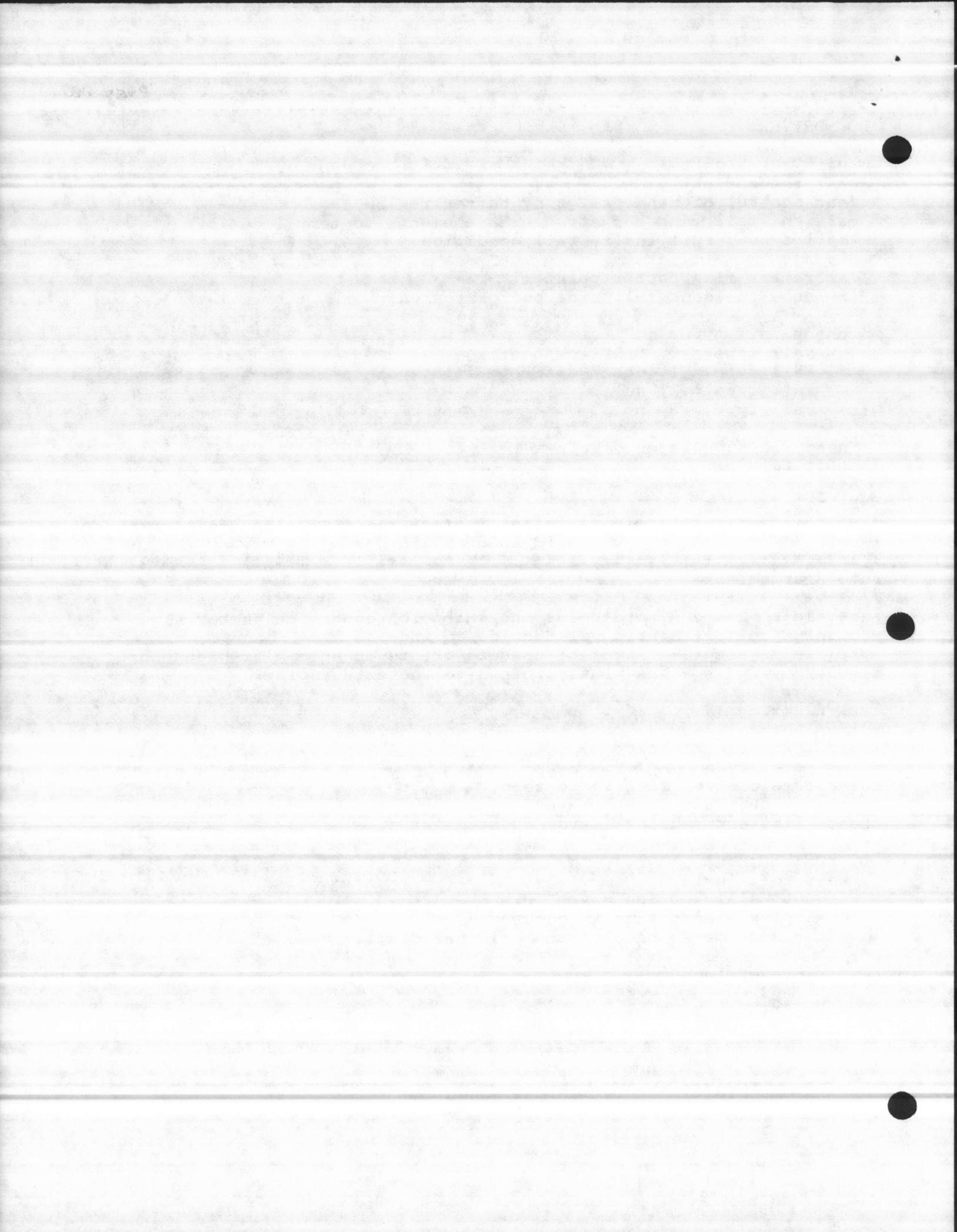
Task Algorithm:
-----
set up exception handler.

set alarms_display_flag(1) to zero;
create aa auto alarm semaphore.
create list entry using alarms_update_list_t.

DO FOREVER;
    wait for a unit.
    If auto_alarm_option = yes THEN
        display alarms when it occurs.
        After silence go back to the original display.

END; Forever loop

END Auto_Alarms_Task;
```



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 11

DISK & FILE SYSTEM

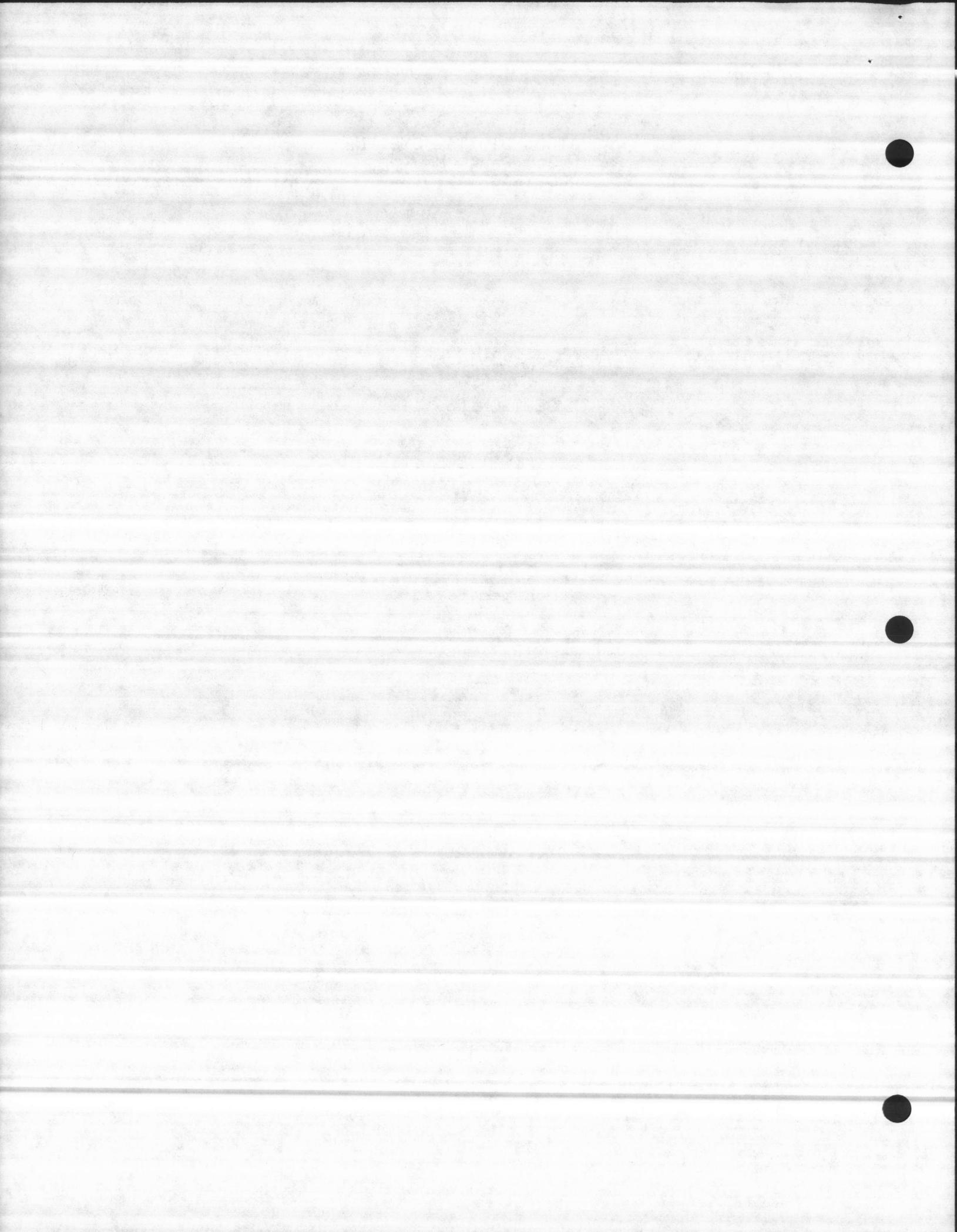
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

The Disk & File System - written by Bob Ryan, Peter Wollenzien.

(11.1) MODULE NAME : CxxxxDISK.Pyy

=====

MODULE DESCRIPTION :

=====

This task provides a system for centralized disk operation.

This disk system makes use of disk headers defined in CxxxxGLOB.PUB. These headers are used in most system calls to this task. This task waits at a mailbox for a parameter segment.

This parameter segment should be region protected for data integrity. The task will then execute the function specified by the parameter segment.

The header\_info is a ram structure which contains the following entries for each file element:

    struc\_name - [Ascii] A six byte ascii name for the structure.

    struc\_loc\_p - [Pointer] The address of the structure.

    struc\_reg\_p - [Pointer] The address of the token to a region protecting the data structure. If a zero is specified no region is used.

The header\_data resides on disk and ram, and contains the following entries for each file element:

    struc\_size - [Word] The total number of bytes required for the full data structure.

    element\_size - [Word] The number of bytes required by a single element of the data structure.

    num\_elements - [Word] The total number of elements within the data structure.

All functions recognize :FD0: as the logical name of the floppy diskette drive, and will automatically attach and detach that device.

Currently the task functions are :

#####

1. Create\_Directory - If no file by the specified name exists, a new directory or data file will be created in that name.

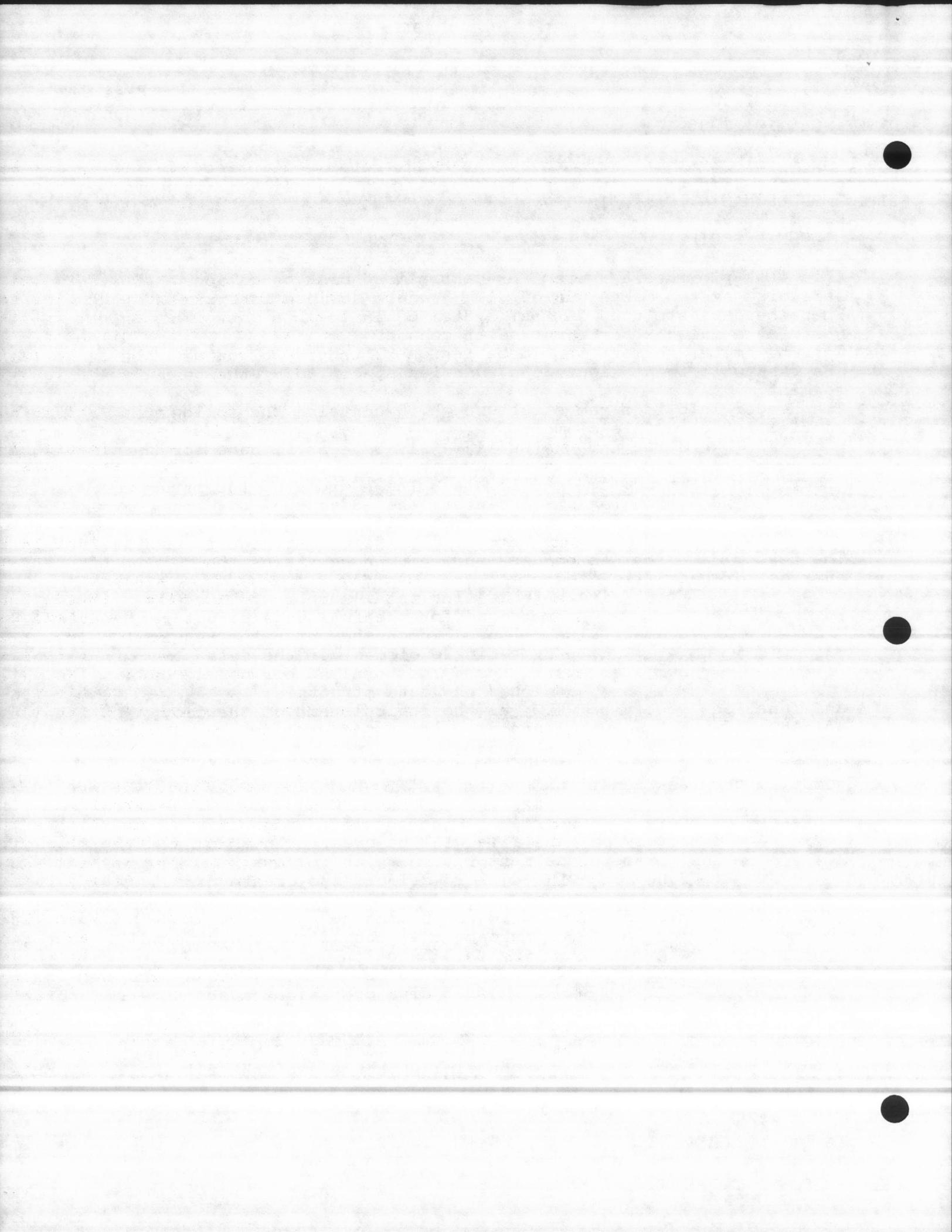
The parameter segment should be as follows:

    function - [Word] Zero selects the create\_directory function.

    disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

    disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

    pathname\_p - [Pointer] The address of an rmx string containing the name of the file to be created.



2. Delete\_File - The specified file will be deleted.

The parameter segment should be as follows:

function - [Word] One selects the delete\_file function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

pathname\_p - [Pointer] The address of an rmx string containing the name of the file to be deleted.

3. Write\_File - The specified file will be written in accordance with the header data and info stored in ram. The file is written to zeros before the data is written. After the data is written, the file pointer is moved back to the start, and the header is written in accordance with the data written.

The parameter segment should be as follows:

function - [Word] Two selects the write\_file function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

pathname\_p - [Pointer] The address of an rmx string containing the name of the file to be written.

header\_info\_p - [Pointer] The address of the standard header info structure for the file to be written.

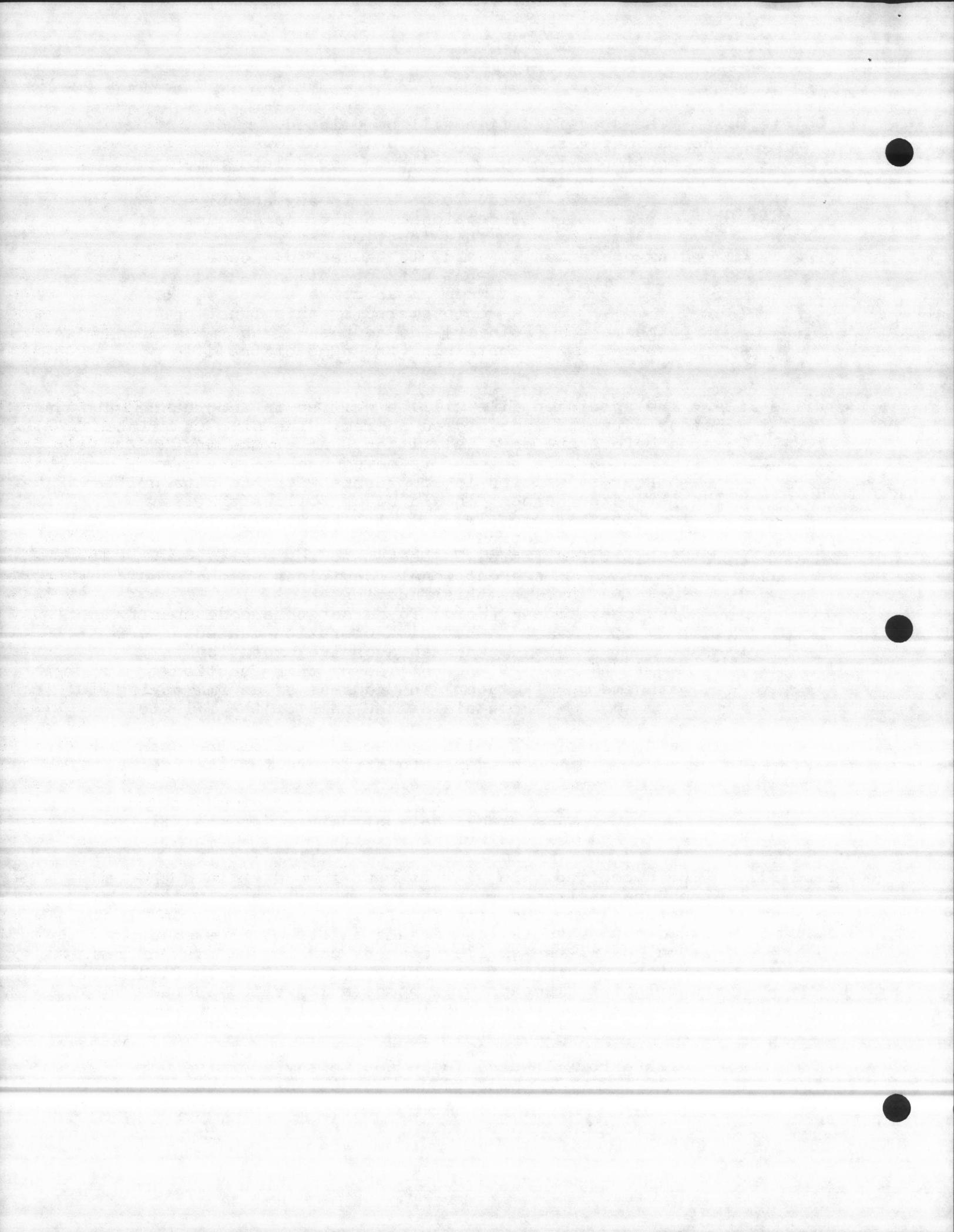
header\_data\_p - [Pointer] The address of the standard header data structure for the file to be written.

4. Read\_File - The specified file will be read from disk into the ram structures selected. The function will default fill the structures from the zeroeth elements, and then read in the disk data. Data is read from disk structure element by element using the smaller element\_count from disk or ram. Therefore, any longer disk elements will not be read, and shorter disk elements will be default filled in ram.

The parameter segment should be as follows:

function - [Word] Three selects the read\_file function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.



disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

pathname\_p - [Pointer] The address of an rmx string containing the name of the file to be read.

header\_info\_p - [Pointer] The address of the standard header info structure for the file to be read.

header\_data\_p - [Pointer] The address of the standard header data structure for the file to be read.

5. Write\_Block - A specified number of blocks will be written at a specified start location within a selected file. Should the initial offset exceed the existing element length the function will return with the initial offset set to  $\text{FFFFH}$  and the bytes per write set to  $0$ . Should the header element or the data structure element not exist on disk, or the data structure element offset exceed the element size on disk, the function will return with the number of bytes of new data set to zero and the erroneous value set to  $\text{FFFFH}$ . Also, if the number of bytes of new data plus the data structure element offset exceeds the disk element size, the number of bytes of new data will be truncated so as not to overwrite the data structure element on disk.

The parameter segment should be as follows:

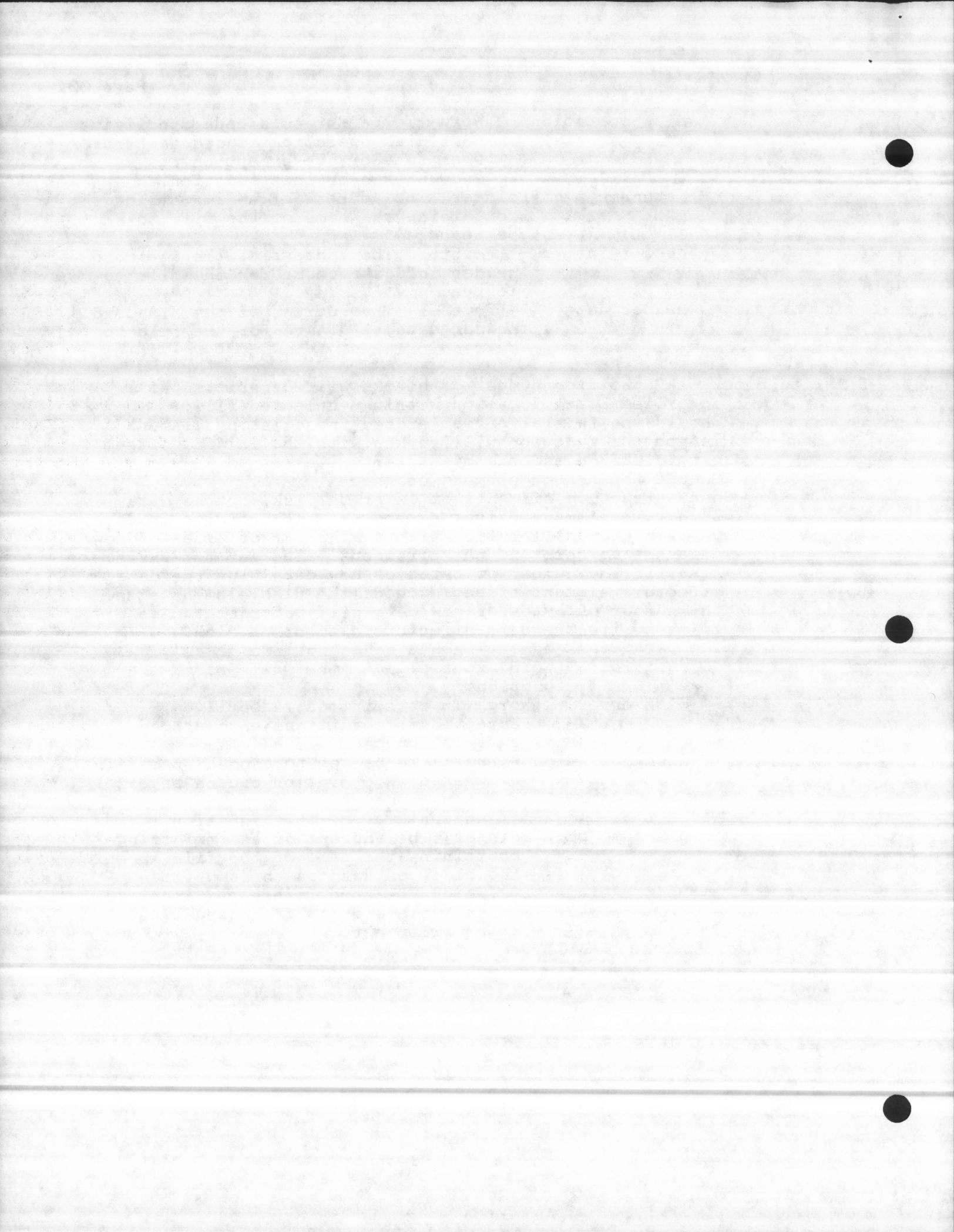
function - [Word] Four selects the write\_block function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

pathname\_p - [Pointer] The address of an rmx string containing the name of the file in which the block of data is to be written.

header\_element - [Word] The zero referenced index to the file element data structure to be written to.

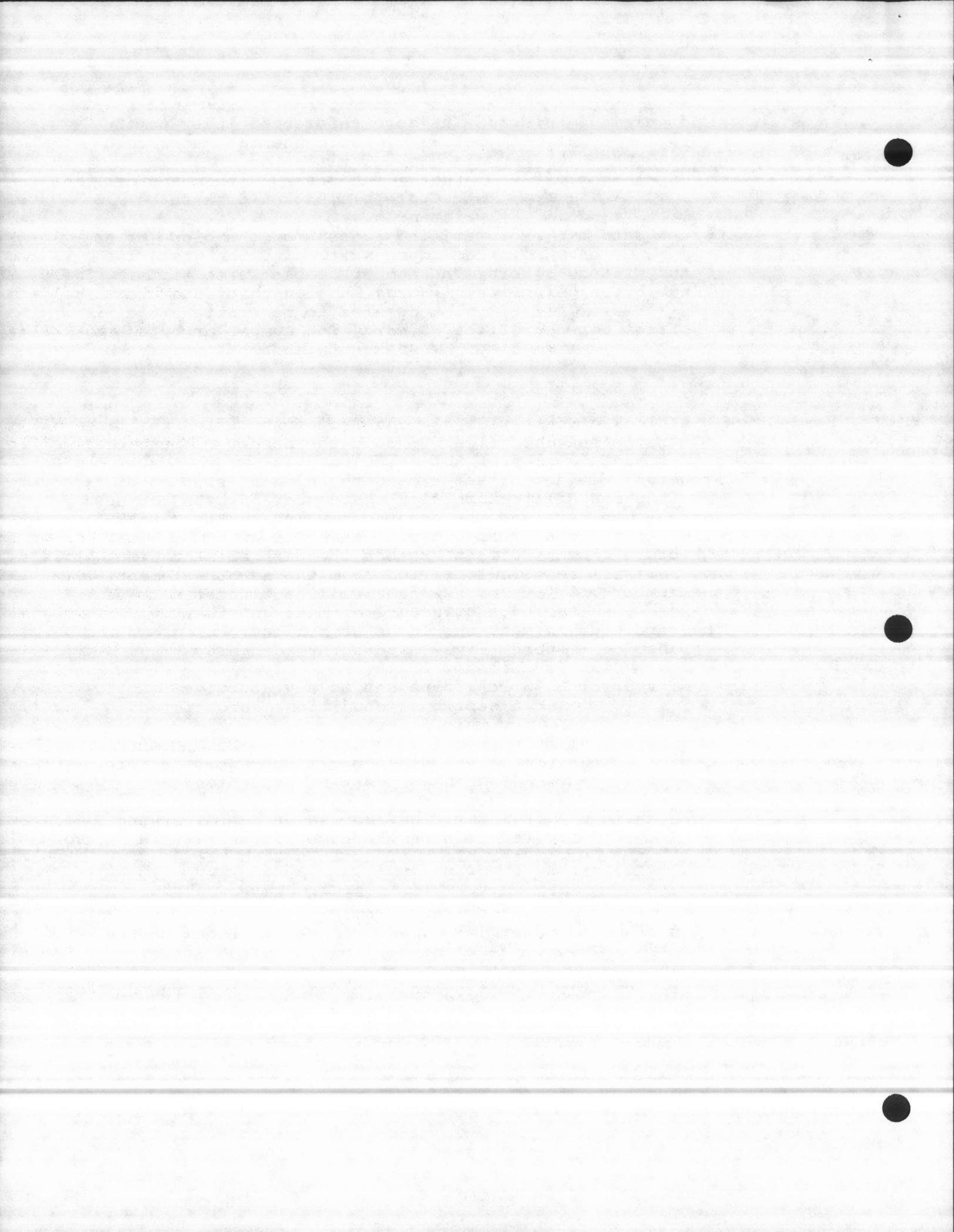


ds\_element - [Word] The zero referenced index to the element within the data structure on disk to be written to.  
ds\_element\_offset - [Word] The offset within the data structure element on disk to be written to.  
new\_data\_p - [Pointer] The address of the block of data in ram to be written to the disk file.  
num\_writes - [Word] The number of blocks to write.  
num\_bytes\_per\_write - [Word] The length of the block desired to be written to disk.  
offset\_between\_writes - [Word] The number of bytes to seek between block writes.

6. Read\_Block - A block of data will be read from a specified position within the selected file. Should the header element or the data structure element not exist on disk, or the data structure element offset exceed the element size on disk, the function will return with the number of bytes of new data set to zero and the erroneous value set to 0FFFFH. Also, if the number of bytes of new data plus the data structure element offset exceeds the disk element size, the ram will be default filled from the zeroth element, and the number of bytes of new data will be truncated so as not to read past the data structure element on disk.

The parameter segment should be as follows:

function - [Word] Five selects the read\_block function.  
disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.  
disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.  
pathname\_p - [Pointer] The address of an rmx string containing the name of the file in which the block of data is to be read.  
header\_element - [Word] The zero referenced index to the file element data structure to be read from.  
ds\_element - [Word] The zero referenced index to the element within the data structure on disk to be read from.  
ds\_element\_offset - [Word] The offset within the data structure element on disk to be read from.



new\_data\_p - [Pointer] The address of the block of data  
in ram to be filled from the disk read.  
num\_bytes\_new\_data - [Word] The length of the block  
desired to be read from disk.

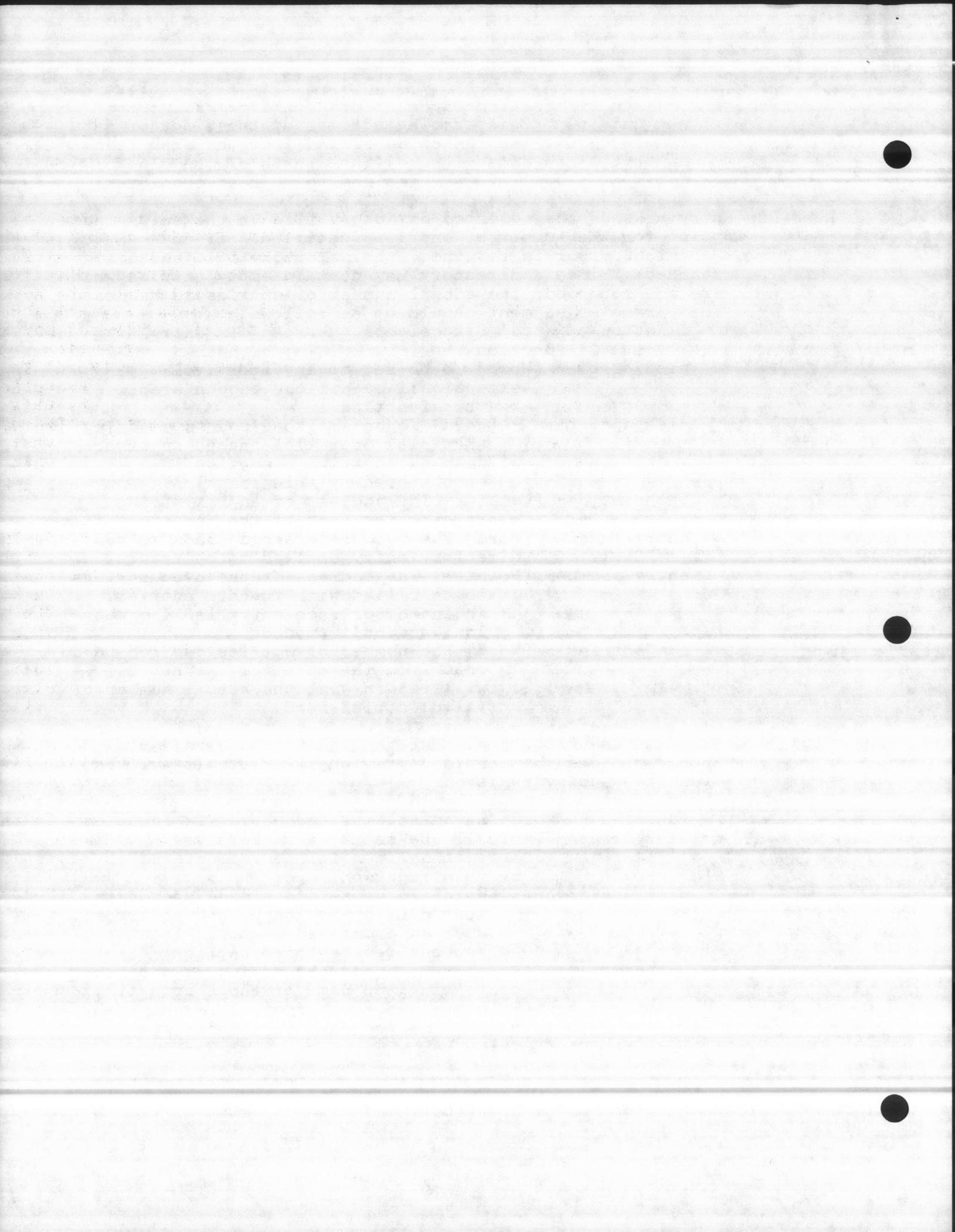
7. Read\_Dir\_Entry - Reads the selected directory file at the entry index point for the number of entries specified. If an E\$DIR\$END error is returned, the directory file end has reached. Since a directory may contain empty entries which are not returned, the actual number of entries is returned. The parameter segment should be as follows:

function - [Word] Six selects the read\_dir\_entry function.  
disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.  
disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.  
pathname\_p - [Pointer] The address of an rmx string containing the name of the directory file.  
entry\_index - [Word] The zero referenced index to the directory entry desired to be read.  
entry\_asc\_p - [Pointer] The address of the byte strings to be filled with the dir entries. The name for each entry is 14 bytes long and null padded  
num\_entries - [Word] The number of entries desired to be read from the directory. The call will return with the actual number of valid entries read.

8. Copy\_File - Copies one file to another through the disk buffer. Should the destination file previously exist, it will be truncated and overwritten.

The parameter segment should be as follows:

function - [Word] Seven selects the copy\_file function.  
disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.  
disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.  
pathname1\_p - [Pointer] The address of an rmx string containing the name of the source file.  
pathname2\_p - [Pointer] The address of an rmx string containing the name of the destination file.



9. Floppy\_Format - Formats the floppy diskette as a named volume.

The parameter segment should be as follows:

function - [Word] Eight selects the floppy\_format function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

vol\_name\_count - [Byte] The length of the volume name

vol\_name\_asc - [Ascii] A six byte ascii string of the desired volume name for the floppy diskette.

10. Rename\_File - Changes the name of a disk file.

The parameter segment should be as follows:

function - [Word] Nine selects the rename\_file function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

pathname\_p - [Pointer] The address of an rmx string containing the name of the file to be changed.

new.pathname\_p - [Pointer] The address of an rmx string containing the desired new name or the file.

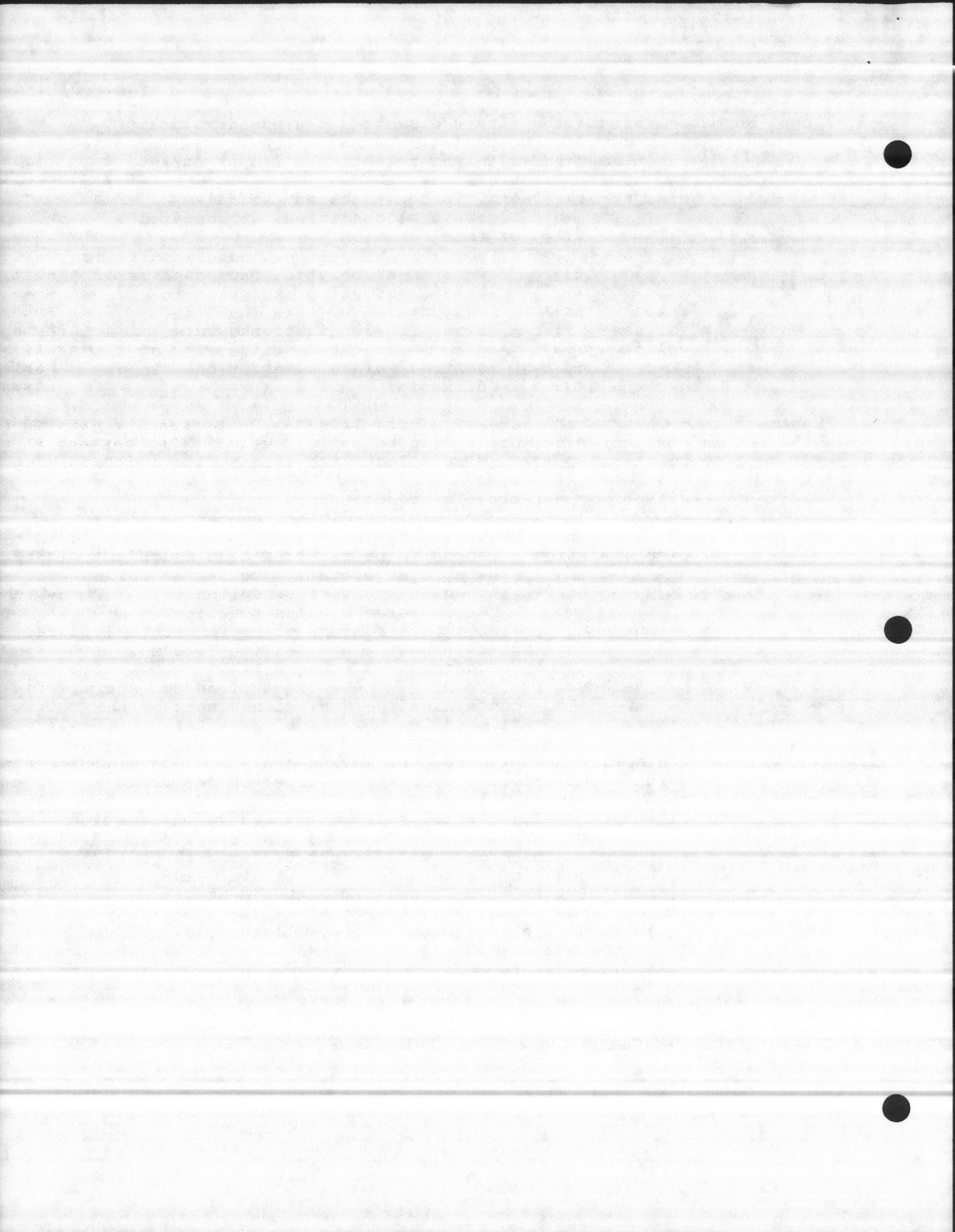
11. Add\_Serial - This file type is used for saving alarm logs on disk.

This function will keep up a small header on the front of the file of the following format :

file\_type (WORD) - This is for future usage in case some kind of id is needed for a historical read. For now, this field will be set to zero. It could hold the number of bytes in the file's header.

element\_size (WORD) - Bytes per element entry.

num\_elements (WORD) - Number of elements held in the current file. Since this file will grow, every time elements are appended to the end of the file this field is incremented.



The actual function of this file is to store structure entries by appending them to the end of the file.

The parameter segment should be as follows:

function - [Word] Ten selects the add\_serial function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

pathname\_p - [Pointer] The address of an rmx string containing the name of the file to be written. If the file does not exist the file is created and its header initialized.

header\_data\_p - [Pointer] The address of a header data structure for the file. This will be used primarily for file initialization.

data\_info\_p - [Pointer] The address of the data to be written. This data should be an one referenced structure, i.e., a Øth element default element should always exist (even though this element is never written).

num\_elements - [Word] This is the number of elements to append to the end of the file.

12. Read\_Serial - This function will read elements off of a serial file

The parameter segment should be as follows:

function - [Word] 11 selects the read\_serial function.

disk\_error - [Word] To be set to any abortive exceptional conditions encountered by this function.

disk\_operation - [Word] To be set to a code specifying what disk operation caused an error when an abortive exception is encountered by this function.

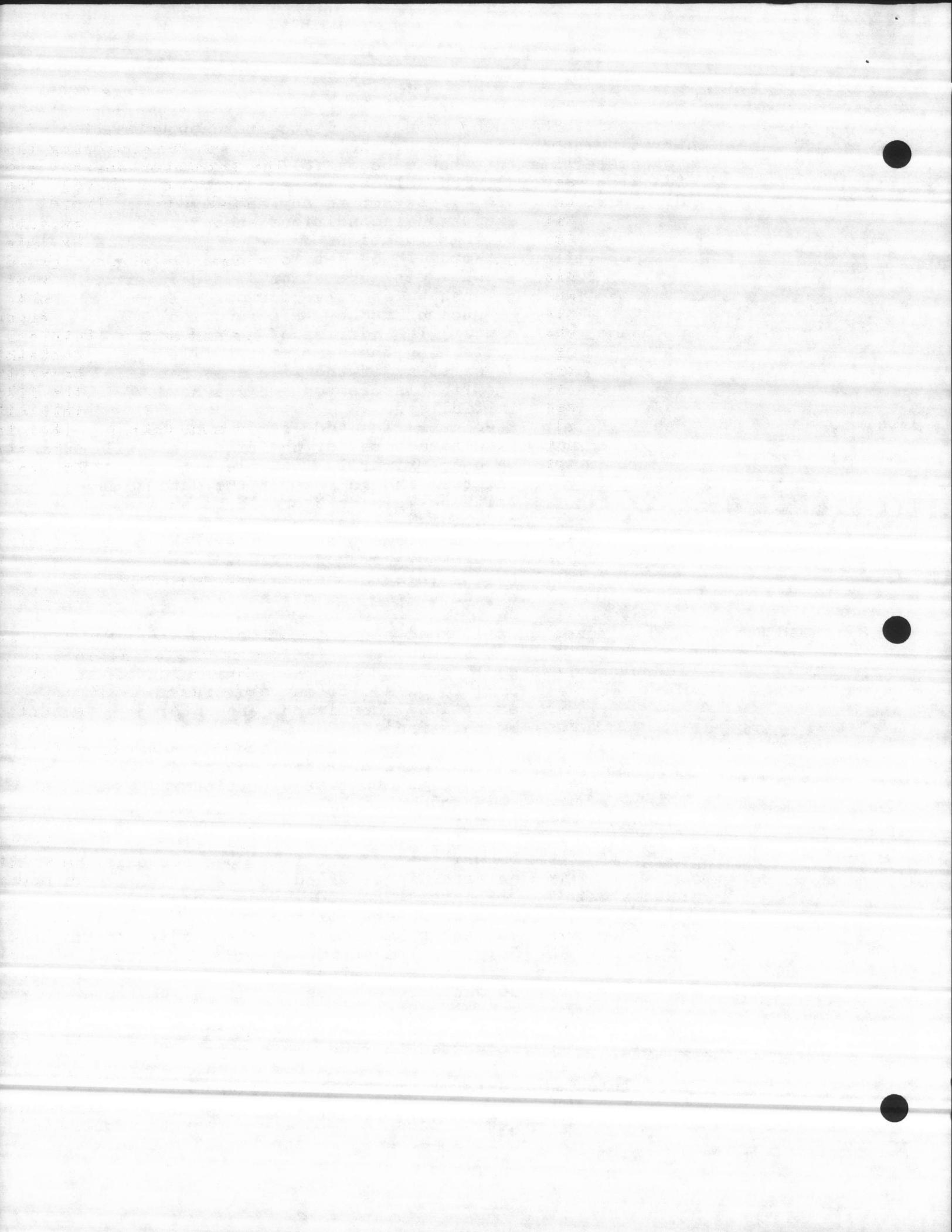
pathname\_p - [Pointer] The address of an rmx string containing the name of the file to be written. If the file does not exist an error id returned.

header\_data\_p - [Pointer] The address of a header data structure for the file. This will be used primarily for file justification.

data\_info\_p - [Pointer] The address to where the read in data is written . This data should be an one referenced structure so that the Øth element can be used for defaults.

num\_elements\_read - [Word] This is the number of elements requested to read. This entry will return with the actual number of entries read.

num\_elements\_offset - [Word] This is the number of elements to skip before reading. The idea behind this is to allow a task to request a file perhaps ten entries at a time.



## FORMAT DISK ERROR

\*\*\*\*\* W8016 SYSTEM DISK LIBARIES \*\*\*

\*\*\*\*\* Format\$Disk\$Error \*\*\*\*\*

This procedure will format a message from a disk operation. The message is broken into 4 parts, the time, a user defined message, the error message, and the operation performed at time of error.

- 1) The time will be the time that the message is formatted.
- 2) The user defined message will probably be a description of the disk application being performed.
- 3) The error message is determined from the system exception code. Specific errors are checked and the following messages are formatted.

E\$FACCESS - File Access.  
 E\$IO\$HARD, E\$IO\$SOFT, E\$IO\$UNCLASS, E\$IO - IO.  
 E\$IO\$OPRINT, E\$MEDIA - Device Offline.  
 E\$IO\$WRPROT - Write Protect.  
 E\$SPACE - No Space Available.  
 E\$FNEXIST - File Not Found.

All other error simply print out an error message with the system error code.

If no error then the word successful is formatted in place of an error message.

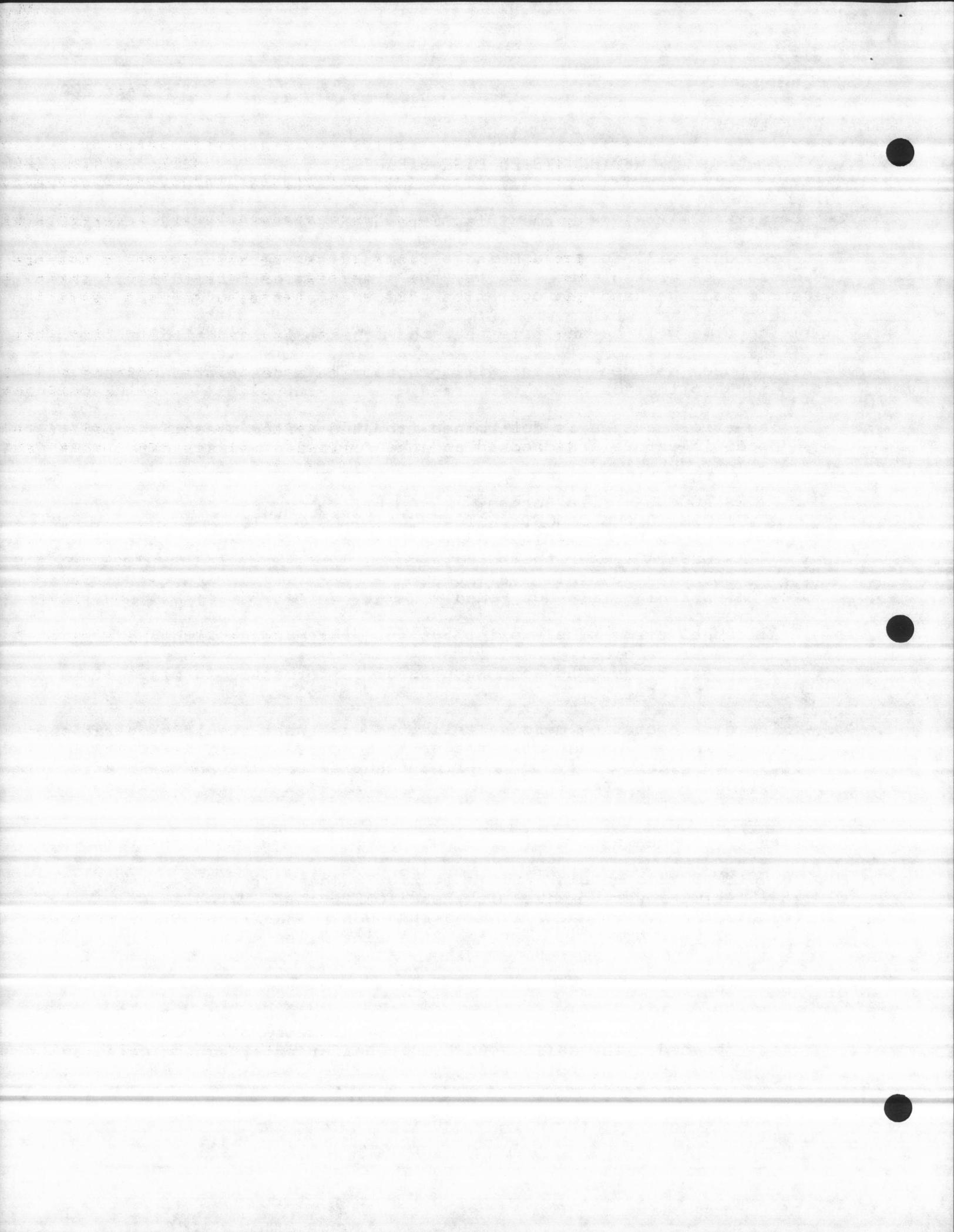
- 4) The disk operation message tells the user what specific operation was being performed while the error occurred. This message will be printed only if an error occurred and the short format option is not specified.

Examples of the messages are:

010178 00:00:00 System Disk Initialization Successful  
 010178 16:45:01 Power Up File Disk Read IO Error  
 010178 20:20:00 Historical File Read File Not Found Error  
    while Attaching File

## INPUT PARAMETERS:

buffer\_t - A token to an open buffer where the message is to be placed. The calling task must buffer prior to the call, and send the buffer after the call.



**message\_str\_p** - A pointer to a string to be placed within the formatted message. The string shall be Rmx format with maximum length of 40 bytes.

**format\_type** - A byte with valid values of 0 for short message format, 1 for long message format, and 2 for extended message format.

0) - The short message format will not format the disk operation message.

1) - The long message format will contain a CR,LF for 80 column output.

2) - The extended message format will format the full message on one line.

**exception\_p** - A pointer to the 2 word array disk exception used by the disk procedures.

#### \*\*\*\*\* Disk\$Create\$File \*\*\*\*\*

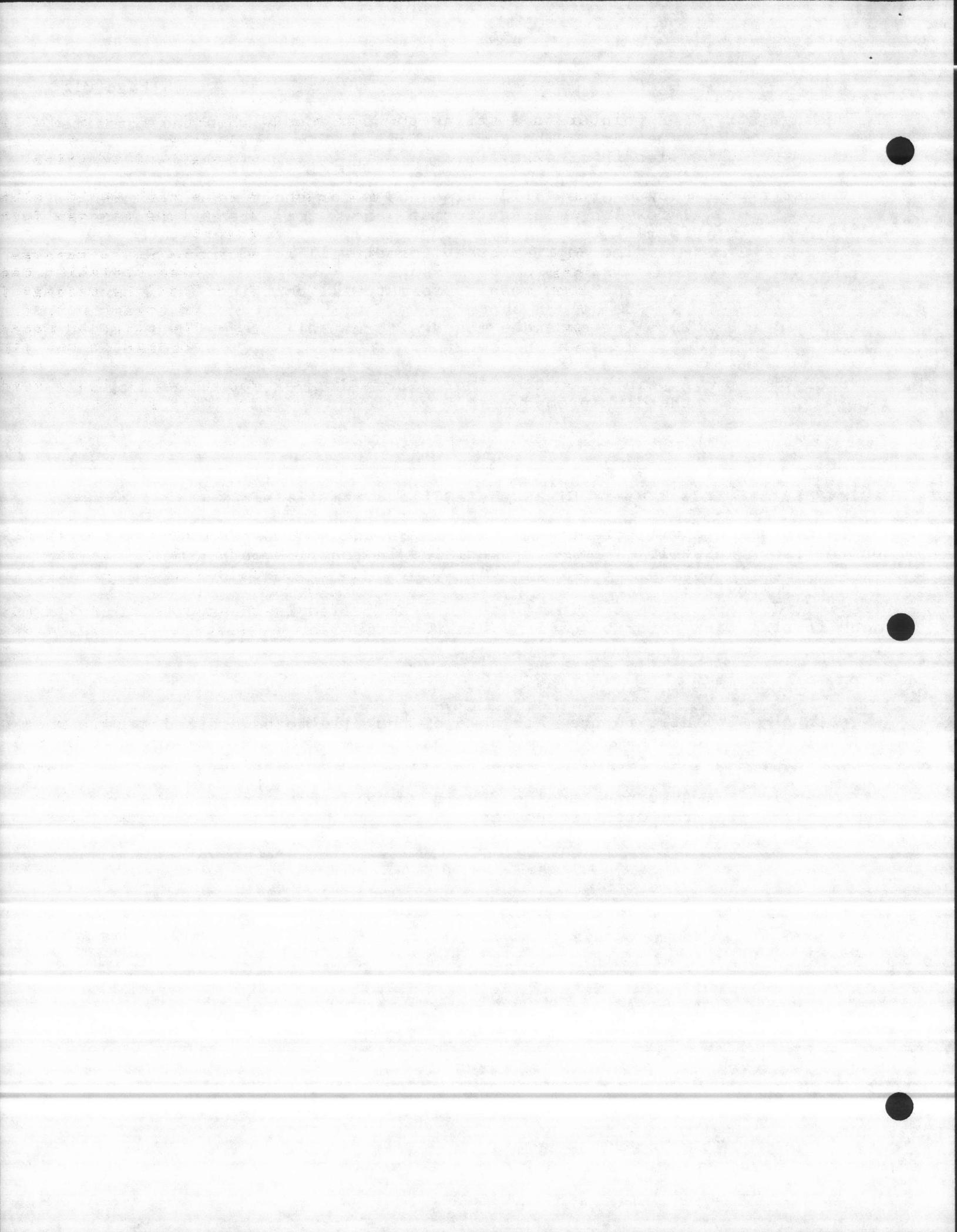
This procedure will create a disk file on the winchester disk, or on the floppy disk if ':FD0:' is specified in the file path name. If the specified file already exists, it will be truncated to a length of 0 bytes.

The init library procedure Init\$Disk must be called in the application jobs initial task prior to calling Disk\$Create\$File.

#### INPUT PARAMETERS:

**path\_p** - A pointer to the path name of the file to be created. The path name should be a RMX string with maximum length of 20 bytes.

**exception\_p** - A pointer to the 2 word array disk exception used by the disk procedures.



## D I S K   T A S K

## MODULE DECLARATIONS :

exit\_flags => A byte holds the exit\_flags value.  
exception => A word holds the exception condition number.

response\_mbx\_t => A token used for receiving response mbx in main mbx  
receive message call

param\_seg\_t => A token used to access function code of parameter  
segment received by main mbx receive message call

segment created by init for holding headers and structure elements  
disk\_buffer BASED disk\_buffer\_seg\_t) (1000H) BYTE;

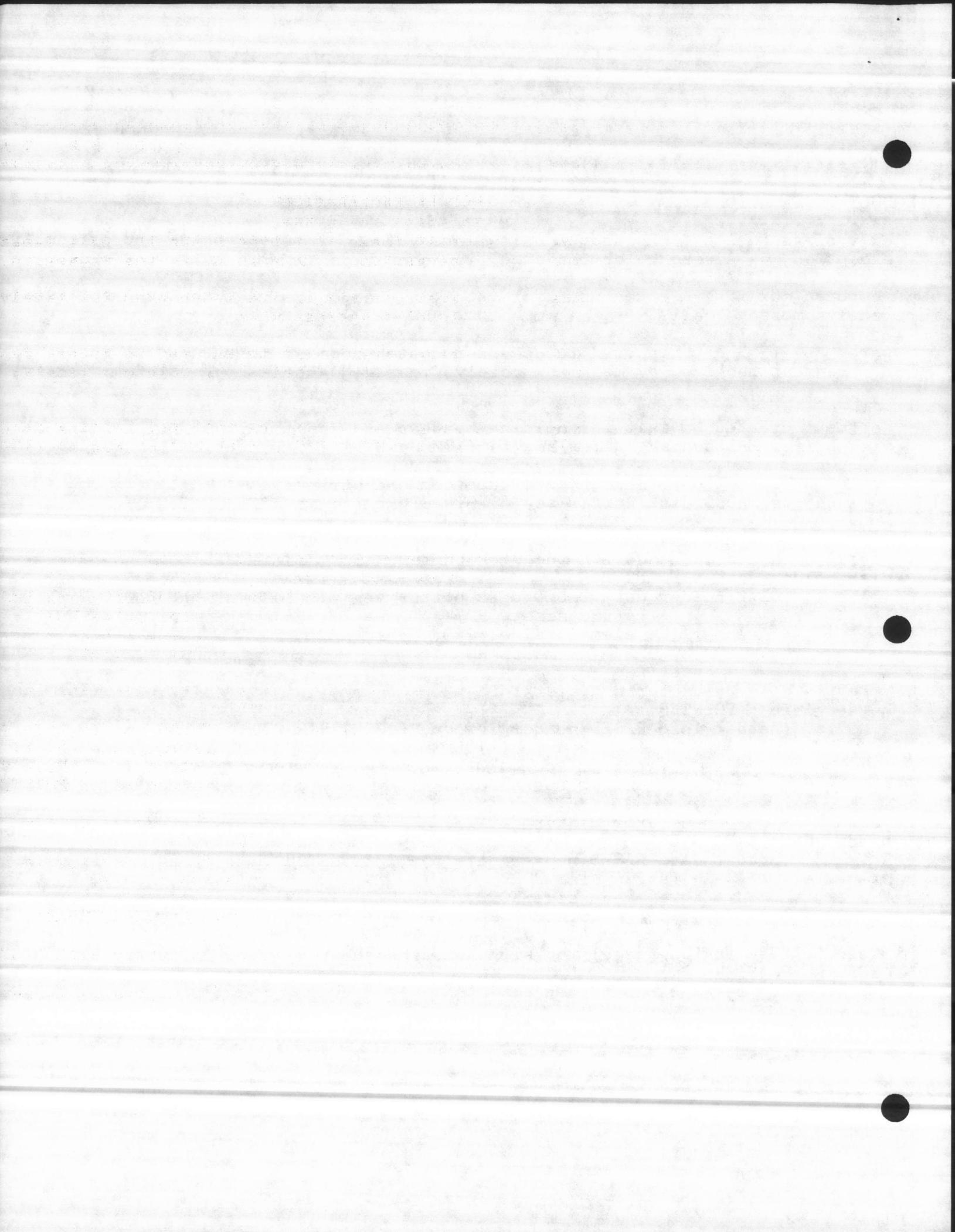
used for formation of human interface command strings  
command\_str BASED disk\_buffer\_seg\_t) STRUCTURE (count BYTE,  
asc (79) BYTE);

used to base disk header in the disk buffer memory segment  
disk\_header\_data\_p POINTER,  
(disk\_header\_data BASED disk\_buffer\_seg\_t) (100H) STRUCTURE  
(struc\_size WORD,  
element\_size WORD,  
num\_elements WORD);

used to base parameter's header\_info\_p  
header\_info\_p POINTER,  
(header\_info BASED header\_info\_p) (100H) STRUCTURE  
(struc\_name (6) BYTE,  
struc\_loc\_p POINTER,  
struc\_reg\_p POINTER);

used to base parameter's header\_data\_p  
header\_data\_p POINTER,  
(header\_data BASED header\_data\_p) (100H) STRUCTURE  
(struc\_size WORD,  
element\_size WORD,  
num\_elements WORD);

used to base region associated with header element's data structure  
ds\_reg\_p POINTER,  
(ds\_reg\_t BASED ds\_reg\_p) TOKEN;



+-----  
+ Create Directory Procedure +  
+-----

ALGORITHM:

=====

Create\_Directory\_Proc:

-----  
set exit\_flags = Ø;

attach the floppy diskette drive if needed

attempt to attach the file

file doesn't exist so create it

Exit:

delete the file connection

detach the floppy diskette drive

END Create\_Directory\_Proc;

+-----  
+ Delete File Procedure +  
+-----

ALGORITHM :

=====

Delete\_File\_Proc:

-----  
exit\_flags = Ø;

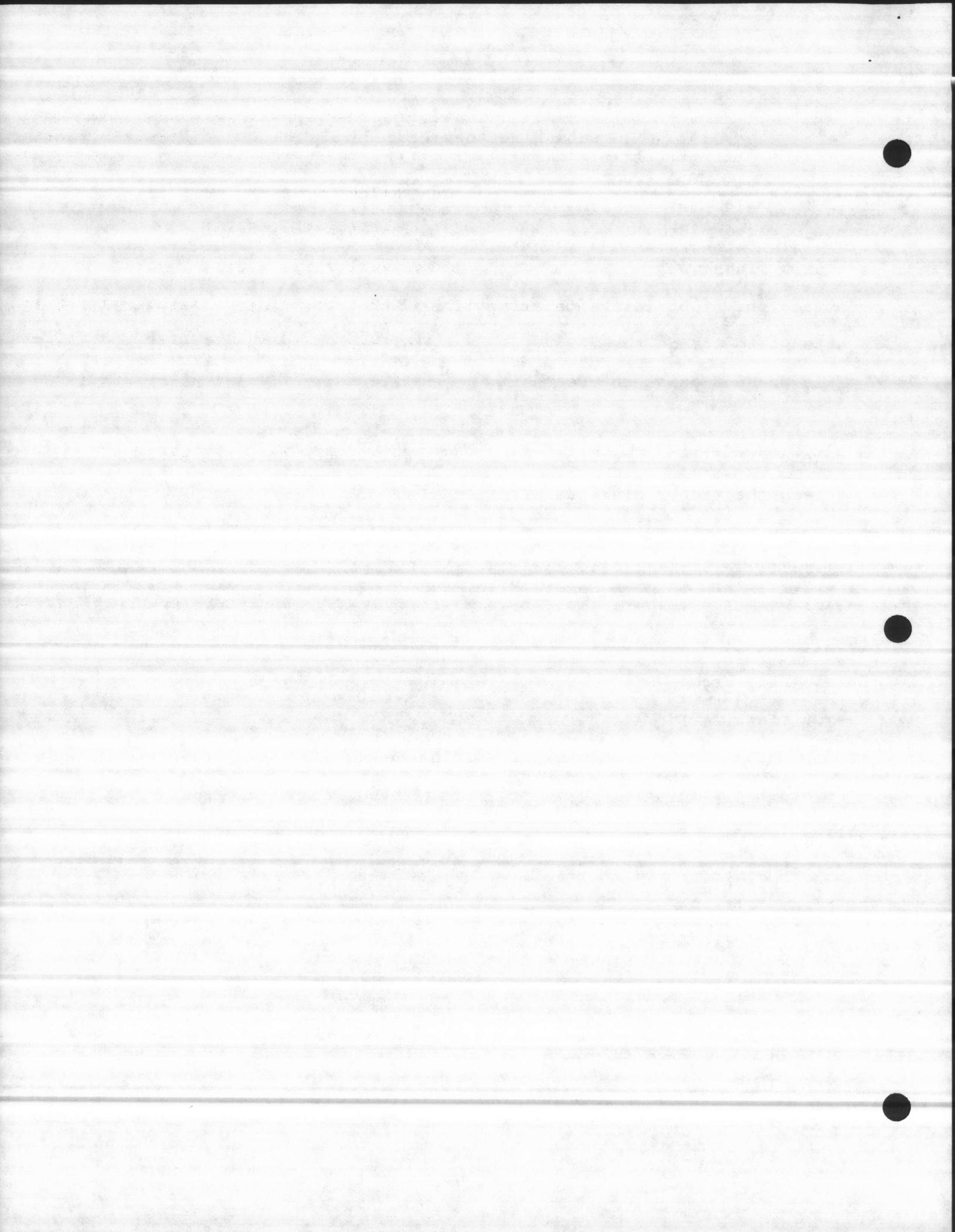
attach the floppy diskette drive if needed

delete the file

Exit:

detach floppy diskette drive

END Delete\_File\_Proc;



```
+++++
          Write File Procedure
+
ALGORITHM :
=====
Write_File_Proc :
=====
    exit_flags = Ø;
    transfer pointers for local basing
    attach the floppy diskette drive if needed
        GOTO Exit;
    exit_flags = 1;

    attach the file to be written
    GOTO Exit;

    create file if necessary
    GOTO Exit;

    exit_flags = exit_flags OR 2;

    open the file connection for writing
    GOTO Exit;

    write zeros to disk header
    element_num = Ø;
    GOTO Exit;

    valid_element_num = Ø;

    get control of region if needed
    write to disk
    send control of region

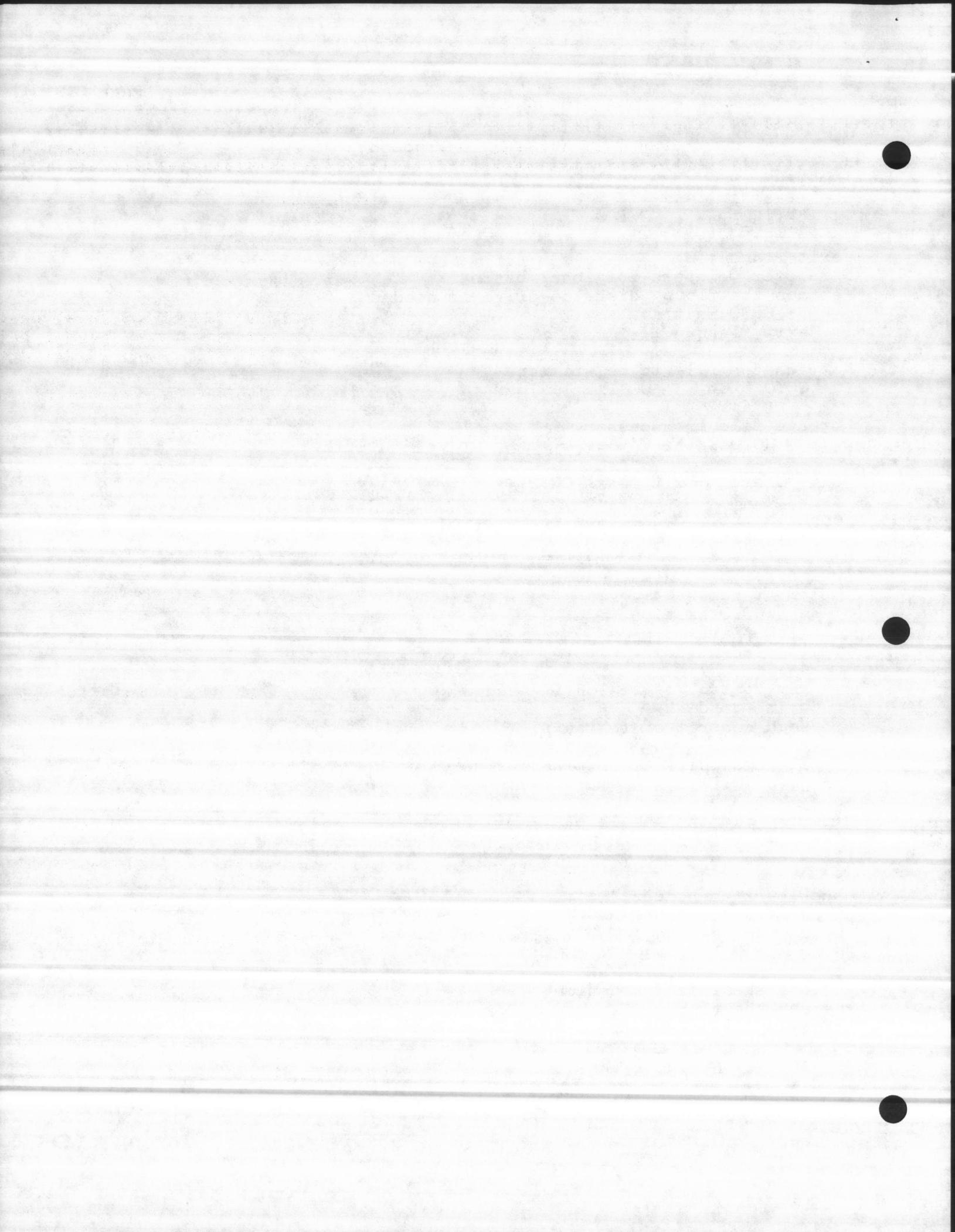
    IF param.disk_error <> E$OK THEN
        GOTO Fill;

    END;   element_num loop

Fill:
    seek back to header
    ELSE   write header

Exit:
    delete the file connection
    detach the floppy diskette drive
    IF exit_flags THEN

END Write_File_Proc;
```



+  
+++++ Read File Procedure +  
+++++

ALGORITHM:

=====

Read\_File\_Proc:

=====

exit\_flags = Ø;

transfer pointers for local basing  
header\_info\_p = param.header\_info\_p;  
header\_data\_p = param.header\_data\_p;

attach the floppy diskette drive if needed

attach the file to be read

open the file connection for reading

zero fill disk header buffer

READ:

read in first six bytes of header

check disk header parameters

read in all of header (use read disk header's size)

Fill:

go thru ram header and fill elements

get region if defined

default fill ram structure from Øth structure element  
Read from disk

read in structure block

OR read structure element by element.

default fill ram structure from Øth structure element

seek forward past zeroeth disk structure element

seek forward to start of next element

Read from disk ended.

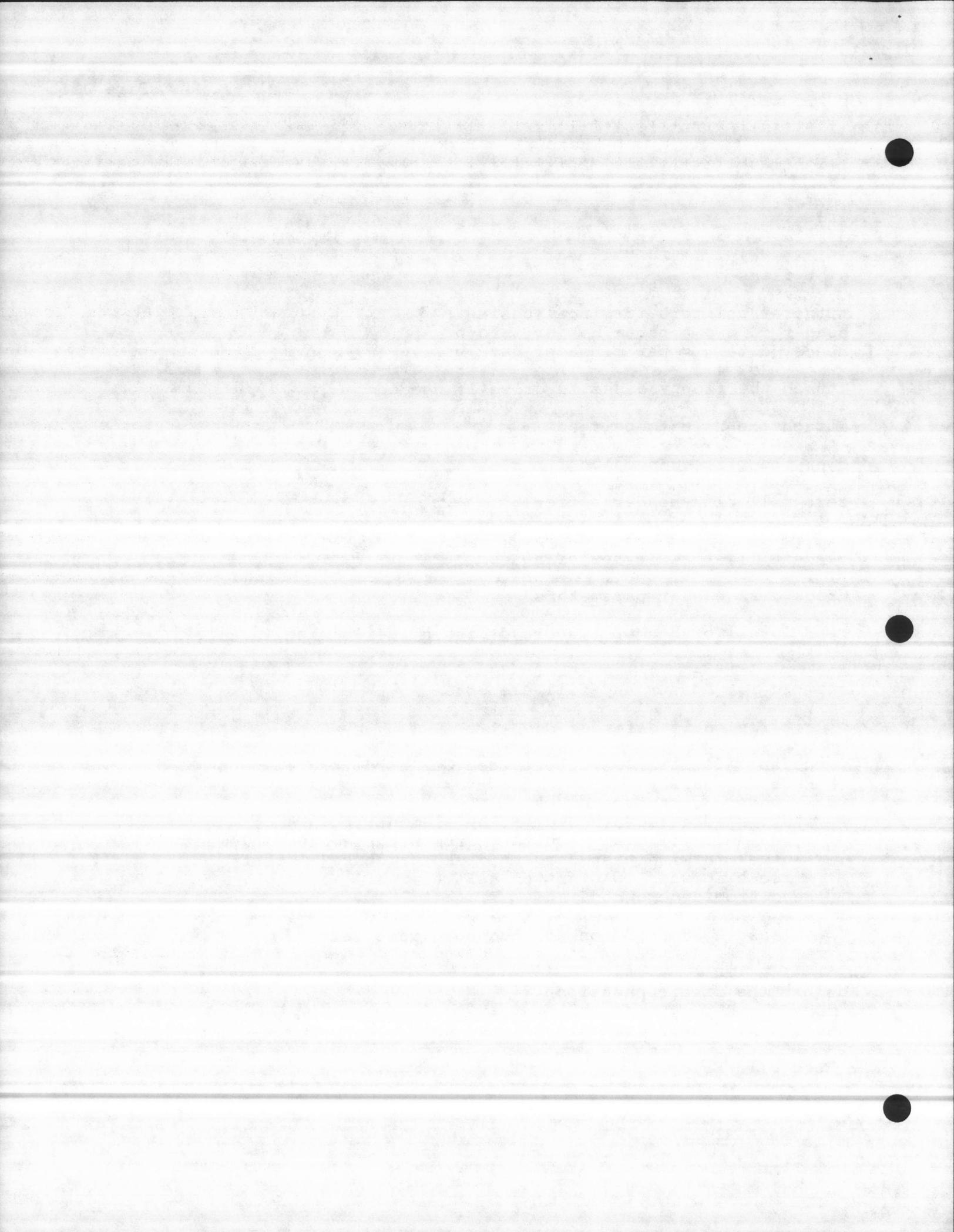
release region if control had been received

Exit:

delete the file connection

detach the floppy diskette drive

END Read\_File\_Proc;



---

+++++  
+ Write Block Procedure +  
+++++

ALGORITHM:

=====

Write\_Block\_Proc:

-----  
exit\_flags = 0;

attach the floppy diskette drive if needed

attach the file to be written

open the file connection for reading and writing

read in first six bytes of header

check if element exists on disk

read in all of header (use read disk header's size)

calculate seek to wanted header element

check for wanted data structure element on disk

check element offset size

calculate initial seek to first entry wanted data

seek to wanted position

keep track of bytes from start of element

loop through number of separate writes  
DO counter = 1 TO param.num\_writes;    write in block  
    seek to next write

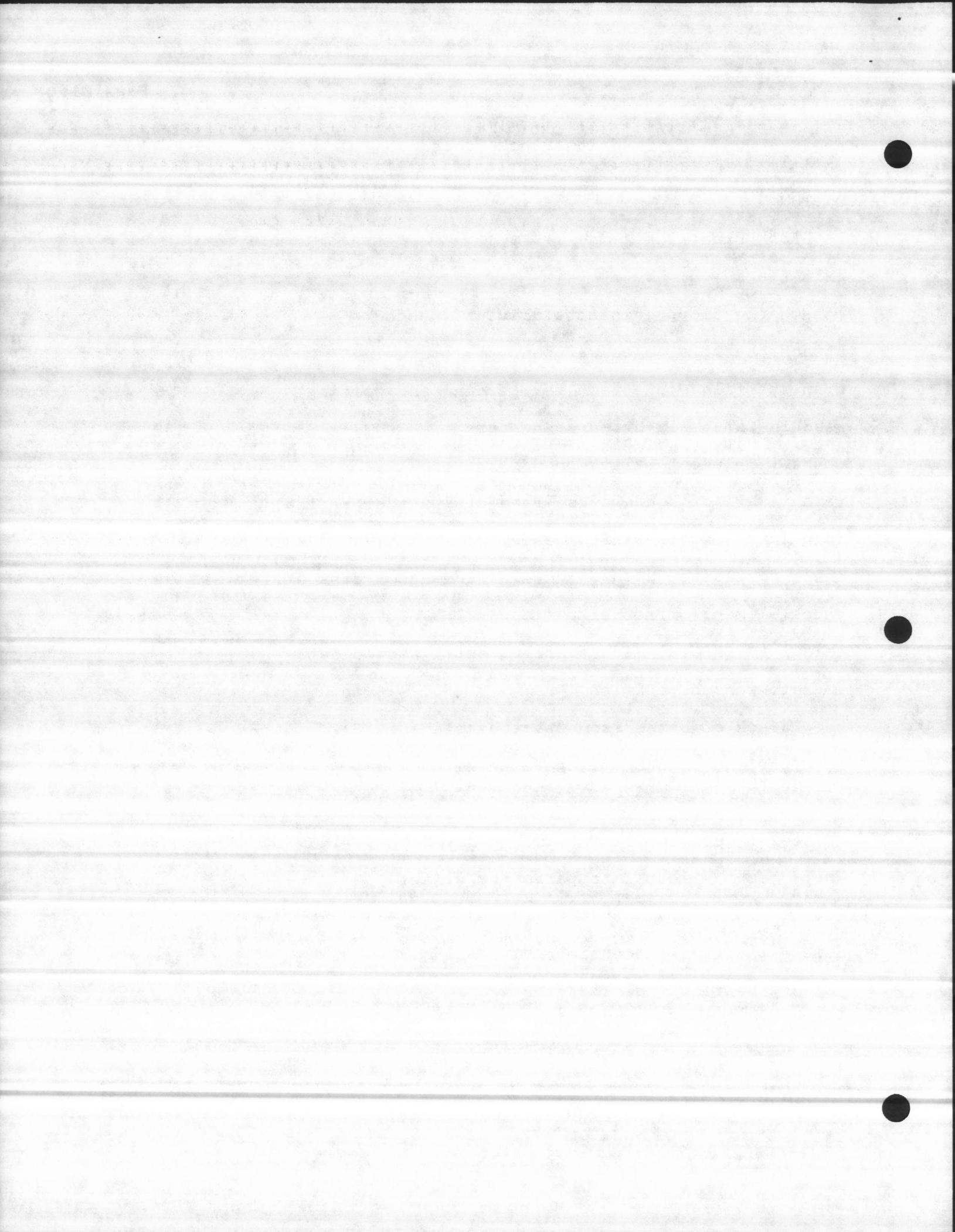
END; counter loop 1 to param.num\_writes

Exit:

delete the file connection

detach the floppy diskette drive

END Write\_Block\_Proc;



-----  
+++++ Read Block Procedure +  
+++++

ALGORITHM:

=====

Read\_Block\_Proc:  
-----

exit\_flags = Ø;

attach the floppy diskette drive if needed

attach the file to be read

open the file connection for reading

read in first six bytes of header

check if element exists on disk

read in all of header (use read disk header's size)

calculate seek to wanted header element

check for wanted data structure element on disk

check element offset size

calculate seek to wanted data

seek to wanted position

default fill if requested block is longer than disk element

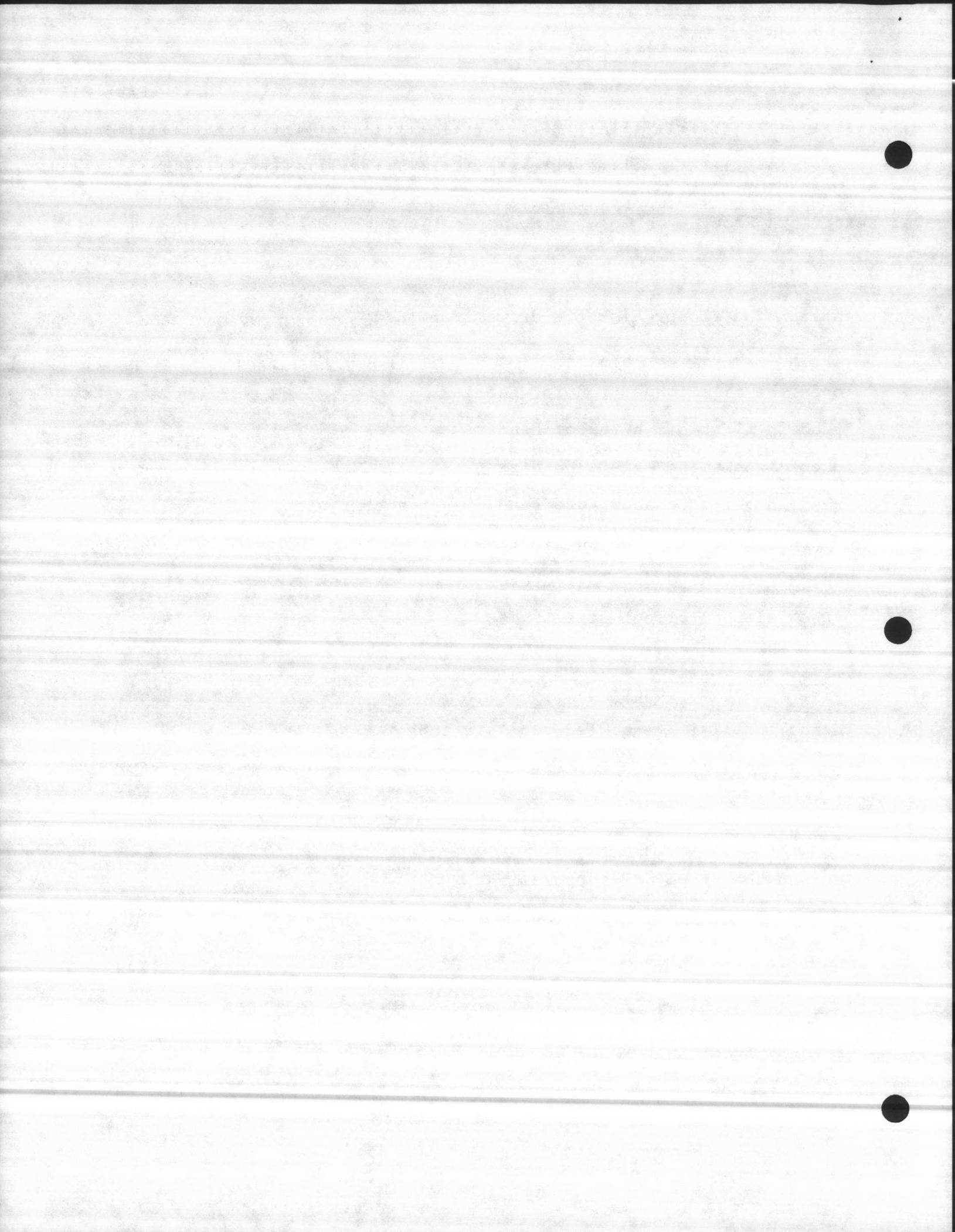
read in block

Exit:

delete the file connection

detach the floppy diskette drive

END Read\_Block\_Proc;



---

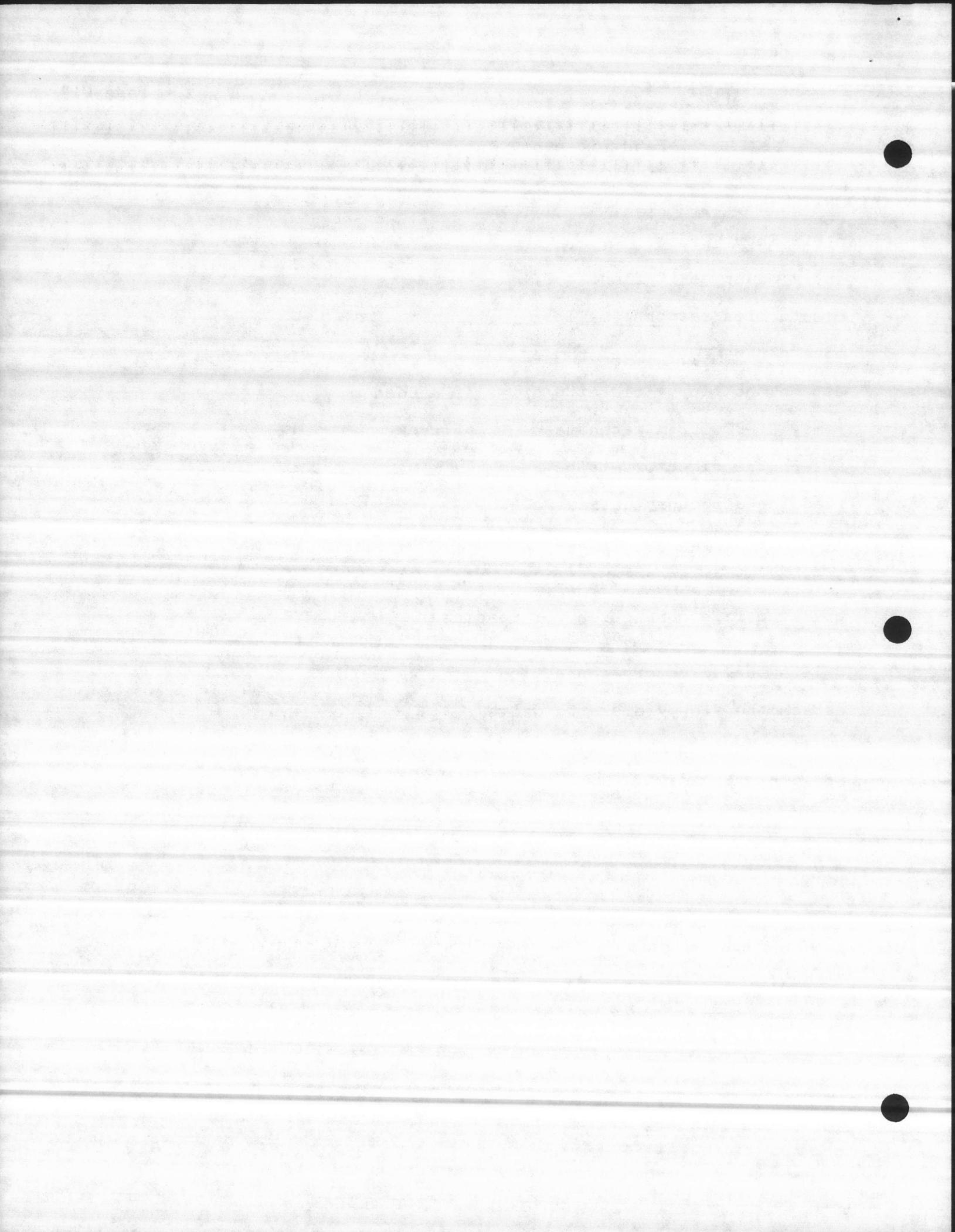
```
+++++ Read Directory Entry Procedure +  
+++++
```

ALGORITHM:

=====

Read\_Dir\_Entry\_Proc:

```
-----  
entry_count, exit_flags = 0;  
  
create bios response mailbox  
  
move pathname to local ram  
  
attach the floppy diskette drive if needed  
  
remove :FD0: from pathname  
  
attach the directory to be read  
  
read the directory entries  
  
attach the file to be read  
  
attach the floppy diskette drive if needed  
  
open the file connection for reading  
  
t:  
  
delete the directory connection  
  
detach the floppy diskette drive  
  
delete response mailbox  
  
END Read_Dir_Entry_Proc;
```



-----  
+++++  
Copy File Procedure  
+++++

ALGORITHM:

=====

Copy\_File\_Proc:

-----

exit\_flags = Ø;

check to assure different path names

attach the floppy diskette drive if needed

open the file connection for reading

get file size

open the file connection for writing

determine size of disk buffer

copy block loop

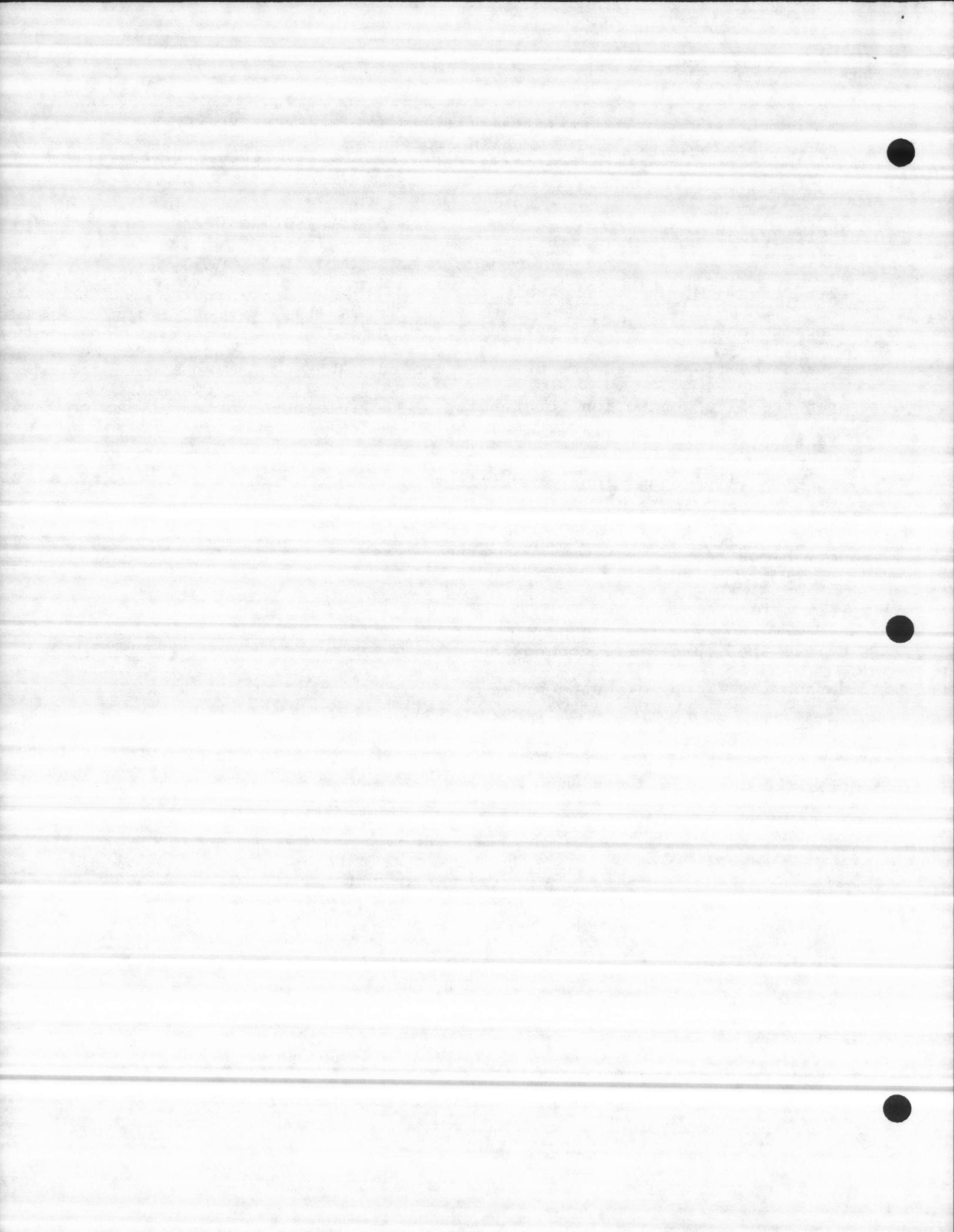
    Check to read in the final bytes remaining

Exit:

    delete the source file connection

    detach the floppy diskette drive

END Copy\_File\_Proc;



+  
+++++ Floppy Format Procedure +  
+++++

ALGORITHM:  
=====

Floppy\_Format\_Proc:

-----  
exit\_flags = Ø;  
attach the floppy diskette drive  
check to assure that a diskette is mounted  
get file connection for human interface input  
open input connection for reading  
get file connection for human interface output  
open input connection for writing  
get human interface command connection  
formulate command string to format diskette as named volume  
send format command

Exit:

delete the human interface command connection  
delete the human interface output connection  
delete the human interface input connection  
detach the floppy diskette drive

END Floppy\_Format\_Proc;

+  
+++++ Rename File Procedure +  
+++++

ALGORITHM:  
=====

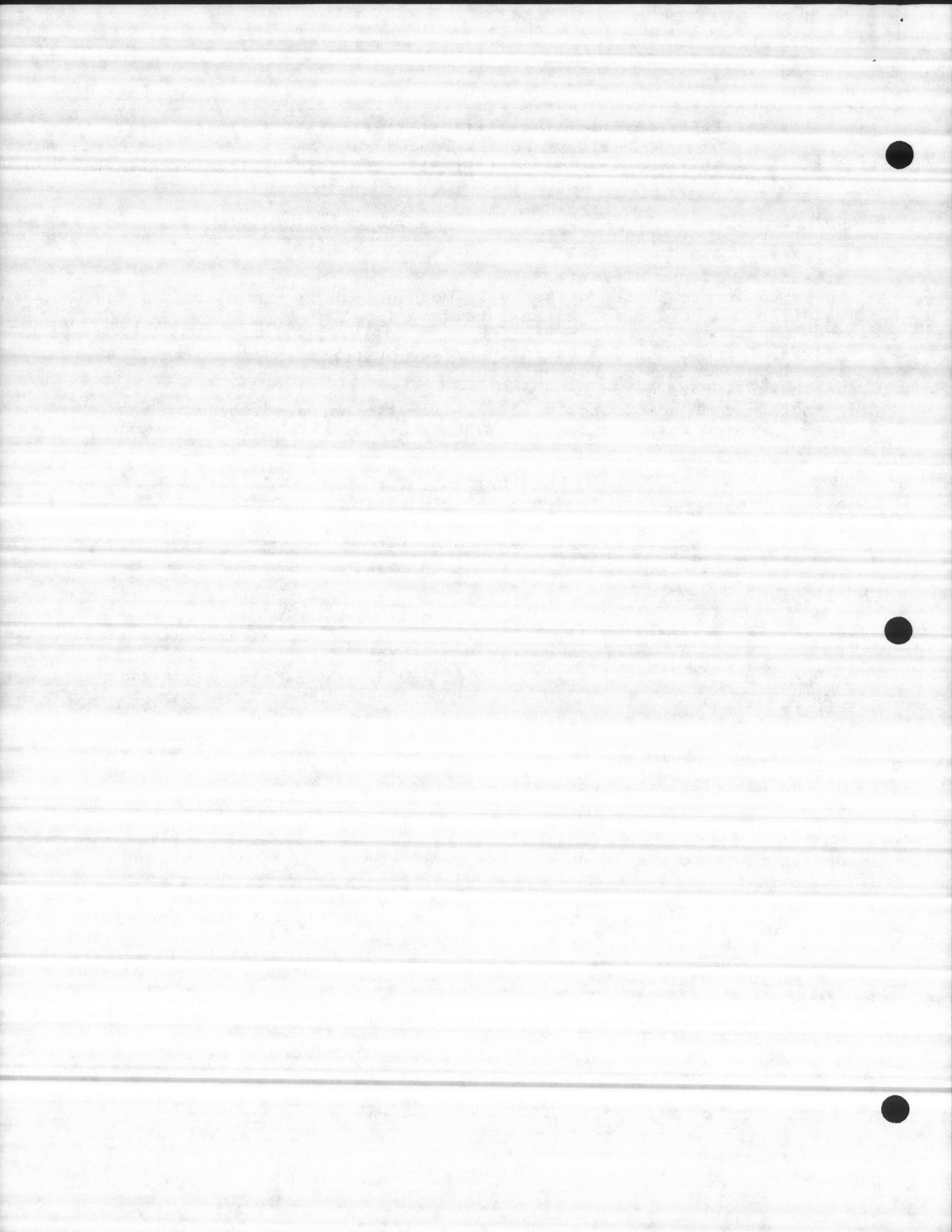
Rename\_File\_Proc:

-----  
exit\_flags = Ø;  
attach the floppy diskette drive if needed  
rename the file

Exit:

    detach floppy diskette drive

END Rename\_File\_Proc;



+-----  
+----- Add Serial Procedure +-----  
+-----

ALGORITHM:

=====

Add\_Serial\_Proc:

-----  
exit\_flags = Ø;

transfer pointers for local basing

attach the file to be written

create file if necessary

signifies that file has been connected

open the file connection for reading writing

write header to disk

header MUST be initialized correctly (i.e. num\_elements = Ø)

initialize local header

ELSE

file exists

signifies that file has been connected

open the file connection for reading writing

read in disk header

check if a write is needed

increment number of elements in disk header

seek back 2 bytes

write in new number of elements

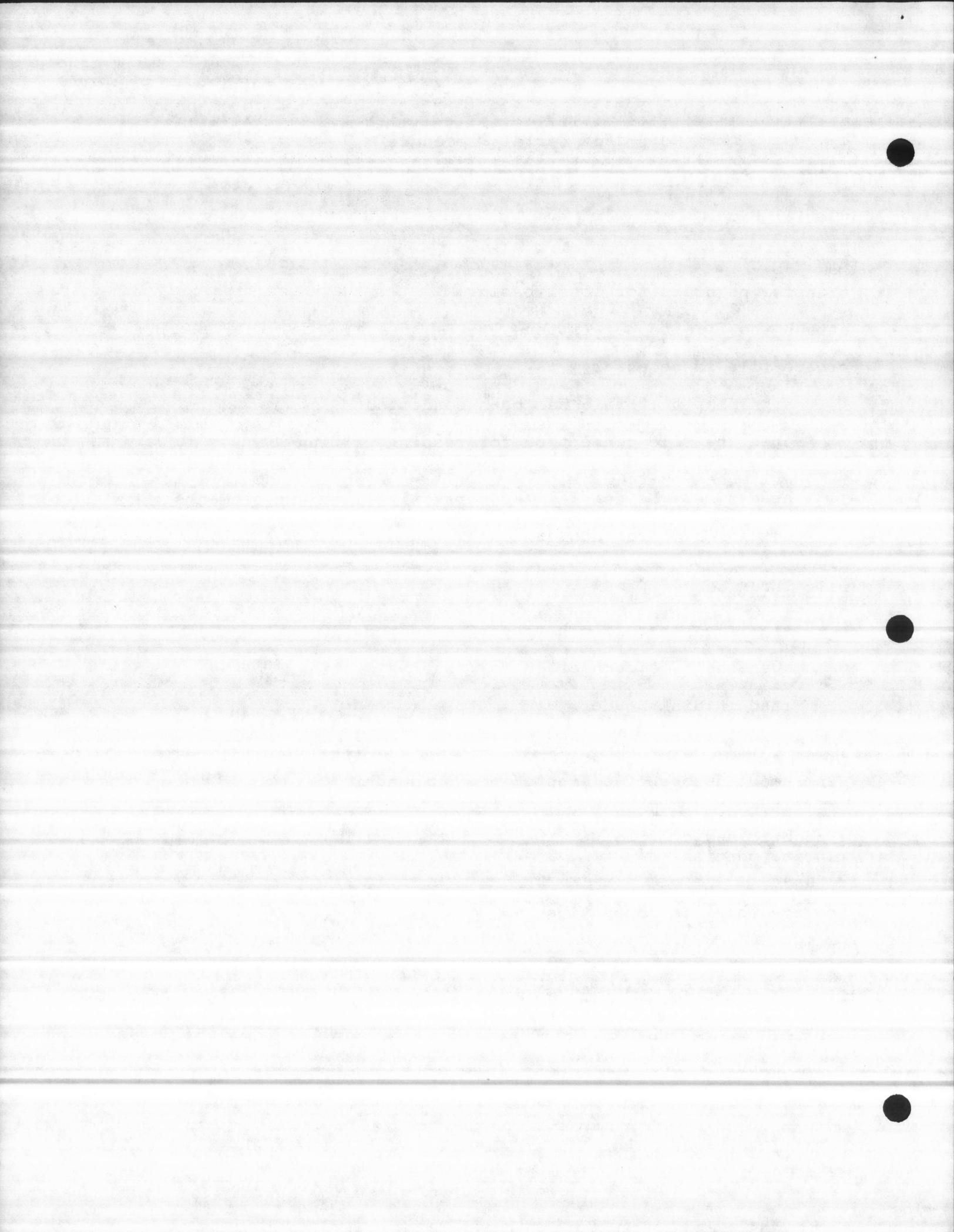
seek to end of file

write to disk

Exit:

delete the file connection

END Add\_Serial\_Proc;



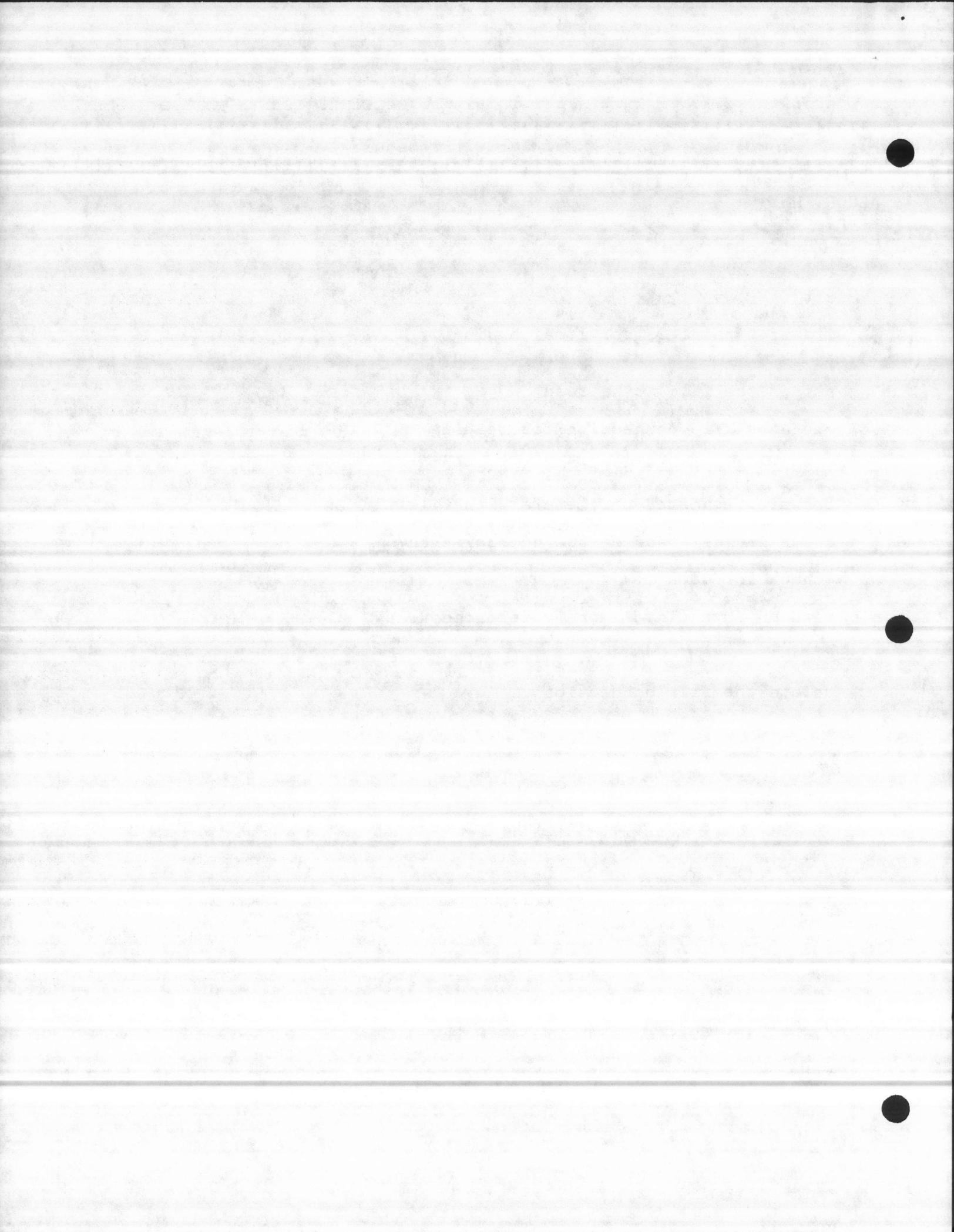
---

+  
+++++ Read Serial Procedure +  
+++++ALGORITHM:  
=====Read\_Serial\_Proc:  
-----

```
exit_flags = 0;  
  
transfer pointers for local basing  
  
attach the file to be read  
  
open the file connection for reading  
  
read in first six bytes of header  
  
compare requested parameters with file  
  
check if full read can be accomplished  
  
check for seek  
  
read in data (into a one referenced array)
```

Exit:

```
delete the file connection  
  
END Read_Serial_Proc;
```



-----  
+-----  
D I S K   T A S K  
+-----  
-----

ALGORITHM:

=====

DISK TASK:

-----  
DO FOREVER;        main loop

wait for mailbox

disable exception handler

function\_code is first word in param\_seg

DO CASE disk\_function;

case 0 = CALL Create\_Directory\_Proc;

case 1 = CALL Delete\_File\_Proc;

case 2 = CALL Write\_File\_Proc;

case 3 = CALL Read\_File\_Proc;

case 4 = CALL Write\_Block\_Proc;

case 5 = CALL Read\_Block\_Proc;

case 6 = CALL Read\_Dir\_Entry\_Proc;

case 7 = CALL Copy\_File\_Proc;

case 8 = CALL Floppy\_Format\_Proc;

case 9 = CALL Rename\_File\_Proc;

case 10 = CALL Add\_Serial\_Proc;

case 11 = CALL Read\_Serial\_Proc;

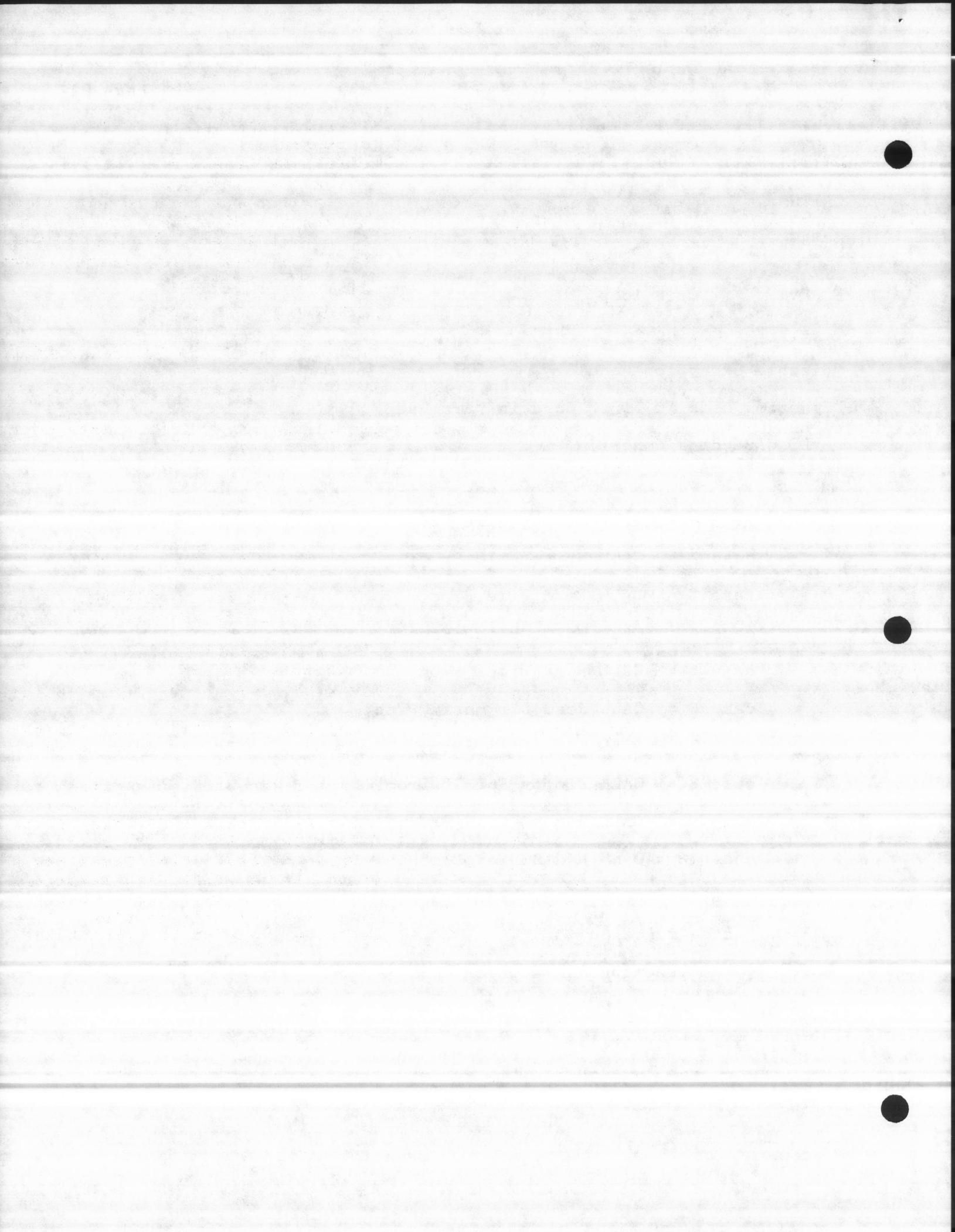
END;     case function

END;     param.function limit check

enable exception handler

END;     forever

END Disk\_Task;



(11.2) MODULE NAME : CxxxxFILE.Pyy

=====

MODULE DESCRIPTION :

=====

Controls the menu of disk commands available to the user, and  
uses the disk task procedures for the following functions:

Copy files, Display directory, Delete files, Format named floppy,  
Load historical files, Rename files, Save historical and  
powerup files.

EXTERNAL PROCEDURES :

=====

Enter\_Password\_And\_Verify: PROCEDURE (crt\_num) WORD EXTERNAL;  
DECLARE (crt\_num) BYTE;  
END Enter\_Password\_And\_Verify;

Prompt\_Entry: PROCEDURE (max\_count, crt\_num) EXTERNAL;  
DECLARE (max\_count, crt\_num) BYTE;  
END Prompt\_Entry;

Format\_Disk\_Error: PROCEDURE (buffer\_t, message\_str\_p, disk\_error,  
disk\_operation) EXTERNAL;  
DECLARE (disk\_error, disk\_operation) WORD,  
buffer\_t TOKEN,  
message\_str\_p POINTER;  
END Format\_Disk\_Error;

MODULE DECLARATION :

=====

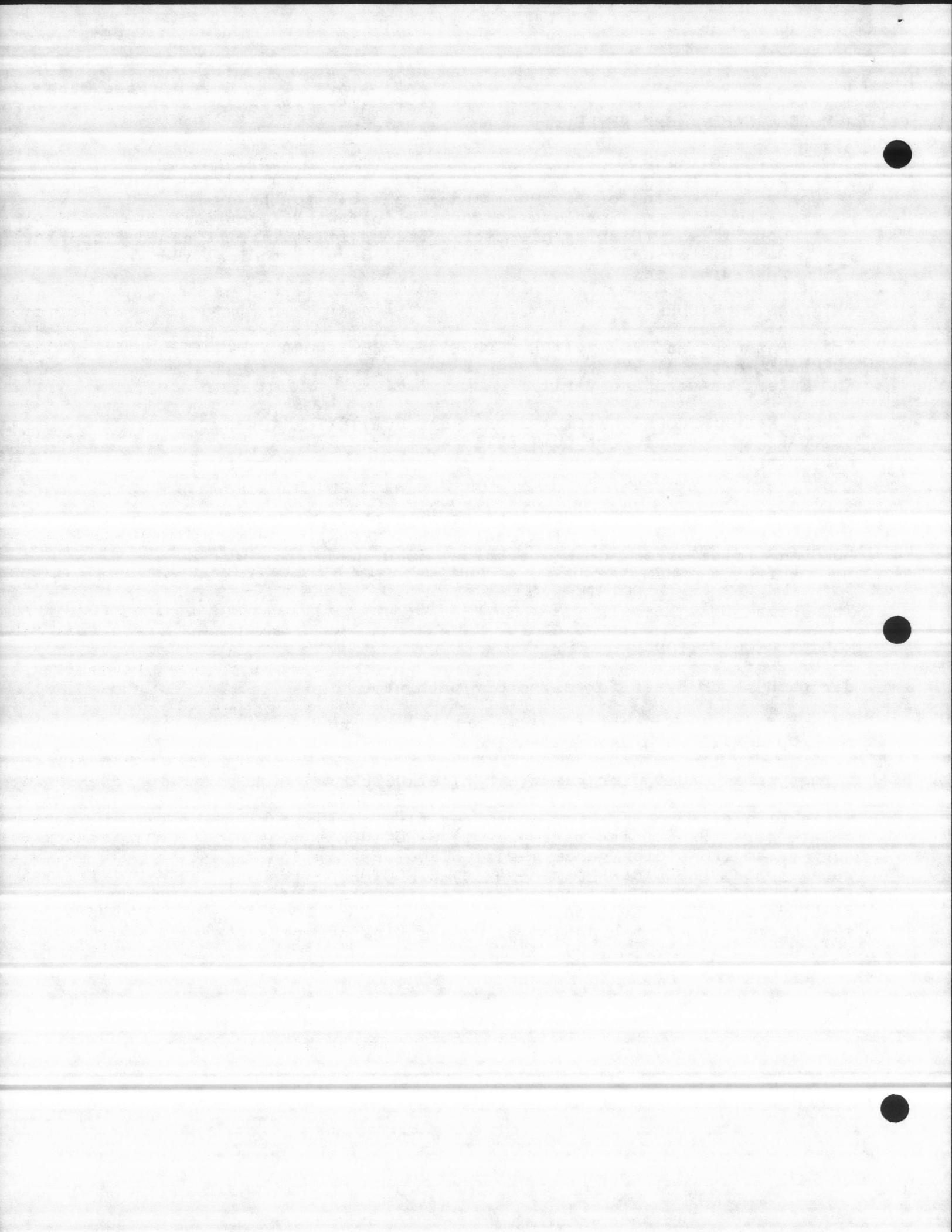
dir\_path => 60 bytes for directory path name

exception => A word holds the exception code number.  
temp\_word => A word used as an index.

format\_param BASED disk\_param\_seg\_t) STRUCTURE  
read\_file\_param BASED disk\_param\_seg\_t) STRUCTURE  
write\_file\_param BASED disk\_param\_seg\_t) STRUCTURE  
delete\_param BASED disk\_param\_seg\_t) STRUCTURE  
copy\_param BASED disk\_param\_seg\_t) STRUCTURE  
rename\_param BASED disk\_param\_seg\_t) STRUCTURE

All the above structures have been described in the beginning of the  
first part of this section.

system\_err\_msg => 18 bytes hold 'System Disk Error!'



---

**MODULE PUBLIC PROCEDURES :**

```
=====
Prompt_Disk_Error:
+++++
```

crt\_num => A byte holds the crt number to display on.

message\_p => A pointer points to the message to be displayed on the crt.

**Proc description :**

This Procedure called at the end of each command when disk error occurs  
and displays the following message  
'Disk Error! Type C to continue :'

END Prompt\_Disk\_Error;

```
Historical_Load_Command:
+++++
```

crt\_num => A byte, the number of the crt to display on.

flag => A byte holds a flag value.

temp\_word => A word value holds a temp value.

time\_dword=> A dword holds the date & time value.

**Proc Description :**

This procedures used to LOAD a historical file from the hard disk or  
floppy disk to the historical RAM.

END Historical\_Load\_Command;

```
Display_Directory:
+++++
```

dir\_param BASED disk\_param\_seg\_t STRUCTURE

function => A word value holds the function.

disk\_error => A word value holds the disk error value

disk\_operation => A word value holds the disk\_opration.

pathname\_p => A pointer points to the directory path name.

entry\_index => A word value as an entry index.

entry\_asc\_p => A pointer points to an entry.

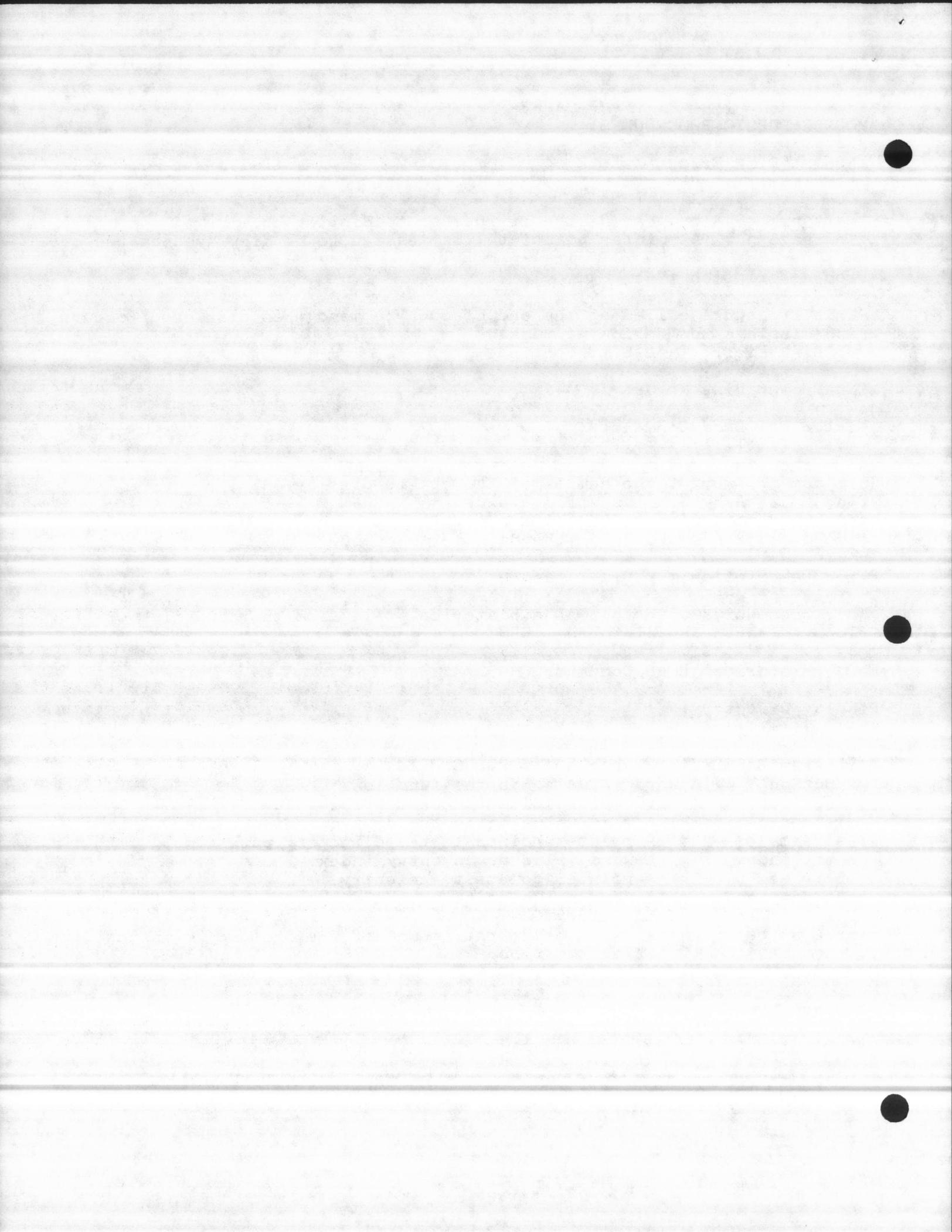
num\_entries => A word value contains the number of entries in the  
directory.

free\_space => A dword holds the amount of the free space on the disk  
or on the floppy

**Proc description :**

Displays Directories.

END Display\_Directory;



Adjust\_Disk\_Name:

++++++

crt\_num => A byte, the number of the crt to display on.

Proc description :

This procedure used to adjust the disk name

END Adjust\_Disk\_Name;

Keyin\_File\_Sub\_Menu\_Proc:

++++++

crt\_num => A byte, the number of the crt to display on.

flag => A byte holds a flag value.

pass\_level => A byte holds the pass\_level value.

result\_word => A word used as an index.

file\_command\_priorities => Seven bytes,

Copy	= 2
Delete	= 3
Directory	= 1
Format	= 2
Load	= 1
Rename	= 3
Save	= 1

source\_path => sixty bytes for source path name.

display\_menu\_count => A byte holds a value of seven.

display\_menu\_prompt\_asc (7) STRUCTURE :

menu\_asc => ten bytes for the command string.

Copy = Copies the contents of one file to another

Delete = Deletes a file from disk

Directory = Displays a disk directory

Format = Formats the floppy diskette

Load = Loads the specified archive file into memory

Rename = Changes the name of a disk file

Save = Saves archive and power-up files

dir\_menu\_count => A byte holds a value of two.

dir\_menu\_prompt\_asc (2) STRUCTURE :

menu\_asc => 10 bytes for the sub command string.

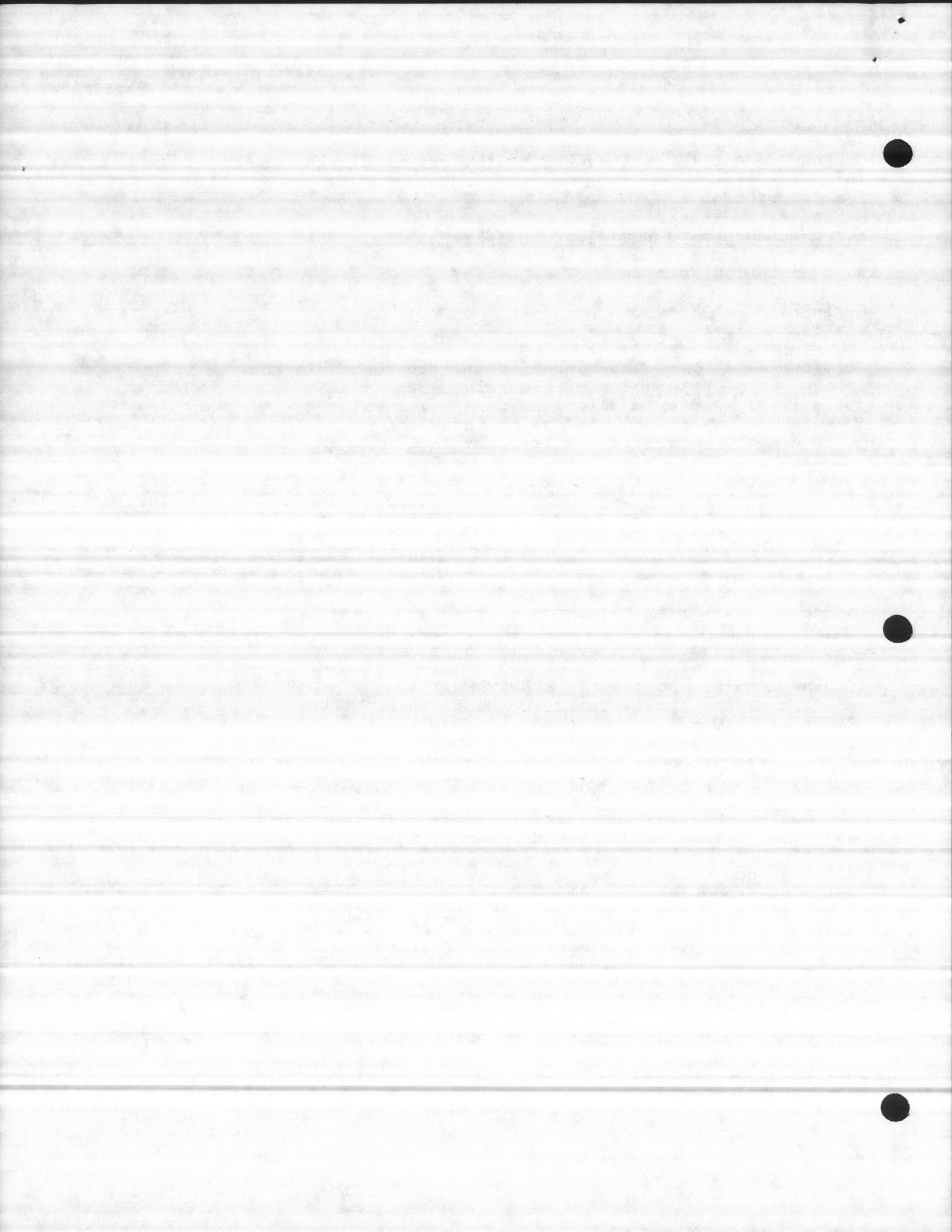
Archive = Displays the archive directory on hard disk

Floppy = Displays the floppy diskette directory

Proc description :

This procedure displays the file menu and when using any command or sub command the proper calls are taken place in the procedure too.

END Keyin\_File\_Sub\_Menu\_Proc;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 12

D A T A  
===== B A S E  
===== S Y S T E M  
=====

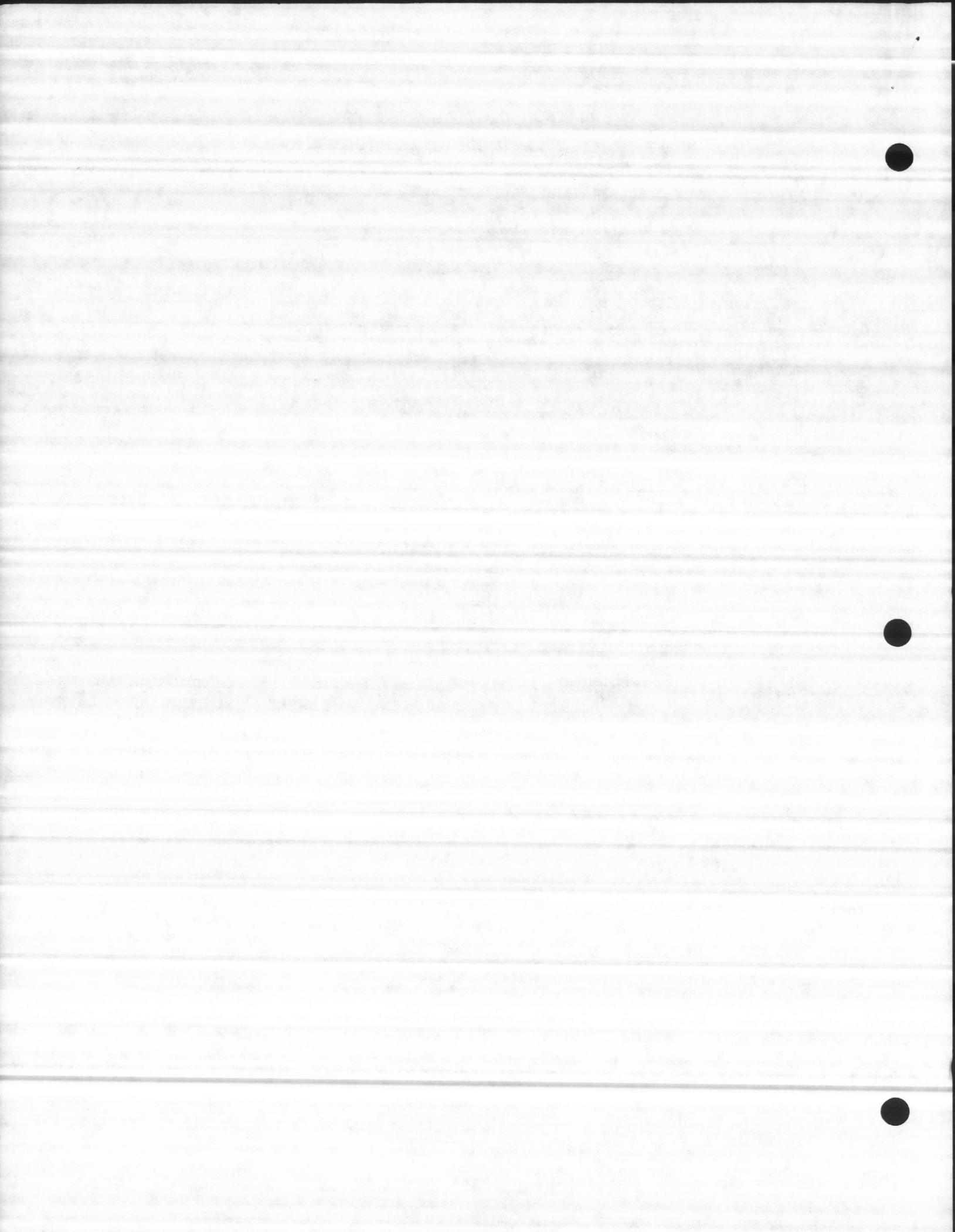
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



INTRODUCTION :

=====

The data base system is a user definable system and it contains the following groups :

- 1) ANALOG INPUT.
- 2) DISCRETE INPUT.
- 3) ANALOG OUTPUT.
- 4) DISCRETE OUTPUT.
- 5) YEAR TREND
- 6) CONTROL.
- 7) LEVEL CONTROL.

DATE : Oct 10, 1985

----  
The Data Base System - Written by Bob Ryan, Peter Wollenzien, Mohamed Fayad.

DATA BASE SYSTEM FLOW :

=====

12.Ø DATA BASE  
system

<>

.4 DISCRETE  
OUTPUT

.3 ANALOG  
OUTPUT

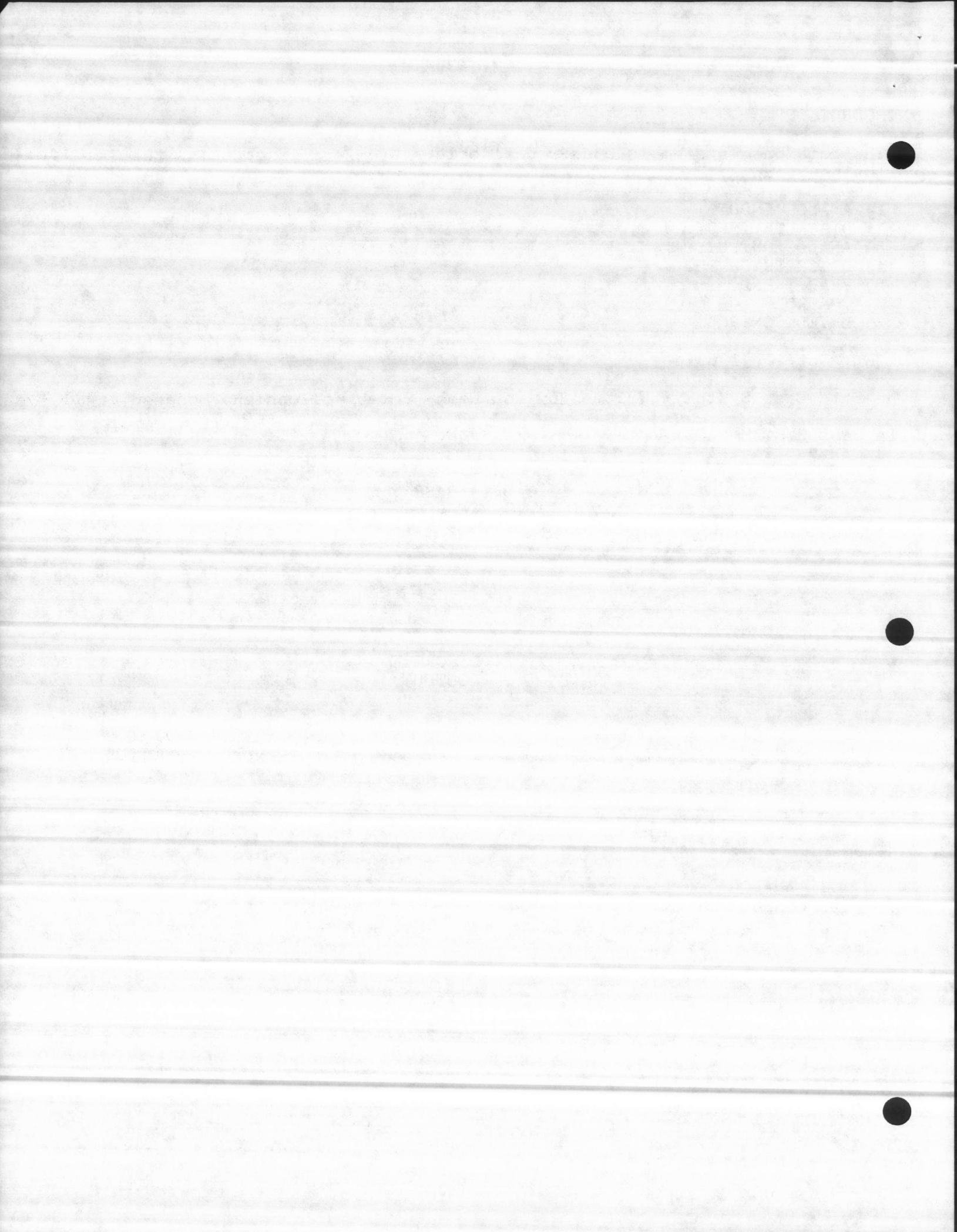
.5 YEAR  
TREND

.2 DISCRETE  
INPUT

.6 CONTROL

.1 ANALOG  
INPUT

.7 LEVEL  
CONTROL



12.0 : MODULE NAME : CxxxxDBAS.Pyy

=====

DISCREPTION : Displays the data base submenu, points and the related  
information to each point.

EXTERNAL PROCEDURES :

=====

```
Display_Table: PROCEDURE (table_p, param_p,
                           table_reg_t, buf_t, mbx_t) EXTERNAL;
    DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
            (table_p, param_p)           POINTER;
End Display_Table;

Table_Fill: PROCEDURE (field_number, element,
                       struc_type, table_p, param_p) WORD EXTERNAL;
    DECLARE (element,field_number)   WORD,
            (struc_type)           BYTE,
            (table_p, param_p)     POINTER;
END Table_Fill;

Hist_Table_Fill: PROCEDURE (field_number, element,
                            struc_type, table_p, param_p) WORD EXTERNAL;
    DECLARE (element,field_number)   WORD,
            (struc_type)           BYTE,
            (table_p, param_p)     POINTER;
END Hist_Table_Fill;
```

NOTE : crt\_base\_display\_type (3) BYTE is global variable which  
holds index to which display type selected in main menu

MODULE DECLARATION :

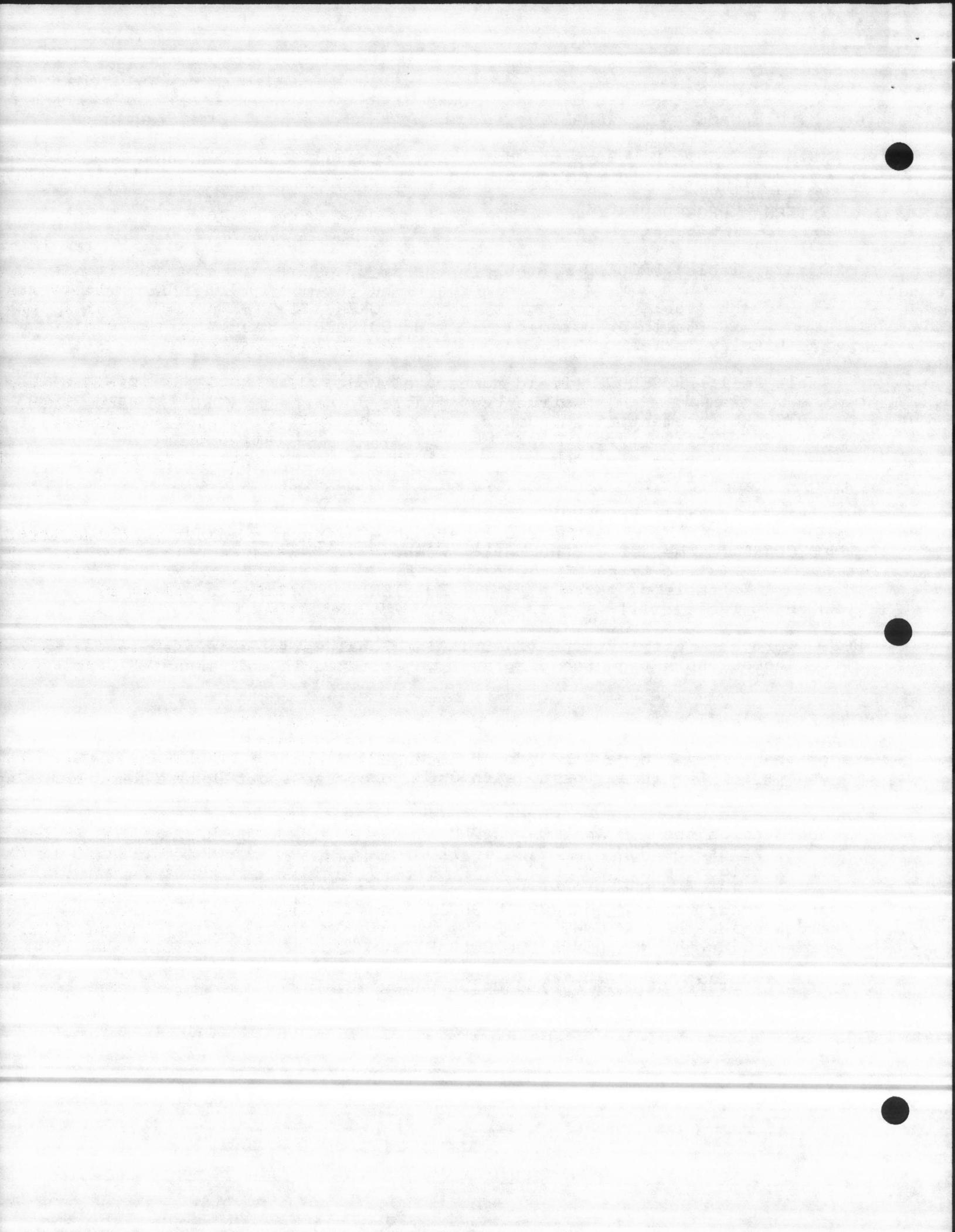
=====

exception => Three words, each holds the exception condition number.  
crt\_update\_sem\_t=> Three token, each used to create a crt update semaphore.

menu\_command\_count => Seven words

word 0 holds the NUM\_ANALOGS\_IN\_PLUS1,  
word 1 holds the NUM\_DISCRETES\_IN\_PLUS1.  
word 2 holds the NUM\_ANALOGS\_OUT\_PLUS1.  
word 3 holds the NUM\_DISCRETES\_OUT\_PLUS1.  
word 4 holds the NUM\_ANALOGS\_IN\_PLUS1.  
word 5 holds the NUM\_CONTROLS\_PLUS1.  
word 6 holds the NUM\_LEVEL\_CONTROLS\_PLUS1.

main\_menu\_command\_count => A byte holds a seven value.



main\_menu\_prompt\_asc (7) STRUCTURE :

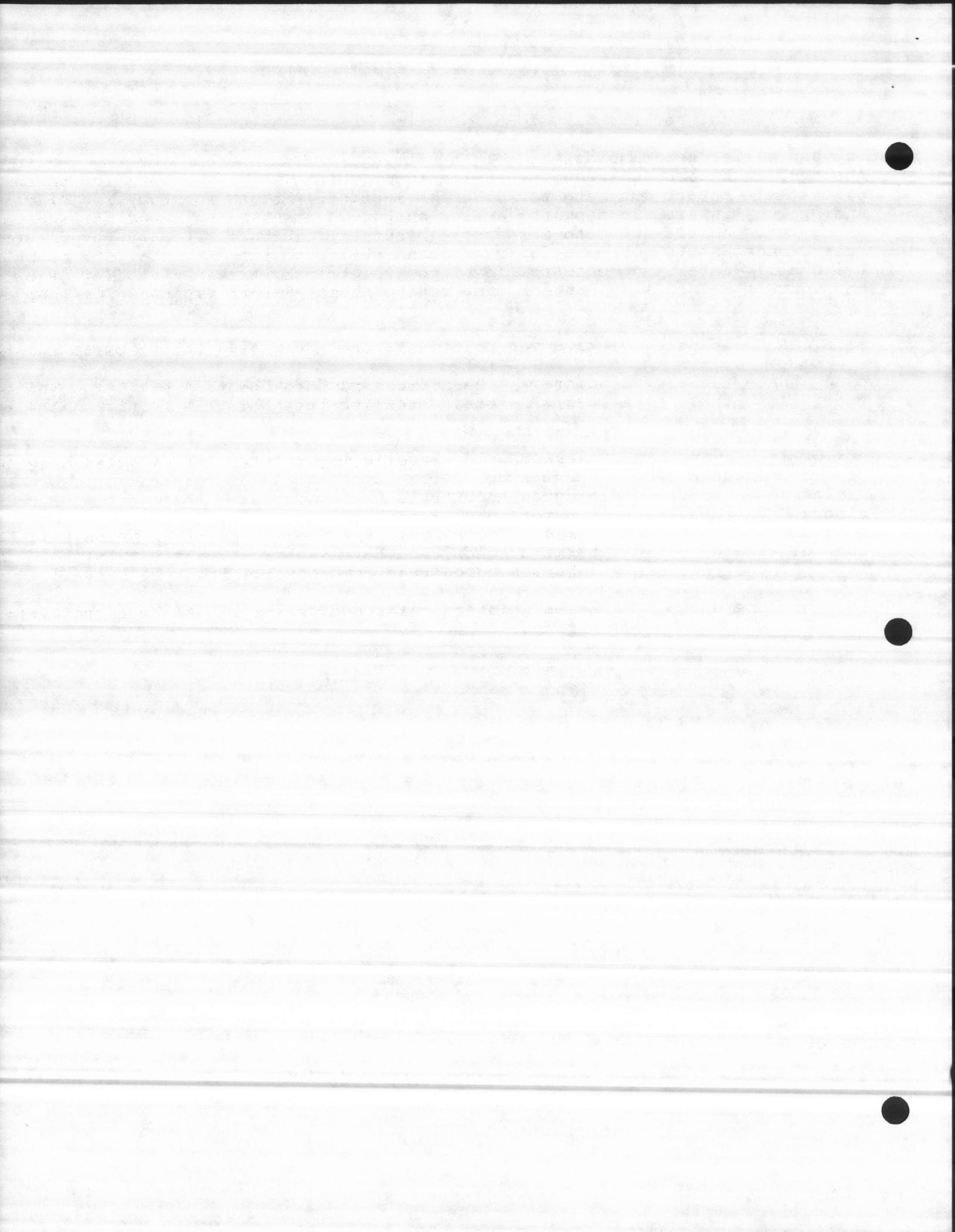
menu\_asc => Ten bytes holds the command string.  
Analog-I = Enters sub-menu for analog input points  
Discrete-I= Enters sub-menu for discrete input points  
Analog-O = Enters sub-menu for analog output points  
Discrete-O= Enters sub-menu for discrete output points  
Year Trend= Enters sub-menu for year trend analog points  
Lvl Contrl= Enters sub-menu for level control points  
Pmp Contrl= Enters sub-menu for pump control points

main\_menu\_help\_desc\_tbl => Seven pointers - the addresses to help page.

base\_update\_table => A pointer - the address to Base\_Update.  
base\_delete\_table => A pointer - the address to Base\_Delete.

analog\_in\_heading => 24 bytes hold 'Base Analog Input Point'.  
discrete\_in\_heading => 26 bytes hold 'Base Discrete Input Point'.  
analog\_out\_heading => 25 bytes hold 'Base Analog Output Point'.  
discrete\_out\_heading=> 27 bytes hold 'Base Discrete Output Point'.  
year\_trend\_heading => 23 bytes hold 'Base Year Trend Point'.  
control\_heading => 25 bytes hold 'Base Level Control Point'.  
level\_control\_heading=> 24 bytes hold 'Base Pump Control Point'.

base\_heading\_str\_p => Seven pointers - the addresses to base heading strings.



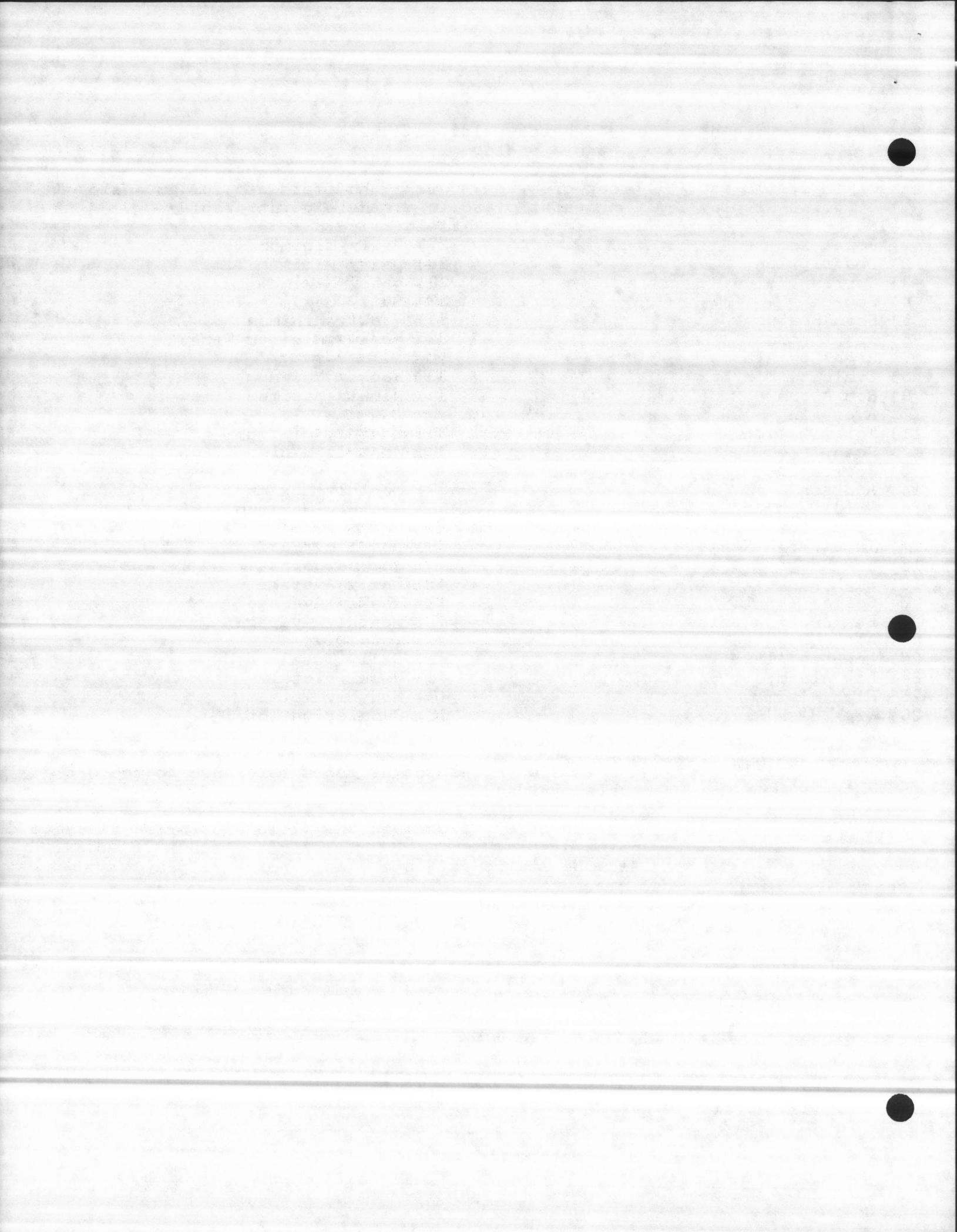
`analog_in_field_xref => 49 bytes holds an X reference value to the crt table field.`

`analog_in_static_asc - a group of bytes contain the field asc for an analog point display. All the static asc in a single point display`

Field-----	Field-----
1) Point Id	27) Daily Minimum
2) Description	28) Daily Minimum Time
3) Current Value	29) Daily Maximum
4) Source	30) Daily Maximum Time
5) Scan Status	31) Weekly Minimum
6) Input Format	32) Weekly Minimum Time
7) High/Low Alarms	33) Weekly Maximum
8) Evaluate Min/Max	34) Weekly Maximum Time
9) Roc Alarm	35) Monthly Minimum
10) Averaging	36) Monthly Minimum Time
11) Totalization	37) Monthly Maximum
12) Scan Time	38) Monthly Maximum Time
13) Source Id	39) Roc Time Period
14) Address	40) Roc Setpoint
15) Word/Port	41) Daily Average
16) Input Low Range	42) Weekly Average
17) Input High Range	43) Monthly Average
18) Low Range	44) Total Type
19) High Range	45) Daily Total
20) Scale X * 10^	46) Weekly Total
21) Engineering units	47) Monthly Total
22) Alarm Deadband	48) Cumulative Total
23) Low Low Alarm	
24) Low Alarm	
25) High Alarm	
26) High High Alarm	

`discrete_in_field_xref - A number of bytes act as index reference to the crt table fields`

`discrete_in_static_asc - a group of bytes contain the field asc for an discrete point display.  
All the static asc in a single point display`



---

Field-----

- 1) Point Id
- 2) Description
- 3) Current Value
- 4) Source
- 5) Scan Status
- 6) Normal Contact
- 7) Alarm
- 8) Printed
- 9) Accumulate Starts
- 10) Accumulate Run Time
- 11) Scan Time
- 12) Source Id
- 13) Address
- 14) Word/Port
- 15) Bit
- 16) Annunciation Delay
- 17) Alarm Priority
- 18) Alarm Dialer
- 19) Off Condition
- 20) On Condition
- 21) Current Cond

## Field-----

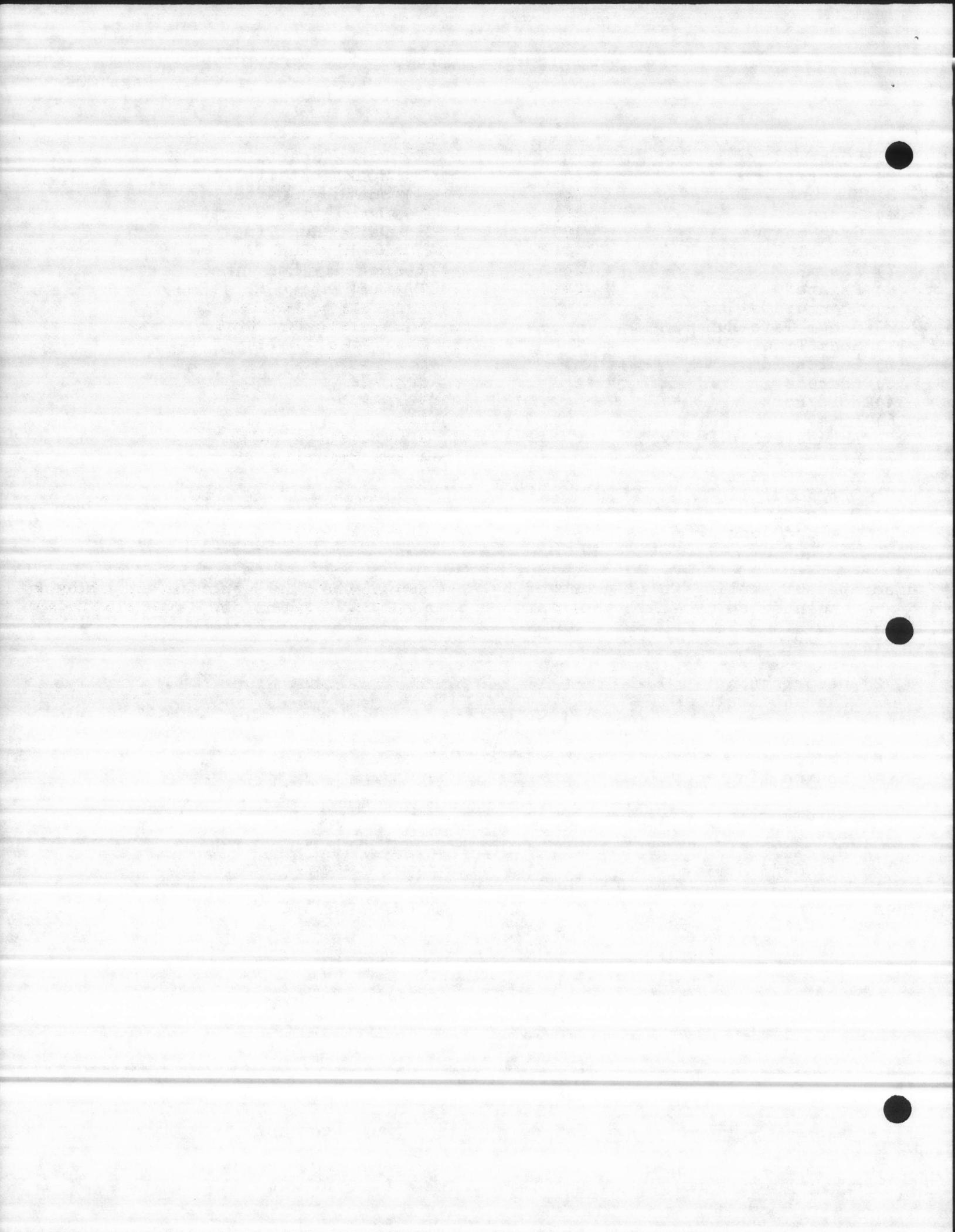
- 22) Daily Starts
- 23) Weekly Starts
- 24) Monthly Starts
- 25) Cumulative Starts
- 26) Daily Run Time
- 27) Weekly Run Time
- 28) Monthly Run Time
- 29) Cumulative Run Time

discrete\_out\_static\_asc \_ a group of bytes contain the field asc for an discrete point display. All the static asc in a single point display

---

## Field

- 1) Point Id
- 2) Description
- 3) Current Status
- 4) Scan Status
- 5) Scan Time
- 6) Source
- 7) Source Id
- 8) Address
- 9) Word Port
- 10) Bit
- 11) Normal Contact
- 12) Off Cond String
- 13) On Cond String
- 14) Current Cond



**analog\_out\_static\_asc** - a group of bytes contain the field asc for an analog point display. All the static asc in a single point display

**Field**

- 1) Id Number
- 2) Description
- 3) Scan Status
- 4) Scan Time
- 5) Source
- 6) Source Id
- 7) Hex/Bcd
- 8) Low Range Out
- 9) High Range Out
- 10) Address
- 11) Word Port
- 12) Output Value
- 13) Eng. Units
- 14) Scale

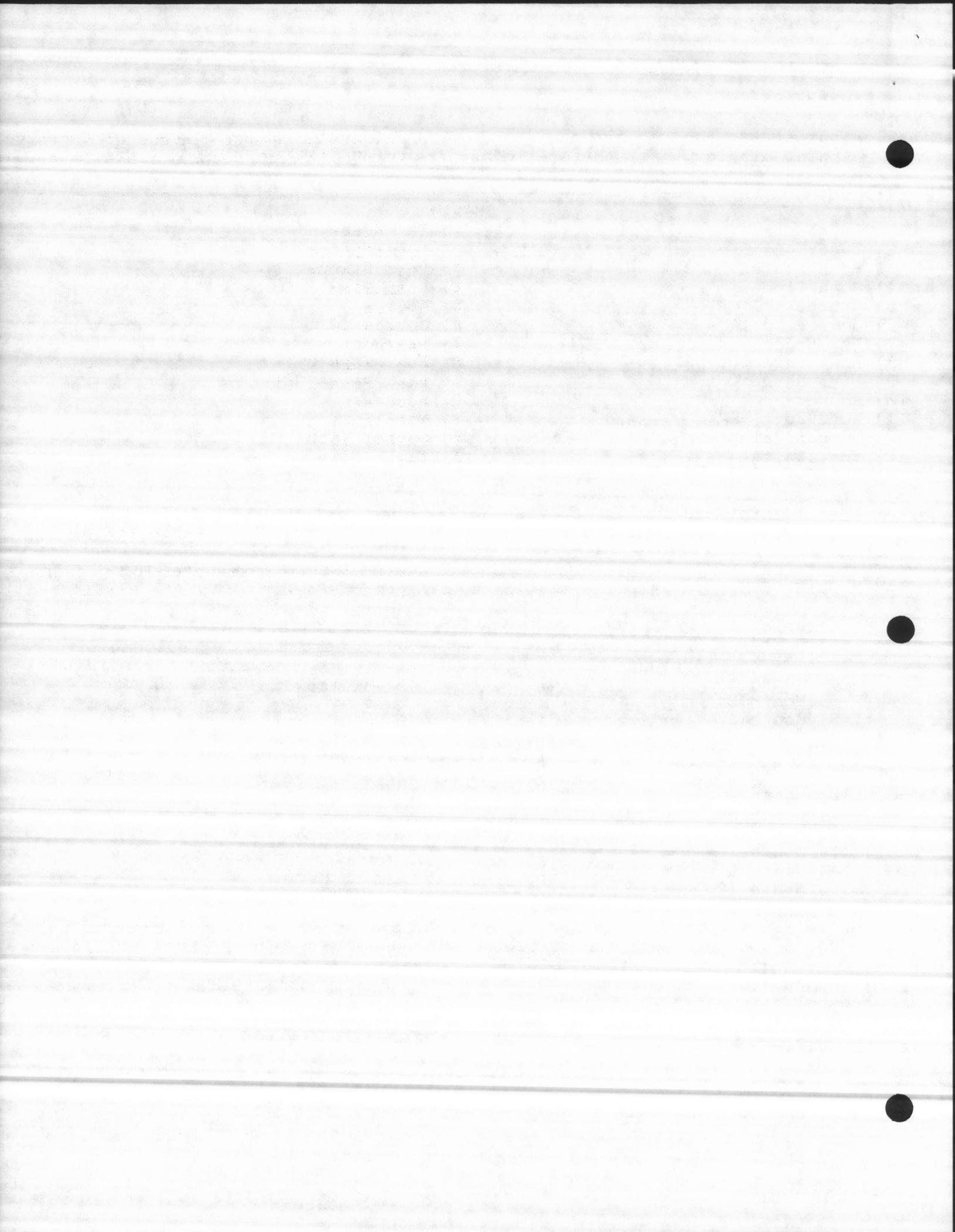
**yearly\_field\_xref** - A number of bytes act as index reference to the crt table fields

**year\_static\_asc** => a group of bytes contain the field asc for an analog yearly point display. All the static asc in a single point display

	Average	Total	2) Source Id Camp Lejeune, Holcomb Blvd Yearly Summary for = 19XX
4) Month 1=	16)		
5) Month 2=	17)		
6) Month 3=	18)		
7) Month 4=	19)		
8) Month 5=	20)		
9) Month 6=	21)		
10) Month 7=	22)		
11) Month 8=	23)		
12) Month 9=	24)		
13) Month 10=	25)		
14) Month 11=	26)		
15) Month 12=	27)		

**control\_static\_asc** => a group of bytes contain the field asc for control point display. All the static asc in a single point display

**level\_ctrl\_static\_asc** => a group of bytes contain the field asc for level control point display. All the static asc in a single point display



1) Point Id  
2) Description  
3) Scan Status  
4) Emergency Power Run  
5) Pump Control Status  
6) Manual Req Status  
7) Auto Req Status  
8) Pump HOA Id  
9) Pump HOA Status  
10) Pump Run Id  
11) Pump Run Status  
12) Pump Fail Timeout      Secs.  
13) Pump Fail Id  
14) Pump Fail Status  
15) Pump Start Id  
16) Pump Start Status  
17) Pump Enable Id  
18) Pump Enable Status  
19) Low Cutout Level Id  
20) Cutout Level  
21) Restore Level  
22) Low Cutout Id  
23) Low Cutout Status

## PUBLIC PROCEDURES :

=====

Clear\_Screen:

++++++

crt\_num => A byte, the number of the crt to display on.

Proc description :

-----

This procedure clears the entire crt screen.

Clear\_Screen only called by mvi's (48 lines).

END Clear\_Screen;

Menu\_Display:

++++++

crt\_num => A byte, the number of the crt to display on.

result => A word used as an index.

off\_set => A word holds the off\_set value.

bytes\_per\_element => A word holds the number of bytes / element.

num\_elements      +> A word holds the number of elements.

start\_id\_p      => A pointer points to the zeroth element of structure.

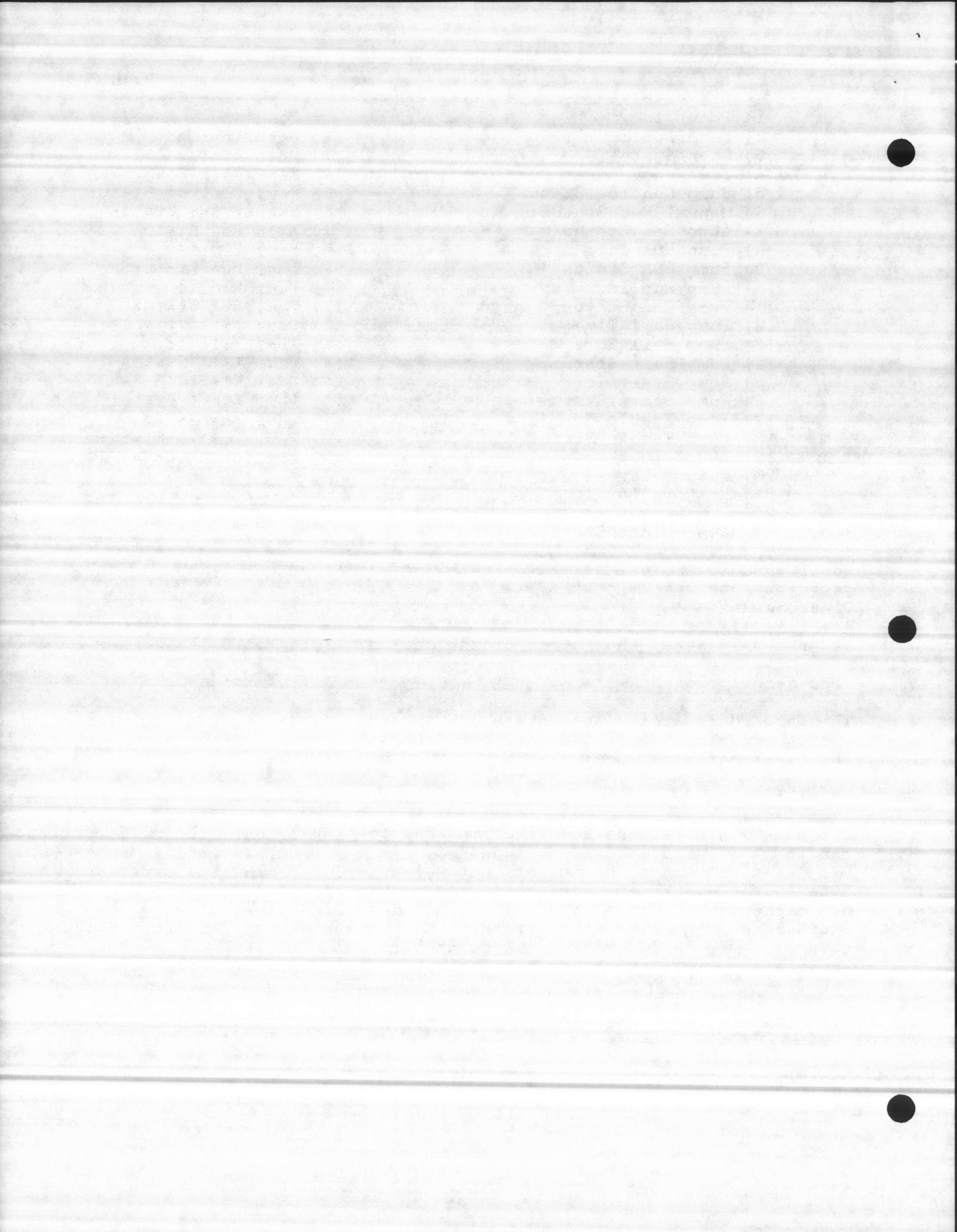
counter      => A word holds a counter.

exception      => A word holds an exception code number.

prompt\_t      => A prompt token.

help\_t      => A help page token.

desc\_t      => A description table token.



The call to this procedure would be:

```
selection = Menu_Display (crt_num,
                          selection, (w/ off_set added OR 0FFFFH)
                          off_set,
                          @structure,
                          SIZE(structure(0)),
                          (SIZE(structure) / SIZE(structure(0))));
```

selection returned would be 0FFFFH (esc - no selection) OR  
a range of off\_set to (num\_elements - off-set);

Proc description :

-----  
This procedure generates the menu display.

END Menu\_Display;

Keyin\_Data\_Base\_Sub\_Menu\_Proc:

+++++

Proc description :

-----  
Displays the current data base menu.

END Keyin\_Data\_Base\_Sub\_Menu\_Proc;

Hist\_Data\_Base\_Sub\_Menu\_Proc:

+++++

Proc description :

-----  
Displays the historical data base menu.

END Hist\_Data\_Base\_Sub\_Menu\_Proc;

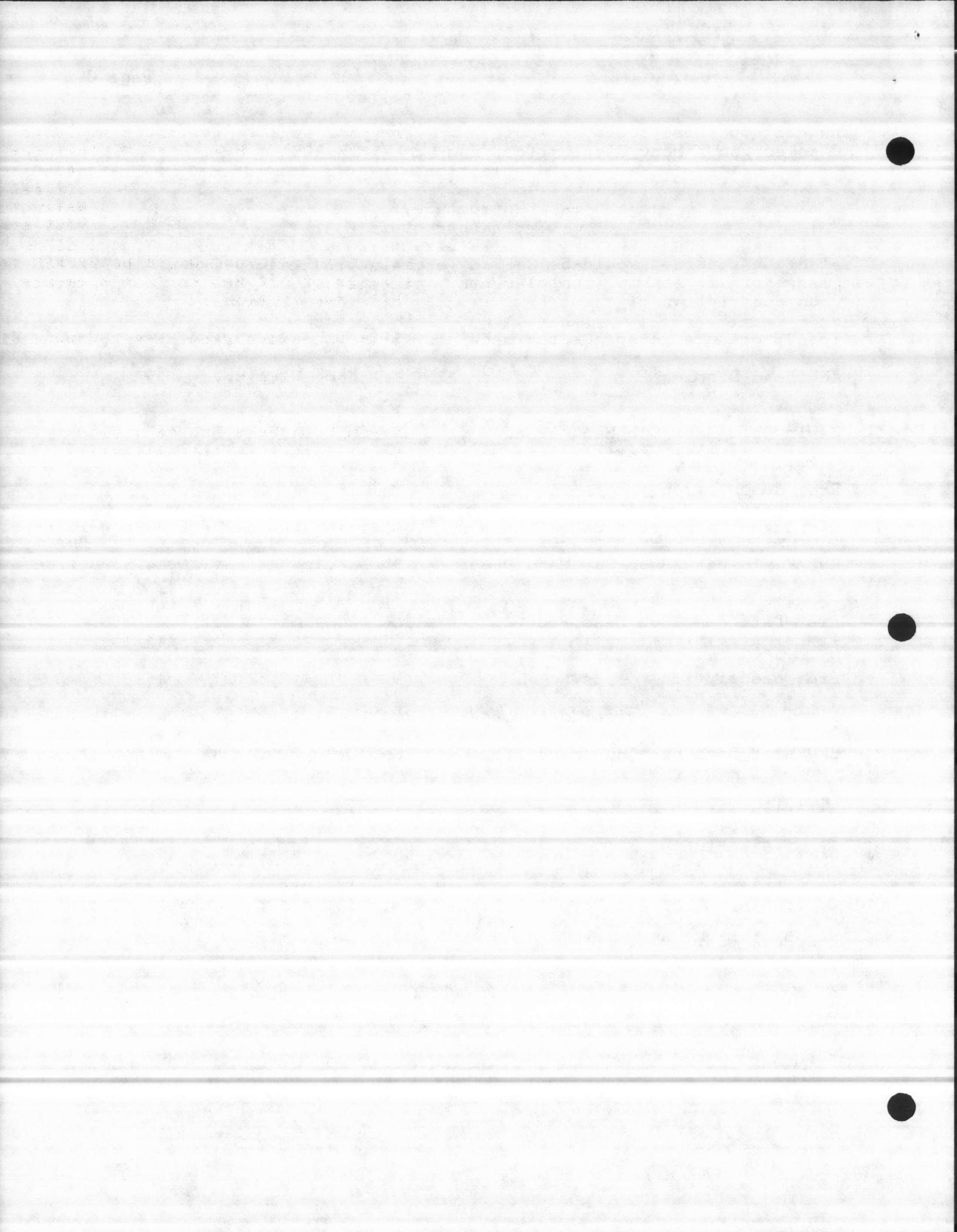
Base\_Display:

+++++

Proc. description :

- 1. displays the data base point header.  
2. displays the data base point static ascii.

End Base\_Display;



Base\_Update:  
++++++

Proc. description :

- 
1. fills the data base points with the proper data.
  2. updates the data base every 10 sec.
  3. displays the crt table for the data base point.
  4. displays the line graph for yearly analog points.

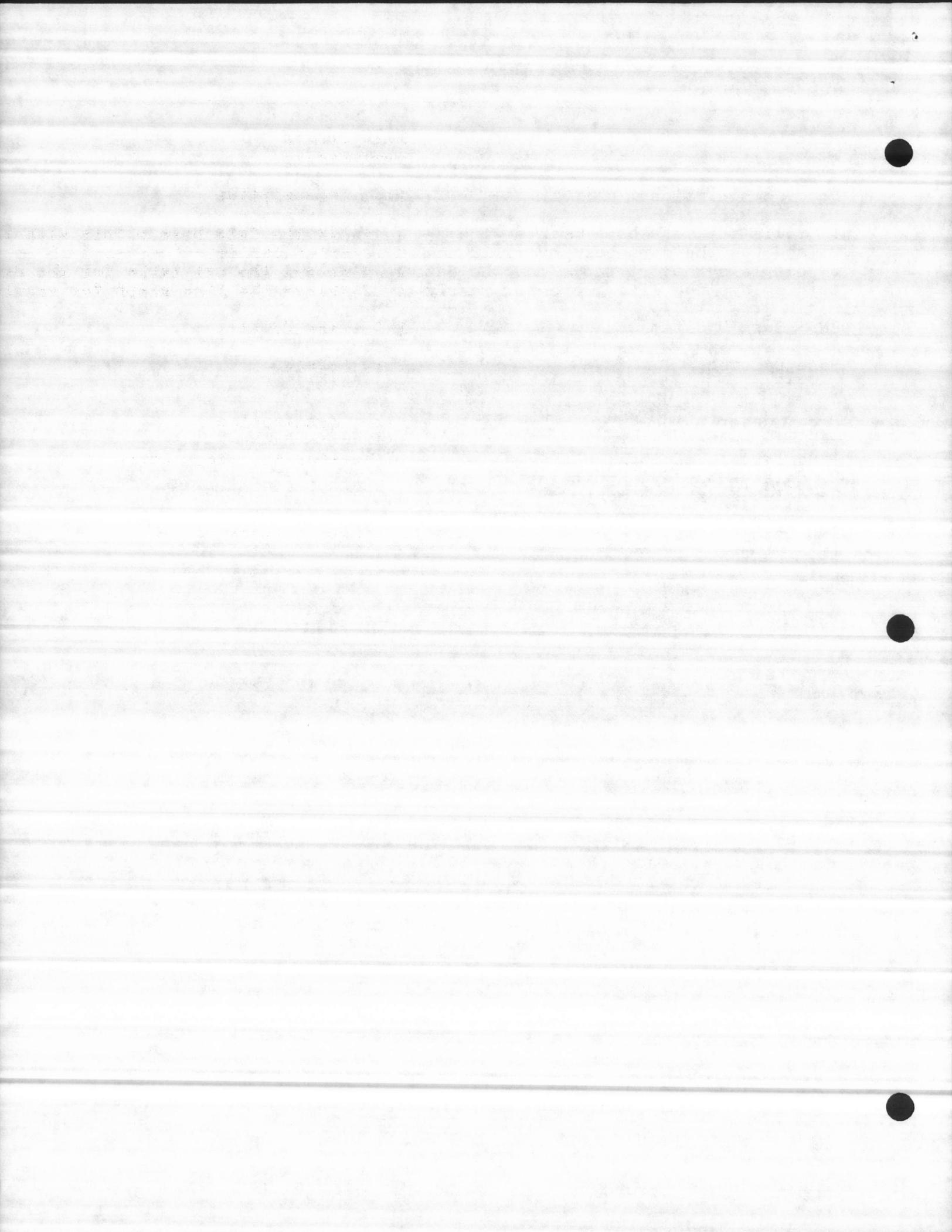
END Base\_Update;

Base\_Delete:  
++++++

Proc. description :

- 
1. deletes the clock entry.
  2. deletes the crt update semaphore.
  3. deletes data segment.

END Base\_Delete;



12.2 : MODULE NAME : CxxxxFILL.Pyy

=====

DISCREPTION : Fills the data base point with the proper data for  
current data & historical data.

PUBLIC PROCEDURES :

=====

Table\_Fill:

++++++

INPUT PARAMETERS :

field\_number, element, struc\_type, table\_p, param\_p

OUTPUT PARAMETER

result => A word, Ø or ØFFFFH.

Proc declaration :

element => A word value holds the element index.

field\_number=> A word value holds the field number.

struc\_type => A byte holds the structure type.

table\_p => A pointer - the address of the crt table.

param\_p => A pointer - the address of the parameter table.

pump\_index => A word used as pump index.

analog\_input\_scale\_xref => A number of words hold the field numbers  
to be scaled.

total\_analog\_input\_scale\_xref => A number of words hold the total  
field numbers to be scaled.

analog\_output\_scale\_xref => A number of words hold the analog out  
field numbers to be scaled.

level\_control\_scale\_xref => A number of words hold the level control  
field numbers to be scaled.

crt\_table BASED table\_p STRUCTURE :

This structure holds the crt table display.

crt\_param BASED param\_p STRUCTURE :

This structure holds the crt parameter.

Proc description :

=====

Fills the crt tables with the current data.

END Table\_Fill;

Hist\_Table\_Fill:

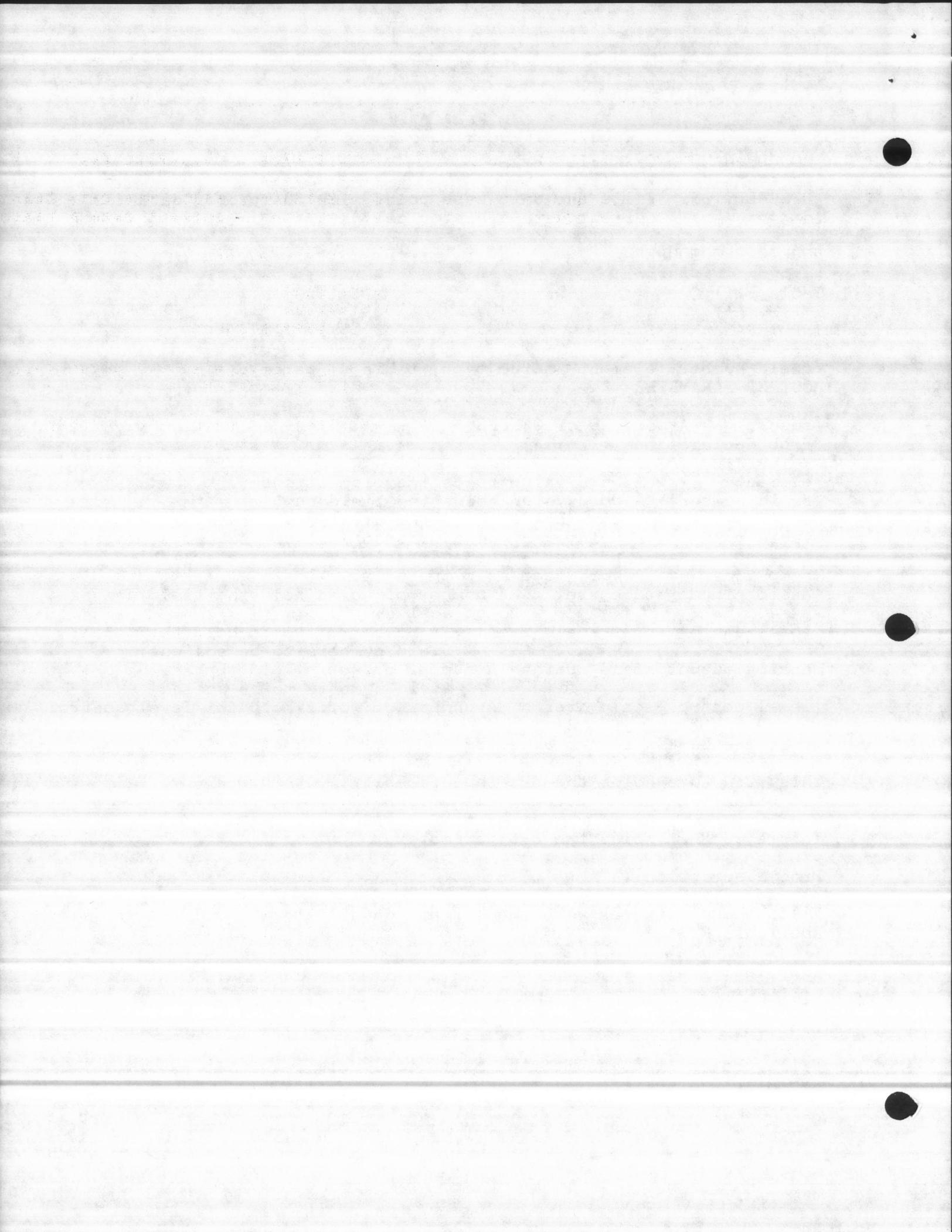
++++++

Proc description :

=====

Fills the crt tables with the historical data.

END Hist\_Table\_Fill;



---

AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

---

SECTION No. 13

D A T A   B A S E  
M O N I T O R   &   C O N T R O L S

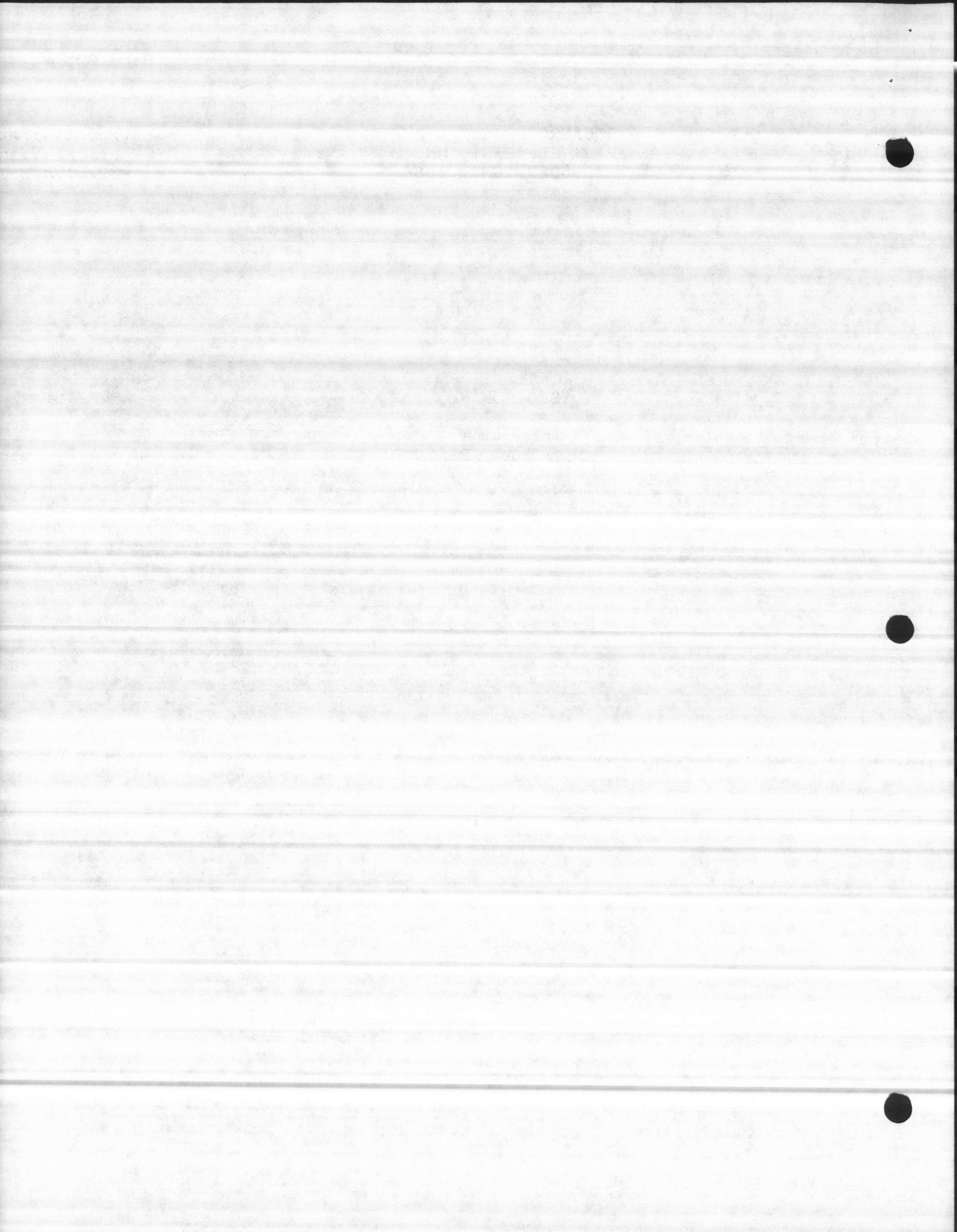
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



## INTRODUCTION :

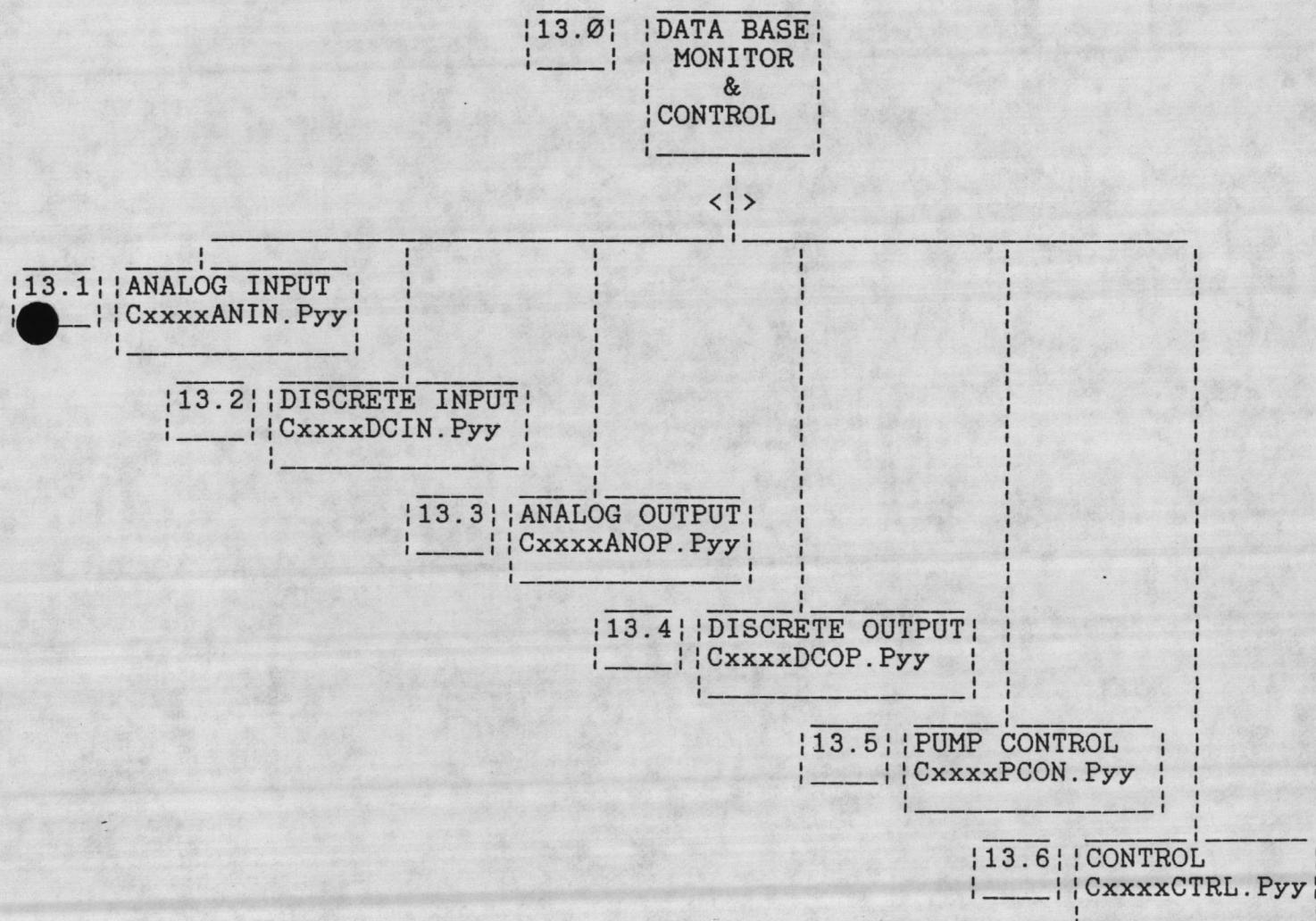
Our data base system is monitored internally by the following Tasks:

- 1) ANALOG INPUT TASK.
- 2) DISCRETE INPUT TASK.
- 3) ANALOG OUTPUT TASK.
- 4) DISCRETE OUTPUT TASK.
- 5) CONTROL TASK.
- 6) PUMP CONTROL TASK

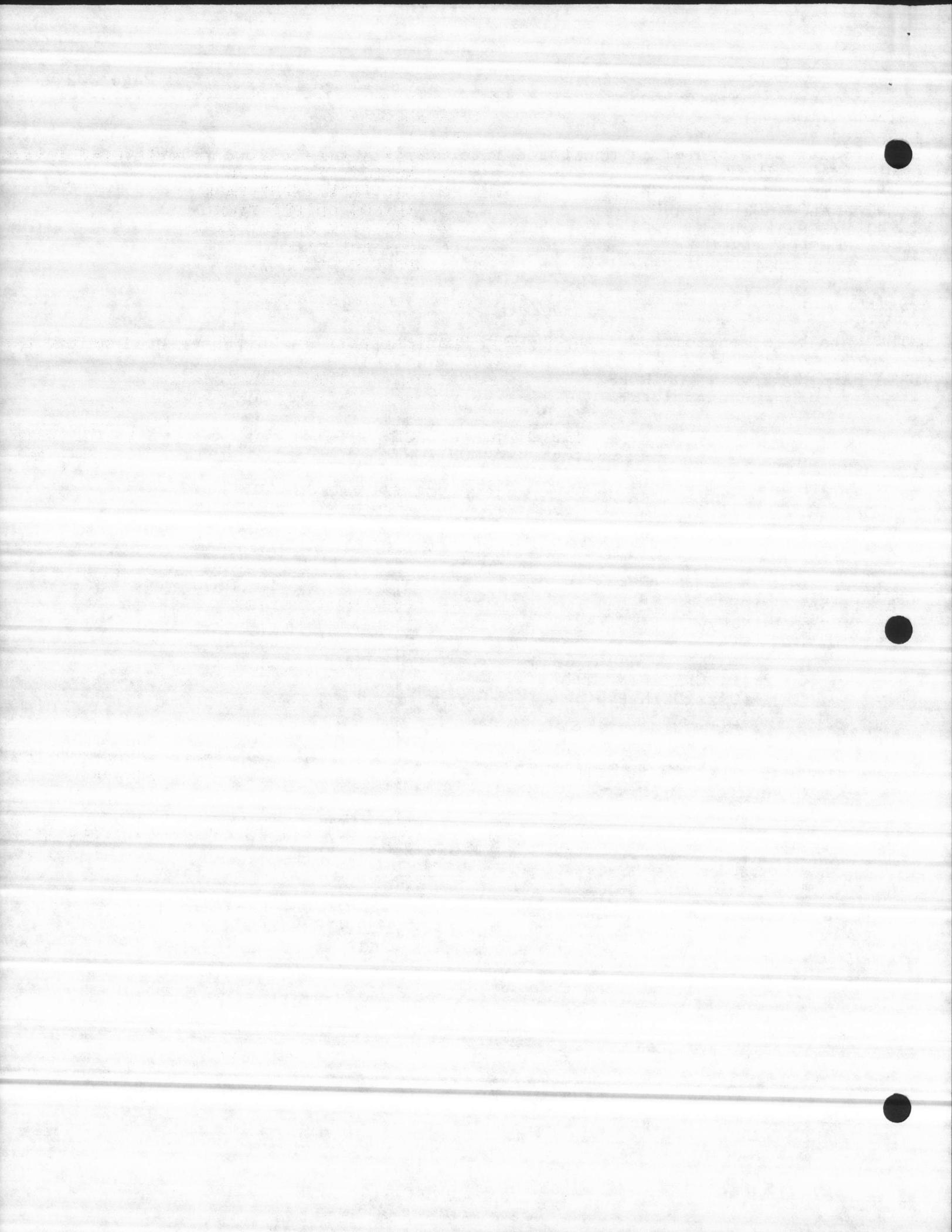
DATE : Oct 10, 1985

The data base monitor - Written by Bob Ryan, Mohamed Fayad, Peter Wollenzien.

## DATA BASE MONITOR &amp; CONTROLS:



WHERE      Cxxxx = C4758  
               &  
               Pyy = P86



## (13.1) MODULE NAME : CxxxxANIN.Pyy

=====

DESCRIPTION : This module contains the analog\_input\_task which monitors  
===== all the analog input points in the system.

## PUBLIC PROCEDURES :

=====

## 1) Input\_W1300: PROCEDURE

(data\_word\_p, remote\_index, word\_index) BYTE PUBLIC REENTRANT;

remote\_index - A word, which contains the index for the remotes.  
word\_index - A word, which contains the index for the word.  
data\_word\_p - A Pointer, which points to acutal data word.  
data\_word - A word, acutal data (BASED data\_word\_p).

## Proc Description :

This procedure checks the data change entry in the remote structure.  
If data has not changed a Ø is returned, else a ØFFH is returned.

END Input\_W1300;

## 2) Input\_W1300\_Data\_Fail: PROCEDURE

(data\_word\_p, remote\_index) BYTE PUBLIC REENTRANT;

remote\_index - A word, which contains the index for the remotes.  
data\_word\_p - A Pointer, which points to acutal data word.  
data\_word - A word, acutal data (BASED data\_word\_p).

## Proc Description :

This procedure checks the 8000H bit in the remote structure.  
If 8000H is high then data\_fail, else is no data\_fail.

END Input\_W1300\_Data\_Fail;

## 3) Input\_Io\_Rack: PROCEDURE

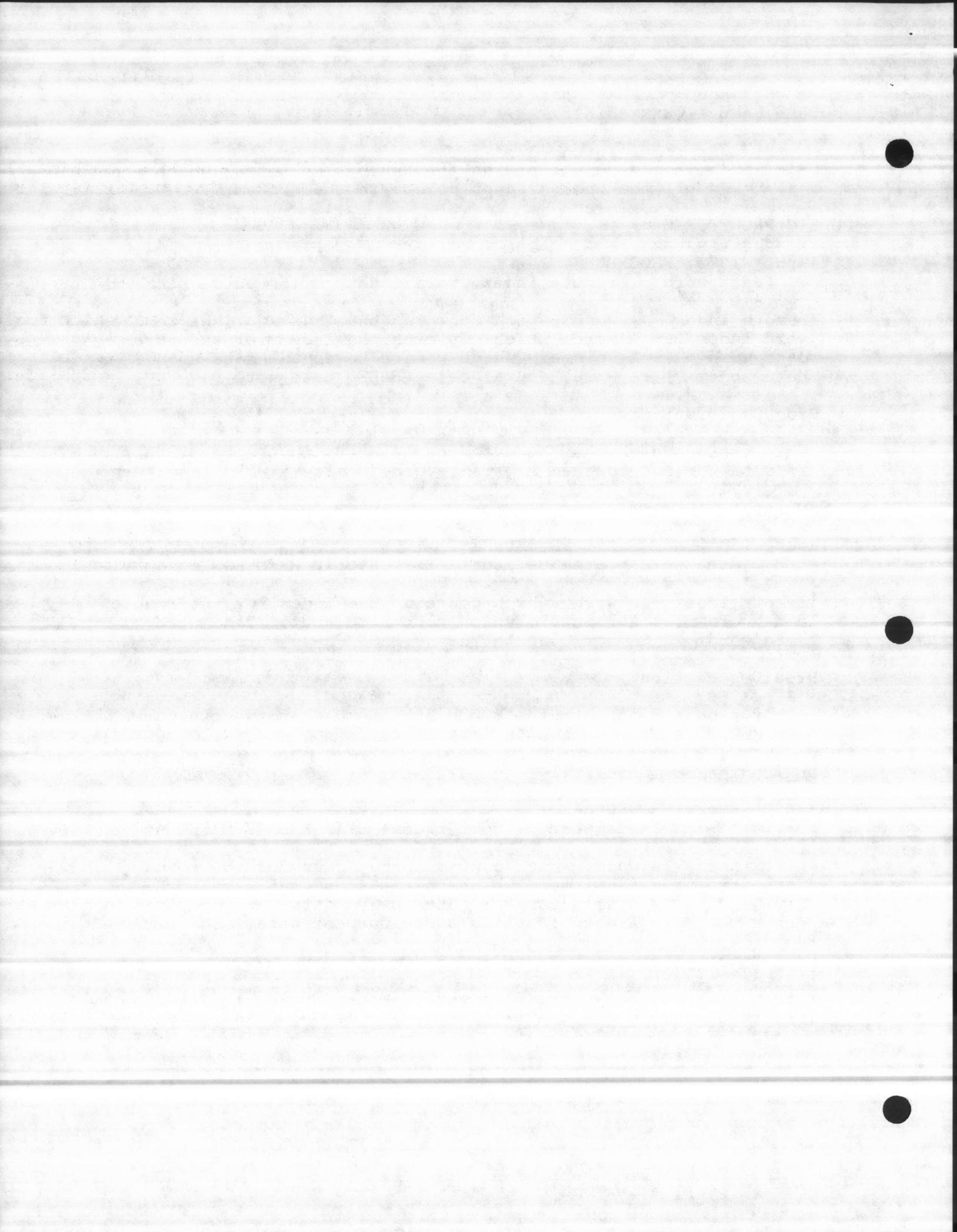
(data\_word\_p, port\_index) BYTE PUBLIC REENTRANT;

port\_index - A word, which contains the index for the ports.  
data\_word\_p - A Pointer, which points to acutal data word.  
data\_word - A word, acutal data (BASED data\_word\_p).  
new\_data - A word contains a new word of data.

## Proc Description :

This procedure checks the data change entry in the io\_ports structure.  
If data has not changed a Ø is returned, else a ØFFH is returned.

END Input\_Io\_Rack;



---

**GLOBAL DATA REFERENCE :**

=====

analog\_in Structure (CxxxxGLOB.Pyy).

**ANALOG\_INPUT\_TASK :**

=====

**INPUT\_PARAMETERS :**

===== None.

**OUTPUT\_PARAMETERS :**

===== None.

**LOCAL VARIABLES :**

=====

change\_flag - A byte, this flag is set when a value change.  
convert\_flag - A byte, this flag is set when a value converted.  
count - A word, is used as a counter.  
task\_tbl\_index - A word, is used as an index  
temp\_word - A word, is used to hold temp value.  
exception - A word, is used for exception messages.  
range - A word, is used for holding the range.  
low\_deadband - A word, is used for holding the low deadband value.  
high\_deadband - A word, is used for holding the high deadband value.  
scan\_sem\_t - A token, is used for creating a semaphore.  
temp\_dword - A dword, is used to hold temp dword value.  
current\_time - A dword, is used to hold the current time.

**TASK LOCAL PROCS :**

=====

1) Check\_Low\_Alarm : PROCEDURE  
(status\_p, current, setpoint, deadband, mask);

Proc Logic :

-----

check for low alarm set  
low alarm setpoint MUST be greater than deadband  
check for low alarm reset

END Check\_Low\_Alarm;

2) Check\_High\_Alarm : PROCEDURE

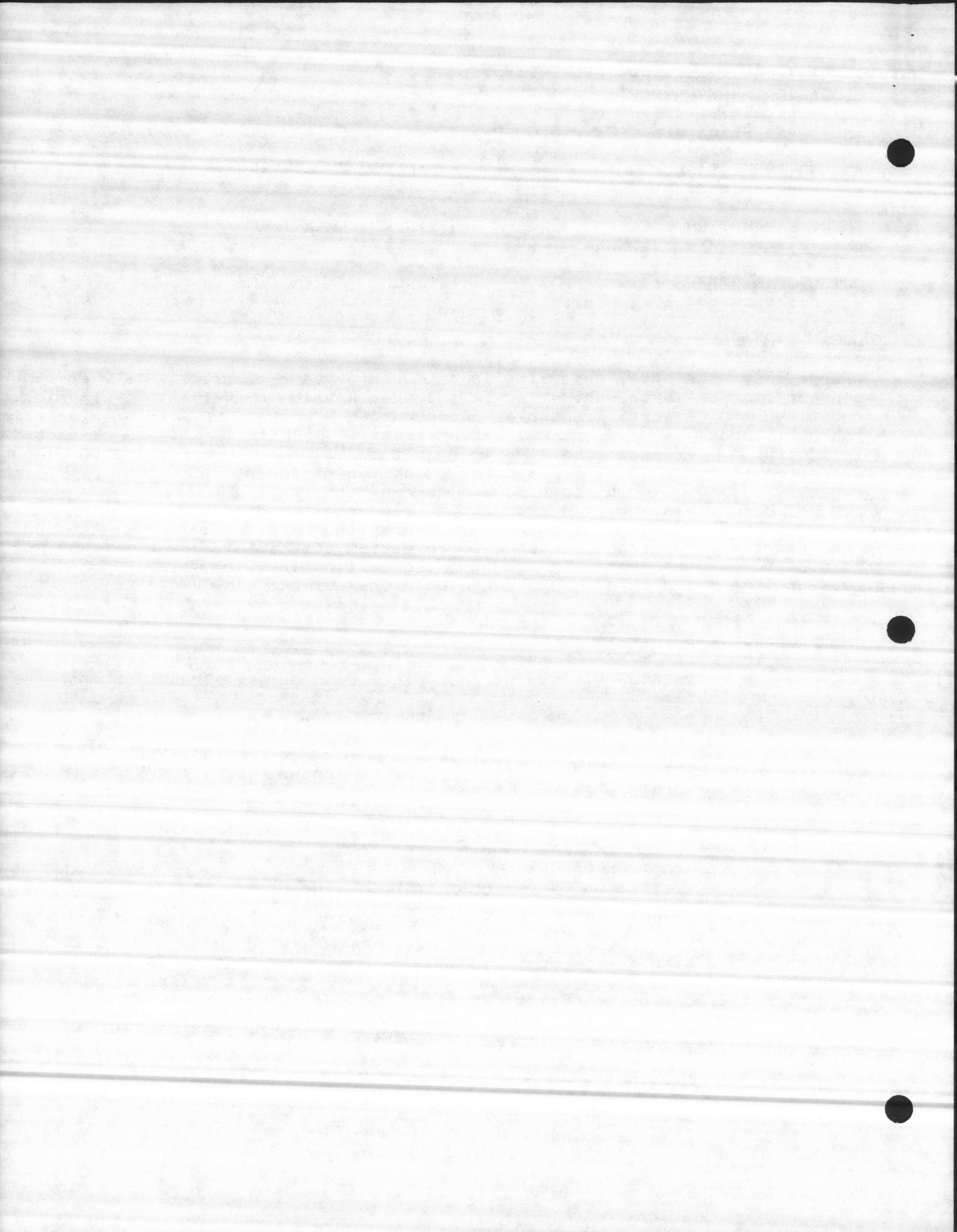
(status\_p, current, setpoint, deadband, mask);

Proc Logic :

-----

check for high alarm set  
check for high alarm reset  
high alarm setpoint MUST be greater than deadband

END Check\_High\_Alarm;



ALGORITHM :

=====

```
set exception handler
create a semaphore.
setup list entry for defined scan time.
```

Loop FOREVER.

```
wait at the list entry semaphore
get current time for min/max determination usage.
receive region for analog input definition
go through all the analog input points.
Loop1:DO count = 1 TO NUM_ANALOGS_IN.
```

```
Ø value in source means analog undefined
otherwise analog defined.
```

```
Loop2: point defined
increment scan time count by task interval time
```

```
initialize flags
check if scan time up
```

```
Loop3:scan time up
decrement scan time count by scan time
test if input activated (1H bit of input flag)
```

```
Loop4: input activated
```

```
bring in actual data.
```

```
Loop case: of sources.
```

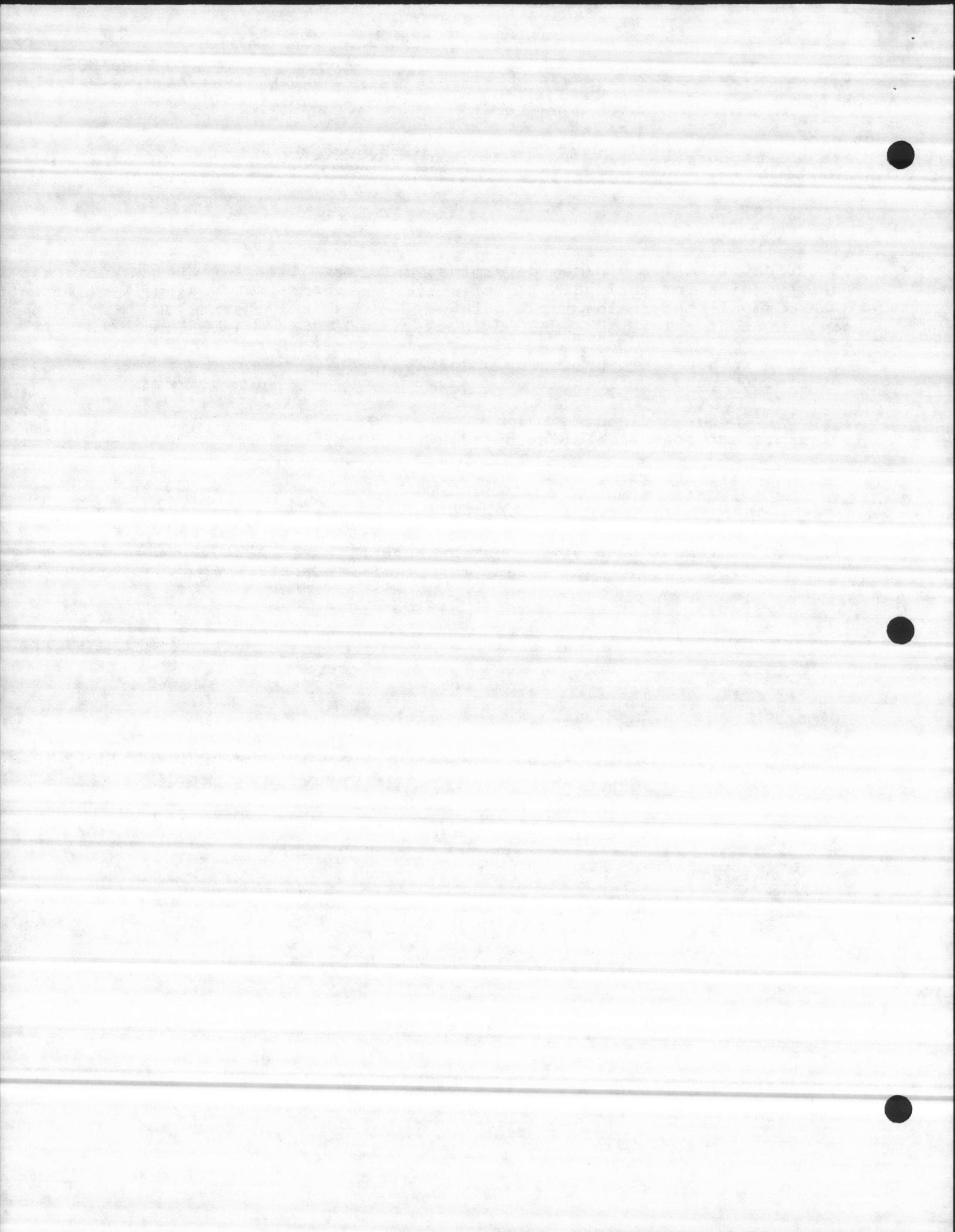
```
case Ø - undefined point
```

```
case 1 - W1300 telemetry input
case 2 - io rack input
case 3 - calculation input
case 4, manual input
```

```
END Loop case.
```

```
END Loop4
```

```
END Loop3
```



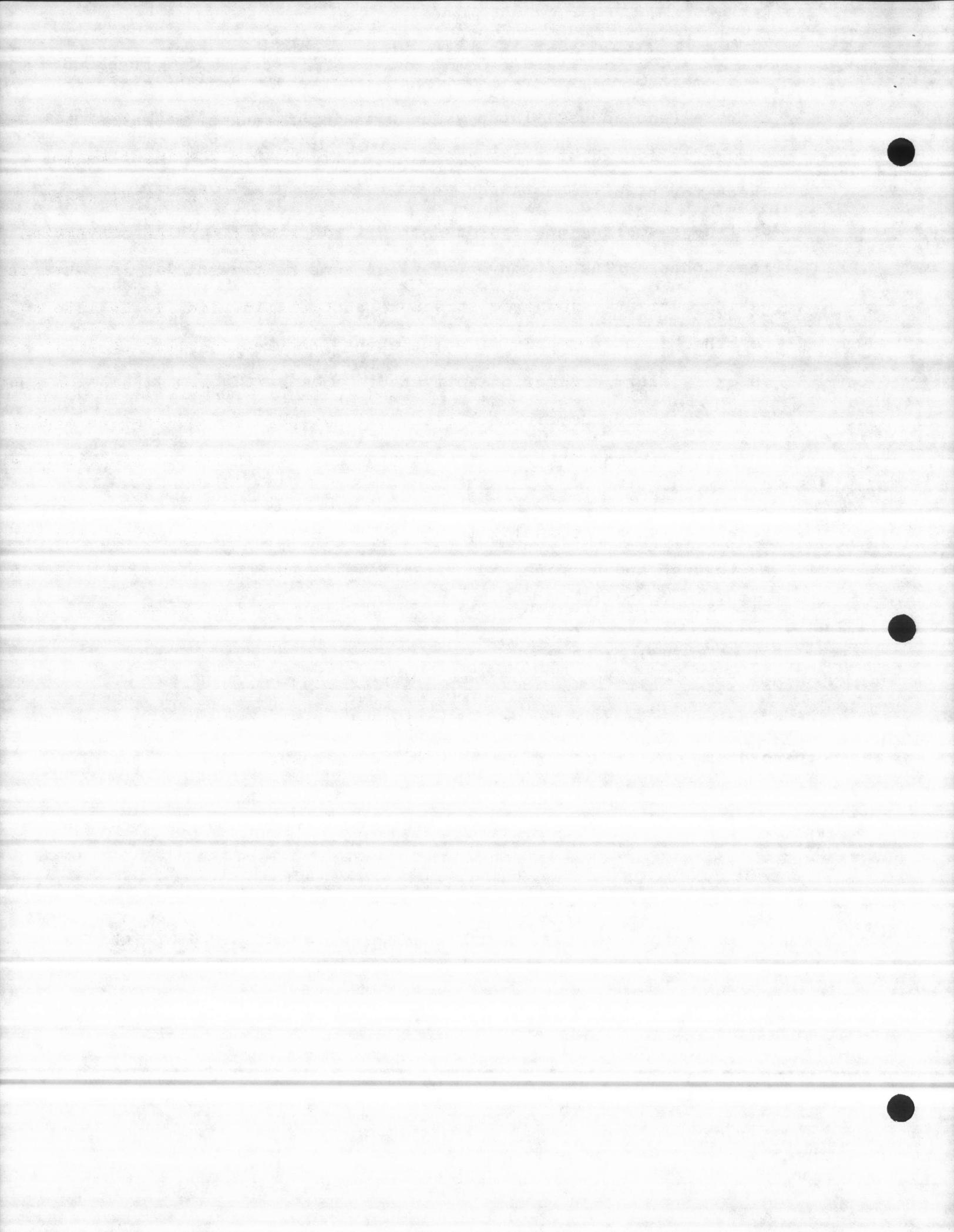
```
compare (and adjust) input value with input ranges
check for low input range
check for high input range
do linear scaling conversion
reset change flag if converted value same as present.
assign current value into structure and rest invalid flag.
do other functions only if value change occurred.
check if alarm generation activated.
go through and compare each alarm bit
check if min/max generation activated
receive min/max region token
receive region for analog input definition
go through and compare each min/max
release min/max region

END Loop2

END Loop1

release region for analog input definition

END Loop forever
```



(13.2) MODULE NAME : CxxxxDCIN.Pyy

=====

DESCRIPTION : This module contains the discrete\_input\_task which monitors  
===== all the discrete input points in the system.

EXTERNAL PROCEDURES :

=====

Send\_To\_Fail: PROCEDURE (fail\_type, index) EXTERNAL;  
/\* CxxxxFAIL.Pyy \*/

DECLARE fail\_type BYTE,  
index WORD;

END Send\_To\_Fail;

Input\_W1300: PROCEDURE

(data\_word\_p, remote\_index, word\_index) BYTE EXTERNAL;  
/\* CxxxxANIN.Pyy \*/

DECLARE (remote\_index, word\_index) WORD,  
(data\_word\_p) POINTER;

END Input\_W1300;

Input\_W1300\_Data\_Fail: PROCEDURE

(data\_word\_p, remote\_index) BYTE EXTERNAL;  
/\* CxxxxANIN.Pyy \*/

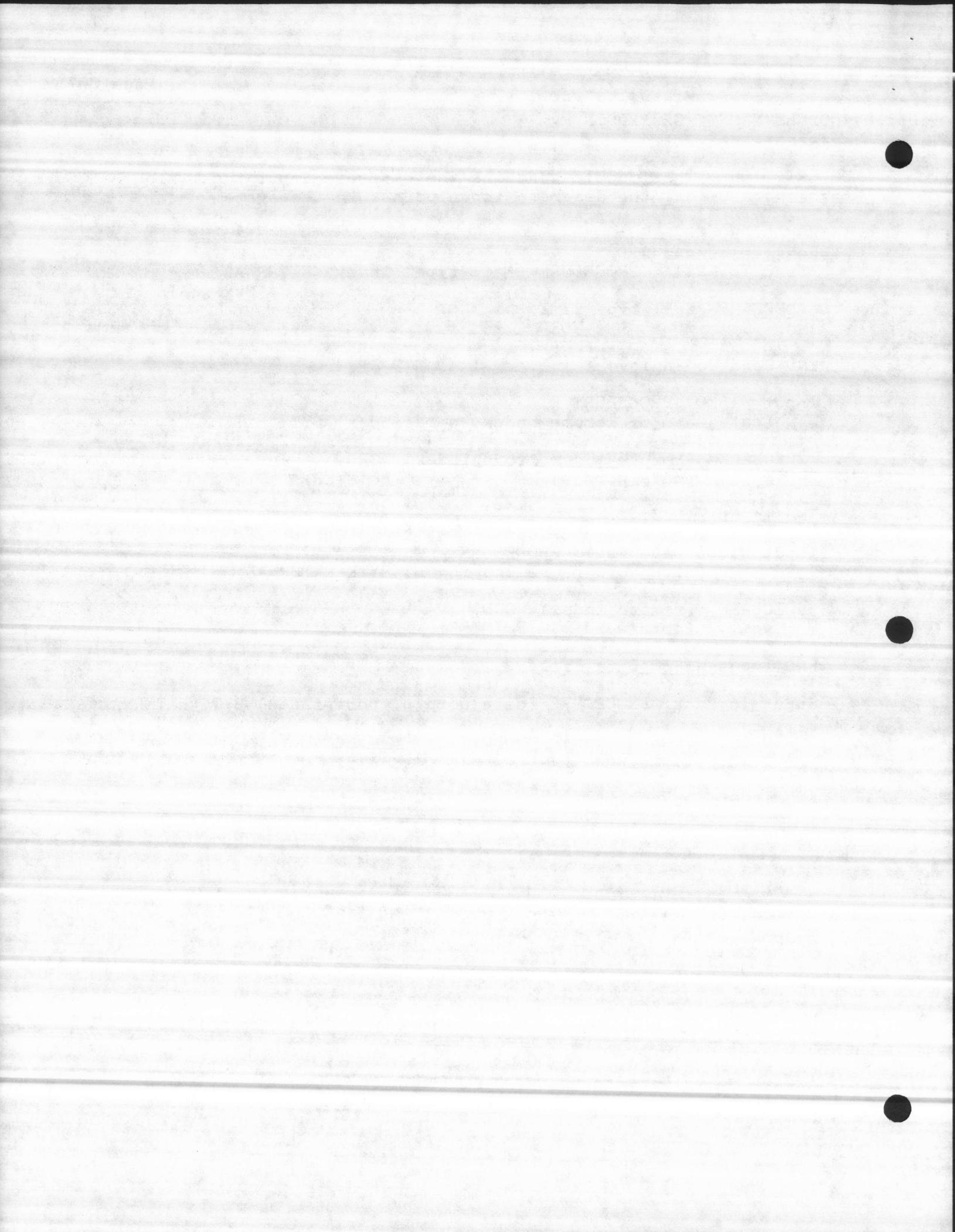
DECLARE (remote\_index) WORD,  
(data\_word\_p) POINTER;

END Input\_W1300\_Data\_Fail;

Input\_Io\_Rack: PROCEDURE (data\_word\_p, port\_index) BYTE EXTERNAL;  
/\* CxxxxANIN.Pyy \*/

DECLARE (port\_index) WORD,  
(data\_word\_p) POINTER;

END Input\_Io\_Rack;



---

GLOBAL DATA REFERENCE :

```
=====
discrete_in Structure (CxxxxGLOB.Pyy).
```

## DISCRETE\_INPUT\_TASK :

```
=====
```

## INPUT\_PARAMETERS :

```
===== None.
```

## OUTPUT\_PARAMETERS :

```
===== None.
```

## LOCAL VARIABLES :

```
=====
```

```
change_flag - A byte, this flag is set when a value change.  
count - A word, is used as a counter.  
task_tbl_index - A word, is used as an index  
temp_word - A word, is used to hold temp value.  
new_current_value - A word, is used to hold the new current value.  
exception - A word, is used for exception messages.  
scan_sem_t - A token, is used for creating a semaphore.  
current_time - A dword, is used to hold the current time.
```

## ALGORITHM :

```
=====
```

```
set exception handler.
```

```
create input scan semaphore.
```

```
setup clock entry for defined scan time
```

```
Loop FOREVER
```

```
    wait at the list entry semaphore
```

```
    get current time for min/max determination usage
```

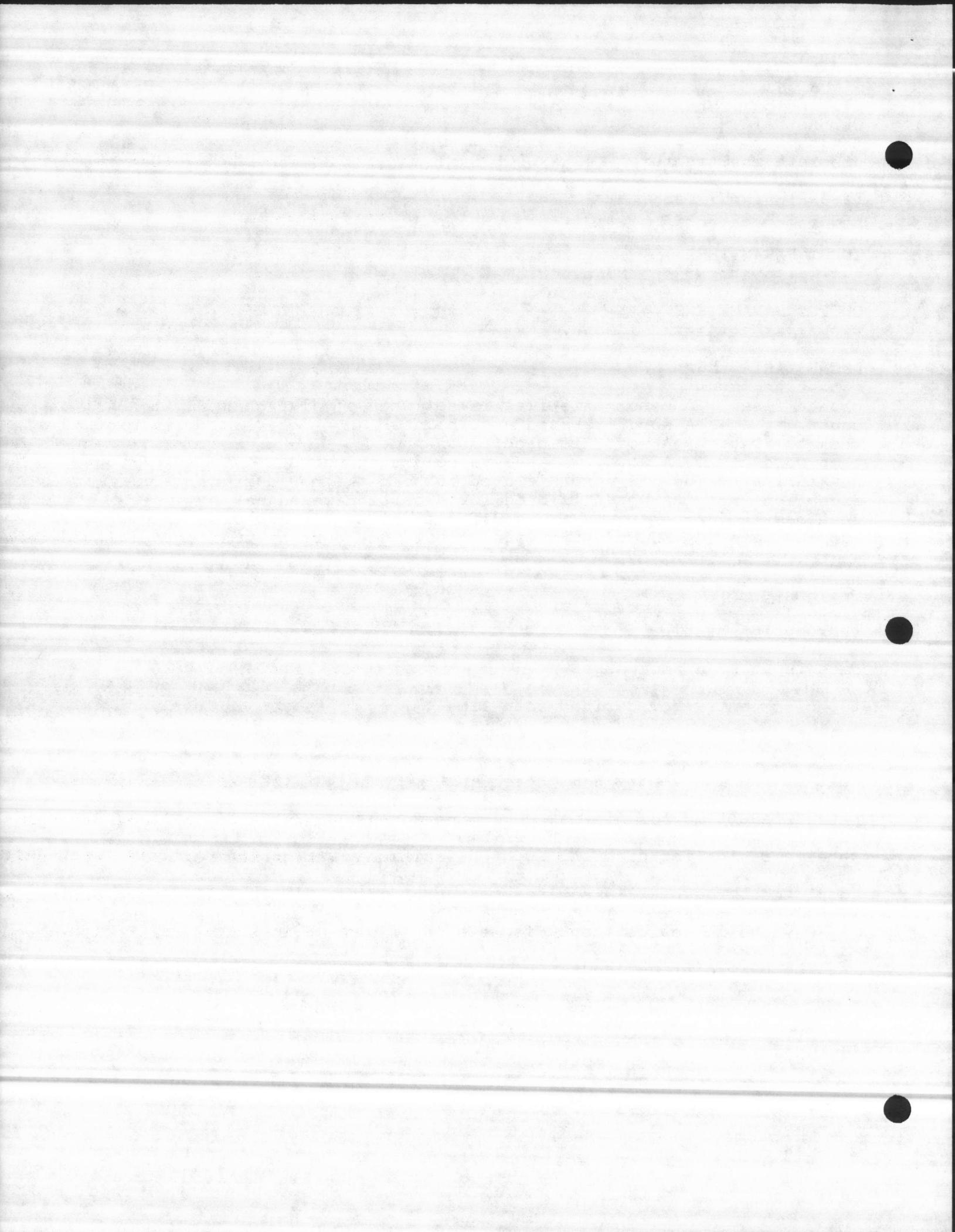
```
    receive region for discrete input definition
```

```
    Loop1 : go through all discretes that are defined
```

```
        Ø value in source means discrete undefined
```

```
        Loop2 : point defined
```

```
            increment scan time count by task interval time
```



```
Loop3 :check if scan time up

    test if input deactivated (1H bit of input flag)

    input activated, bring in actual data

    Loop CASE discrete input source.

        case 0 - undefined point

        case 1 - W1300 telemetry input
                  and input W1300 data fail

        case 2 - io rack input

        case 3 - analog alarm

        case 5 - manual

    END Loop Case.

    check for normally open / normally closed

    determine new value

    set change flag

    If an alarm is present determine when to annunciate
    Loop5:
        Check for send to fail

    Loop6 :check for start accumulation

    END Loop6.

    END Loop5.

    END Loop3.

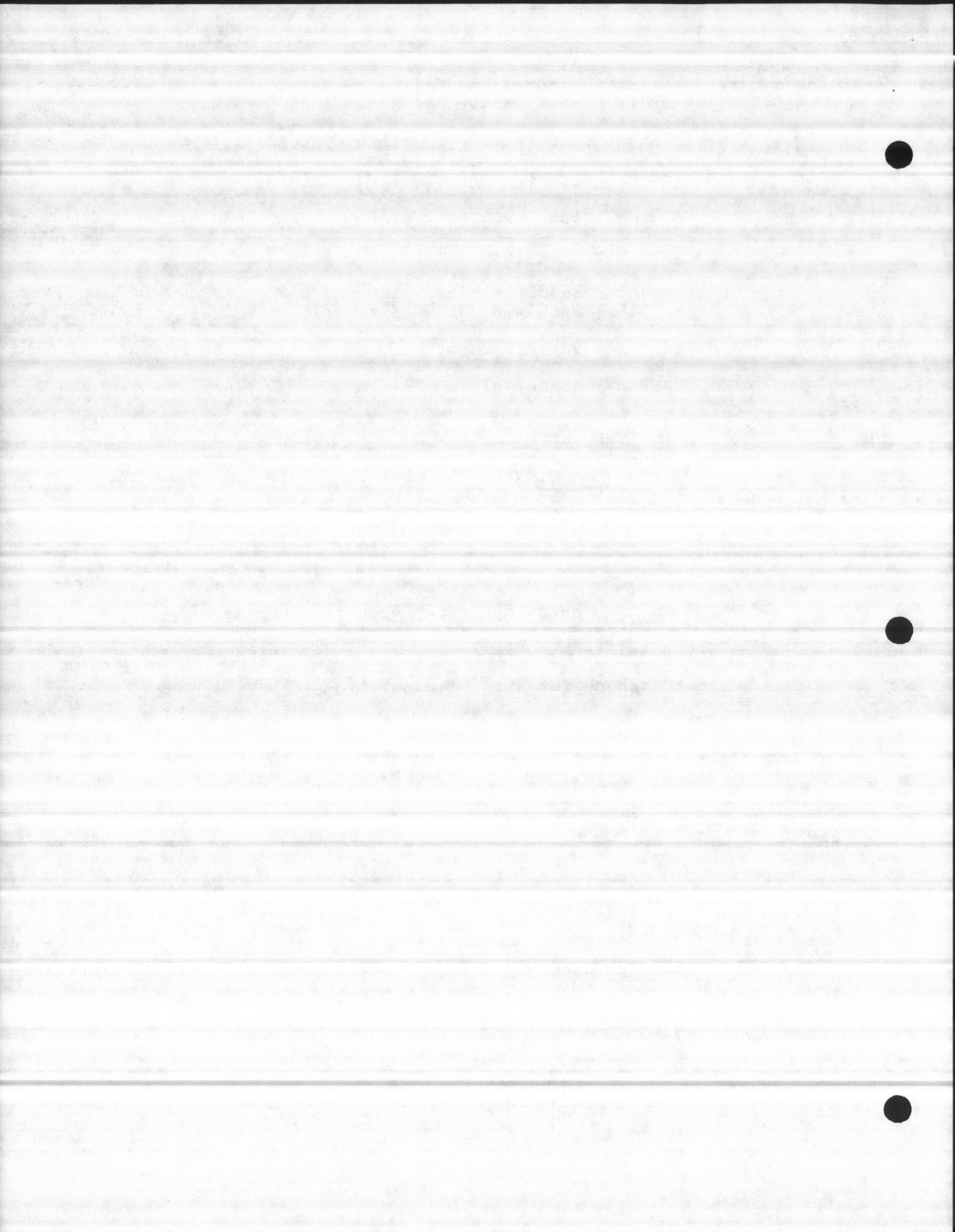
    END Loop2.

    END Loop1.

    Hardware acknowledge Bit
    Check Ack Bit (2)

    release region for discrete input definition

END Loop forever
```



(13.3) MODULE NAME : CxxxxANOP.Pyy

=====

DESCRIPTION : This module contains the analog\_output\_task which monitors  
===== all the analog output points in the system.

PUBLIC PROCEDURES :

=====

1) Output\_W1300: PROCEDURE

(data\_word\_p, master\_index, word\_index, mask) PUBLIC REENTRANT;  
master\_index - A word, holds a master index.  
word\_index - A word, holds a word index.  
mask - A word, holds the mask for each word value.  
data\_word\_p - A pointer points to data word.  
data\_word - A word holds the data (BASED data\_word\_p).

Proc Description:

-----  
This procedure carry the data to the master board, and mask  
on it.

END Output\_W1300;

2) Output\_Io\_Rack: PROCEDURE

(data\_word\_p, port\_index, mask) PUBLIC REENTRANT;  
port\_index - A word holds the port index.  
new\_data - A word holds a new data in the case if data change.  
mask - A word holds the mask value;  
exception - A word holds the exception message.  
data\_word\_p - A pointer points to the data word.  
data\_word - A word holds the data word(BASED data\_word\_p).

Proc Description :

-----  
This procedure gains control of the io rack region and sends the new  
data to its port. And then releases the control of io rack region.

END Output\_Io\_Rack;

GLOBAL DATA REFERENCE :

=====

analog\_out Structure (CxxxxGLOB.Pyy).

ANALOG\_OUTPUT\_TASK :

=====

INPUT\_PARAMETERS :

===== None.

OUTPUT\_PARAMETERS :

===== None.

LOCAL VARIABLES :

=====

count - A word, is used as a counter.

src\_index - A word, is used as a source index.

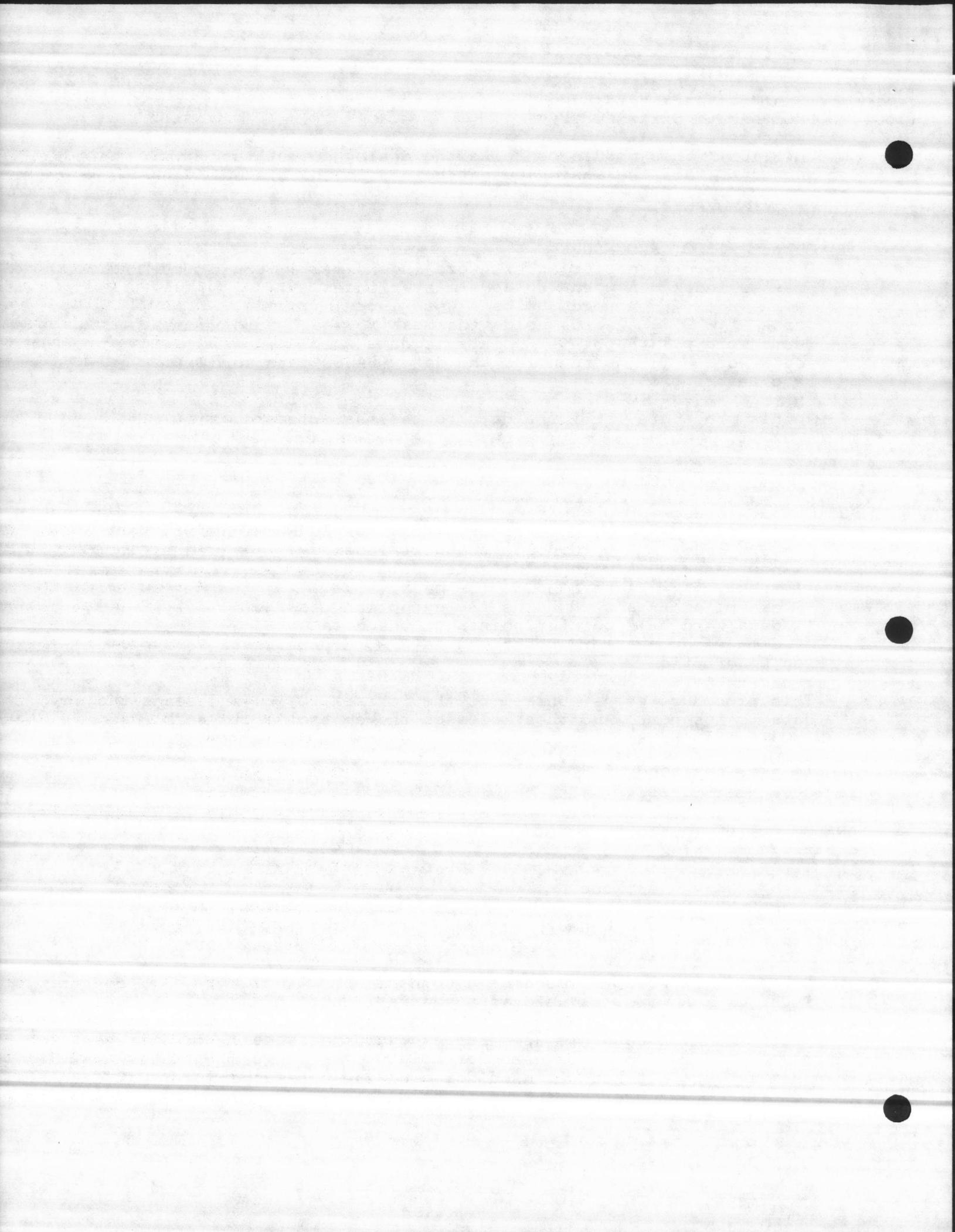
temp\_word - A word, is used to hold temp value.

exception - A word, is used for exception messages.

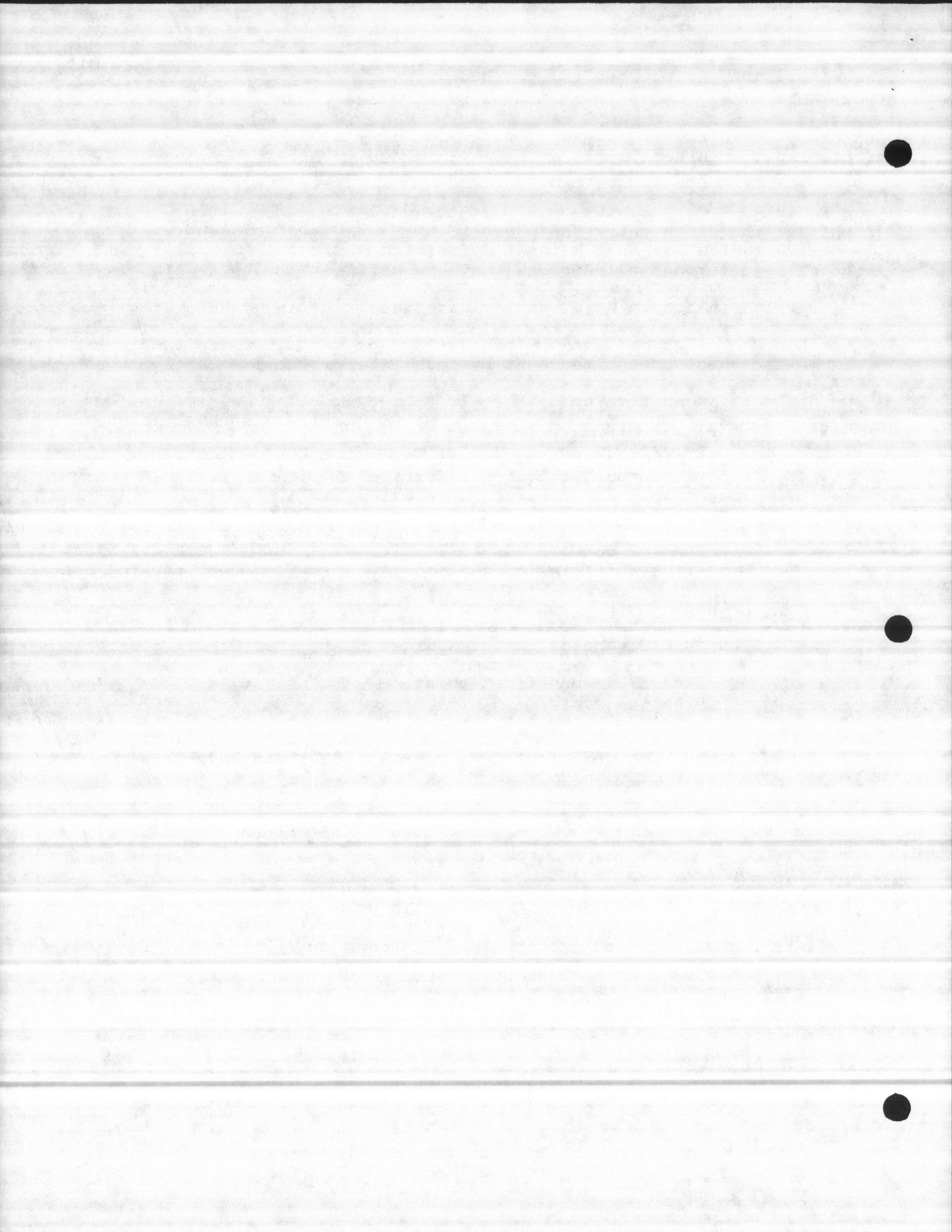
range - A word, is used for holding the range.

scan\_sem\_t - A token, is used for creating a semaphore.

temp\_dword - A dword, is used to hold temp dword value.



```
-----  
ALGORITHM :  
=====  
    set exception handler.  
    create input scan semaphore  
    setup clock entry for defined scan time  
  
Loop FOREVER:  
    wait at the clock entry semaphore  
    receive region for analog input definition  
    Loop1: go through all analogs that are defined  
  
        Ø value in output means analog undefined  
  
        Loop2: point defined  
            increment scan time count by task interval time  
  
        Loop3: scan time up.  
            check if scan time up  
  
        Loop4: point activated  
            test if output activated (1H bit of input flag)  
            output activated, bring in actual data  
            output activated, bring in actual data  
            compare (and adjust) output value with output ranges  
            check for conversions  
            do linear scaling conversion  
  
            Check for BCD conversion  
  
        Loop CASE analog output sources.  
            case Ø - undefined point  
            case 1 - W130Ø telemetry input  
            case 2 - io rack input  
  
        END Loop CASE.  
  
    END Loop4  
  
    END Loop3  
  
    END Loop2  
  
END Loop1  
  
release region for analog output definition  
send list.  
  
END Loop forever.
```



(13.4) MODULE NAME : CxxxxDCOP.Pyy  
=====

DESCRIPTION : This module contains the discrete\_output\_task which monitors  
===== all the discrete output points in the system.

EXTERNAL PROCEDURES :

=====

Output\_W1300: PROCEDURE

(data\_word\_p, master\_index, word\_index, mask) EXTERNAL;  
===== /\* CxxxxANOP.Pyy \*/

DECLARE (master\_index, word\_index, mask) WORD,  
===== (data\_word\_p) POINTER;

END Output\_W1300;

Output\_Io\_Rack: PROCEDURE (data\_word\_p, port\_index, mask) EXTERNAL;  
===== /\* CxxxxANOP.Pyy \*/

DECLARE (port\_index, mask) WORD,  
===== (data\_word\_p) POINTER;

END Output\_Io\_Rack;

GLOBAL DATA REFERENCE :

=====

discrete\_out Structure (CxxxxGLOB.Pyy).

DISCRETE\_OUTPUT\_TASK :

=====

INPUT\_PARAMETERS :

===== None.

OUTPUT\_PARAMETERS :

===== None.

LOCAL VARIABLES :

=====

count - A word, is used as a counter.

src\_index - A word, is used as a source index

temp\_word - A word, is used to hold temp value.

exception - A word, is used for exception messages.

scan\_sem\_t - A token, is used for creating a semaphore.

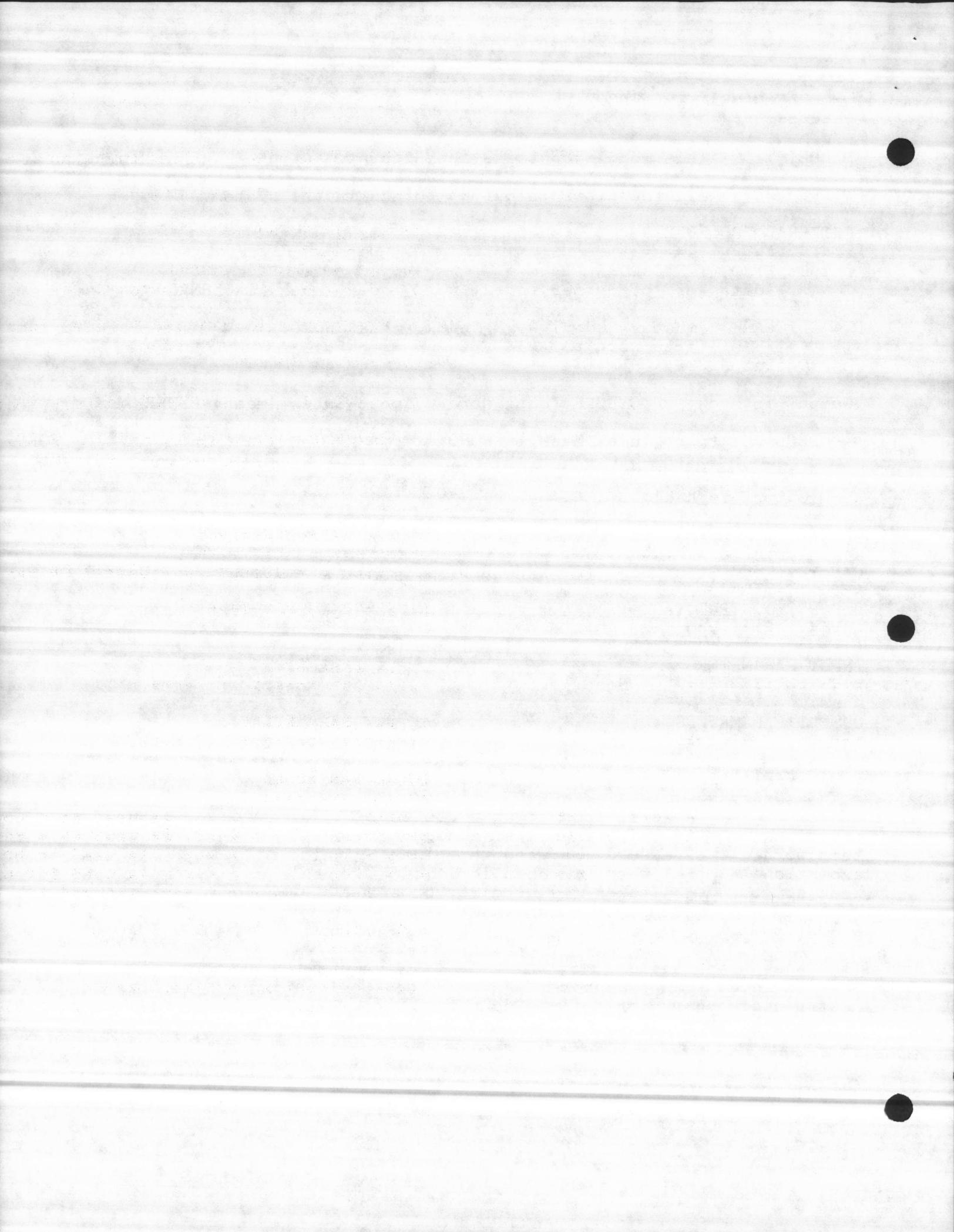
ALGORITHM :

=====

set exception handler.

create input scan semaphore

setup clock entry for defined scan time



Loop FOREVER :

wait at the clock entry semaphore  
receive region for discrete output definition  
Loop1: go through all the dialers in the system.

SEND OUT DIALER

END Loop1.  
Check the sonalert  
if the first sonalert is greater than zero then

Sound Common Sonalert

Sound all of the appropriate sonalerts by looping through all the sonalerts.

Silence all the Sonalerts  
Computer Pulsed Output Bit  
Loop2: go through all analogs that are defined

Ø value in output means discrete undefined

Loop3: point defined

increment scan time count by task interval time

Loop4: scan time up  
check if scan time up

test if output activated (1H bit of input flag)

Loop5: output activated  
bring in actual data  
check for normally open / normally closed  
determine new value

Loop CASE discrete output sources

case Ø - undefined point  
case 1 - W1300 telemetry input  
case 2 - io rack input  
END Loop Case.

END Loop5.

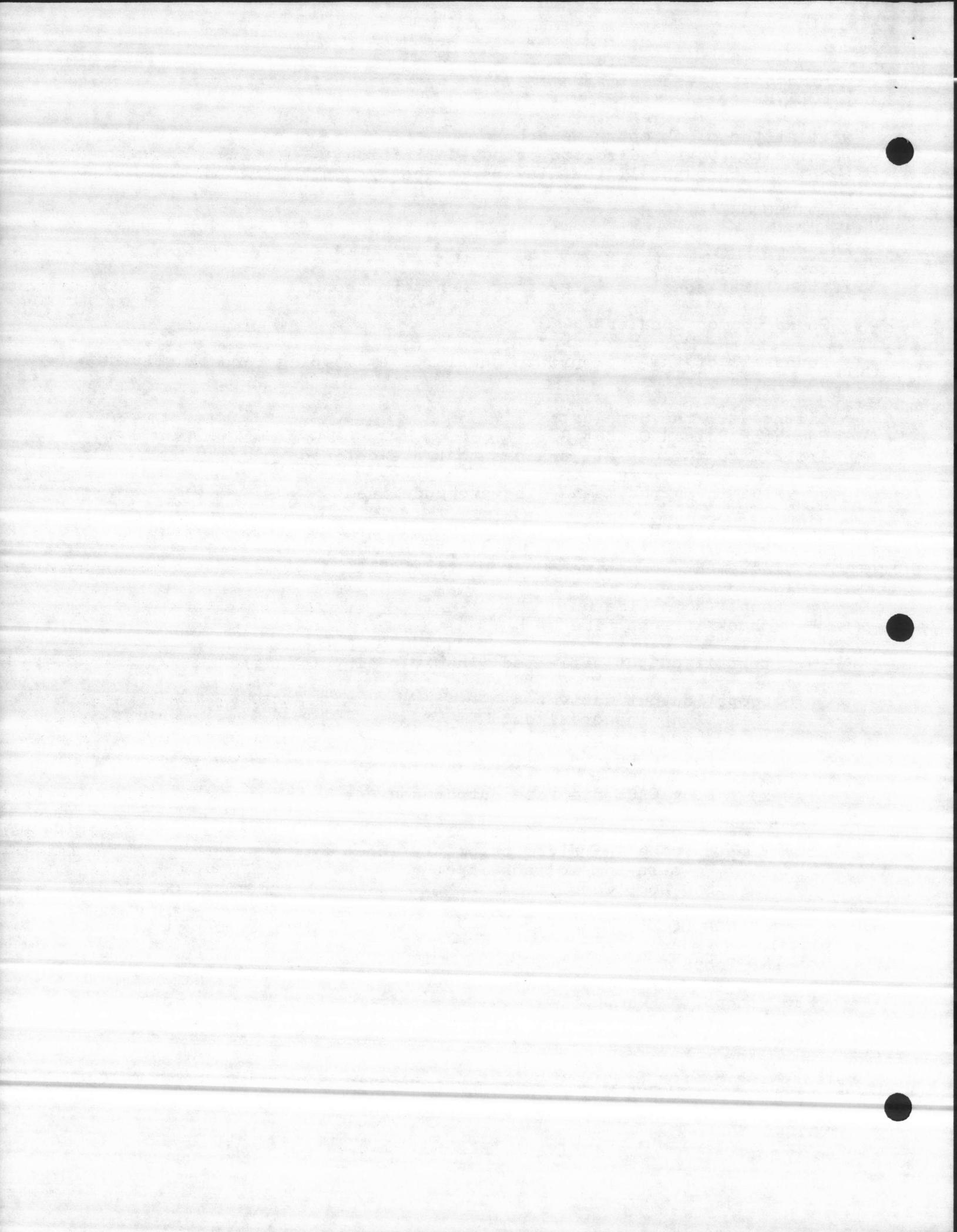
END Loop4

END Loop3

END Loop2

release region for discrete output definition  
send list.

END Loop forever.



(13.5) MODULE NAME : CxxxxPCON.Pyy

---

=====

DESCRIPTION : This module contains the Pump control task which controls all the control points in the system. The module CxxxxPCON.Pyy is user definable in the nature of applying all the tag names into the level control point display and activate.

EXTERNAL PROCEDURES :

---

```
Prompt_Entry: PROCEDURE (entry_max_count,crt_num) EXTERNAL;
    DECLARE (crt_num,entry_max_count) BYTE;
END Prompt_Entry;
```

```
Point_Id_Search: PROCEDURE (id_p,struc_type_p,struc_index_p)
    BYTE EXTERNAL;
    DECLARE (id_p,struc_type_p, struc_index_p) POINTER;
END Point_Id_Search;
```

GLOBAL REFERENCE :

---

Level\_control structure (See CxxxxGEXR.Pyy)

CONTROL PROCEDURES :

---

Perform\_Start:

+++++

Proc description :

This procedure used to start the pumps.

END Perform\_Start;

Perform\_Stop:

+++++

Proc description :

This procedure stops the runing pumps.

END Perform\_Stop;

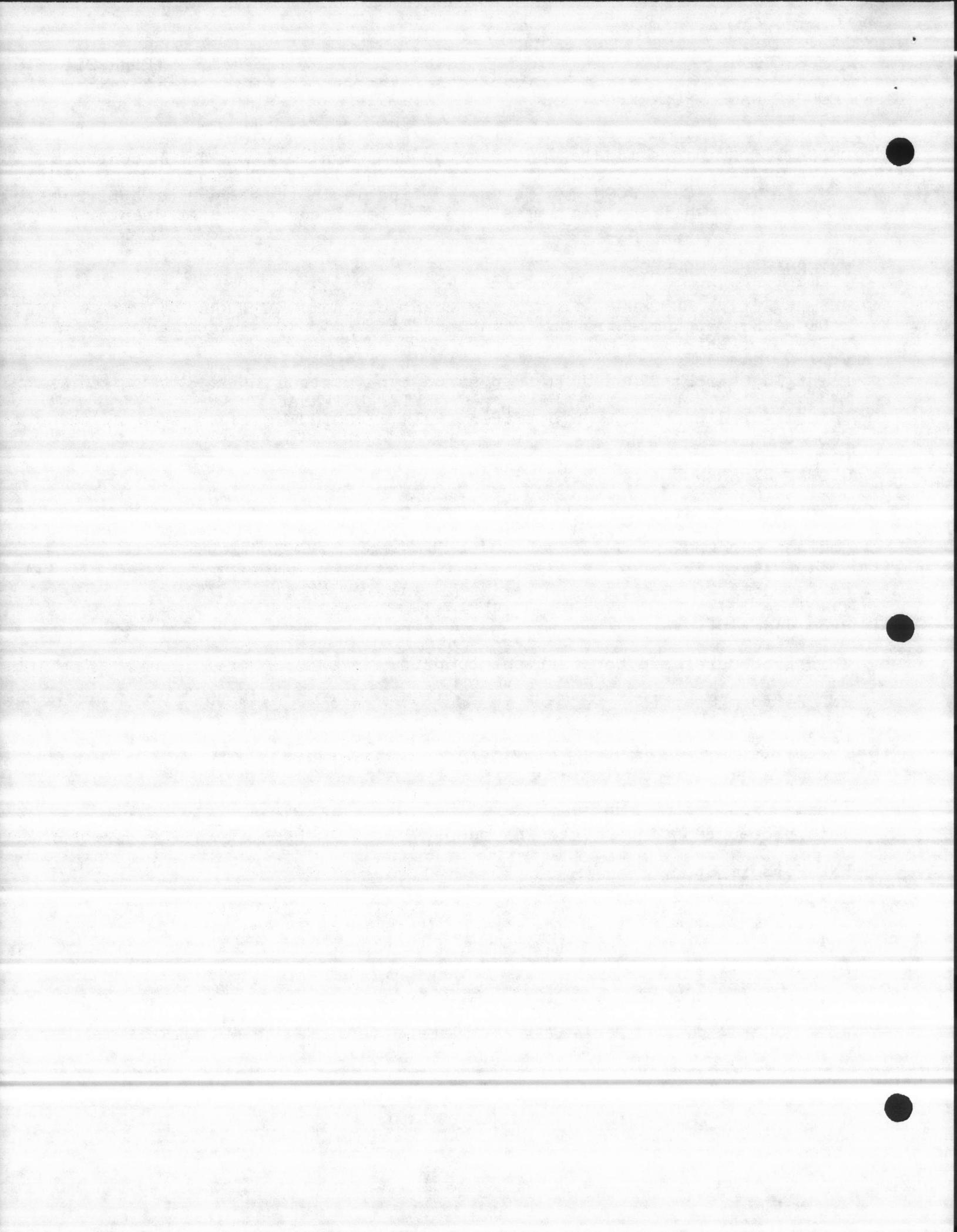
Perform\_Fail\_Reset:

+++++

Proc description :

This procedure performs a manual fail reset.

END Perform\_Fail\_Reset;



```
Perform_Enable_Backwash:  
++++++
```

Proc description :

```
This procedure performs a manual enable for the backwash pump.  
END Perform_Enable_Backwash;
```

```
Perform_Disable_Backwash:  
++++++
```

Proc description :

```
This procedure performs a manual disable for the backwash pump.  
END Perform_Disable_Backwash;
```

```
Perform_Low_Cutout:  
++++++
```

Proc description :

```
This procedure performs the low cutout and stop all the pumps.  
END Perform_Low_Cutout;
```

MODULE TASKS :

```
=====
```

```
Pump_Control_Task:  
++++++
```

Task\_description :

```
This task controls all the pumps in the entire system in auto mode.
```

```
END Pump_Control_Task;
```

```
Keyin_Start_Stop_Proc:  
++++++
```

```
This procedure is the keyboard command to operate pumps in the manual  
mode.
```

input parameters:

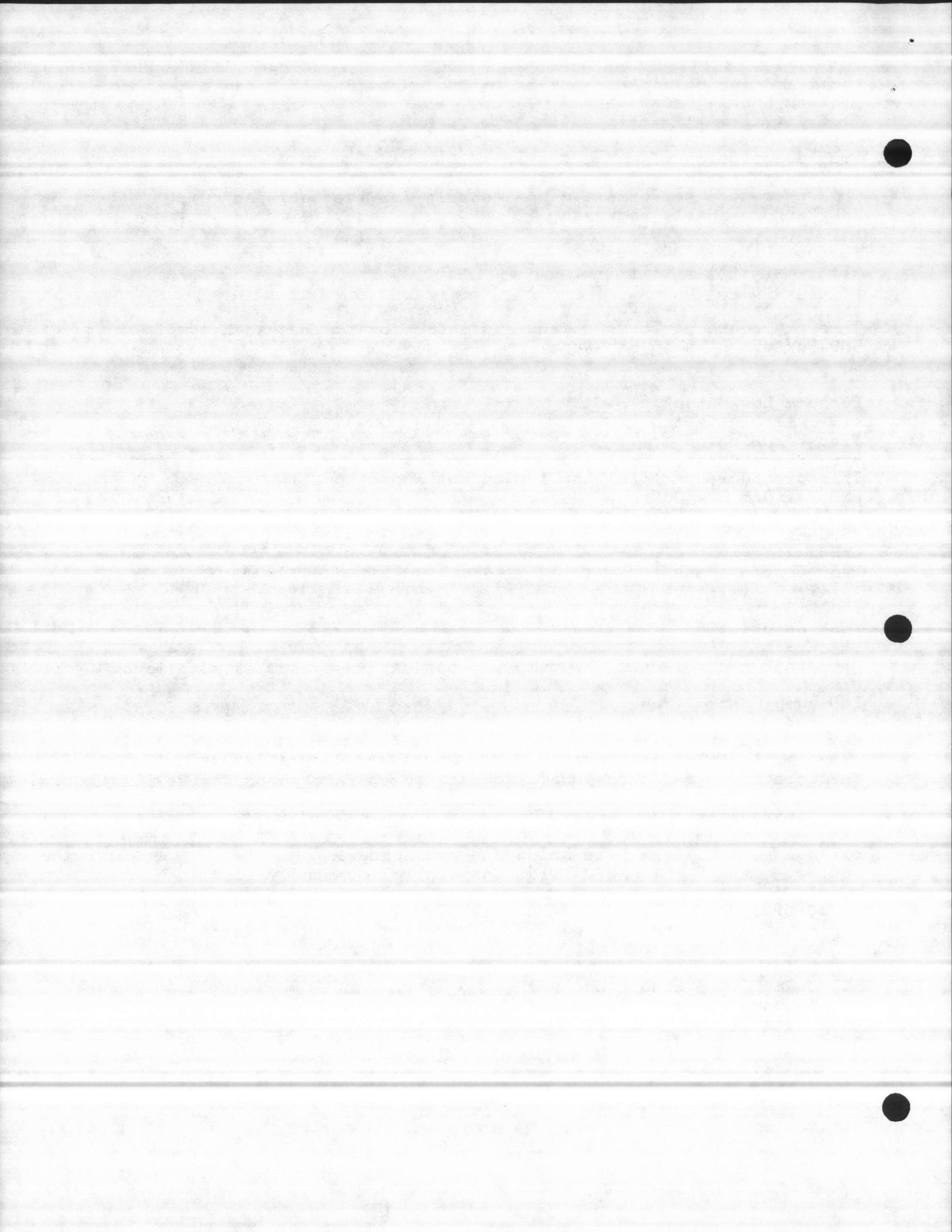
```
crt_num      => A byte holds the crt number.
```

```
control_command -> A word holds the control command value.
```

control\_command values :

```
command 1 - start pump  
command 2 - stop pump  
command 3 - fail reset  
command 4 - enable backwash  
command 5 - disable backwash
```

```
END Keyin_Start_Stop_Proc;
```



(1'3.6) MODULE NAME : CxxxxCTRL.Pyy

=====

DESCRIPTION : This module contains the control\_task which monitors  
===== all the controls in the system.

MODULE LOCAL PROCEDURES :

Set\_Pump\_Req:  
++++++

Proc description:

Sets the pump require.

END Set\_Pump\_Req;

Alt\_Pumps:  
++++++

Proc description :

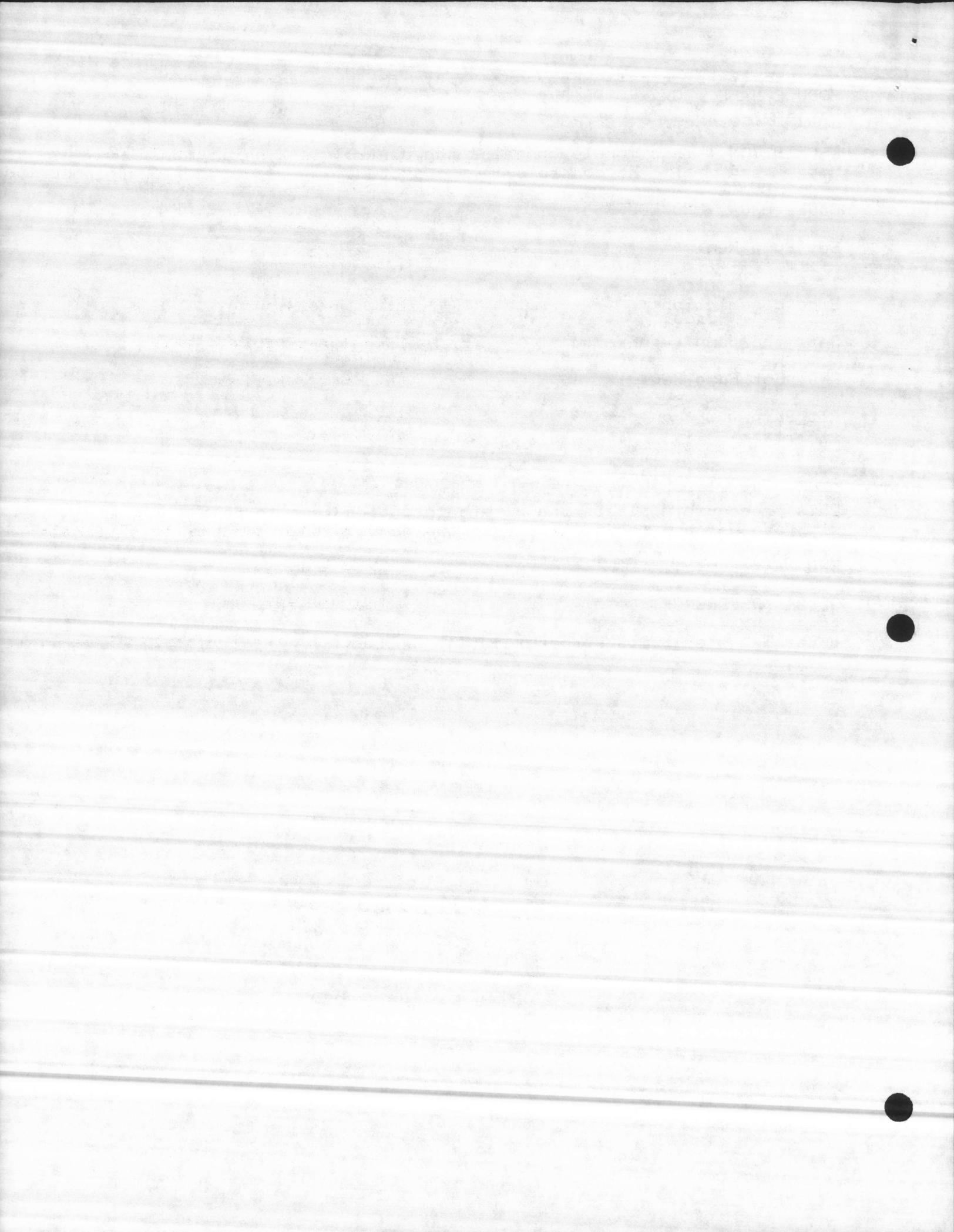
This procedure monitors the alternate option to alternate one pump  
with another pump or to alternate two pumps with other two.

END Alt\_Pumps;

GLOBAL REFERENCE :

=====

control & step\_pumps structures (See CxxxxGEXR.Pyy)



MODULE TASKS :

=====

Control\_Task:

++++++

/\* these flags are used to indicate whether a step's timer counter is  
counting the ON setpoint or OFF setpoint.  
0 = neither ON/OFF met, 1 = ON setpoint met, 2 = OFF setpoint met \*/

Task Algorithm :

-----

start of task initialization .

create input scan semaphore .

initialize timer flags .

setup list entry for defined scan time .

DO FOREVER;

wait at the list entry semaphore .

Loop1 go through all controls that are defined .

receive region for analog input definition .

get analog level for setpoint comparisons .

release analog region .

find level to be used .

falling level - find lowest level .

rising level - find highest level

do setpoint comparisons .

flag - 0 = single step, 1 = multiple step .

check ON setpoint of next step above current step .

check OFF setpoint .

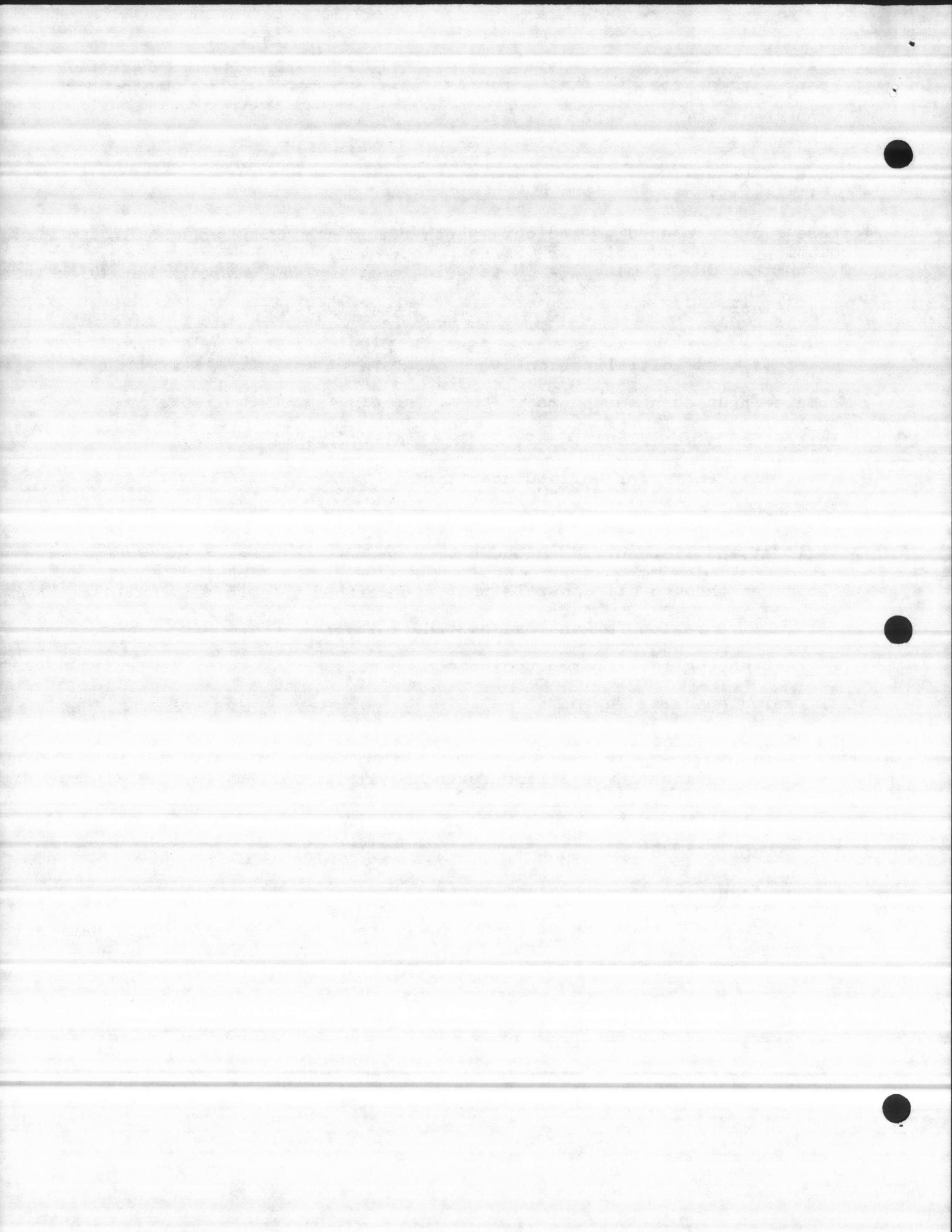
check for alternation .

release control region .

END; loop1 .

END; forever .

END Control\_Task;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 14

D I S P L A Y  
&  
G R A P H I C  
&  
T R E N D  
  
G E N E R A T O R

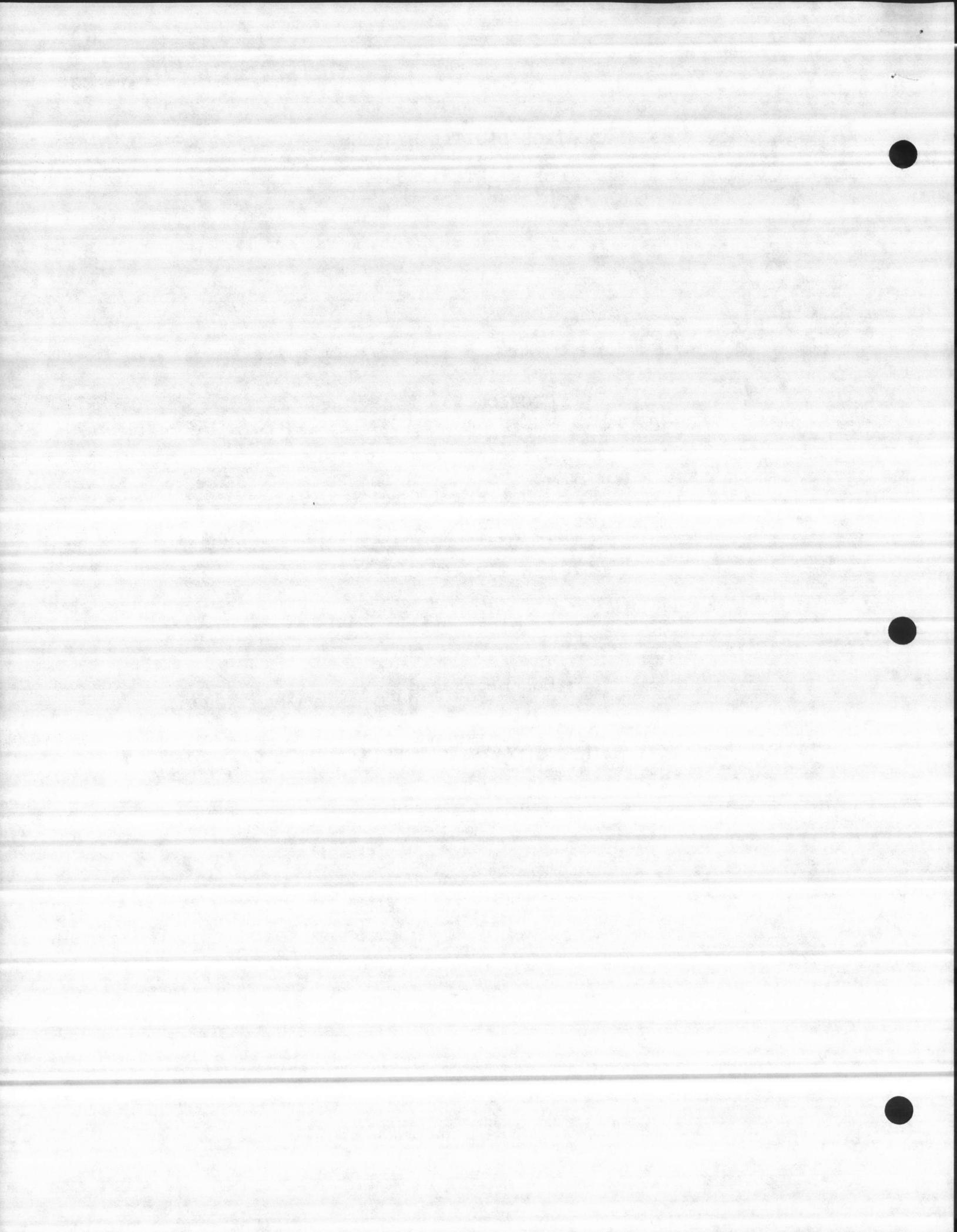
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

DISPLAY GENERATOR - written by Bob Ryan, Mohamed Fayad.

(14.1) MODULE NAME : CxxxxGGEN.Pyy

=====

DESCRIPTION :

=====

This module constructs all kinds of screen displays  
group displays  
graphics  
trends

EXTERNAL PROCEDURES :

=====

Menu\_Display:

```
PROCEDURE (crt_num, result, off_set,
           start_id_p, bytes_per_element, num_elements) WORD EXTERNAL;
DECLARE (crt_num) BYTE,
        (result, off_set) WORD,
        (bytes_per_element, num_elements) WORD,
        (start_id_p) POINTER;
END Menu_Display;
```

Clear\_Screen: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;

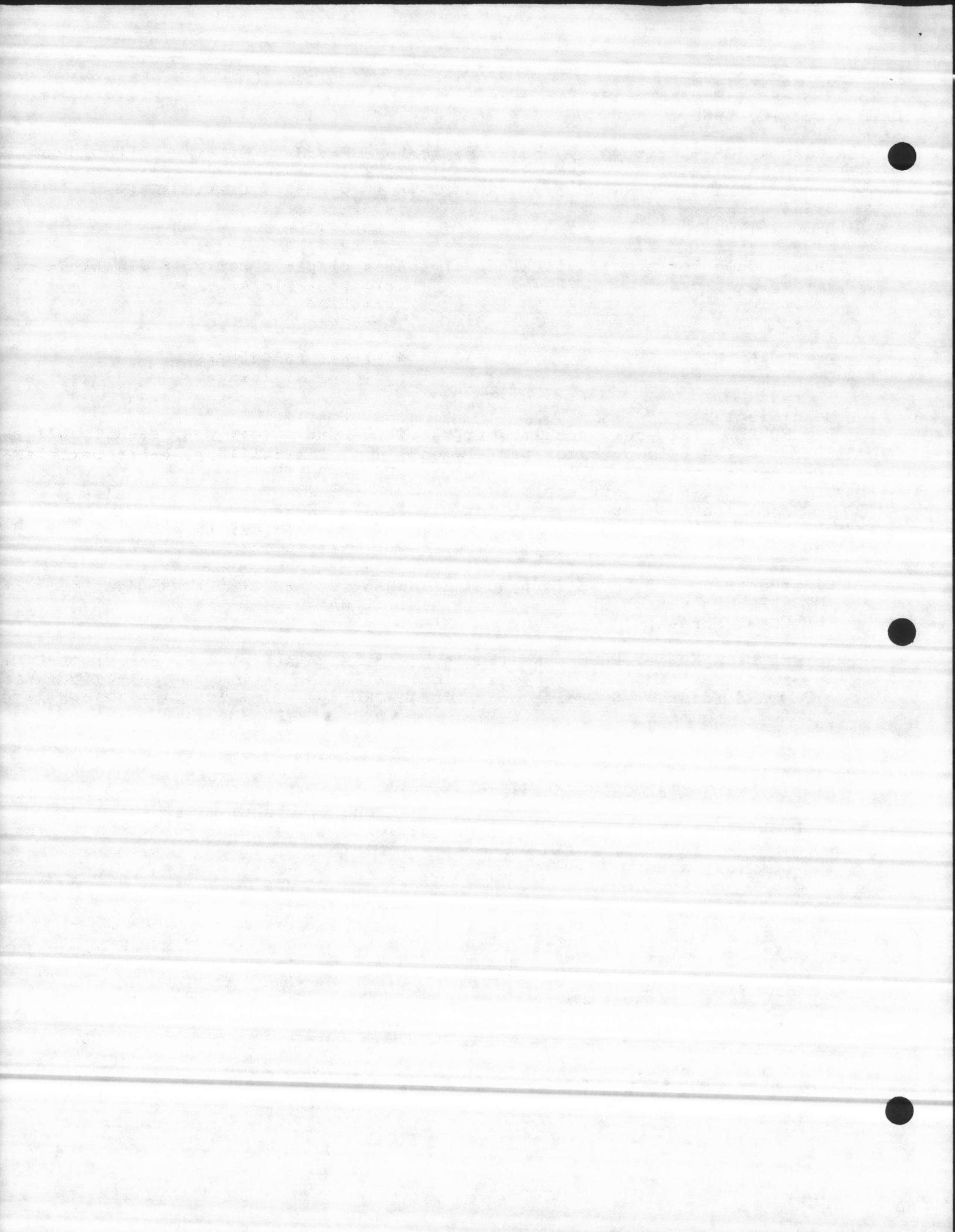
END Clear\_Screen;

```
Format_Disk_Error: PROCEDURE (buffer_t, message_str_p, disk_error,
                               disk_operation) EXTERNAL;
DECLARE (disk_error, disk_operation) WORD,
        (buffer_t) TOKEN,
        (message_str_p) POINTER;
END Format_Disk_Error;
```

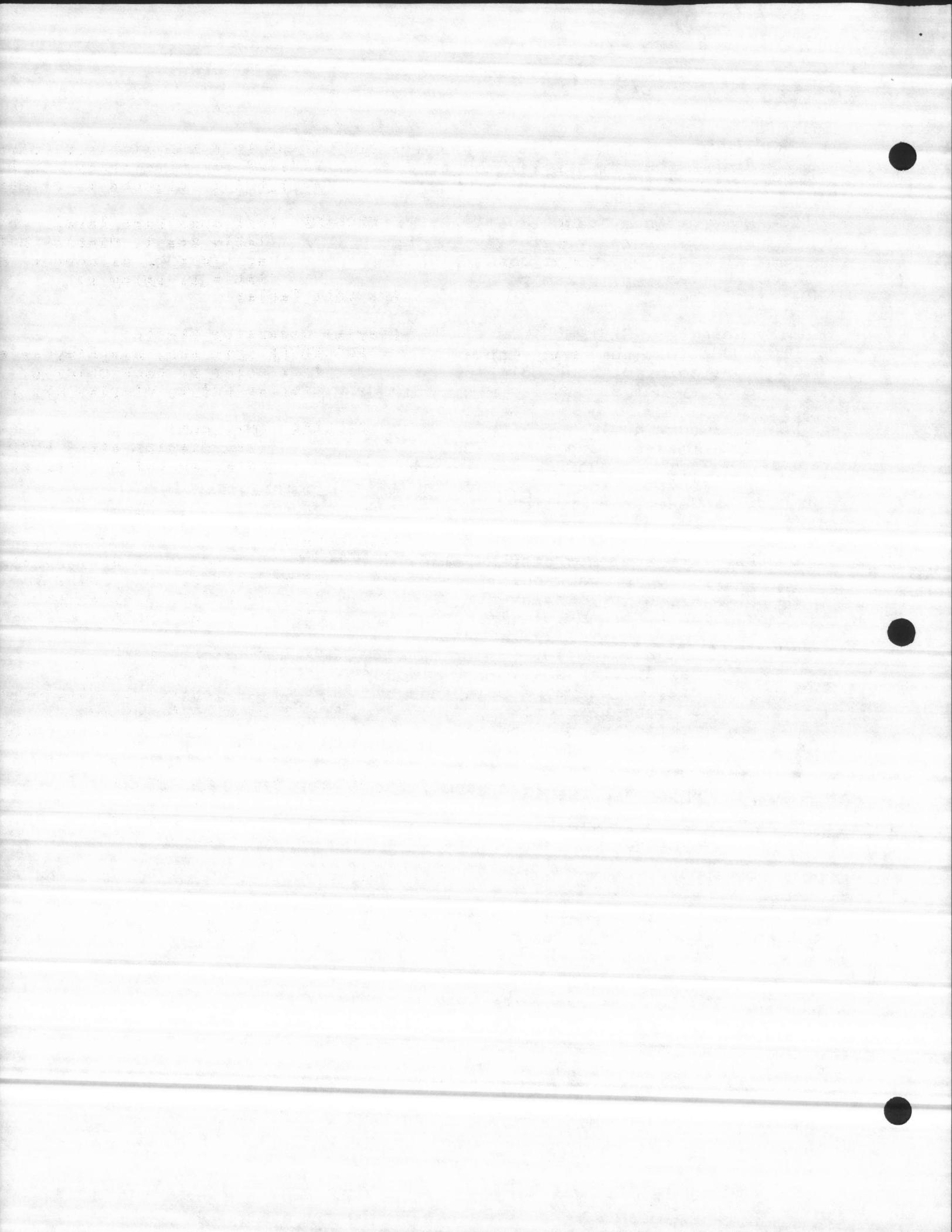
```
Display_Table: PROCEDURE(table_p, param_p, table_reg_t,
                           buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p) POINTER;
END Display_Table;
```

Get\_Password:

```
PROCEDURE (crt_num, table_p, table_reg_t,
           display_buf_t, display_mbx_t,
           ci_conn_t, ci_mbx_t) BYTE EXTERNAL;
DECLARE (crt_num) BYTE,
        (table_reg_t,
         display_buf_t,
         display_mbx_t,
         ci_conn_t,
         ci_mbx_t) TOKEN,
        (table_p) POINTER;
END Get_Password;
```



```
Edit_Table:  
    PROCEDURE (crt_num, table_p, param_p, table_reg_t,  
              display_buf_t, display_mbx_t, pass_level,  
              edit_buf_t, ci_conn_t, ci_mbx_t)      EXTERNAL;  
  
    DECLARE  (crt_num, pass_level)  BYTE,  
             (table_reg_t, display_buf_t, display_mbx_t,  
              edit_buf_t, ci_conn_t, ci_mbx_t)          TOKEN,  
             (table_p, param_p)                      POINTER;  
END Edit_Table;  
  
Display_Generator_Static:  
    PROCEDURE (crt_num, max_display_size, static_asc_p,  
              fore_color_p, back_color_p,  
              display_buf_t, display_mbx_t) EXTERNAL;  
  
    DECLARE  (crt_num)                  BYTE,  
             (max_display_size) WORD,  
             (display_buf_t, display_mbx_t)      TOKEN,  
             (static_asc_p, fore_color_p, back_color_p)  POINTER;  
END Display_Generator_Static;  
  
Edit_Screen:  
    PROCEDURE (crt_num, max_display_size, static_asc_p,  
              fore_color_p, back_color_p,  
              display_buf_t, display_mbx_t,  
              ci_conn_t, ci_mbx_t) EXTERNAL;  
  
    DECLARE  (crt_num)  BYTE,  
             (max_display_size) WORD,  
             (display_buf_t, display_mbx_t,  
              ci_conn_t, ci_mbx_t)          TOKEN,  
             (static_asc_p, fore_color_p, back_color_p)  POINTER;  
END Edit_Screen;  
  
Draw_Trend_Task: PROCEDURE (crt_num , trend_p) EXTERNAL;  
    DECLARE crt_num BYTE,  
            trend_p POINTER;  
END Draw_Trend_Task;  
  
GLOBAL REFERENCE :  
=====  
Check CxxxxGLOB.EXT,  
Check CxxxxGRPH.EXT
```



## MODULE DECLARATION :

=====

trend\_menu\_command\_count = A byte holds the value of four.

## trend\_menu\_prompt\_asc (4) STRUCTURE :

menu\_asc => 10 bytes hold the trend sub command string.

Cross Hair= Enters cross hair report mode

Edit Parm = Enters editor to allow parameter changes

Next = Display next page of screen parameters

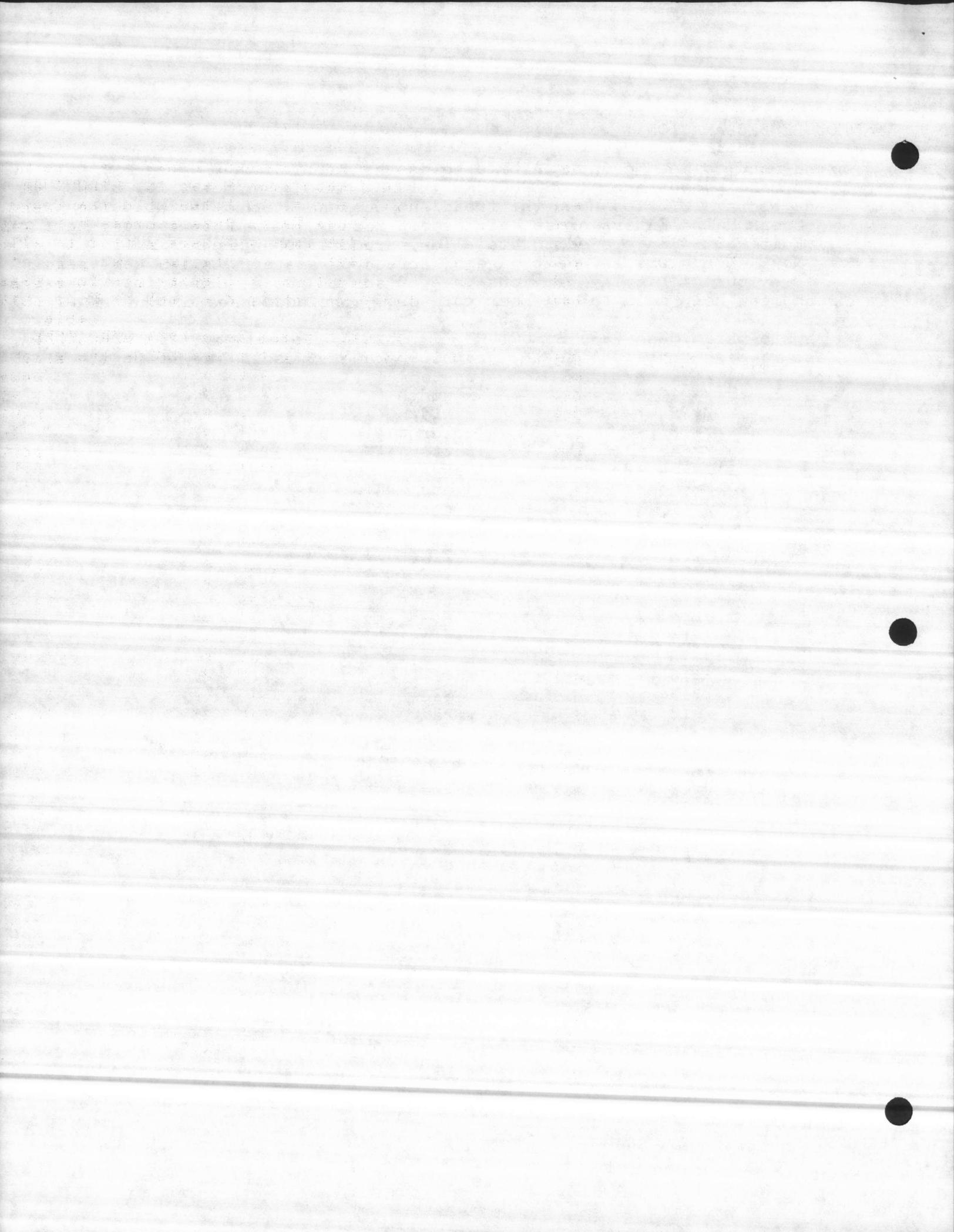
Previous = Display previous page of screen parameters

trend\_menu\_help\_desc\_tbl => Four pointers - the addresses of the help table.

## graphic\_param\_desc (38) STRUCTURE

asc => Eight bytes holds the graphic parameter ascii string.

/* 0 */	'unused ',
/* 1 */	'x width ',
/* 2 */	'y width ',
/* 3 */	'color # ',
/* 4 */	'pattern ',
/* 5 */	'bakgnd ',
/* 6 */	'#vecs ',
/* 7 */	'x coord ',
/* 8 */	'y coord ',
/* 9 */	'width ',
/*10 */	'height ',
/*11 */	'fill ',
/*12 */	'x tip ',
/*13 */	'y tip ',
/*14 */	'radius ',
/*15 */	'angle 1 ',
/*16 */	'angle 2 ',
/*17 */	'x rad ',
/*18 */	'y rad ',
/*19 */	'option ',
/*20 */	'rotate ',
/*21 */	'# sides ',
/*22 */	'degrees ',
/*23 */	'size ',
/*24 */	'offset ',
/*25 */	'out top ',
/*26 */	'old val ',
/*27 */	'#digits ',
/*28 */	'scale ',
/*29 */	'field ln',
/*30 */	'length ',
/*31 */	'interval',
/*32 */	'max ',
/*33 */	'min ',
/*34 */	'x inter ',
/*35 */	'y inter ',
/*36 */	'# sides ',
/*37 */	'# bars ',



**graphic\_desc\_xref (28) STRUCTURE**

index => Ten bytes holds the x reference index to each param value for constructing a graph.

```
/* Ø = Undefined */          0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
/* 1 = VALVE */            23, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 2 = CHECK VALV */        23, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 3 = CNTRL VALV */       23, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 4 = MOTOR */             14, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 5 = PUMP */              14, 24, 25, 20, 0, 0, 0, 0, 0, 0,
/* 6 = SCREW PUMP */        37, 23, 20, 0, 0, 0, 0, 0, 0, 0,
/* 7 = PROPELLER */         14, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 8 = GATE */               23, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 9 = BAR SCREEN */        37, 23, 9, 20, 0, 0, 0, 0, 0, 0,
/* 10 = ARROW HEAD */       23, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 11 = STICK MAN */        23, 20, 0, 0, 0, 0, 0, 0, 0, 0,
/* 12 = BAR GRAPH */        1, 2, 27, 26, 0, 0, 0, 0, 0, 0,
/* 13 = BRUSH SIZE */       1, 2, 0, 0, 0, 0, 0, 0, 0, 0,
/* 14 = GRAPHIC C */        3, 0, 0, 0, 0, 0, 0, 0, 0, 0,
/* 15 = FILL TYPE */        4, 5, 0, 0, 0, 0, 0, 0, 0, 0,
/* 16 = VECTOR TYP */       4, 0, 0, 0, 0, 0, 0, 0, 0, 0,
/* 17 = ABS VECTOR */        6, 7, 8, 7, 8, 7, 8, 7, 8, 0,
/* 18 = REL VECTOR */        6, 7, 8, 7, 8, 7, 8, 7, 8, 0,
/* 19 = POL VECTOR */        6, 30, 22, 30, 22, 30, 22, 30, 22, 0,
/* 20 = DRAW BOX */          9, 10, 11, 12, 13, 0, 0, 0, 0, 0,
/* 21 = DRAW ARC */          14, 15, 16, 19, 0, 0, 0, 0, 0, 0,
/* 22 = REL ARC */           14, 20, 19, 0, 0, 0, 0, 0, 0, 0,
/* 23 = CIRCLE */            14, 11, 0, 0, 0, 0, 0, 0, 0, 0,
/* 24 = ABS EL ARC */        17, 18, 15, 16, 20, 36, 19, 0, 0, 0,
/* 25 = REL EL ARC */        17, 18, 22, 20, 36, 19, 0, 0, 0, 0,
/* 26 = ELIPSE */             17, 18, 20, 36, 11, 0, 0, 0, 0, 0);
```

**graphic\_menu\_command\_count** => A byte holds the number of sub commands under graphic command.

**graphic\_menu\_prompt\_asc (9) STRUCTURE**

menu\_asc -> Ten bytes holds the graphic sub command ascii string.

Cross Hair = Enters cross hair report mode

Edit Parm = Enters editor to allow parameter changes

Add = Adds a new symbol to the end of the list

Delete = Delete a symbol from the list

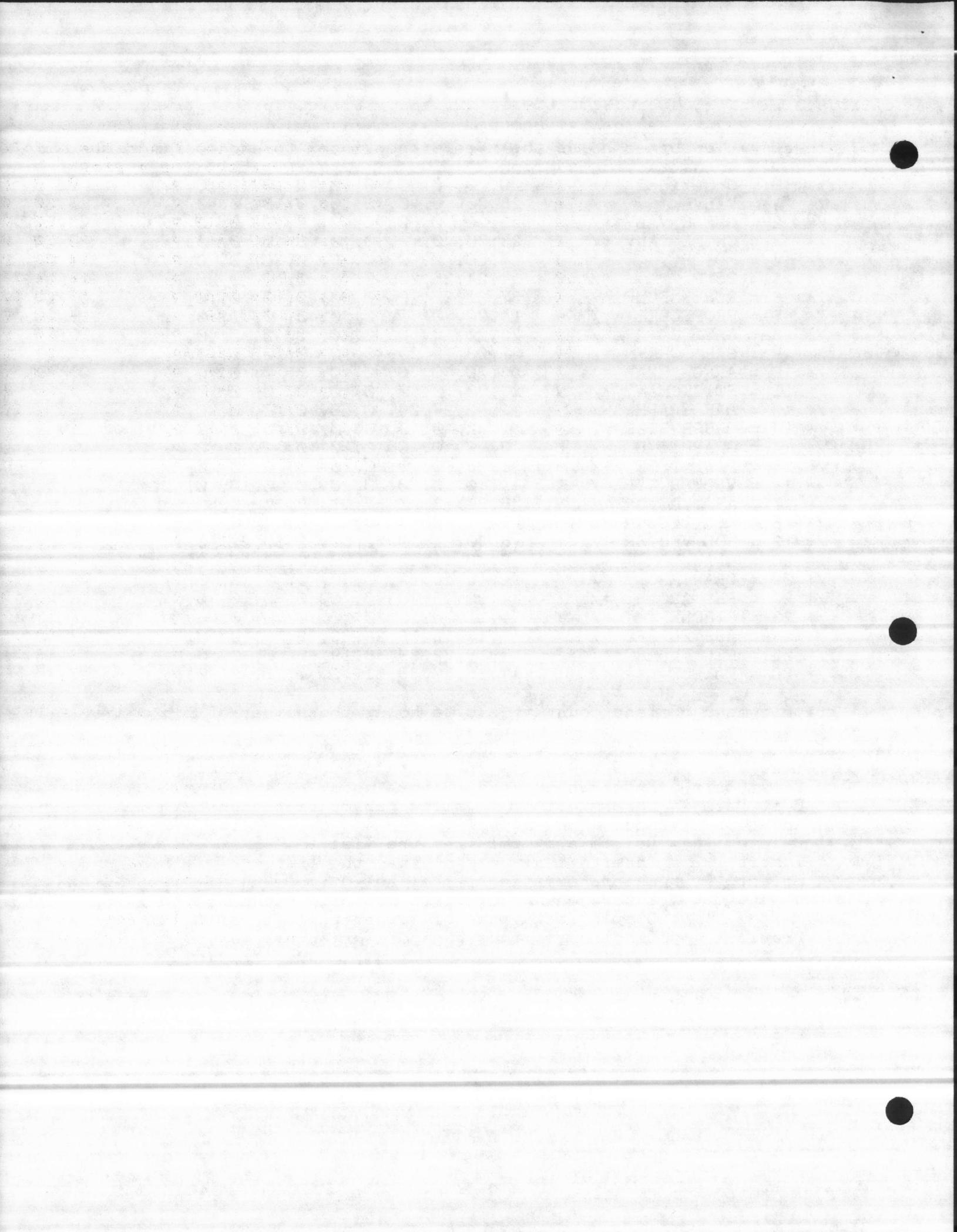
Insert = Inserts a new symbol into the list

Select = Enters existing symbol selection menu

Change C = Enters change of state selection menu

Next = Display next page of screen parameters

Previous = Display previous page of screen parameters



```
graphic_menu_help_desc_tbl => Nine pointers - The addresses to the
                               help table.

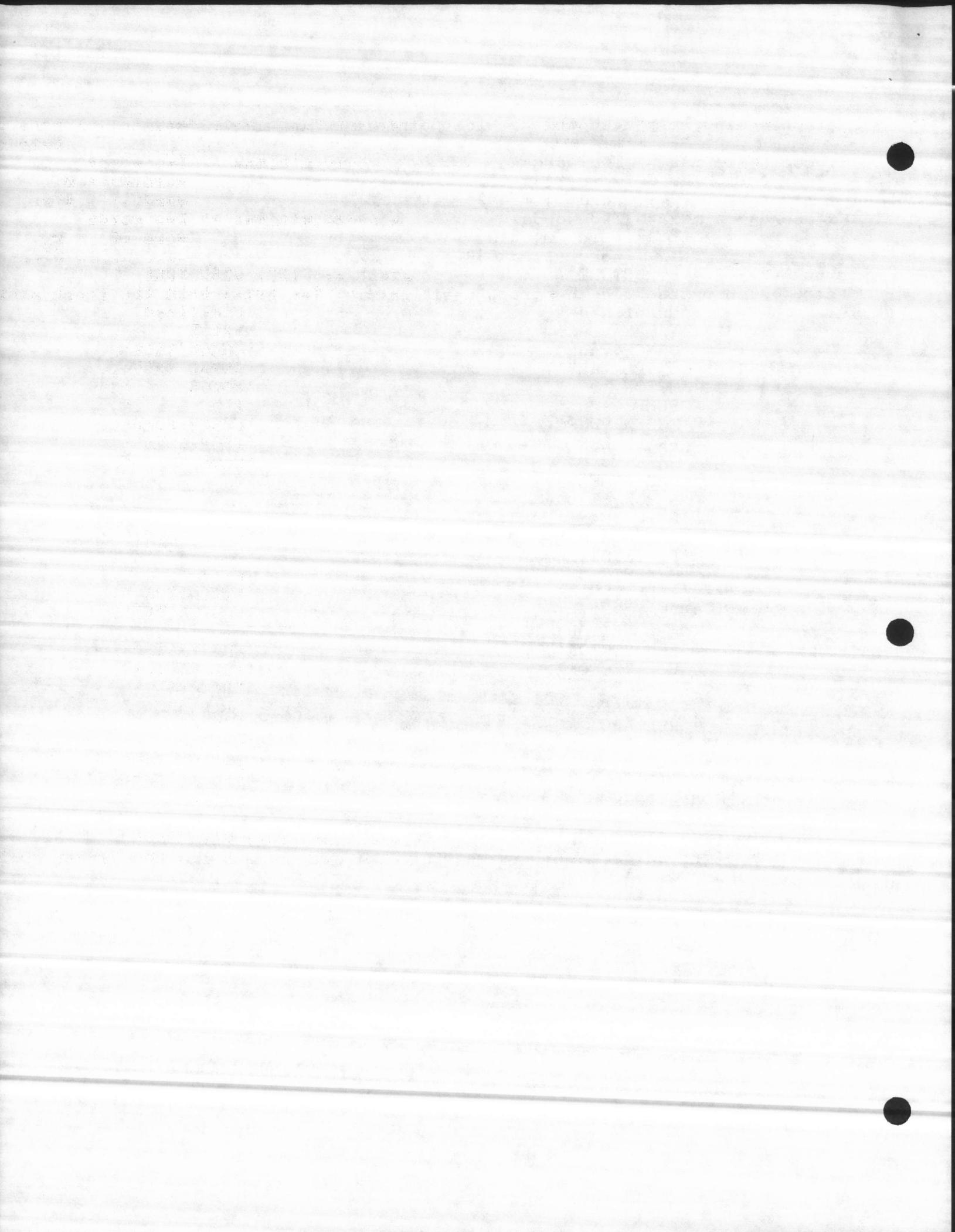
y_coord_min_max => Two words
                     word(0) = 0.
                     word(1) = 479.

x_coord_min_max => Two words
                     word (0) = 0.
                     word (1) = 639.

graph_asc (27) STRUCTURE
str => Ten bytes hold the graph ascii string
'Undefined',
'VALVE',
'CHECK VALV',
'CNTRL VALV',
'MOTOR',
'PUMP',
'SCREW PUMP',
'PROPELLER',
'GATE',
'BAR SCREEN',
'ARROW HEAD',
'STICK MAN',
'BAR GRAPH',
'BRUSH SIZE',
'GRAPHIC C',
'FILL TYPE',
'VECTOR TYP',
'ABS VECTOR',
'REL VECTOR',
'POL VECTOR',
'DRAW BOX',
'DRAW ARC',
'REL ARC',
'CIRCLE',
'ABS EL ARC',
'REL EL ARC',
'ELIPSE',
```

## graph\_rmx (27) STRUCTURE :

This structure holds the 27 rmx strings for graphics.



twenty\_seven => A word holds 27.

graph\_rmx\_tbl => 27 pointers - the addresses of the graphic rmx string table.

graph\_edit\_tbl => Two pointer - the addresses of the graph edit table.

default\_graph STRUCTURE:

symbol\_name => 10 bytes hold the symbol point id (User symbol name).  
function => A word holds the function value.  
color => A word holds the color index.  
x\_coord => A word holds the X coord.  
y\_coord => A word holds the Y coord.  
parameters (10) => Ten words holds the graph parameter value  
change\_flag => A word holds the value of change\_flag.  
point\_id\_type (4) => Four bytes hold the point\_id Types.  
point\_id\_index (4) => Four words hold the point id indexes.  
change\_state (4) => Four words hold the change statuses.  
change\_color (4) => Four words holds the change colors.

default\_trend STRUCTURE :

x\_coord => A word holds the X coord.  
y\_coord => A word holds the Y coord.  
height => A word holds the graph height.  
width => A word holds the trend width.  
int\_type => A word holds the value of the int\_type.  
scale => A word holds the scale factor.  
min\_x => A dword holds minimum X  
max\_x => A dword holds maximum X  
min\_y => A dword holds minimum Y  
max\_y => A dword holds maximum Y  
window\_flag => A word holds the value of the window flag.  
h\_file\_name (30) => Thirty bytes hold the historical file name.  
point\_id\_type (4) => Four bytes hold four point id types.  
point\_id\_index (4) => Four words hold four point id indexes.  
min\_max\_avg (4) => Four words hold four min max averages.

graph\_parameter\_crt\_table (21) STRUCTURE

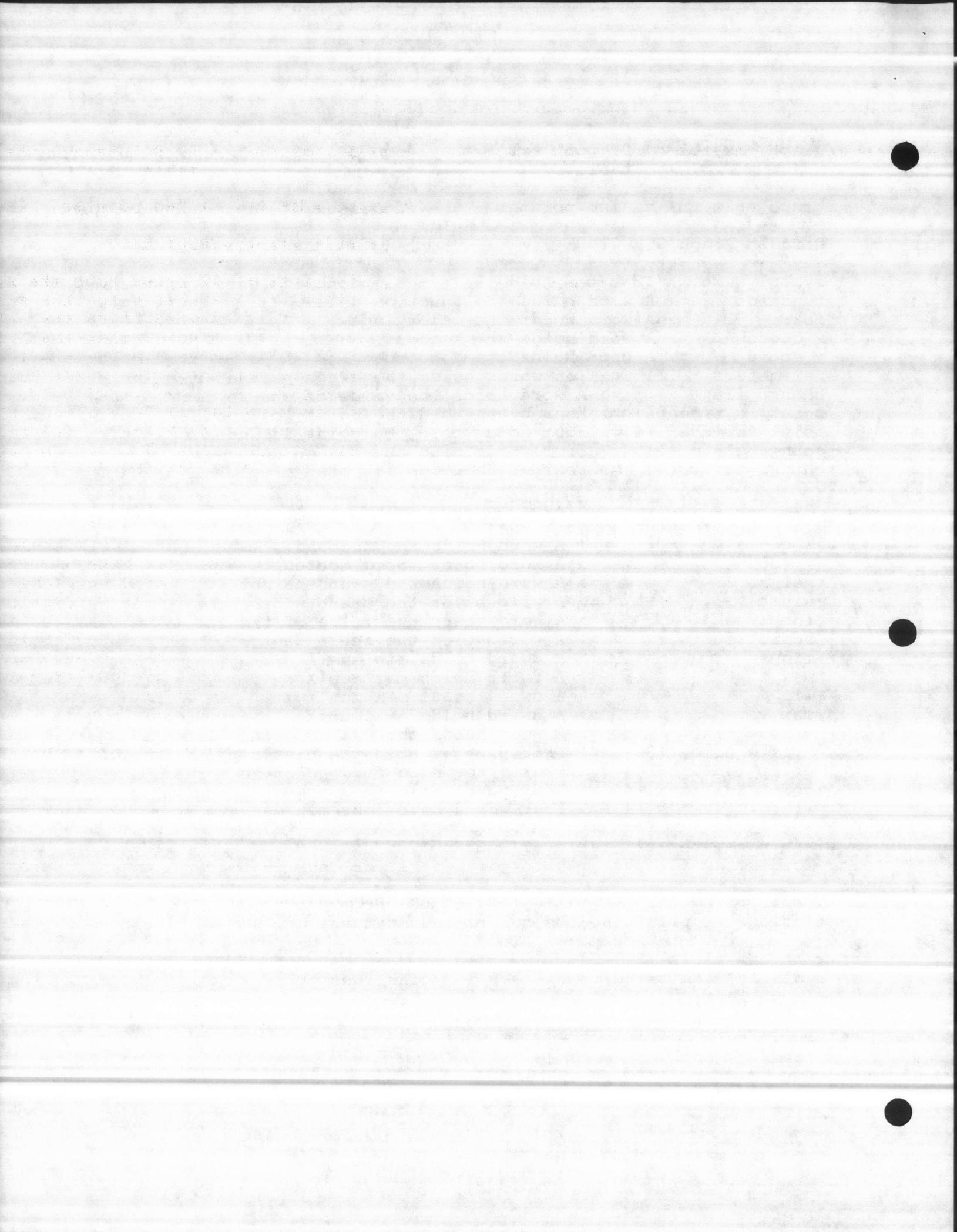
To display all the graphic parameters for construct graph.

dynamic\_parameter\_crt\_table (13) STRUCTURE :

To hold four point id, its color, and statuses.

trend\_crt\_table (21) STRUCTURE :

To display the trend parameters for construct trend.



```
trend_desc =>
  X coord
  Y coord
  Height
  Width
  Interval Type
  Display Scale
  Minimum Time
  Maximum Time
  Minimum Value
  Maximum Value
  Current window ?
  Historical File
  Pen 1 Point Id
  Pen 2 Point Id
  Pen 3 Point Id
  Pen 4 Point Id
```

```
dynamic_desc =>
  Point ID :
  Color :
  Condition:
```

Name	Symbol	Color	Coord	Links
------	--------	-------	-------	-------

cons\_crt\_menu\_command\_count => A byte holds the value 6.

cons\_crt\_menu\_prompt\_asc (6) STRUCTURE :

```
menu_asc => Ten bytes holds the construct screen command string.
Configure = (declared globally)
Parameters= (declared globally)
Edit      = (declared globally)
Next      = (declared globally)
Graphics  = Enters the graphics display generator
Trending   = Enters the video trend questioner
```

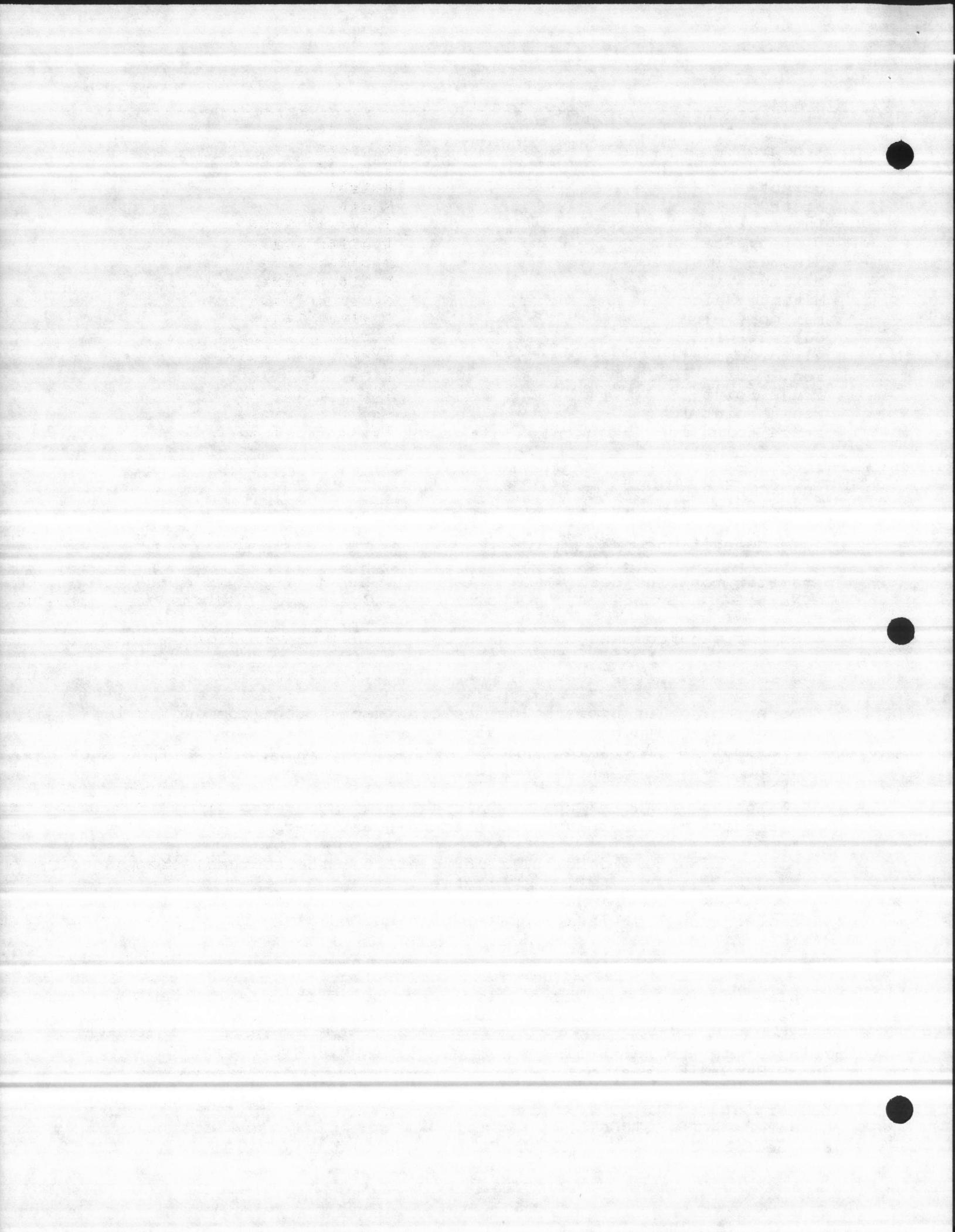
cons\_crt\_menu\_help\_desc\_tbl => 6 pointers - the addresses to the help table.

parameter\_asc\_p1 => The parameter page sub heading

```
edit_flag :           => Three bytes (one for each crt) each holds the
                           value of the edit flag.
display_page        => Three bytes, each holds the display page number.
display_page_limit => Three bytes, each holds the display page limit.
```

file\_parameters BASED disk\_param\_seg\_t STRUCTURE :
See DISK SYSTEM section.

```
header_info (8) STRUCTURE &
header_data (8) STRUCTURE
See Section 17
```



exception => Three words, each holds the exception condition.  
crt\_update\_sem\_t=> Three crt update semaphore tokens, each used to  
create the crt update semaphore.

group\_disp\_heading => 29 bytes hold the group display heading string.  
'Construct Group Point Display'.

## PUBLIC PROCEDURES :

=====

## Keyin\_Construct\_Sub\_Menu:

+++++  
crt\_num => A byte holds the number of the crt to display on.  
old\_echo\_count=> A byte holds the old\_echo\_count.  
result => A word used as an index.

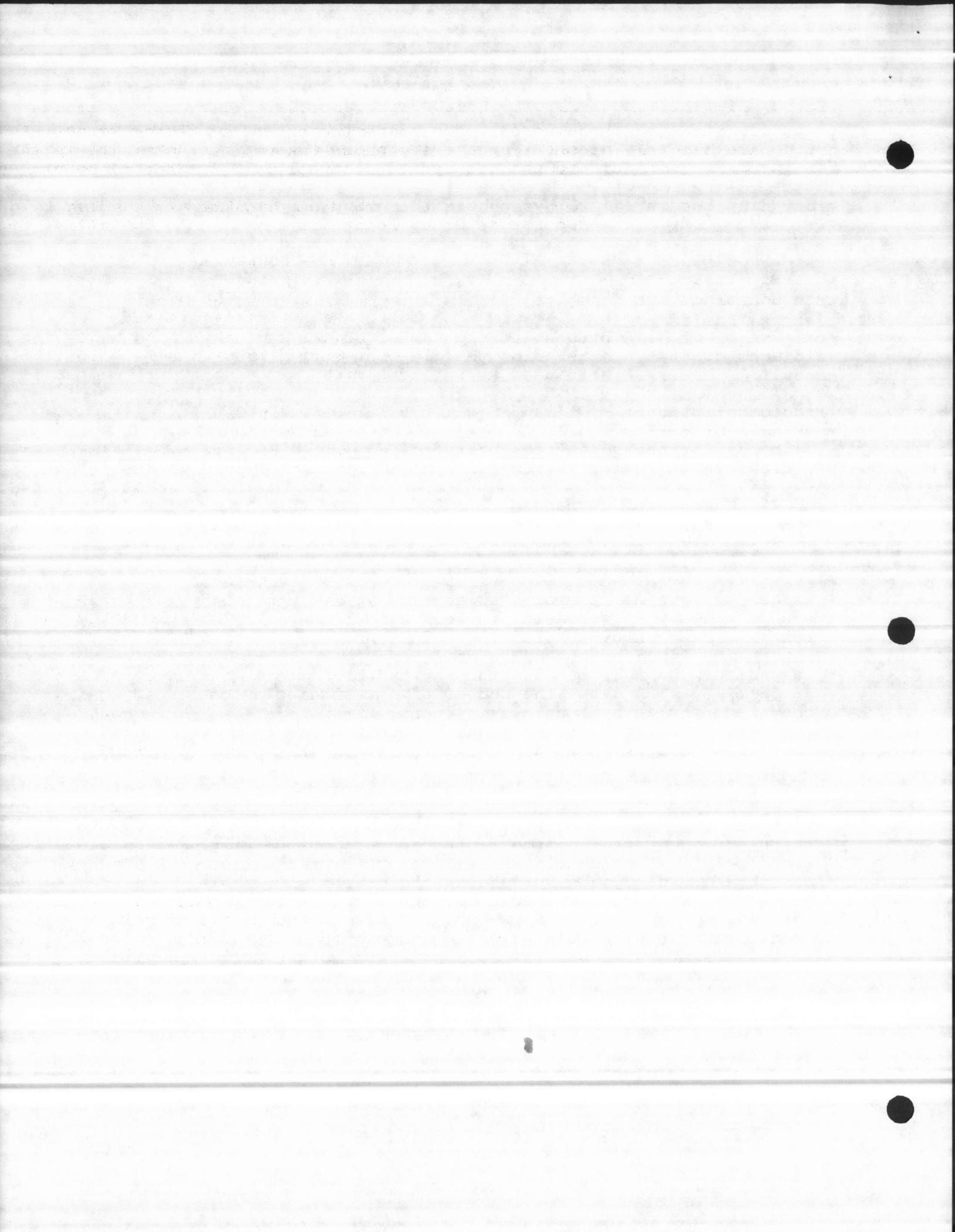
## Proc description :

- 1. Checks for the supervisor password.  
2. displays the displays sub menu.

END Keyin\_Construct\_Sub\_Menu;

## Access\_CRTDP\_File:

+++++  
access\_type => A byte holds an access type value.  
row => A byte holds the row number.  
col => A byte holds the column number.  
num => A byte used as an index.  
tag\_number => A byte holds the tag number.  
crt\_num => A byte, the number of crt to display on.  
count => A byte used as a count.  
fore\_color\_size => A word holds the size of the foeground color.  
back\_color\_size => A word holds the size of the background size.  
trend\_size => A word holds the trend size.  
graph\_size => A word holds the graph size.  
fore\_color\_p => A pointer - the address of the foreground color.  
back\_color\_p => A pointer - the address of the background color.  
graph\_p => A pointer - the address of the graph.  
trend\_p => A pointer - the address of the trend.  
static\_asc\_size => A word holds the static ascii size.  
screen\_param\_size => A word holds the screen parameter size.  
edit\_level\_p => A pointer - the address to the edit\_level  
static\_asc\_p => A pointer - the address to static ascii.  
screen\_param\_p => A pointer - the address to the screen parameter.



```
screen_param BASED screen_param_p (601) STRUCTURE:  
point_id => 10 bytes hold the report id.  
struc_type => A byte holds the structure type value.  
struc_index => A word holds the structure index.  
field => A byte holds the field number.  
level => A byte holds the level number.  
  
error_msg_p => A pointer - the address of the error message.  
  
graph BASED graph_p (500) STRUCTURE :  
and  
trend BASED trend_p (5) STRUCTURE :  
See MODULE DECLARATION.  
  
screen_param BASED screen_param_p (123) STRUCTURE:  
point_id => 10 bytes hold the group id.  
struc_type => A byte holds the structure type value.  
struc_index => A word holds the structure index.  
field => A byte holds the field number.  
color => A byte holds the index of the color.  
level => A byte holds the level number.  
row => A byte holds the row number.  
col => A byte holds the column number.  
  
crt_file_name => 15 bytes - the crt file name of the format  
CRTDP000X.PARM
```

Proc description :

-----  
This procedure reads & saves the display disk file , when you  
construct a display.

END Access\_CRTDP\_File;

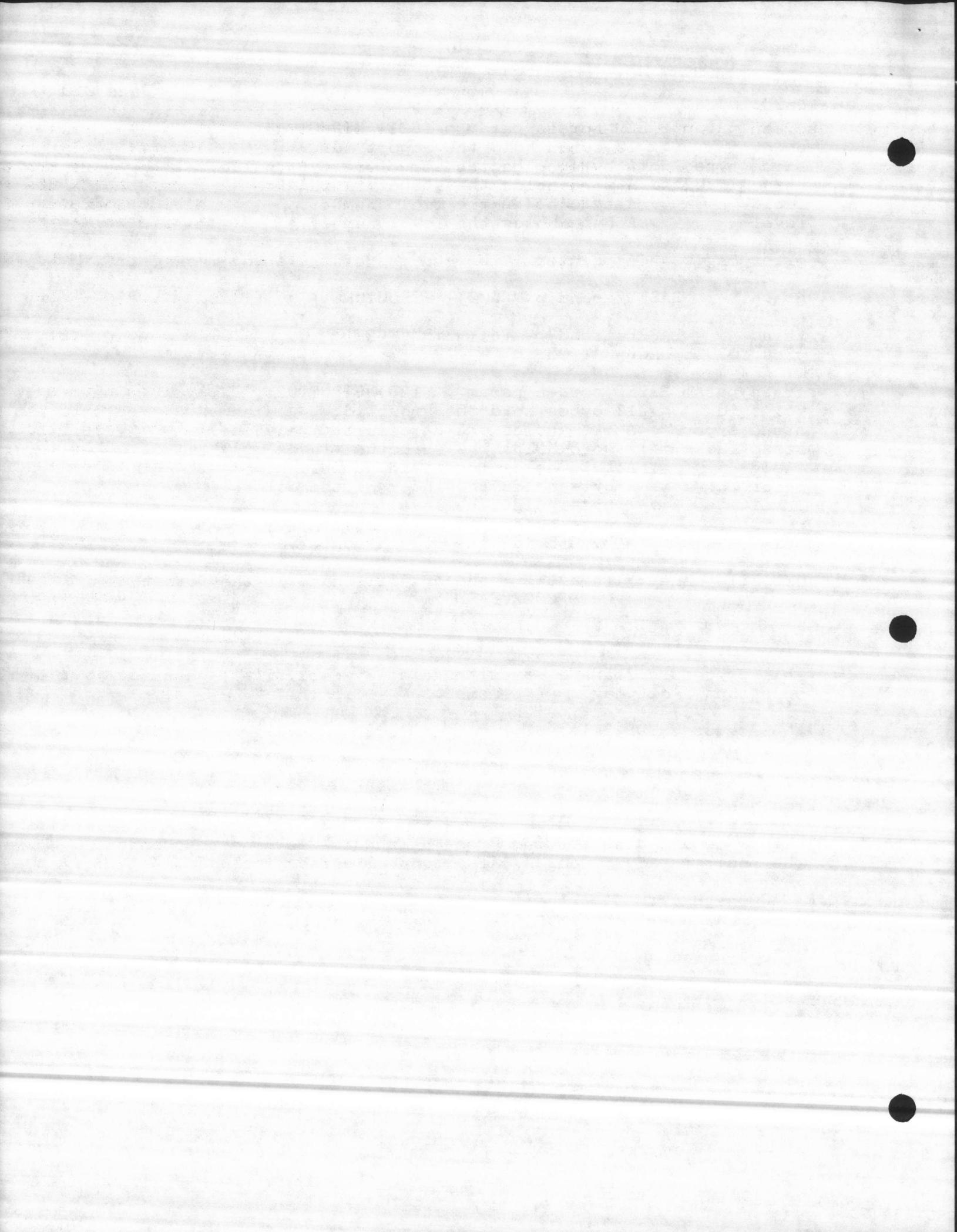
Cross\_Hair:

++++++

Proc description :

-----  
This procedure displays the Cross Hair Which is used to change  
the position X & Y coords. Also control cursor included.

END Cross\_Hair;



```
Display_Graph_Parameters:  
++++++
```

```
/* Type determines what crt table will be displayed */  
/* 0 - Graphic symbols */  
/* 1 - Dynamic point ids */  
Proc description :
```

```
-----  
This procedure displays the graph parameters when you construct a  
graph.
```

```
END Display_Graph_Parameters;
```

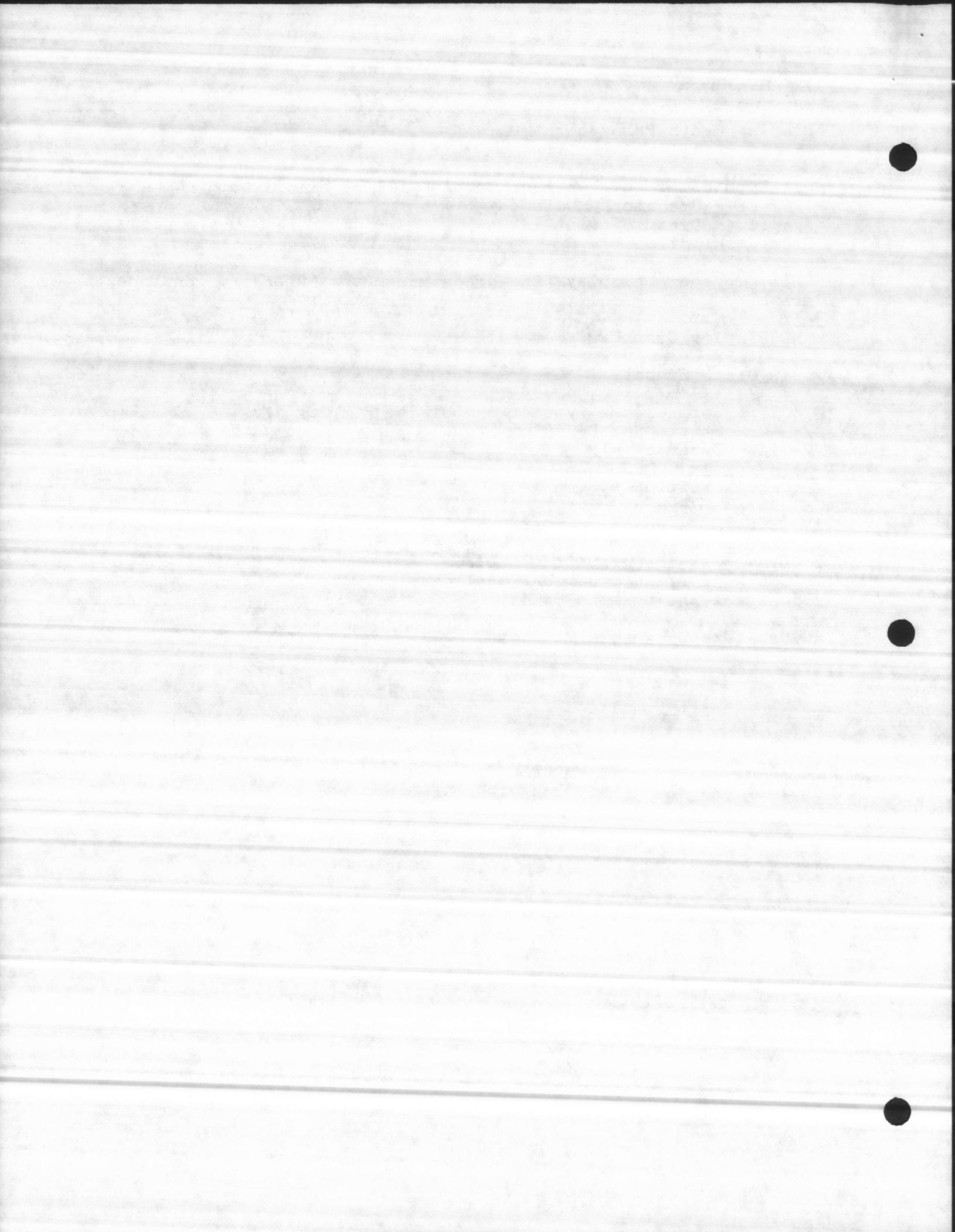
```
Draw_Graph_Symbols:  
++++++
```

```
/* Type defines the type of symbols to display */  
/* 0 - Static and Dynamic */  
/* 1 - Dynamic only */
```

```
Proc description :
```

- ```
-----  
1. SETS THE X PLANE TO BLACK AND BLINKING WHEN UNACKNOWLEDGE ALARMS  
EXIST.  
2. Draws all the graphic symbols.  
3. Checks for the dynamic symbol display modes.  
4. changes the dynamic color.
```

```
END Draw_Graph_Symbols;
```



```
Graphic_Generator_Editor:
```

```
++++++
```

```
Proc description :
```

- ```
-----  
1. creates menu data segments for the crt table, help table, description  
table, and menu prompt.  
2. Displays the current parameters table.  
3. monitors the graphic menu.  
4. generates graphic symbols.  
5. deletes the menu data segments.
```

```
END Graphic_Generator_Editor;
```

```
Trend_Generator_Editor:
```

```
++++++
```

```
Proc description :
```

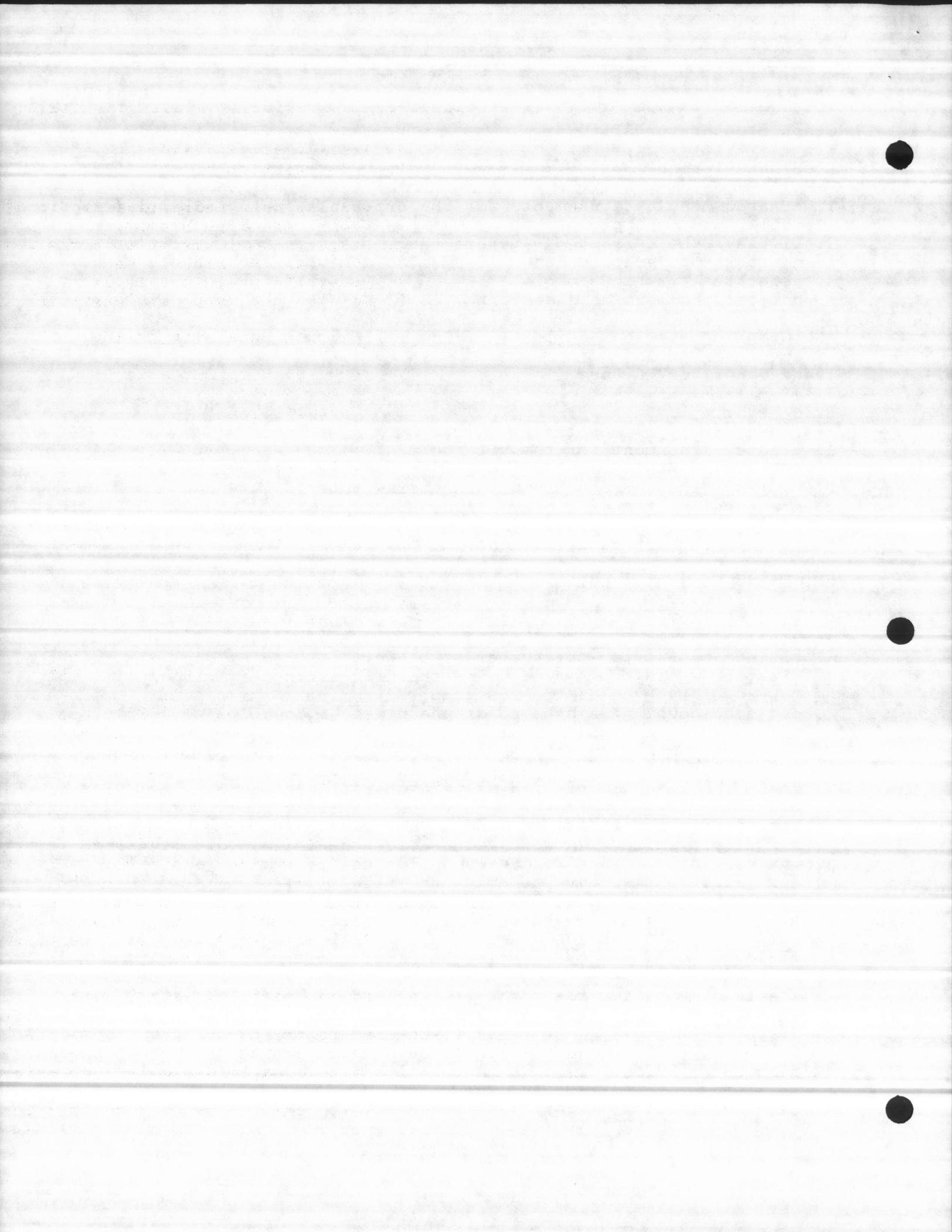
- ```
-----  
1. creates a segment for the crt table.  
table, and menu prompt.  
2. Displays the current trend parameters table.  
3. monitors the trending menu.  
4. generates graphic symbols.  
5. deletes the crt table segment.
```

```
END Trend_Generator_Editor;
```

```
Construct_Display:
```

```
++++++
```

```
crt_num => A byte, the number of the crt to display on.  
disk_error_flag => A byte holds the disk_error_flag value.  
row => A byte holds the row number.  
old_echo_count => A byte holds the old echo count.  
count => A word used as a count.  
field => A word used as a field number.  
left_index => A word used as parameter left index.  
right_index => A word used as parameter right index.  
off_set => A word value holds an off_set value.  
remaining_units => A word used as an index.  
max_display_size => A word value holds the maximum display size.  
max_page_size => A word value holds the maximum page size.  
file_size => A word value holds the file size.  
result => A word used as an index.  
graph_count => A word holds the graph count.  
temp_word => A word used as a temp. location.  
field_asc => Three bytes hold the field ascii.  
  
static_disk_buffer_seg_t => A token used to create the static disk  
buffer segment.  
graph_seg_t => A token used to create the graph segment.  
trend_seg_t => A token used to create the trend segment.  
fore_color_buffer_seg_t => A token used to create fore color buffer  
segment.  
back_color_buffer_seg_t => A token used to create the back color  
buffer segment.
```



edit\_level => Two bytes hold the edit level  
static\_asc BASED static\_disk\_buffer\_seg\_t => 8713 bytes.  
  
table\_p => A pointer - the address to the crt display table.  
  
crt\_table\_seg\_t => A token used to create the crt table segment.  
  
screen\_param\_seg\_t => A token used to create the screen parameter  
segment.

Proc description :

- 
1. displays construct\_display.
  2. calls stop\$update.
  3. initializes the parameter crt table and creates a segment for it.
  4. paging definition.
  5. Creates the data segment.
  6. defines the file\_size & max\_page\_size equal to 48 \* 80.
  7. creates the screen parameter segment, the static\_disk\_buffer\_segment,
  8. checks for display disk file if it exists then read it otherwise  
creates a new report file.
  9. displays the construct menu driver.

configure      parameter      Edit      Next      Graphic      Trending

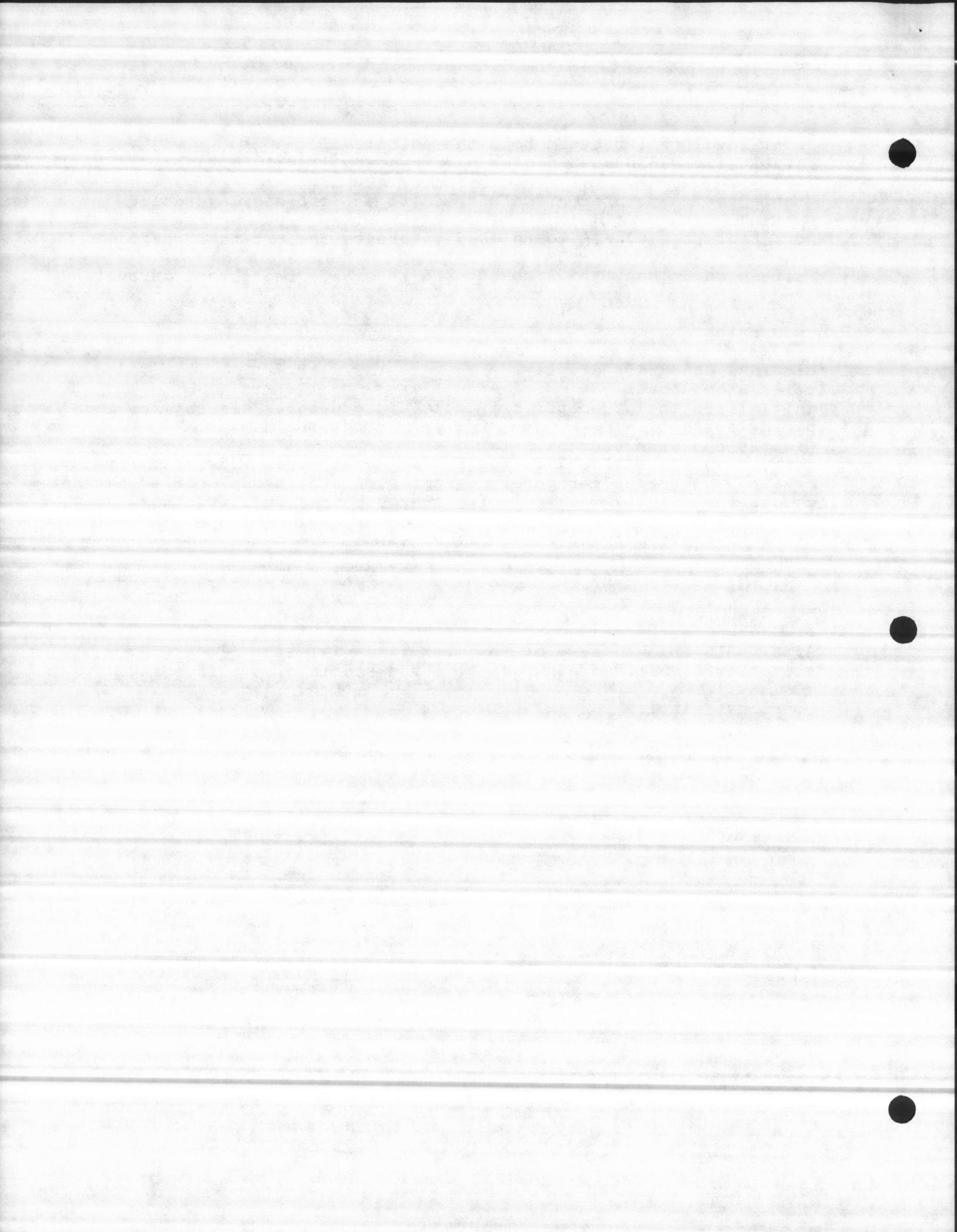
Note That :

=====

After edit an auto save of the disk file will take place.

10. deletes crt table, screen parameter and static disk buffer segments

END Construct\_Display;



(14.2) MODULE NAME : CxxxxDISG.Pyy

=====

DESCRIPTION :

=====

This module constructs daily, weekly, monthly, yearly and periodic reports.

EXTERNAL PROCEDURES :

=====

Menu\_Display:

```
PROCEDURE (crt_num, result, off_set,
          start_id_p, bytes_per_element, num_elements) WORD EXTERNAL;
DECLARE (crt_num) BYTE,
        (result, off_set) WORD,
        (bytes_per_element, num_elements) WORD,
        (start_id_p) POINTER;
```

END Menu\_Display;

```
Format_Disk_Error: PROCEDURE (buffer_t, message_str_p, disk_error,
                               disk_operation) EXTERNAL;
DECLARE (disk_error, disk_operation) WORD,
        (buffer_t) TOKEN,
        (message_str_p) POINTER;
END Format_Disk_Error;
```

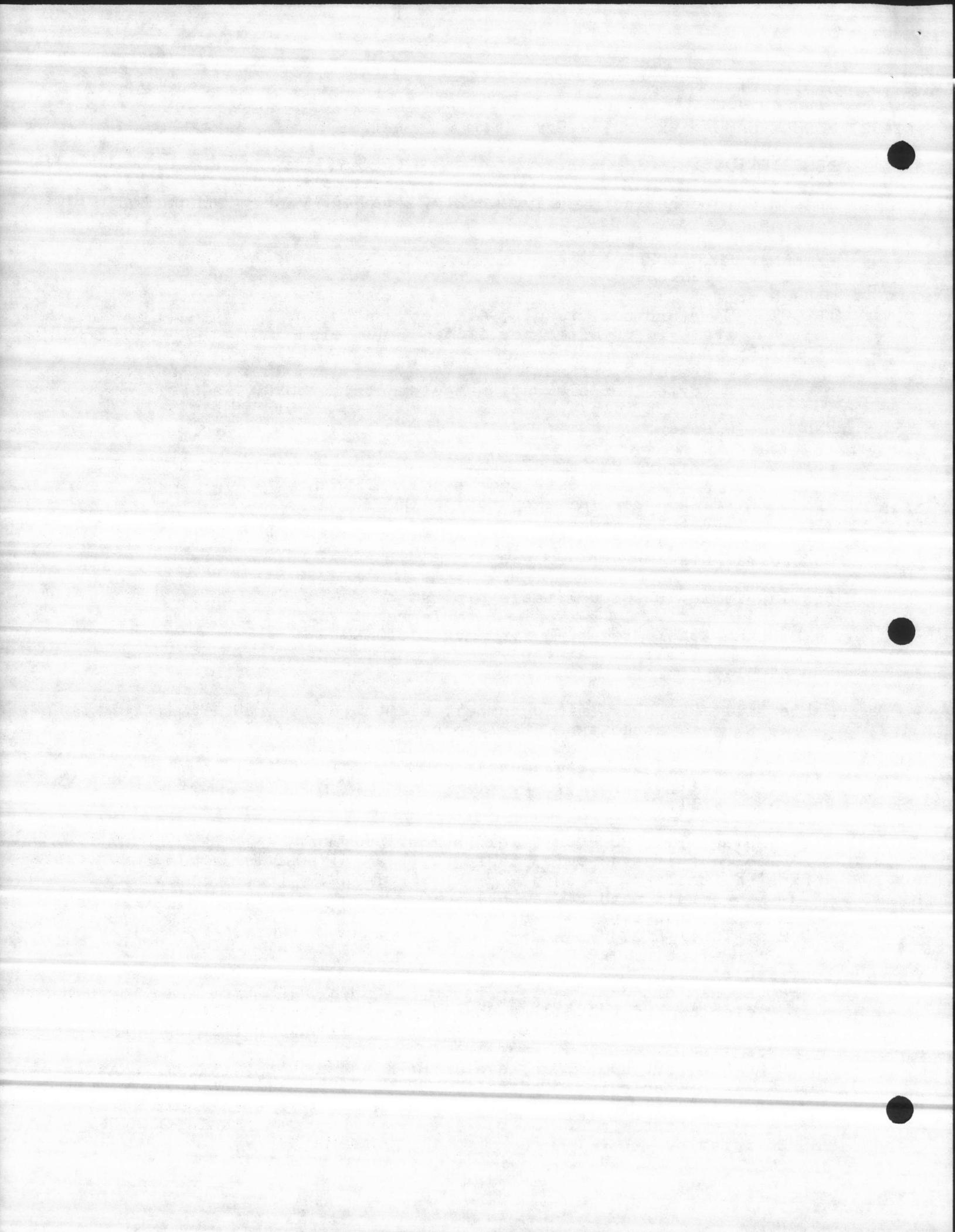
```
Display_Table: PROCEDURE(table_p, param_p,
                         table_reg_t, buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p) POINTER;
END Display_Table;
```

```
Table_Fill: PROCEDURE(field_number, element,
                      struc_type, table_p, param_p) WORD EXTERNAL;
DECLARE (table_p, param_p) POINTER,
        (field_number, element) WORD,
        (struc_type) BYTE;
END Table_Fill;
```

```
Hist_Table_Fill: PROCEDURE(field_number, element,
                           struc_type, table_p, param_p) WORD EXTERNAL;
DECLARE (table_p, param_p) POINTER,
        (field_number, element) WORD,
        (struc_type) BYTE;
END Hist_Table_Fill;
```

Edit\_Table:

```
PROCEDURE (crt_num, table_p, param_p, table_reg_t,
          display_buf_t, display_mbx_t, pass_level,
          edit_buf_t, ci_conn_t, ci_mbx_t) EXTERNAL;
DECLARE (crt_num, pass_level) BYTE,
        (table_reg_t, display_buf_t, display_mbx_t,
         edit_buf_t, ci_conn_t, ci_mbx_t) TOKEN,
        (table_p, param_p) POINTER;
END Edit_Table;
```



```
Display_Generator_Static:
    PROCEDURE (crt_num, max_display_size, static_asc_p,
               fore_color_p, back_color_p,
               display_buf_t, display_mbx_t) EXTERNAL;

    DECLARE (crt_num)                      BYTE,
            (max_display_size) WORD,
            (display_buf_t, display_mbx_t) TOKEN,
            (static_asc_p, fore_color_p, back_color_p) POINTER;
END Display_Generator_Static;

Draw_Graph_Symbols:PROCEDURE (crt_num,graph_p,max_symbols,type,
                             display_buf_t, display_mbx_t) EXTERNAL;

/* Type defines the type of symbols to display */
/* 0 - Static and Dynamic */
/* 1 - Dynamic only */

DECLARE (crt_num,type) BYTE;
DECLARE (max_symbols) WORD;
DECLARE (graph_p) POINTER,
        (display_buf_t,display_mbx_t) TOKEN;
END Draw_Graph_Symbols;

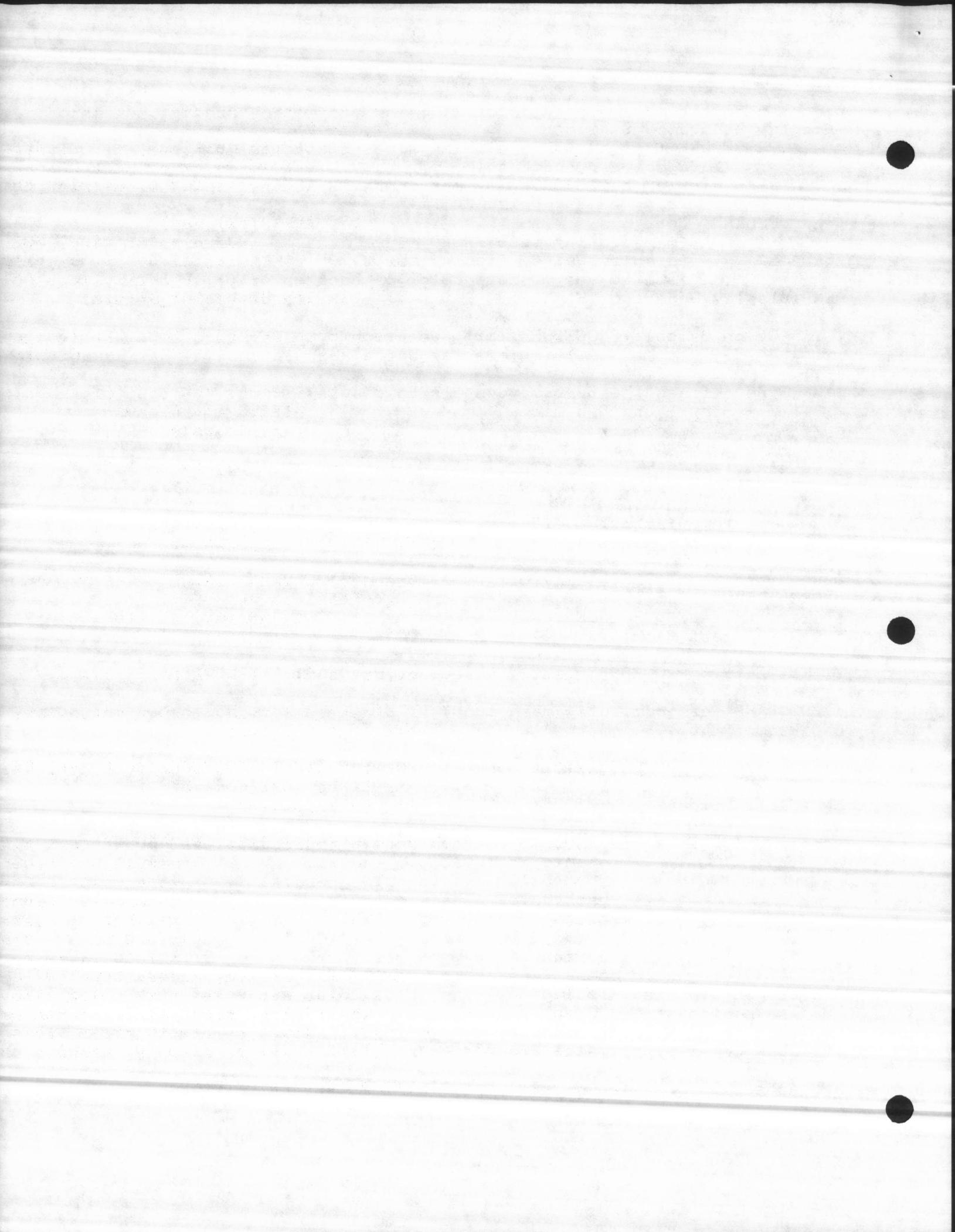
Draw_Trend_Task: PROCEDURE (crt_num , trend_p) EXTERNAL;
    DECLARE crt_num BYTE,
           trend_p POINTER;
END Draw_Trend_Task;

Prompt_Entry: PROCEDURE (entry_max_count,crt_num) EXTERNAL;

    DECLARE (crt_num,entry_max_count) BYTE;
END Prompt_Entry;

Clear_Screen: PROCEDURE (crt_num) EXTERNAL;
    DECLARE crt_num BYTE;
END Clear_Screen;

Access_CRTDP_File: PROCEDURE (crt_num,access_type,edit_level_p,
                               static_asc_p,static_asc_size,
                               fore_color_p,fore_color_size,
                               back_color_p,back_color_size,
                               graph_p      ,graph_size,
                               trend_p      ,trend_size,
                               screen_param_p, screen_param_size) BYTE EXTERNAL;
    DECLARE (crt_num, access_type) BYTE,
            (static_asc_size, fore_color_size, back_color_size,
             graph_size, trend_size, screen_param_size) WORD,
            (edit_level_p, static_asc_p, fore_color_p,
             back_color_p, graph_p, trend_p, screen_param_p) POINTER;
END Access_CRTDP_File;
```



---

GLOBAL REFERENCE :

=====

See CxxxxGLOB.Pyy

## MODULE DECLARATION:

=====

heading\_p => Three pointers - each hold the address of the group heading.

symbol\_count => Three words, each holds the symbol count.

graph\_update\_seg\_t => Three tokens, each used to create the graph update segment.

trend\_update\_seg\_t => Three tokens, each used to create the trend update segment.

crt\_table\_t => A crt table token.

crt\_param\_t => A crt parameter token.

disk\_error\_flag => Three bytes, each holds the disk\_error\_flag.

historical\_display\_flag => Three bytes, each holds the value of the historical display flag.

historical\_display\_called\_flag => Three bytes, each holds the value of the historical display called flag.

read\_file\_param BASED disk\_param\_seg\_t STRUCTURE &

write\_file\_parameters BASED disk\_param\_seg\_t STRUCTURE

(See disk system)

header\_info (4) STRUCTURE &

header\_data (4) STRUCTURE

(See 14.1)

exception => Three words, each holds the exception condition.

crt\_update\_sem\_t => Three crt update semaphore tokens, each used to create the update task semaphore.

Display\_update\_table => A pointer - the address of Display\_Update

Display\_delete\_table => A pointer - the address of Display\_Delete.

Hist\_Display\_Menu:

++++++

crt\_num => A byte, the number of the crt to display on.

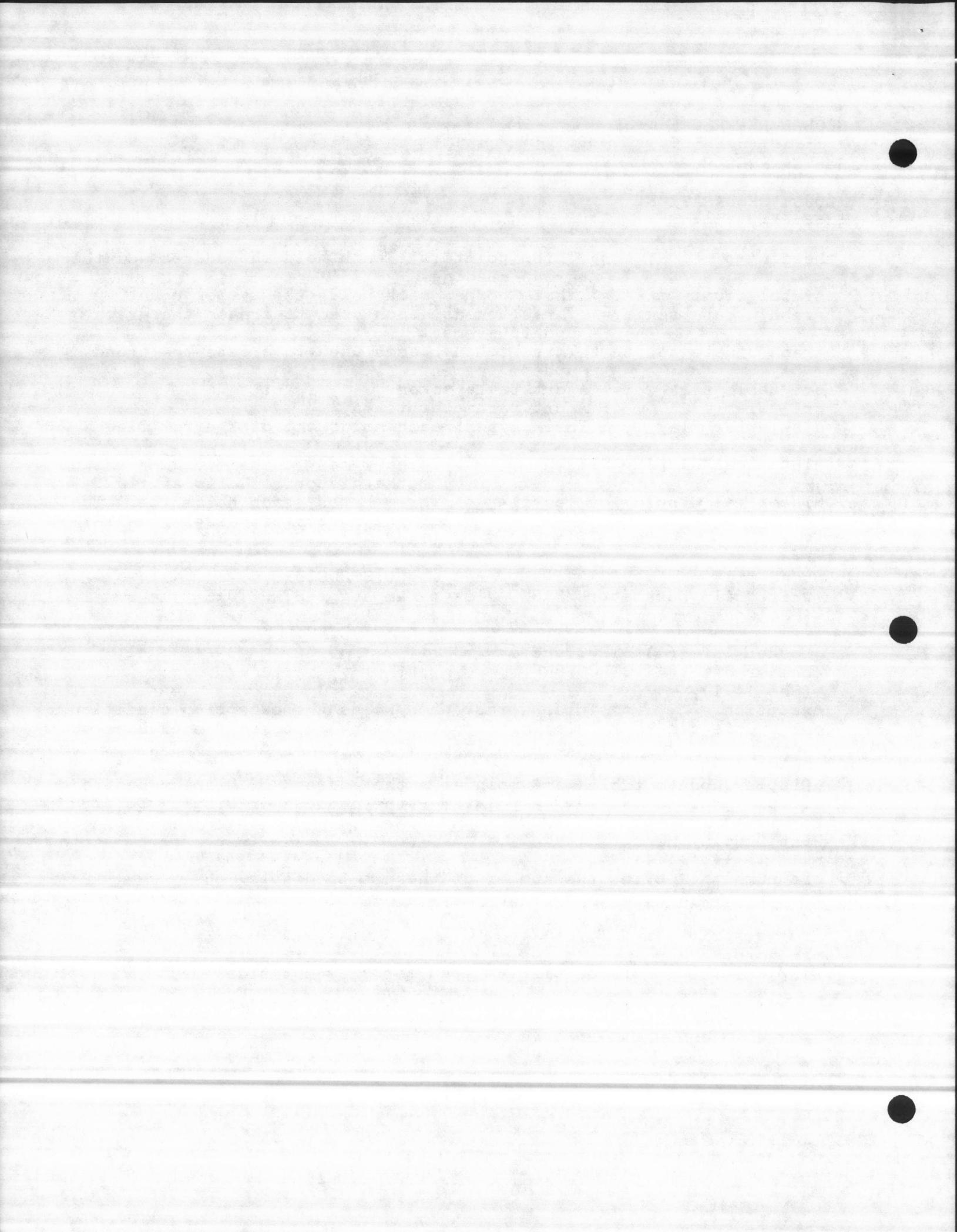
result => A word used as an index.

Proc description :

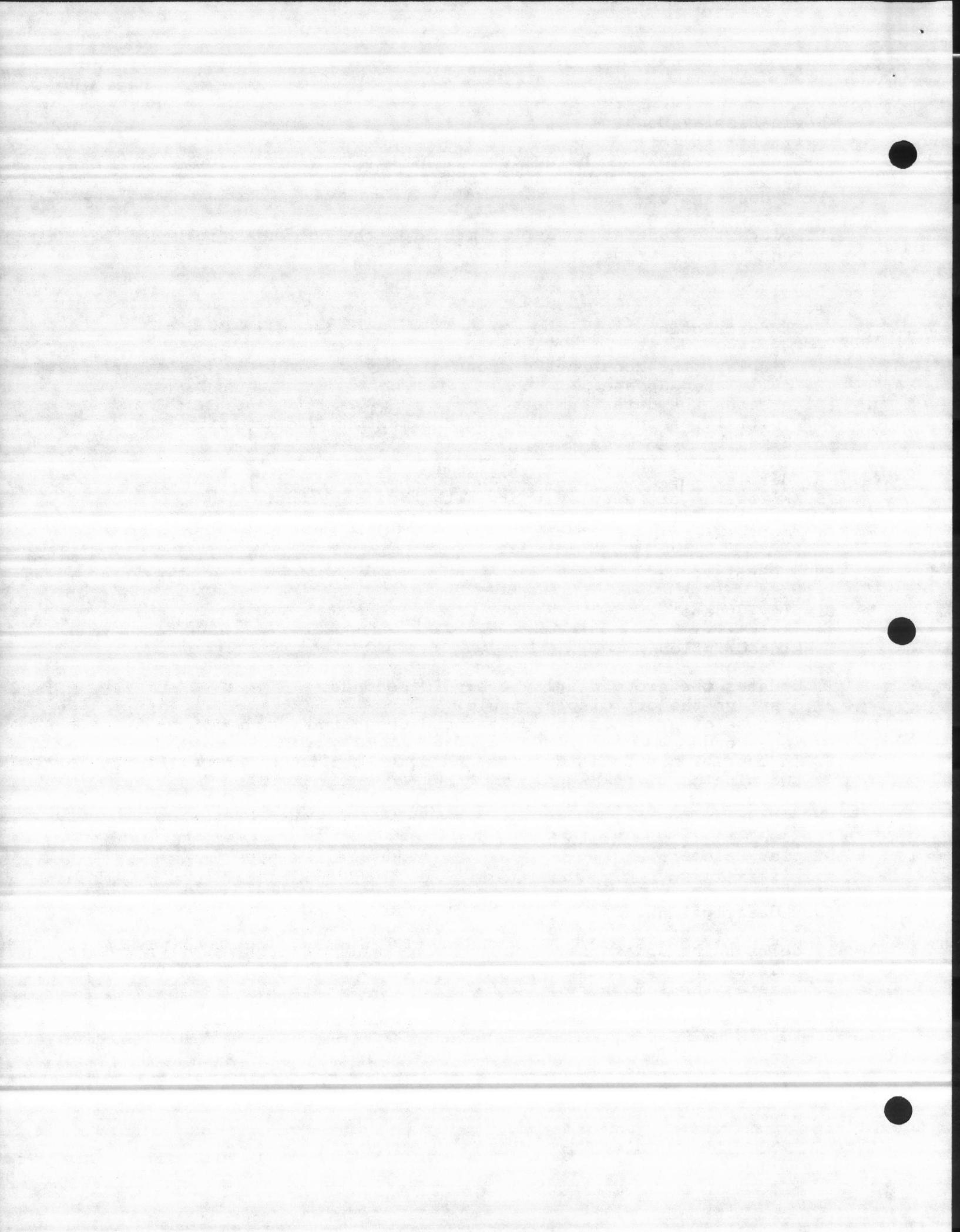
-----

Displays the generator display menu for the historical data.

END Hist\_Display\_Menu;



```
Keyin_Display_Menu:  
+++++  
crt_num => A byte, the number of the crt to display on.  
result => A word used as an index.  
Proc description :  
-----  
Displays the generator display menu for the current data.  
END Keyin_Display_Menu;  
  
Display:  
+++++  
Proc description :  
-----  
1. displays the constructed group display.  
2. calls stop$update.  
3. creates the data segments.  
4. initializes the file size and the maximum display size.  
5. Sets the page mechanism.  
7. Checks the display disk file.  
8. display the group heading and sub heading.  
9. Fills the crt display table with the proper data.  
10. deletes data segments.  
11. Calls set$update.  
  
END Display;  
  
Display_Update:  
+++++  
Task description :  
-----  
1. Updates the group display every 10 seconds.  
2. displays the crt display table.  
3. In case of graphics it updates and draws the graph every  
10 seconds.  
3. In case of trends it updates and draws the trend every  
two minutes.  
  
END Display_Update;  
  
Display_Delete:  
+++++  
  
Task description :  
-----  
1. deletes the update segments.  
2. deletes the clock entry.  
3. deletes the semaphore.  
  
END Display_Delete;
```



(14.3) MODULE NAME : CxxxxESCR.Pyy

=====

DESCRIPTION :

=====

Edit screen is a full screen editor used for developing user definable CRT displays.

EXTERNAL PROCEDURES :

=====

Format\_Cursor\_And\_Color: PROCEDURE (display\_buf\_t, row, column,  
color\_index) EXTERNAL;

DECLARE (row, column) BYTE,  
(display\_buf\_t) TOKEN,  
(color\_index) BYTE;

END Format\_Cursor\_And\_Color;

Enter\_Password\_And\_Verify: PROCEDURE (crt\_num) WORD EXTERNAL;

/\* returns :

0 - password entered is incorrect or display invoked,  
1 to 3 : password entered has matched level

(equal passwords will return highest level).

display invoked by entering PATIODOOR as password  
default password is PASSWORD \*/

DECLARE (crt\_num) BYTE;

END Enter\_Password\_And\_Verify;

Prompt\_Display: PROCEDURE (crt\_num, display\_buf\_t, crt\_out\_mbx\_t,  
row, column, display\_str\_p) EXTERNAL;

DECLARE (row, column, crt\_num) BYTE,  
(display\_buf\_t, crt\_out\_mbx\_t) TOKEN,  
(display\_str\_p) POINTER;

END Prompt\_Display;

GLOBAL REFERENCE :

=====

Check CxxxxGLOB.EXT

Check CxxxxTABL.PUB

Check CxxxxEDIT.Pyy

PUBLIC PROCEDURE :

=====

Edit\_Screen:

++++++

char => A byte holds a character.

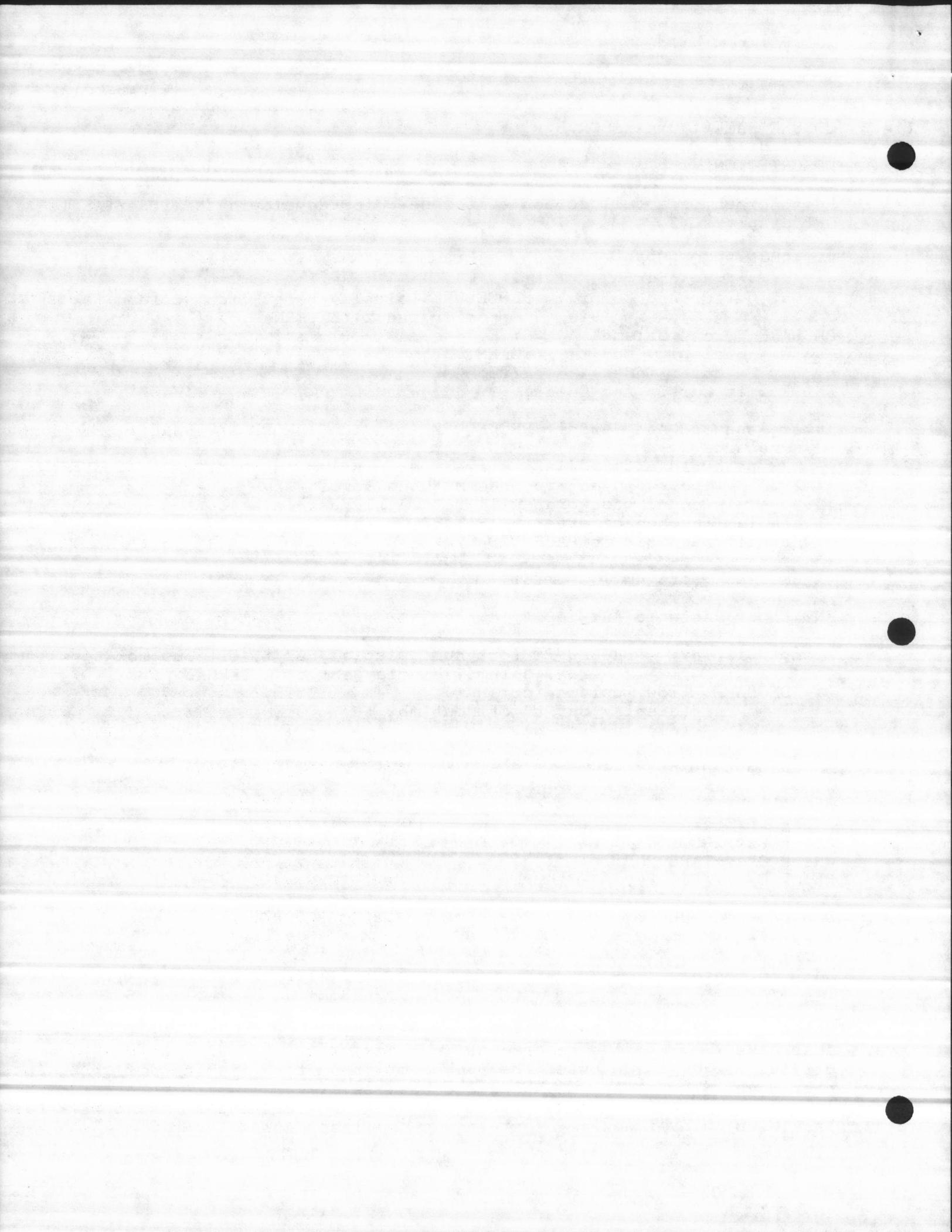
char\_case => A byte used as an index.

crt\_num => A byte, the number of the crt to display on.

pass\_level => A byte holds the pass\_level value.

last\_fore\_color\_index => A byte holds the last fore color index.

last\_back\_color\_index => A byte holds the last back color index.



```
max_display_size=> A word holds the maximum display size.  
display_buf_t => A display buffer token.  
display_mbx_t => A display mailbox token.  
edit_buf_t    => An edit buffer token.  
ci_conn_t     => A CI connection token.  
ci_mbx_t      => A CI mailbox token.  
static_asc_p   => A pointer - the address of the static ascii.  
fore_color_p   => A pointer - the address of the fore color.  
back_color_p   => A pointer - the address of the back color.  
  
char_index      => A word used as an index.  
current_table_index => A word used as an index to the current crt table.  
previous_table_index => A word used as an index to the previous table.  
exception       => A word holds the exception condition.  
result          => A word used as an index.  
remaining_units  => A word used as an index to the remaining units.
```

Maximum of a 43 X 80 screen

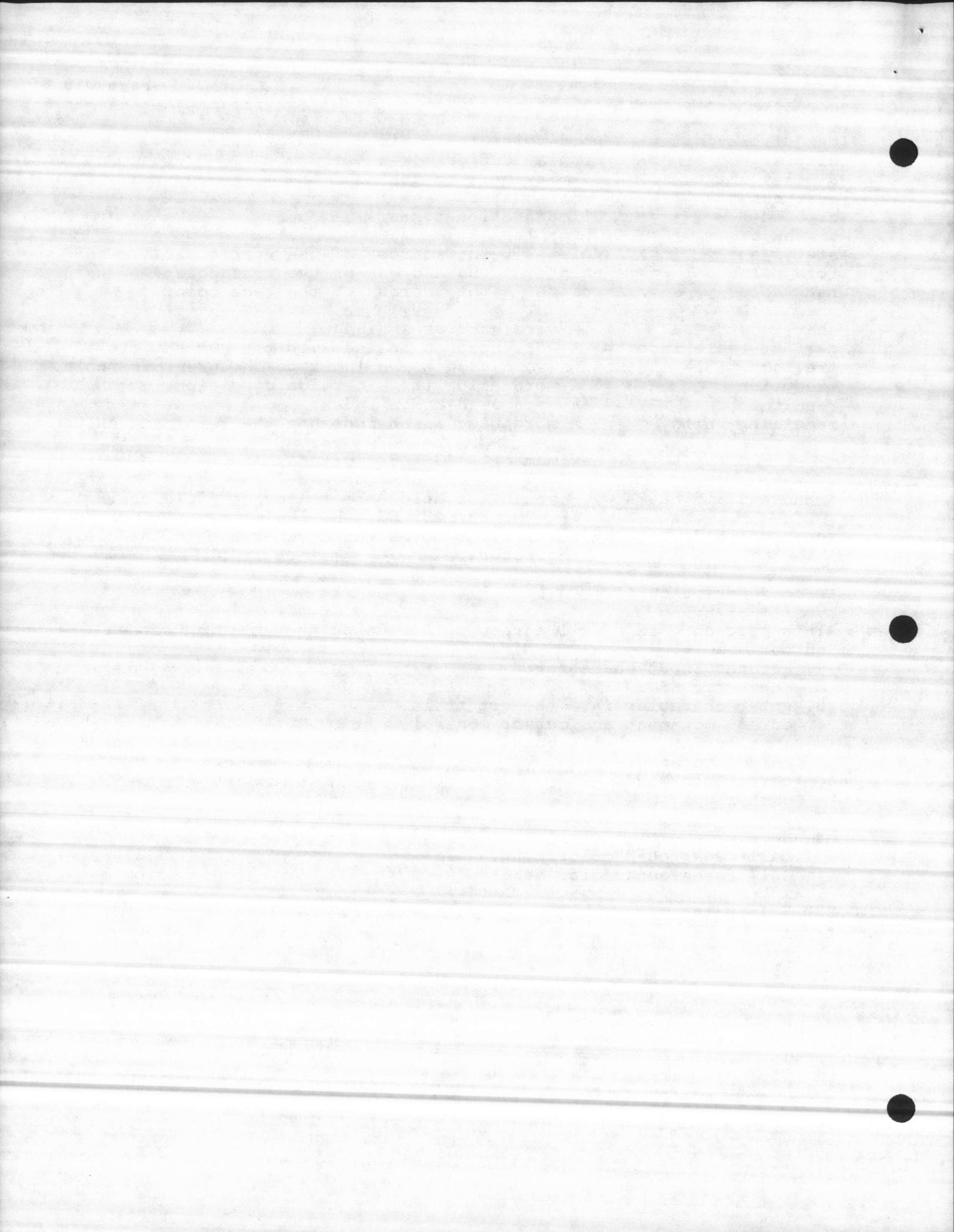
```
change_field_chars => Seven bytes holds the following field characters  
                      1H, 2H, 3H, 4H, 5H, 6H, 7H  
change_field_xref  => Seven bytes holds the x_reference to the field  
                      characters.
```

#### Proc description :

-----  
This procedure is a full screen editor using the existing screen editor module.

1. get the input character.
2. check for ESC.
3. enter character into the crt table.
4. cursor movement and cursor control  
 invalid input, bell sound.  
 cursor forward.  
 cursor backward  
 cursor up.  
 cursor down.  
 Home.  
 call background color menu.  
 call foreground color menu.

END Edit\_Screen;



4.4) MODULE NAME : CxxxxDGEN.Pyy

=====

DESCRIPTION :

=====

This module displays the group generator static.

EXTERNAL PROCEDURES :

=====

Format\_Cursor\_And\_Color:

PROCEDURE (display\_buf\_t, row, column, color\_index) EXTERNAL;

DECLARE (row, column) BYTE,  
(display\_buf\_t) TOKEN,  
(color\_index) BYTE;

END Format\_Cursor\_And\_Color;

MODULE DECLARATION :

=====

row\_xref => 14 bytes hold the row numbers as an X reference.

PUBLIC PROCEDURES :

=====

Display\_Generator\_Static:

+++++-----+

crt\_num => A byte, the number of the crt to display on.

max\_display\_size=> A word holds the maximum display size.

display\_buf\_t => A display buffer token.

display\_mbx\_t => A display mailbox token.

edit\_buf\_t => An edit buffer token.

static\_asc\_p => A pointer - the address of the static ascii.

fore\_color\_p => A pointer - the address of the fore color.

back\_color\_p => A pointer - the address of the back color.

index => A byte used as an index.

old\_fore\_color => A byte holds the index to the old foreground color.

old\_back\_color => A byte holds the index to the old background color.

display\_flag => A byte holds the value of the display flag.

line => A word holds the line number.

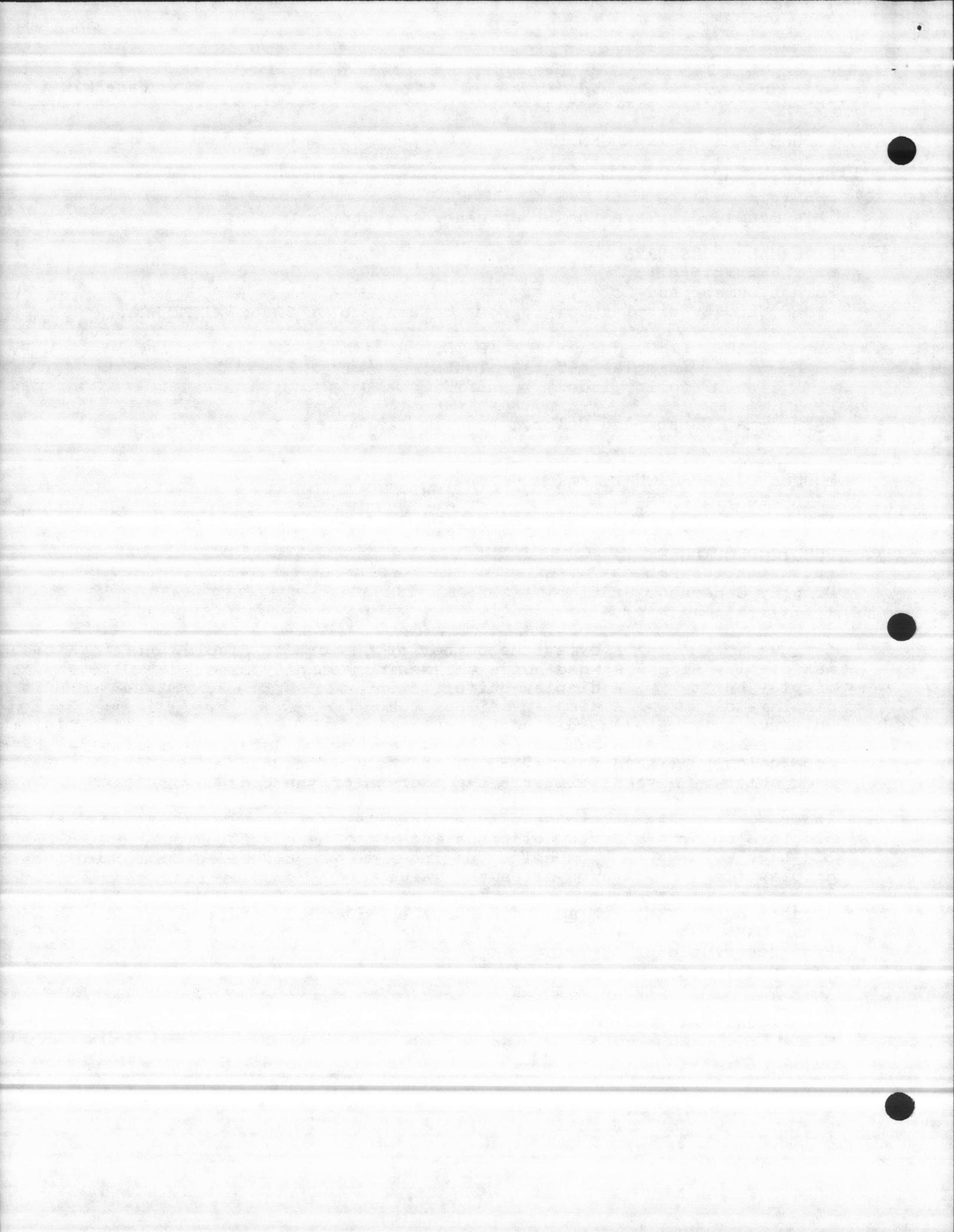
exception => A word holds the exception condition.

Proc description :

-----

This procedure displays the generated group static.

END Display\_Generator\_Static;



AQUATROL DIGITAL SYSTEMS  
Arden Hills, MN.  
COPYRIGHT 1986

SECTION No. 15

G R A P H I C S

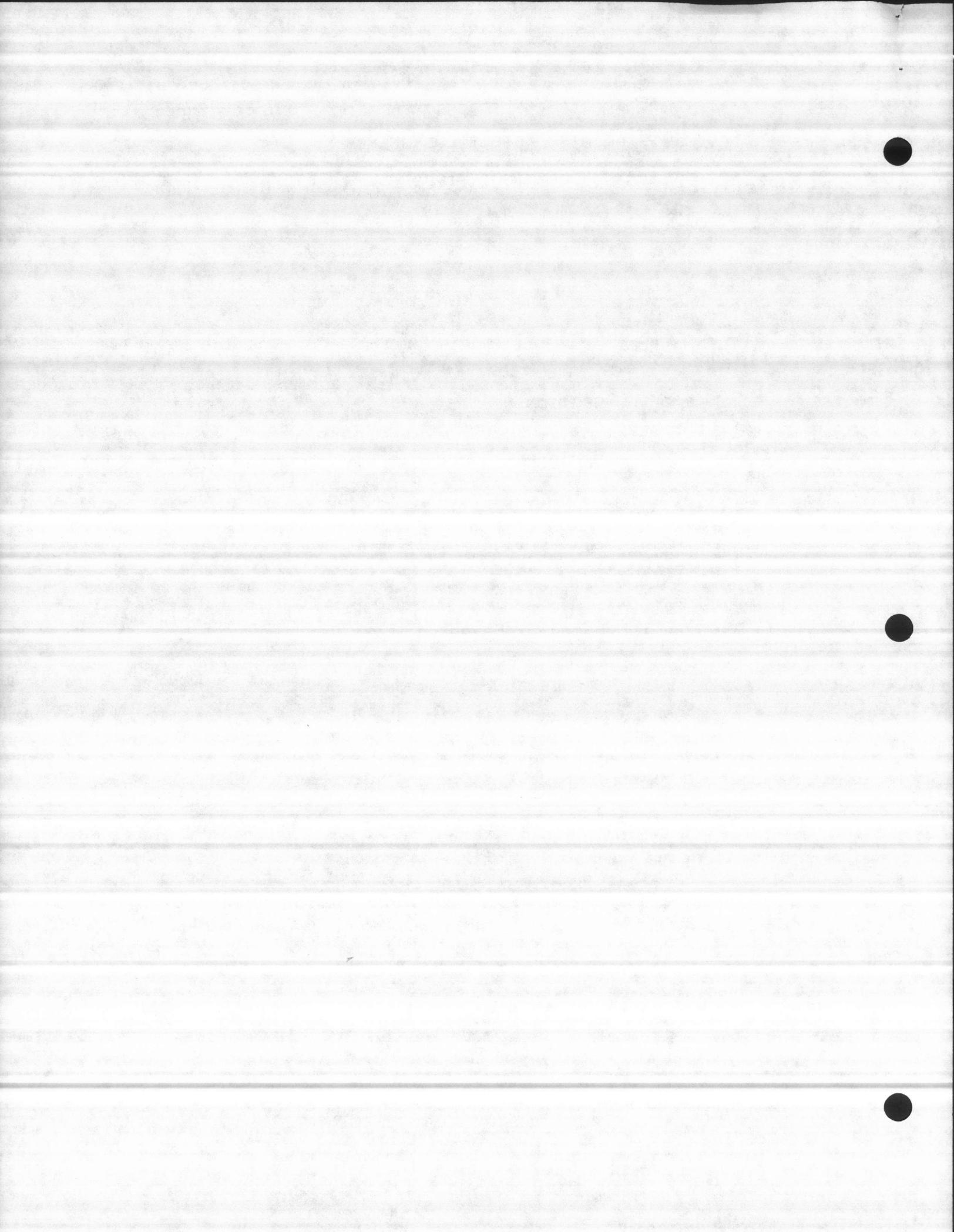
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



INTRODUCTION :

=====

The graphic system is a group of routines forming symbols like pump, valve, gate, ... etc. All the graphic symbols available in what we call a graphic library.

DATE : Oct 10, 1986

====

JOB # : C4758

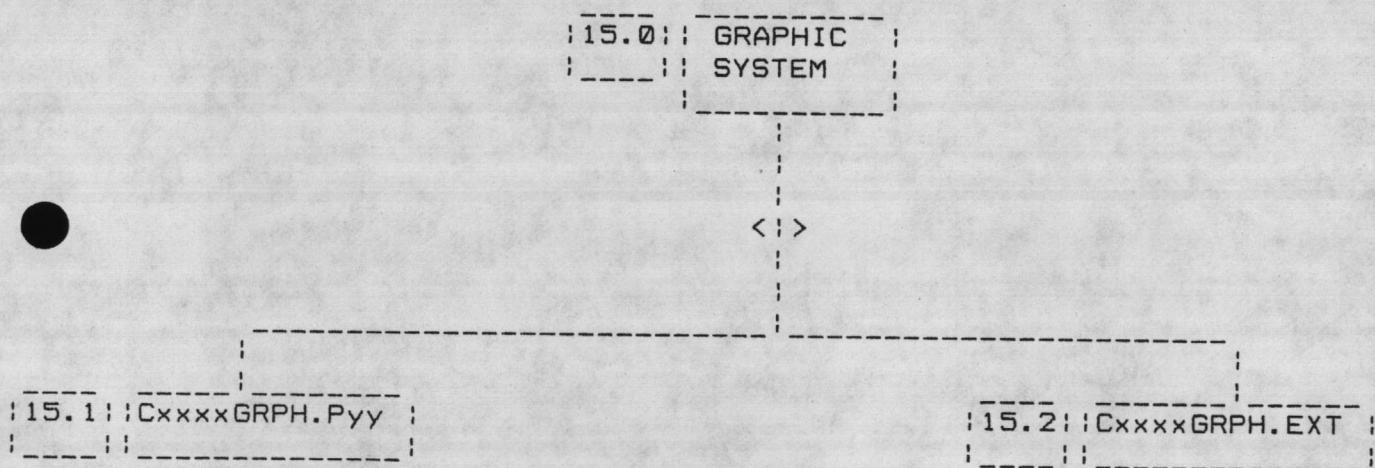
=====

HOLCOMB BLVD, W.W.T.P. CAMP LEJEUNE, NC.

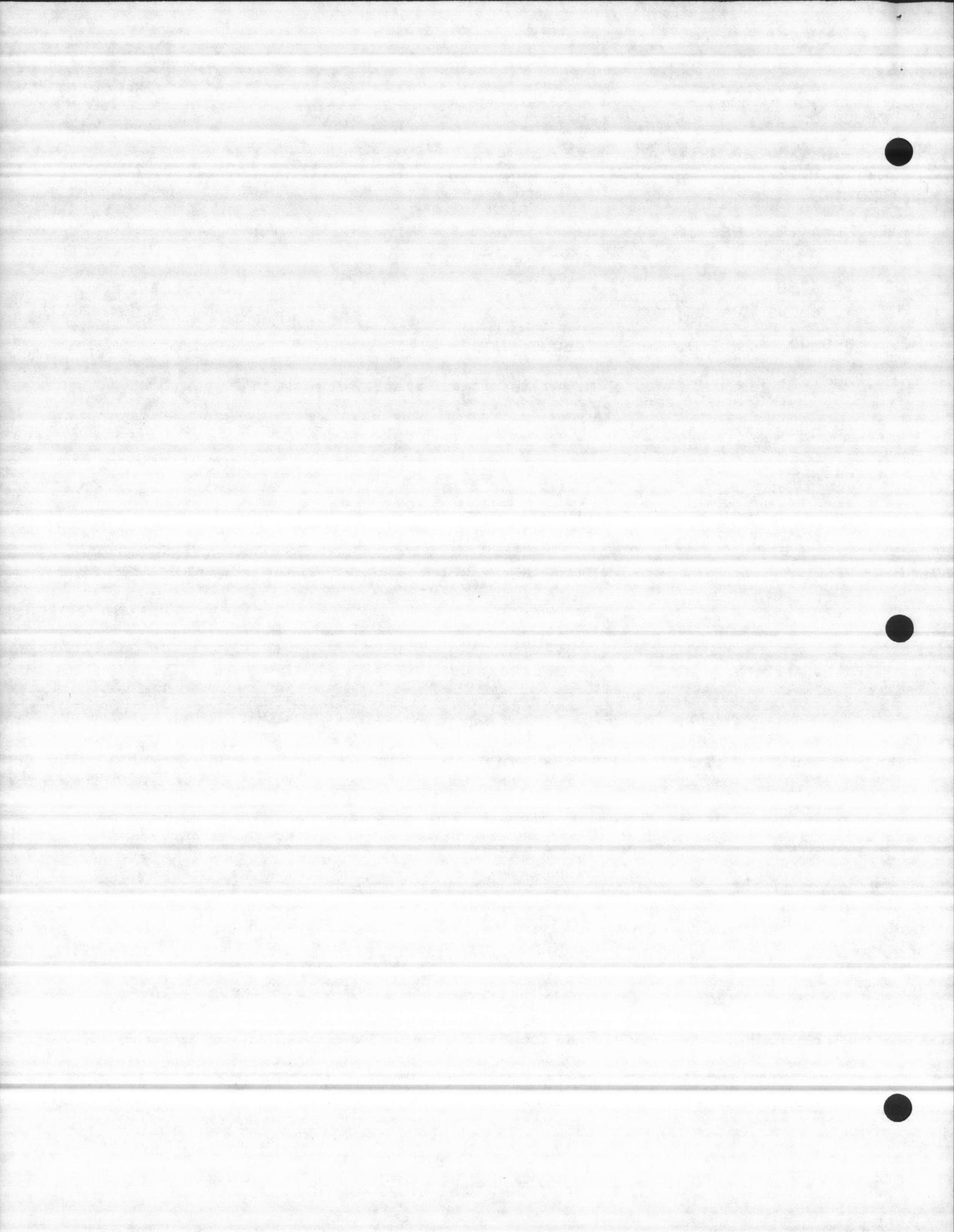
The graphic - written by Bob Ryan, Mohamed Fayad, Peter Wollenzien,  
Greg Jensen.

GRAPHIC SYSTEM :

=====



Where Cxxxx = C4758 & Pyy = P86



(15.1) MODULE NAME : CxxxxGRPH.Pyy

=====

**DESCRIPTION :**

===== This module contains all the public procedures.  
The procedures are dividing into groups as following :

- 1) DRAWING PARAMETER PROCEDURES
- 2) CURSOR POSITION PROCEDURES
- 3) LINE DRAWING PROCEDURES
- 4) GEOMETRIC FIGURE DRAWING PROCEDURES
- 5) FILL PROCEDURES.
- 6) STANDARD SYMBOL PROCEDURES.
- 7) GRAPH DRAWING PROCEDURES.
- 8) NON\_GRAPHIC PROCEDURES.

1) DRAWING PARAMETER PROCEDURES :

=====

**Set\_Brush\_Width:** PROCEDURE (crt\_num, x\_width, y\_width) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
x\_width => a word - holds the width value of x.  
y\_width => a word - holds the width value of y.

**Proc Description :**

This procedure will define the thickness of the line drawings.  
END Set\_Brush\_Width;

**Set\_Graphic\_Color:** PROCEDURE (crt\_num, color\_index, x\_mode) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
color\_index => a word holds the color index, which is used to select the  
drawing color  
x\_mode => a word holds the mode value, which it could be of the range of 0 to  
7. (see page 30 in XL-19 Manual).

**Proc Description :**

This procedure assigns a color to the CURSOR.

END Set\_Graphic\_Color;

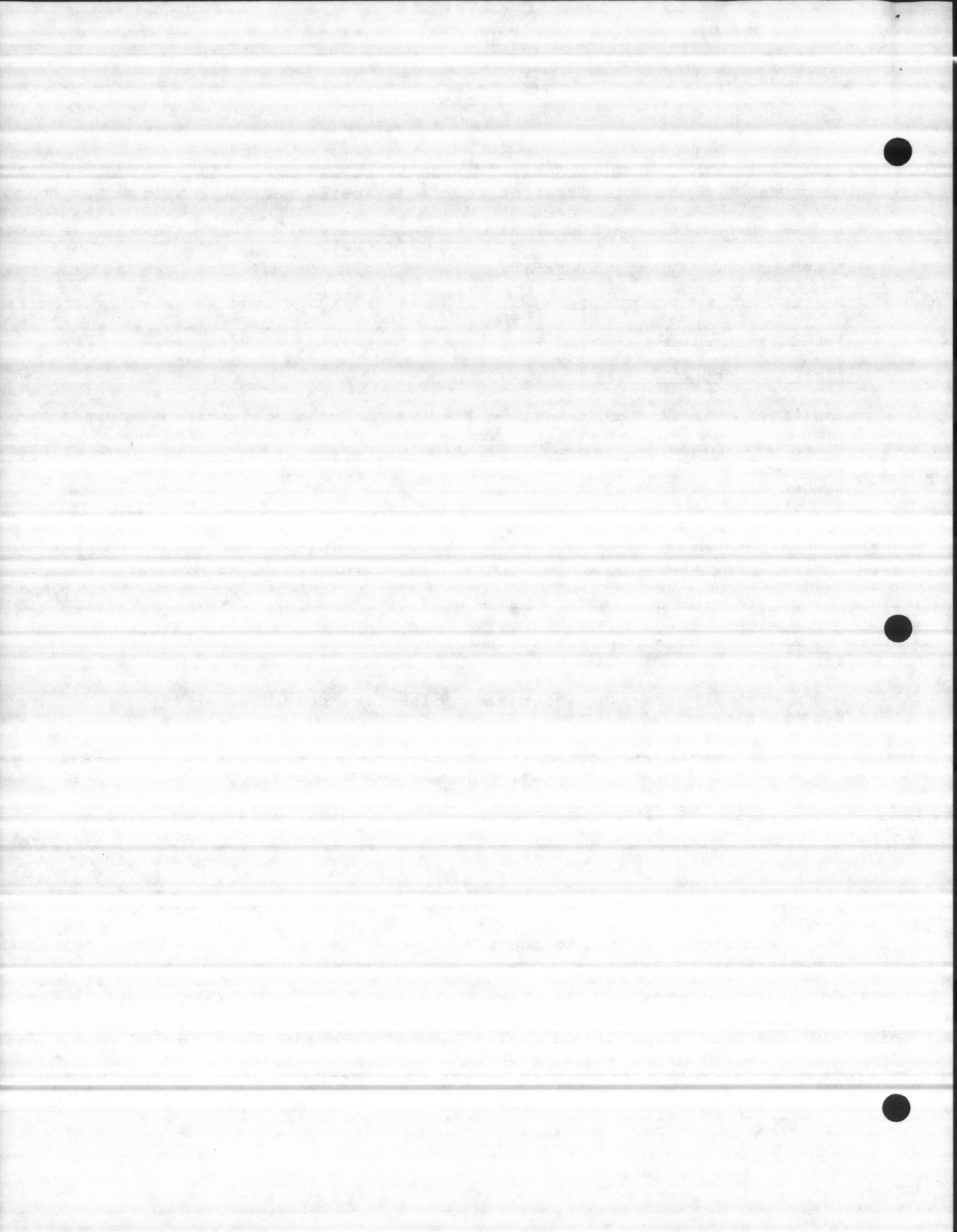
**Set\_Fill\_Type:** PROCEDURE (crt\_num, pattern, background) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
pattern => a word is a parameter from 0 to 255 used to select a fill pattern  
The default pattern, = 0, which is the solid fill.  
background => a word - is an optional parameter used to select a background  
color for a fill operations using a non-solid pattern.

**Proc Description :**

This procedure used to select fill type (see XL-19 Manual pg 32).

END Set\_Fill\_Type;



Set\_Vector\_Type: PROCEDURE (crt\_num, pattern) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
pattern => a word - is a value from 0 to 255 and represents an 8 bit mask used  
to create a dotted or dashed line style. The default value of 255  
generates a solid line and represents an 8 bit mask of all 1's.  
(255 decimal = 11111111 binary).

Proc description :

This procedure used to select vector type (see XL-19 Manual pg 32).

END Set\_Vector\_Type;

2) CURSOR POSITIONING PROCEDURES :

Abs\_Cursor: PROCEDURE (crt\_num, num\_moves, coord\_p) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
num\_moves => a word - holds the number of moves.  
coord\_p => a pointer points to coordinates X,Y.

Proc description :

This procedure causes the current position of the CURSOR to move to the  
coordinates X,Y (see XL-19 Manual pg 33).

END Abs\_Cursor;

Rel\_Cursor: PROCEDURE (crt\_num, num\_moves, coord\_p) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
num\_moves => a word - holds the number of moves.  
coord\_p => a pointer points to coordinates dx1,dy1.

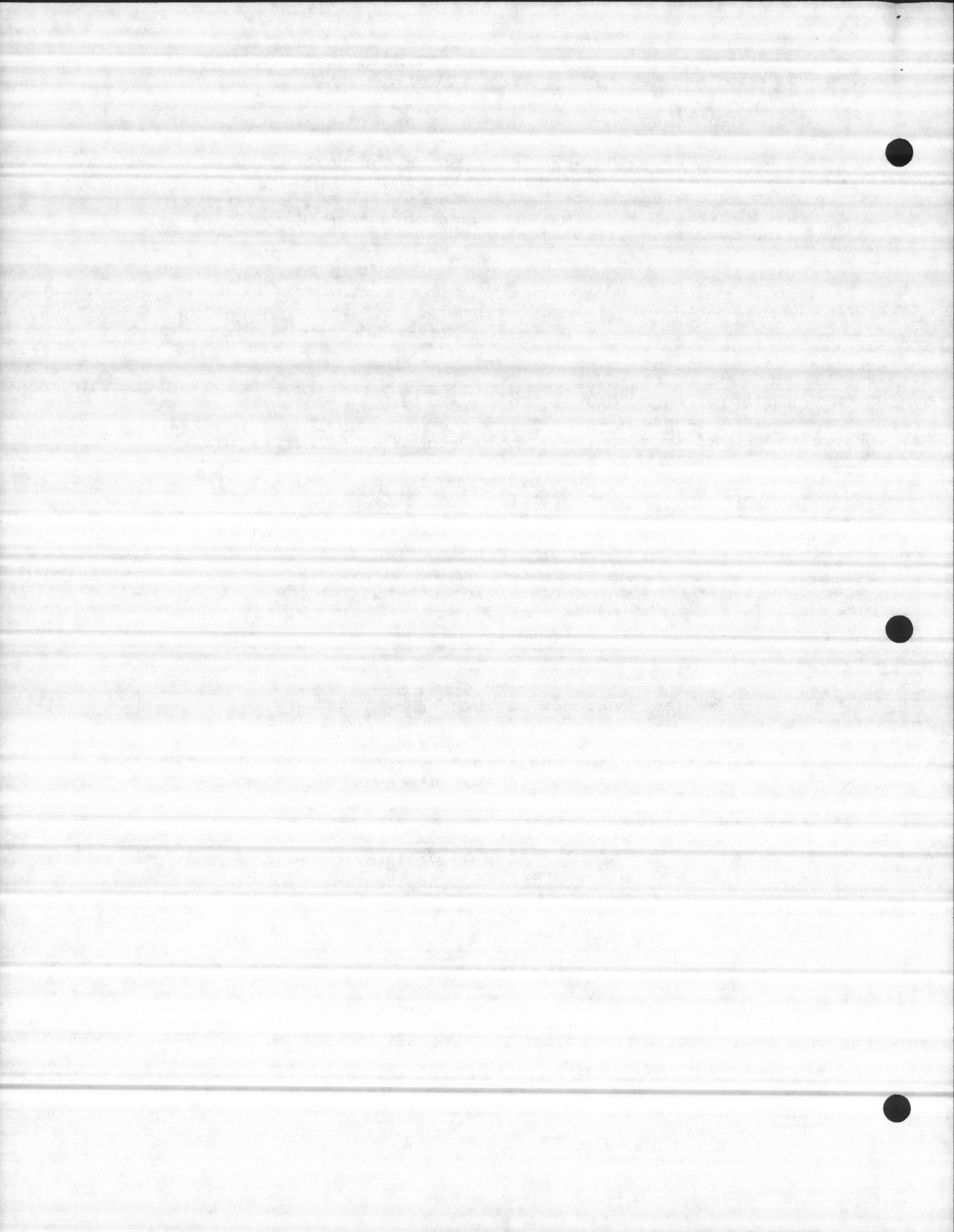
Proc description :

This procedure causes the current position of the CURSOR to move to the  
coordinates dx1,dy1 (see XL-19 Manual pg 33).

END Rel\_Cursor;

Polar\_Cursor: PROCEDURE (crt\_num, num\_moves, coord\_p) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
num\_moves => a word - holds the number of moves.  
coord\_p => a pointer points to coordinates X,Y, Which are the length, and  
the angle.



Proc description :

This procedure causes the current position of the CURSOR to move to the coordinates X,Y (length, angle). (see XL-19 Manual pg 33).

END Polar\_Cursor;

3) LINE DRAWING PROCEDURES :

Draw\_Abs\_Vector: PROCEDURE (crt\_num, num\_vectors, coord\_p) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
num\_vectors => a word - holds how many vector need to be drawing.  
coord\_p => a pointer points to the coordinates

Proc description :

This procedure causes the vector to be drawn from point X1, Y1 to point X2, Y2 and on to X3, Y3 etc. At least two points are required.

END Draw\_Abs\_Vector;

Draw\_Rel\_Vector: PROCEDURE (crt\_num, num\_vectors, coord\_p) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
num\_vectors => a word - holds how many vector need to be drawing.  
coord\_p => a pointer points to the coordinates

Proc description :

The vector is drawn from the current position to the new position specified by the change in x ,dx and the change in y, dy.

END Draw\_Rel\_Vector;

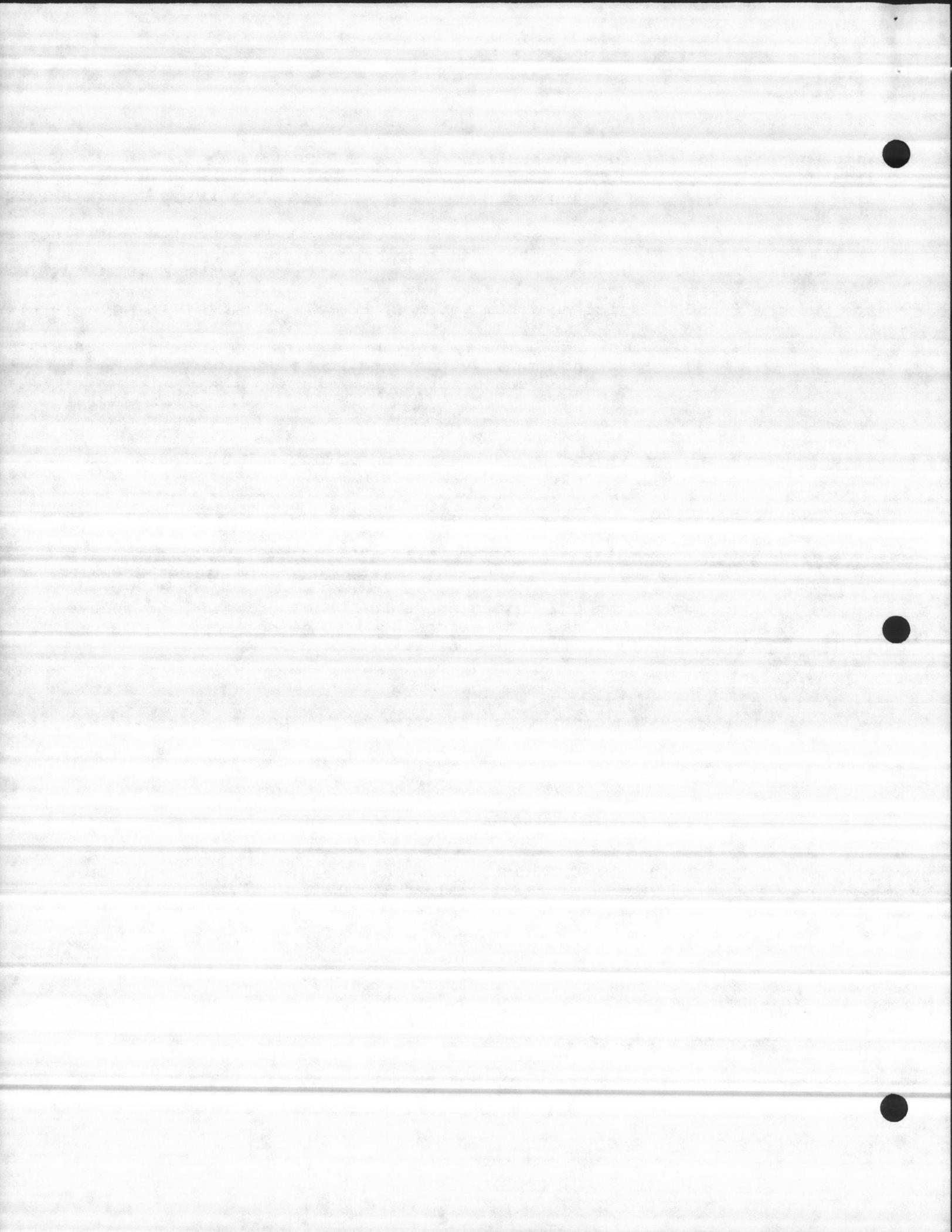
Draw\_Polar\_Vector: PROCEDURE (crt\_num, num\_vectors, coord\_p) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
num\_vectors => a word - holds how many vector need to be drawing.  
coord\_p => a pointer points to the coordinates

Proc description :

The vector is drawn from the current position to the new position specified by the LENGTH and ANGLE.

END Draw\_Polar\_Vector;



## 4) GEOMETRIC FIGURE DRAWING PROCEDURES :

```
=====  
Draw_Bar: PROCEDURE (crt_num, width, height, fill,  
                  xtip, ytip) PUBLIC REENTRANT;
```

crt\_num => a byte - holds the CRT number where the heading is to be displayed.

width => a word holds the width in pixels

height=> a word holds the hieght in pixels.

fill => a word if it is = 0 then 0=solid figure otherwise 1=outline.

xtip => a word = the displacement of the upper left corner of the figure  
            from the vertical.

ytip => a word = the displacement of the upper right corner of the figure  
            from the horizontal.

Proc description :

This procedure represent the command to draw a parallelogram.

END Draw\_Bar;

```
=====  
Draw_Arc: PROCEDURE (crt_num, radius, angle1, angle2, option) PUBLIC REENTRANT;
```

crt\_num => a byte - holds the CRT number where the heading is to be displayed.

radius, angle1, angle2, option => draw an arc parameter where

radius => a word as endpoint.

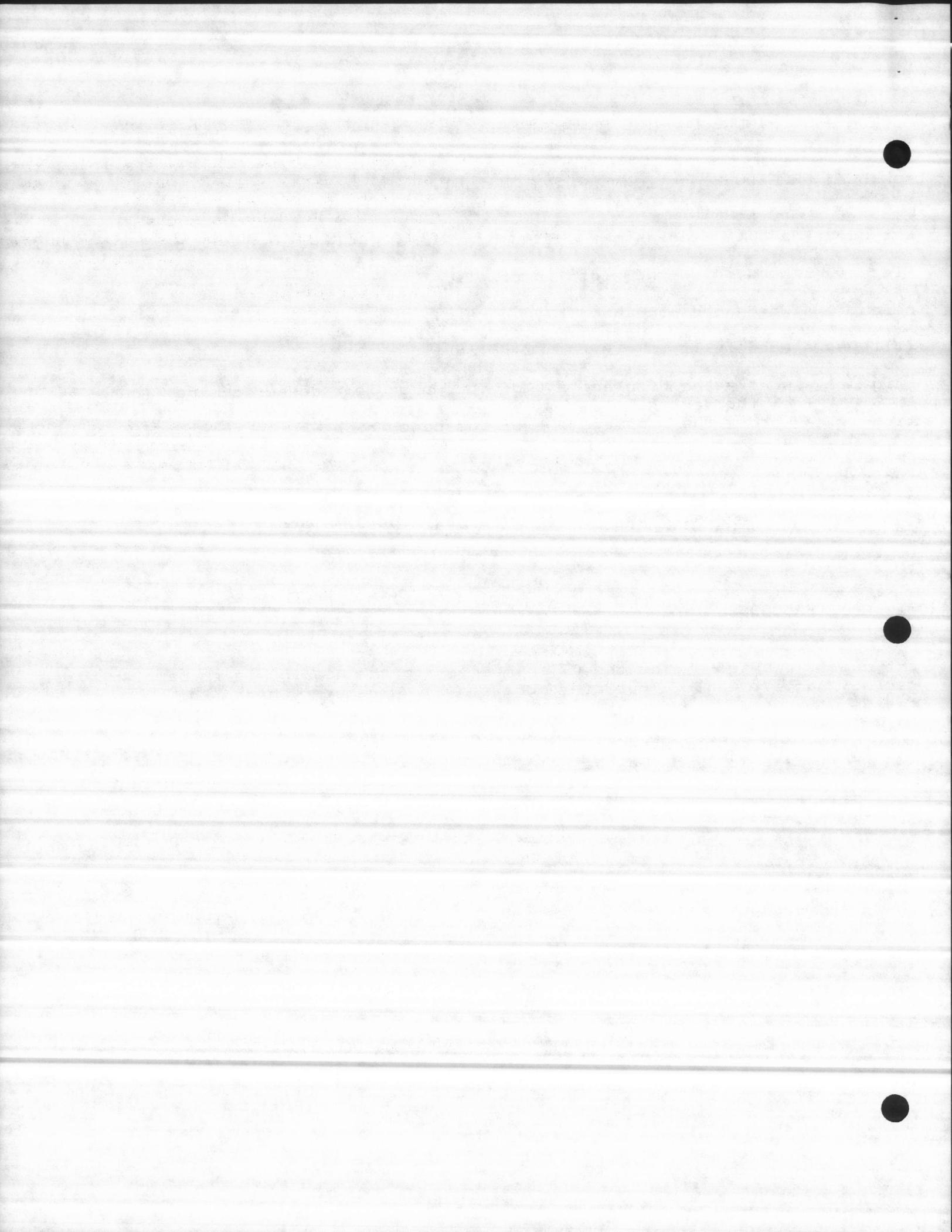
angle1, angle2 => two word hold angles

option => a word holds the option number, which range from 0 to 4.  
            (see XL-19 Manual pg 34).

Proc description :

to draw an arc.

END Draw\_Arc;



Draw\_Rel\_Arc: PROCEDURE (crt\_num, radius, angle, option) PUBLIC REENTRANT;  
-----  
crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
radius, angle1, angle2, option => draw an arc parameter where  
radius => a word as endpoint.  
angle => a word holds angle.  
option => a word holds the option number, which range from 0 to 4.  
(see XL-19 Manual pg 34).

Proc description :  
-----

to draw an arc.  
END Draw\_Rel\_Arc;

Draw\_Circle: PROCEDURE (crt\_num, radius, fill) PUBLIC REENTRANT;  
-----

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
radius => a word holds a radius value.  
fill => a word if 0 then draw circle(default)  
if 1 then draw filled circle.

Proc description :  
-----

to draw a circle.

END Draw\_Circle;

Draw\_Abs\_Elip\_Arc: PROCEDURE (crt\_num, x\_radius, y\_radius, angle1, angle2,  
----- angle, num\_sides, option) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
x\_radius, y\_radius, angle1, angle2, angle, num\_sides, option => proc parameter.

Proc description :  
-----

to draw absolute elipse arc.

END Draw\_Abs\_Elip\_Arc;

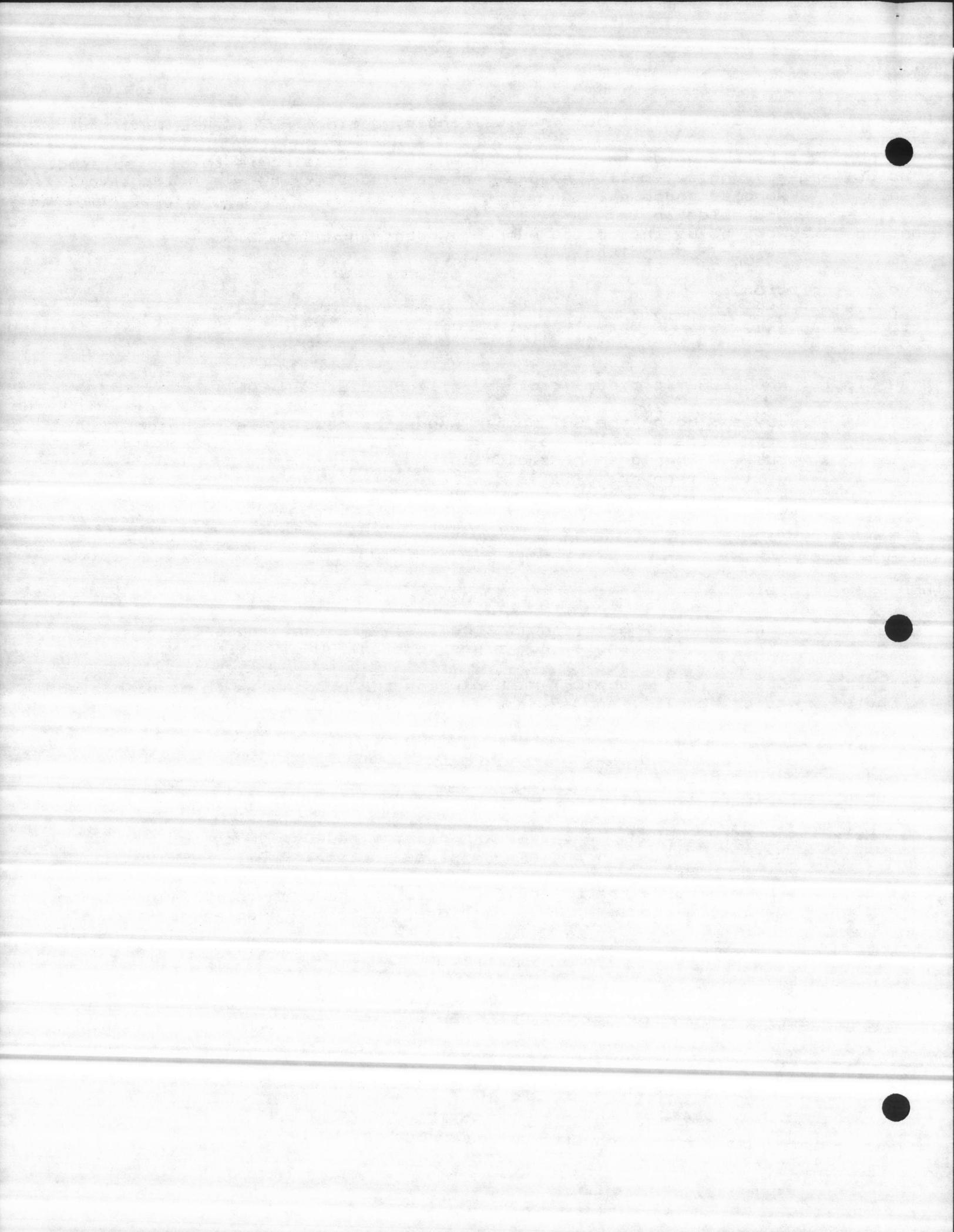
Draw\_Rel\_Elip\_Arc: PROCEDURE (crt\_num, x\_radius, y\_radius, degrees,  
----- angle, num\_sides, option) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
x\_radius, y\_radius, degrees, angle, num\_sides, option => proc parameters

Proc description :  
-----

The same as the previos procedure with one exception the drawing from a  
relative position.

END Draw\_Rel\_Elip\_Arc;



```
Draw_Ellipse: PROCEDURE (crt_num, x_radius, y_radius, angle,
                         num_sides, fill) PUBLIC REENTRANT;
-----
crt_num => a byte - holds the CRT number where the heading is to be displayed.
x_radius, y_radius, angle, num_sides, fill => proc parameters

Proc description :
-----
to draw an Ellipse.

END Draw_Ellipse;

5) FILL PROCEDURES :
=====
Draw_Abs_Fill: PROCEDURE (crt_num, num_coord, coord_p) PUBLIC REENTRANT;
-----
crt_num => a byte - holds the CRT number where the heading is to be displayed.
num_coord => a word - holds the number of coordinates.
coord_p => a pointer points to coordinates.

Proc description :
-----
This procedur is used to fill an area surrounded by a the boundary described
in the coordinate list.

END Draw_Abs_Fill;

Draw_Rel_Fill: PROCEDURE (crt_num, num_coord, coord_p) PUBLIC REENTRANT;
-----
crt_num => a byte - holds the CRT number where the heading is to be displayed.
num_coord => a word - holds the number of coordinates.
coord_p => a pointer points to coordinates.

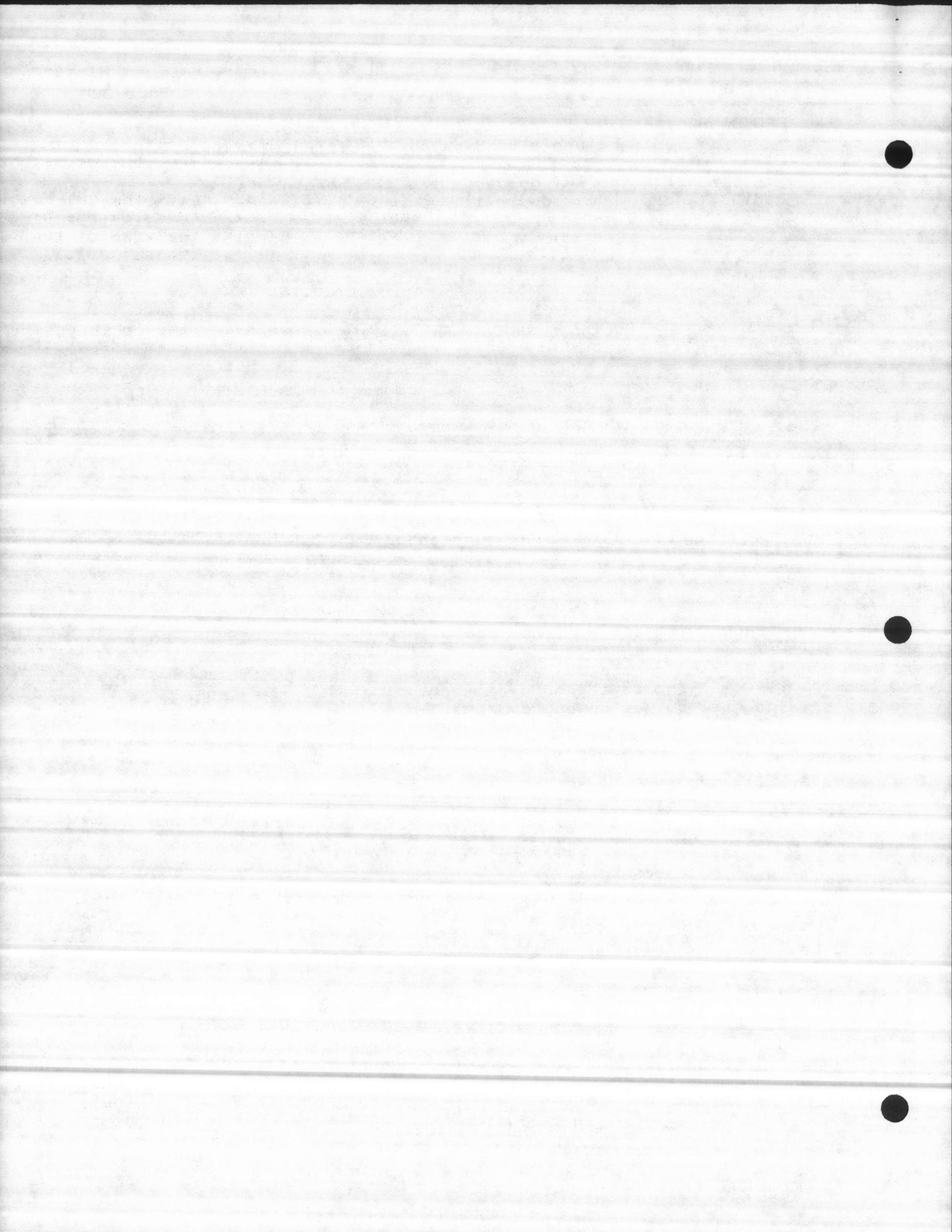
Proc description :
-----
This procedur is used to fill an area surrounded by a the boundary described
in the coordinate list from a relative point.

END Draw_Rel_Fill;

Draw_Bounded_Fill: PROCEDURE (crt_num, pattern, background) PUBLIC REENTRANT;
-----
crt_num => a byte - holds the CRT number where the heading is to be displayed.
pattern, background => two parameters

Proc description :
-----
Used when a figure to be filled is an area of solid color bounded by lines
or areas of different colors.

END Draw_Bounded_Fill;
```



## 6) STANDARD SYMBOL PROCEDURES :

```
=====  
Dr_Valve: PROCEDURE (crt_num, size, heading_angle) PUBLIC REENTRANT;
```

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
heading\_angle => a word holds the direction of the valve.  
size => a word holds the size of the valve.  
temp\_list (10) => a 10 word holds the positions of the valve.

## Proc description :

The valve is drawn around the current graphic cursor position rotated according to the displacement of the heading\_angle. The final graphic position will be the side of the valve in the direction of the heading\_angle. Two filled equalateral triangles drawn outwards from the cursor are used to draw the valve. The size parameter is used as the length for triangle sides.

```
END Draw_Valve;
```

```
Draw_Check_Valve: PROCEDURE (crt_num, size, heading_angle) PUBLIC REENTRANT;
```

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
heading\_angle => a word holds the direction of the valve.  
size => a word holds the size of the valve.  
temp\_list (10) => a 10 word holds the positions of the valve.

## Proc description :

The valve is drawn end to end from the current graphic cursor rotated according to the displacement of the heading\_angle. The final graphic position will be the side of the valve in the direction of the heading\_angle. Three vectors are used to draw the valve. The size parameter is used as the length of the two outside vectors, with the middle vector using length X 2.

```
END Draw_Check_Valve;
```

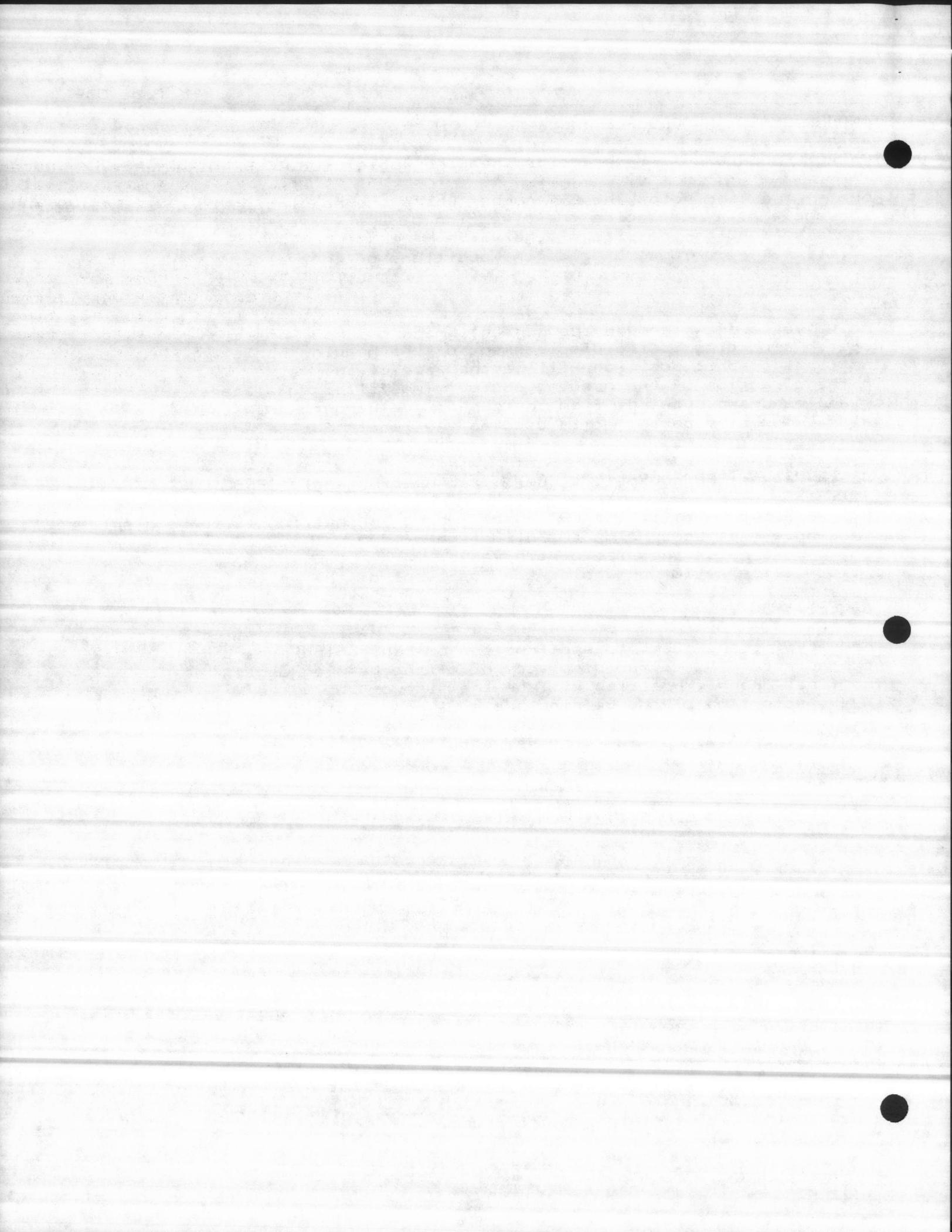
```
Draw_Ctrl_Valve: PROCEDURE (crt_num, size, heading_angle) PUBLIC REENTRANT;
```

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
heading\_angle => a word holds the direction of the valve.  
size => a word holds the size of the valve.  
temp\_list (10) => a 10 word holds the positions of the valve.

## Proc description :

The valve is drawn around the current graphic cursor position rotated according to the displacement of the heading\_angle. The final graphic position will be the side of the valve in the direction of the heading\_angle. Two filled equalateral triangles drawn outwards from the cursor are used to draw the valve. The size parameter is used as the length of the triangle sides, the length of the valve stem, and also the radius of the control knob.

```
END Draw_Ctrl_Valve;
```



```
Draw_Motor: PROCEDURE (crt_num, radius, heading_angle) PUBLIC REENTRANT;
```

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
radius, offset, heading\_angle, offset\_edge\_angle => proc parameters.  
temp\_list (4) => words hold radius and heading\_angles.

Proc description :

The motor is drawn around the current graphic cursor position rotated according to the displacement of the heading\_angle.

The current graphic cursor position is not altered. A filled circle along with two vectors drawn on the edge of the circle at 45 degree displacements is used to create the motor.

The radius parameter is used as the radius of the circular part of the motor and also as the length of the vectors.

```
END Draw_Motor;
```

```
Draw_Pump: PROCEDURE (crt_num, radius, offset, offset_sign,
```

heading\_angle) PUBLIC REENTRANT;

crt\_num => a byte, holds the CRT number where the heading is to be displayed.  
offset\_sign => a byte.

radius, offset, heading\_angle, offset\_edge\_angle,temp\_list (6) => see description

Proc description :

The pump is drawn around the current graphic cursor position rotated according to the displacement of the heading\_angle.

The final graphic cursor position will be placed at the center of the outside edge of the pump output. A filled circle along with bounded filled vectors is used to create the image of the pump. The radius parameter is used as the radius of the circular part of the pump. The offset is used to indicate the pump output offset from the main body radius.

Offset\_sign is a flag that if true, will draw the pump output on top of the heading angle, else it will be drawn underneath.

The pump output is always drawn towards the heading\_angle.

```
END Draw_Pump;
```

```
Draw_Screw_Pump: PROCEDURE (crt_num, blades, radius, heading_angle)
```

PUBLIC REENTRANT;

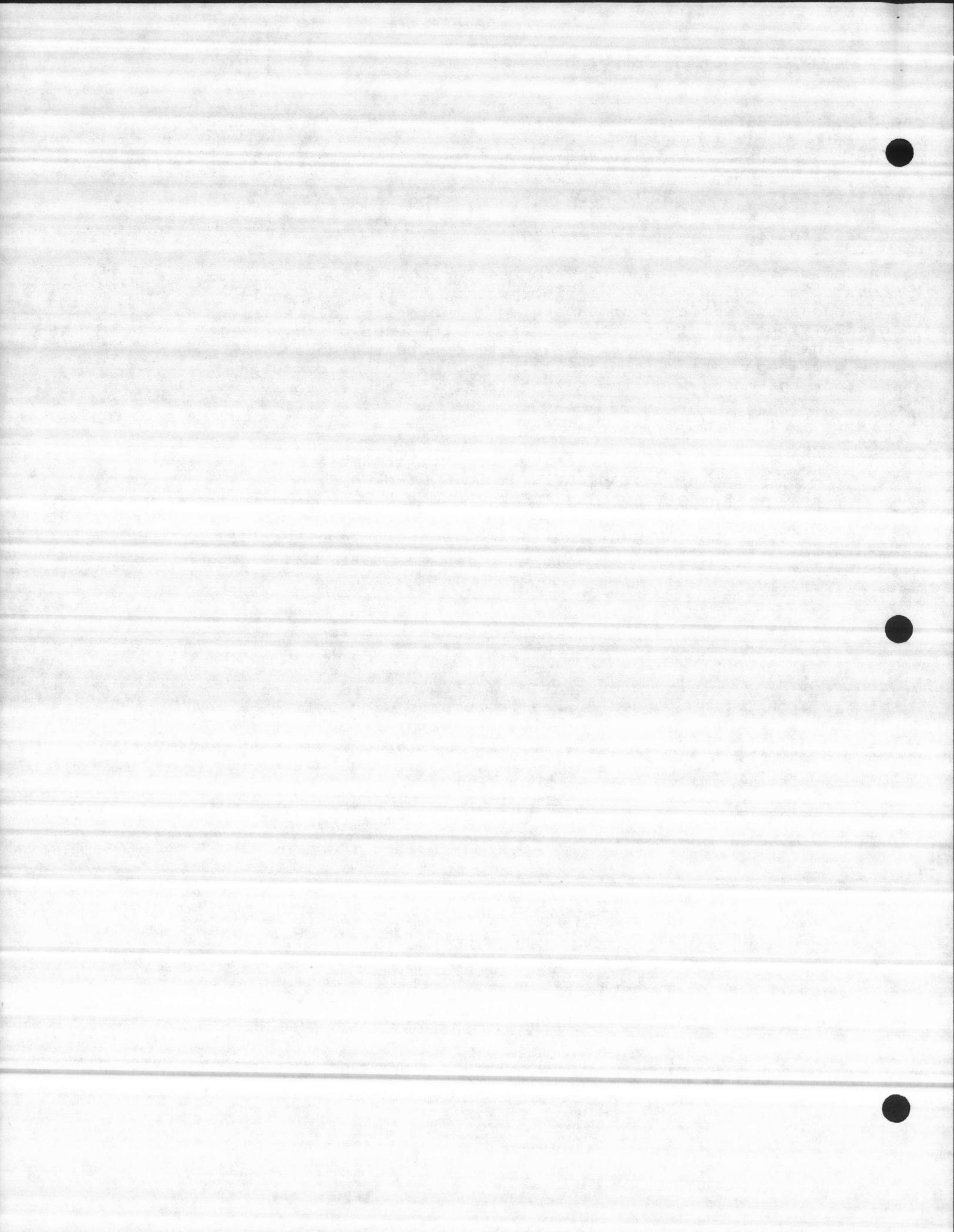
crt\_num => a byte - holds the CRT number where the heading is to be displayed.

blades => a byte - holds the number of blades.

radius, heading\_angle => proc parameter (see description).

Proc description :

The pump is drawn with the radius of the motor around the current graphic rotated according to the heading\_angle. The



final graphic position will be the first auger blade below  
the pump motor. The radius parameter is used for the motor  
radius, and also the blade length and spacing. Nine blades  
are always drawn beneath the pump motor.  
END Draw\_Screw\_Pump;

Draw\_Propeller: PROCEDURE (crt\_num, radius, heading\_angle) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
heading\_angle, radius => proc parameter (see description).

Proc description :

The propeller is drawn around the current graphic position  
rotated according to the displacement of the heading\_angle.  
The final graphic position will be the side of the propeller  
in the direction of the heading\_angle. Two filled  
wedges drawn outwards from the cursor are used to draw the  
prop. The radius parameter is used as the radius of the  
wedges.

END Draw\_Propeller;

Draw\_Gate: PROCEDURE (crt\_num, size, heading\_angle) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
size, heading\_angle => proc parameter (see description).

Proc description :

The gate is drawn around the current graphic cursor position  
rotated according to the displacement of the heading\_angle.  
The final graphic position will be the side of the gate  
in the direction of the heading\_angle. Two filled boxes drawn  
above and below the heading\_angle, with a vector between  
are used to draw the valve. The size parameter is used as  
length of the vector, and the width and height of the boxes.  
END Draw\_Gate;

Draw\_Screen: PROCEDURE (crt\_num, num\_bars, size, distance,  
heading\_angle) PUBLIC REENTRANT;

crt\_num => a byte - holds the CRT number where the heading is to be displayed.

num\_bars=> a byte - holds the number of bars.

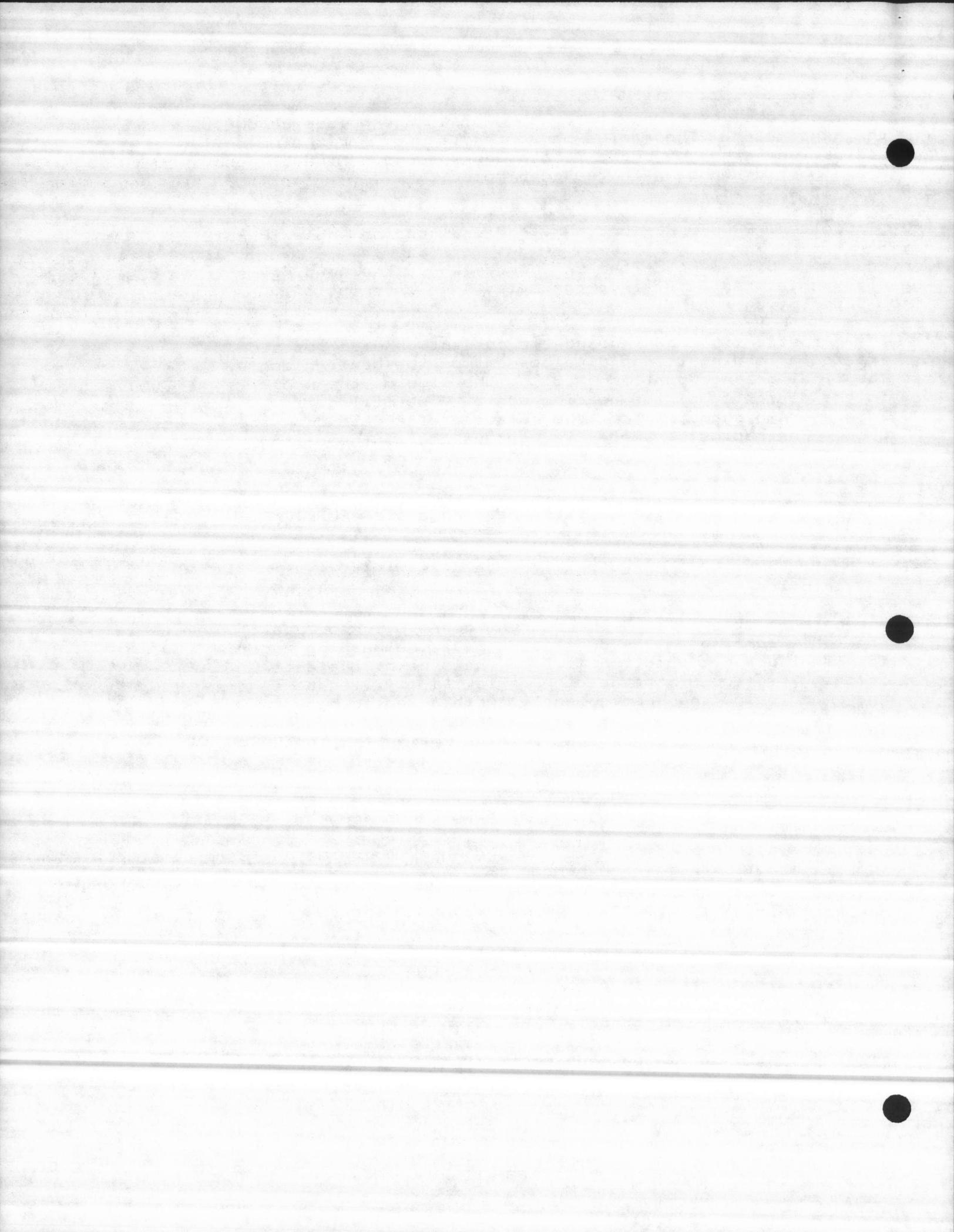
count => a byte holds a counter.

size, heading\_angle, distance => proc paramter (see description).

Proc description :

The screen is drawn from the current graphic cursor position  
rotated according to the displacement of the heading\_angle.  
The final graphic position will be the opposite side of the  
screen from the heading angle. Relative 45 degree angles  
are drawn to create the screen. The size parameter is used  
for the length of the bar angles, and the distance between.

END Draw\_Screen;



Draw\_Arrow\_Head: PROCEDURE (crt\_num, size, heading\_angle) PUBLIC REENTRANT;  
-----  
crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
size, heading\_angle => proc parameter (see description).

Proc description :

-----  
The arrow head is drawn around the current graphic cursor  
rotated according to the displacement of the heading\_angle.  
The final graphic position is not altered. 45 degree angles  
of length size are drawn to create the arrow head.

END Draw\_Arrow\_Head;

Draw\_Stick\_Man: PROCEDURE (crt\_num, size, heading\_angle) PUBLIC REENTRANT;

-----  
crt\_num => a byte - holds the CRT number where the heading is to be displayed.  
size, heading\_angle => see description

Proc description :

-----  
The crotch of the stick man is centered the length of the  
size parameter above the current graphic cursor position and  
drawn upwards rotated according to the displacement of the  
heading\_angle. The final graphic position is not altered.  
The size parameter is used as the length of the arms, legs,  
and neck. The body portion is drawn twice as long as size,  
and the head radius is one third of twice the size.

END Draw\_Stick\_Man;

8) GRAPH DRAWING PROCEDURES :

=====

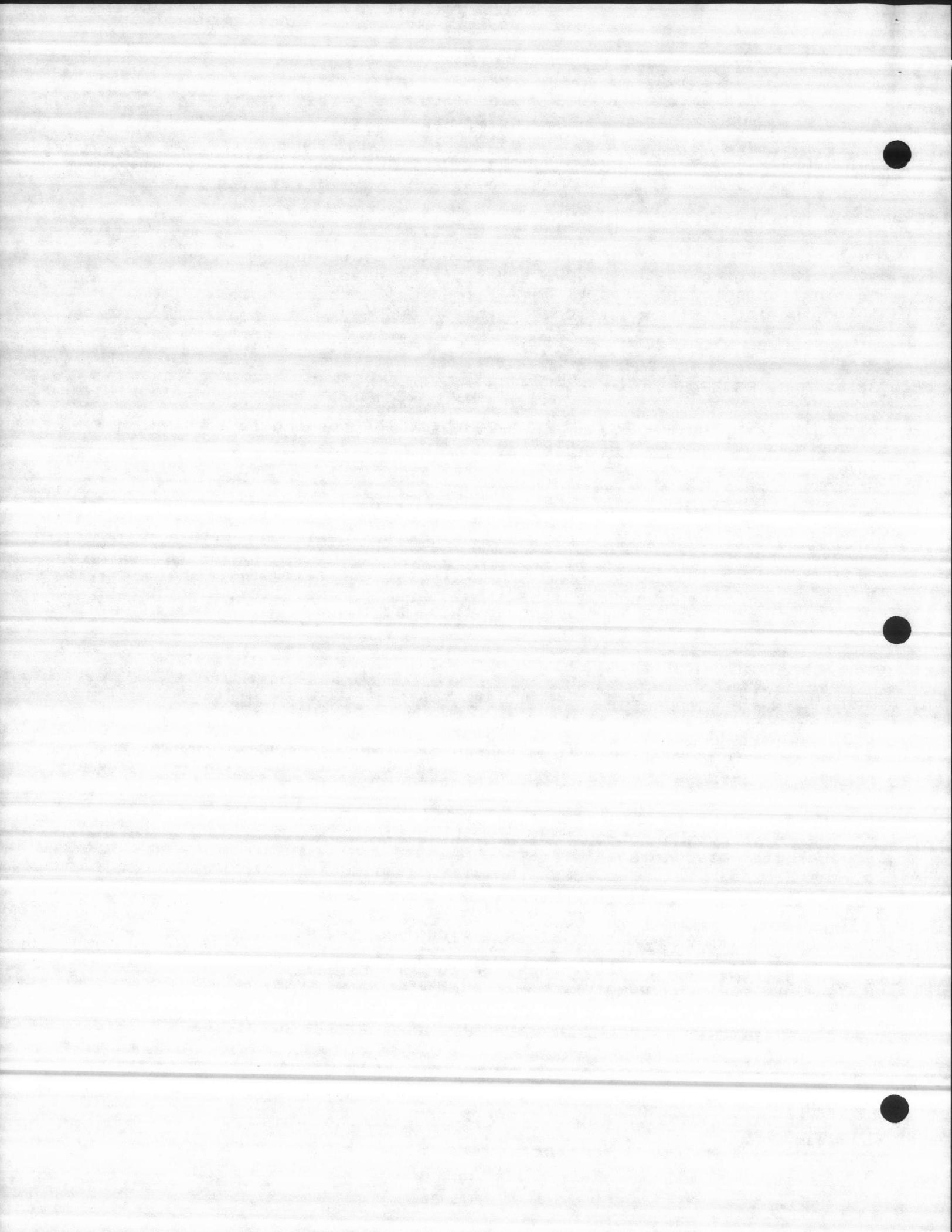
Draw\_Bar\_Graph:PROCEDURE (crt\_num, color\_index, width, height,  
-----  
x\_coord, y\_coord, value, max\_value, min\_value,  
scale, field\_length) PUBLIC REENTRANT;

crt\_num => a byte holds the CRT number where the display should be.  
color\_index => a byte holds an index to the color.  
scale => a byte value holds the scale value.  
field\_length => a byte value holds the field length.  
width => a word value holds the width.  
height => a word value holds the height.  
x\_coord => a word value holds the X coordinate value.  
y\_coord => a word value holds the Y coordinate value.  
current\_height => a word value holds the current height.  
value => an integer represents the value of the point.  
max\_value => an integer represents the max value of the point.  
min\_value => an integer represents the min value of the point.

Proc description :

-----  
draw a bar graph.

END Draw\_Bar\_Graph;



Draw\_X\_Axis: PROCEDURE (crt\_num, color\_index, length, thickness, num\_intervals,  
----- origin\_offset) PUBLIC REENTRANT;  
crt\_num => a byte holds the CRT number where the display should be.  
color\_index => a byte holds an index to the color.

Proc description :

To draw an X axis.

END Draw\_X\_Axis;

Draw\_Y\_Axis: PROCEDURE (crt\_num, color\_index, height, thickness, num\_intervals,  
----- origin\_offset) PUBLIC REENTRANT;  
crt\_num => a byte holds the CRT number where the display should be.  
color\_index => a byte holds an index to the color.

Proc description :

To draw a Y axis

END Draw\_Y\_Axis;

Draw\_Line\_Grid: PROCEDURE (crt\_num, width, height,  
----- x\_intervals, y\_intervals) PUBLIC REENTRANT;  
crt\_num => a byte holds the CRT number where the display should be.  
width => a word value holds the width.  
height => a word value holds the height.

Proc description :

To draw a line grid, which is mostly used in the trends

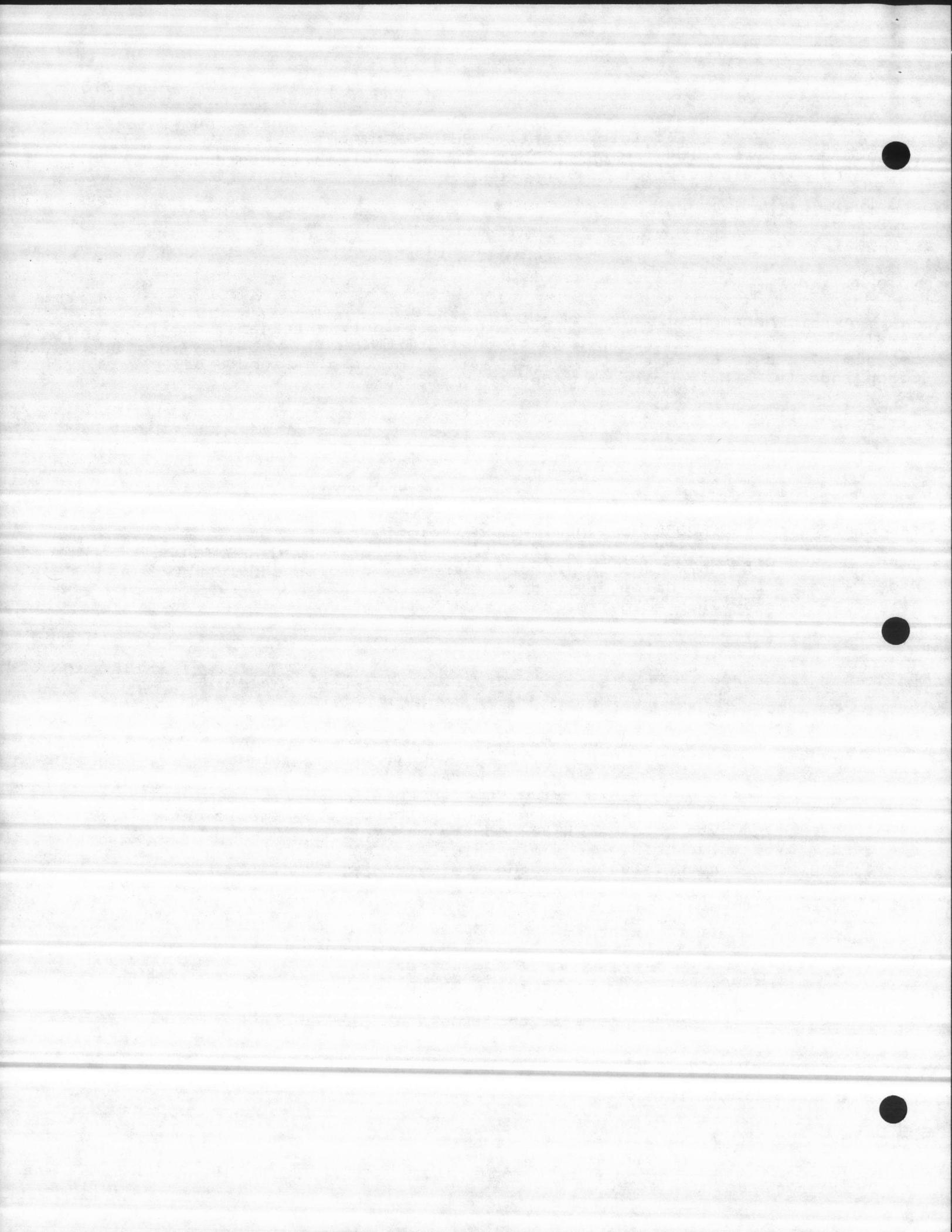
END Draw\_Line\_Grid;

Draw\_Graph\_Line: PROCEDURE (crt\_num, color\_index, width, height,  
----- value\_p, max\_value,  
min\_value, num\_intervals) PUBLIC REENTRANT;  
crt\_num => a byte holds the CRT number where the display should be.  
color\_index => a byte holds an index to the color.  
scale => a byte value holds the scale value.  
field\_length => a byte value holds the field length.  
width => a word value holds the width.  
height => a word value holds the height.  
x\_coord => a word value holds the X coordinate value.  
y\_coord => a word value holds the Y coordinate value.  
current\_height => a word value holds the current height.  
value => an integer represents the value of the point.  
max\_value => an integer represents the max value of the point.  
min\_value => an integer represents the min value of the point.

Proc description :

draw a graph line.

END Draw\_Graph\_Line;



## 9) NON-GRAFIC PROCEDURES :

```
=====
Display_Y_Values: PROCEDURE (crt_num, height, num_intervals, origin_offset,
-----           x_coord, y_coord, max_value, min_value,
           scale, field_length) PUBLIC REENTRANT;
```

crt\_num => a byte holds the CRT number where the display should be.  
color\_index => a byte holds an index to the color.  
scale => a byte value holds the scale value.  
field\_length => a byte value holds the field length.  
width => a word value holds the width.  
height => a word value holds the height.  
x\_coord => a word value holds the X coordinate value.  
y\_coord => a word value holds the Y coordinate value.  
current\_height => a word value holds the current hieghth.  
value => an integer represents the value of the point.  
max\_value => an integer represents the max value of the point.  
min\_value => an integer represents the min value of the point.

## Proc description :

-----  
displays the Y values on the positions of Y axis..

```
END Display_Y_Values;
```

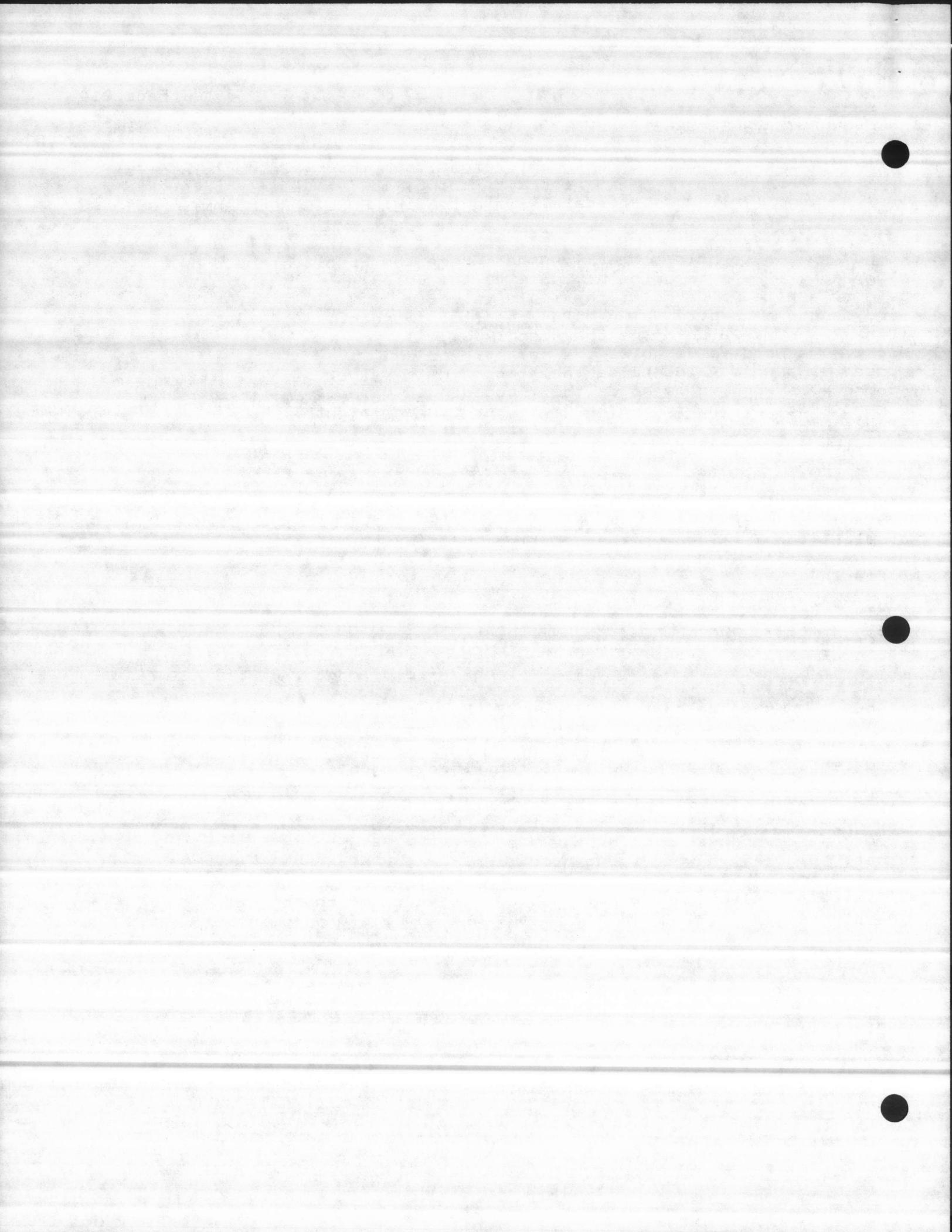
```
=====
Display_X_Times: PROCEDURE (crt_num, width, num_intervals, origin_offset,
-----           x_coord, y_coord, start_time, time_interval,
           time_str_index, field_length) PUBLIC REENTRANT;
```

crt\_num => a byte holds the CRT number where the display should be.  
color\_index => a byte holds an index to the color.  
scale => a byte value holds the scale value.  
field\_length => a byte value holds the field length.  
width => a word value holds the width.  
height => a word value holds the height.  
x\_coord => a word value holds the X coordinate value.  
y\_coord => a word value holds the Y coordinate value.  
current\_height => a word value holds the current hieghth.  
value => an integer represents the value of the point.  
max\_value => an integer represents the max value of the point.  
min\_value => an integer represents the min value of the point.

## Proc description :

-----  
displays the X times on the position of x axis.

```
END Display_X_Times;
```



(15.2) MODULE NAME : CxxxxGRPH.EXT

=====

DESCRIPTION :

===== This module contains all the external procedures of its  
public of the CxxxxGRPH.Pyy

#### DRAWING PARAMETER PROCEDURES

```
Set_Brush_Width: PROCEDURE (crt_num, x_width, y_width) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (x_width, y_width) WORD;  
END Set_Brush_Width;
```

```
Set_Graphic_Color: PROCEDURE (crt_num, color_index, x_mode) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (color_index, x_mode) WORD;  
END Set_Graphic_Color;
```

```
Set_Fill_Type: PROCEDURE (crt_num, pattern, background) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (pattern, background) WORD;  
END Set_Fill_Type;
```

```
Set_Vector_Type: PROCEDURE (crt_num, pattern) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (pattern) WORD;  
END Set_Vector_Type;
```

#### CURSOR POSITIONING PROCEDURES

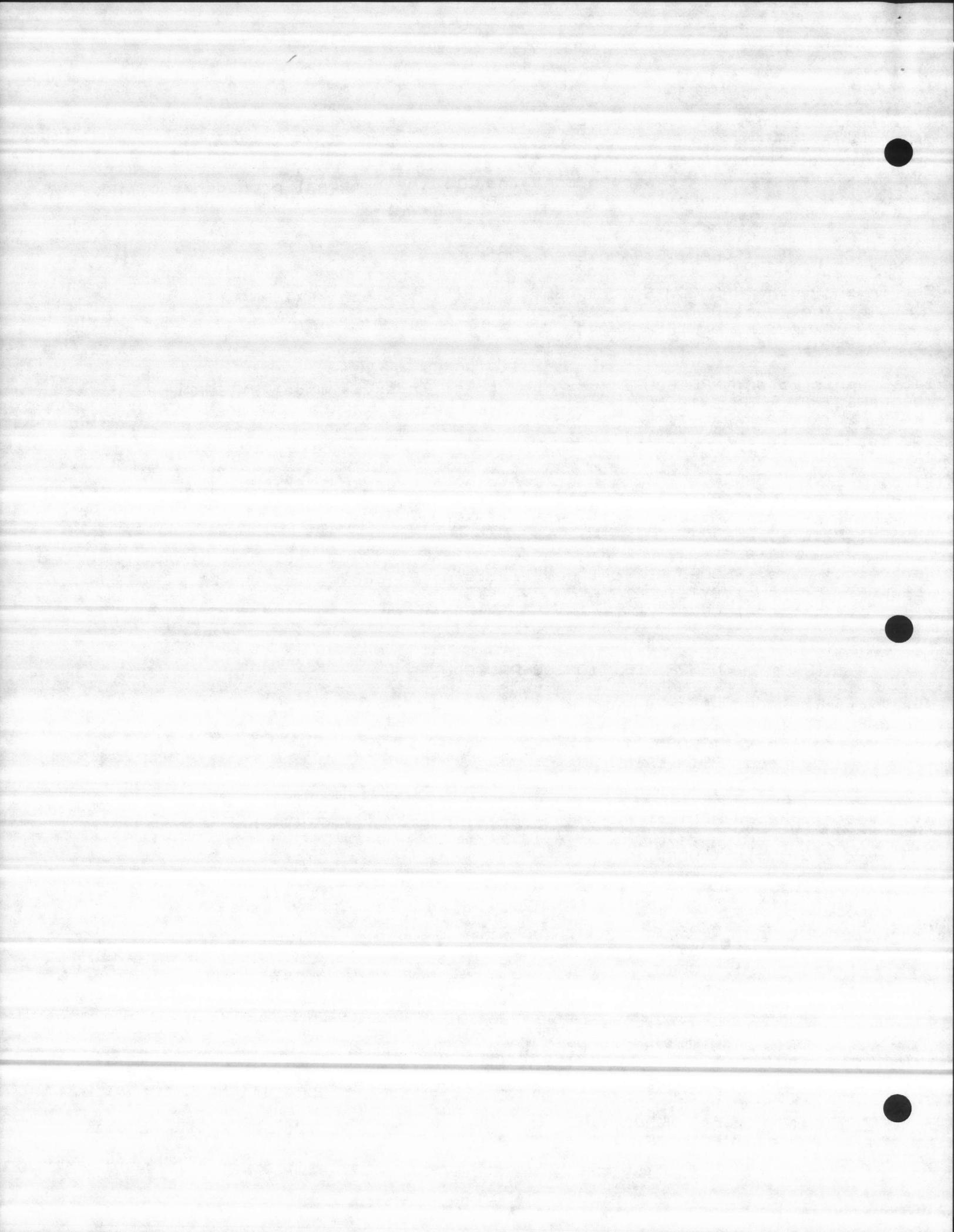
```
Abs_Cursor: PROCEDURE (crt_num, num_moves, coord_p) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (num_moves) WORD,  
       (coord_p) POINTER;  
END Abs_Cursor;
```

```
Rel_Cursor: PROCEDURE (crt_num, num_moves, coord_p) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (num_moves) WORD,  
       (coord_p) POINTER;  
END Rel_Cursor;
```

```
Polar_Cursor: PROCEDURE (crt_num, num_moves, coord_p) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (num_moves) WORD,  
       (coord_p) POINTER;  
END Polar_Cursor;
```

#### LINE DRAWING PROCEDURES

```
Draw_Abs_Vector: PROCEDURE (crt_num, num_vectors, coord_p) EXTERNAL;  
DECLARE (crt_num) BYTE,  
       (num_vectors) WORD,  
       (coord_p) POINTER;  
END Draw_Abs_Vector;
```



```
Draw_Rel_Vector: PROCEDURE (crt_num, num_vectors, coord_p) EXTERNAL;
DECLARE (crt_num) BYTE,
       (num_vectors) WORD,
       (coord_p) POINTER;
END Draw_Rel_Vector;

Draw_Polar_Vector: PROCEDURE (crt_num, num_vectors, coord_p) EXTERNAL;
DECLARE (crt_num) BYTE,
       (num_vectors) WORD,
       (coord_p) POINTER;
END Draw_Polar_Vector;
```

#### GEOMETRIC FIGURE DRAWING PROCEDURES

```
Draw_Bar: PROCEDURE (crt_num, width, height, fill,
                      xtip, ytip) EXTERNAL;
DECLARE (crt_num) BYTE,
        (width, height, fill, xtip, ytip) WORD;
END Draw_Bar;
```

```
Draw_Arc: PROCEDURE (crt_num, radius, angle1, angle2, option) EXTERNAL;
DECLARE (crt_num) BYTE,
        (radius, angle1, angle2, option) WORD;
END Draw_Arc;
```

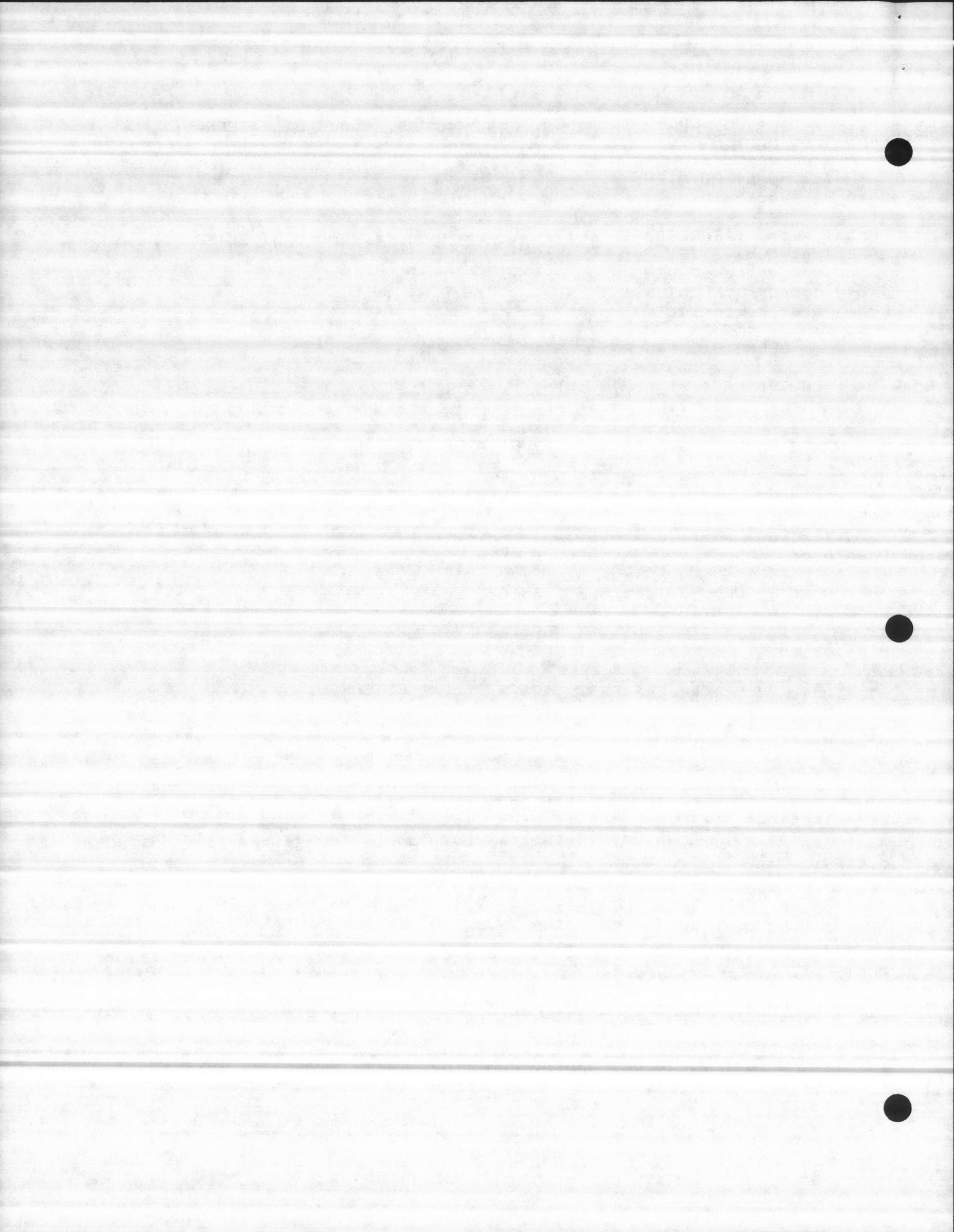
```
Draw_Rel_Arc: PROCEDURE (crt_num, radius, angle, option) EXTERNAL;
DECLARE (crt_num) BYTE,
        (radius, angle, option) WORD;
END Draw_Rel_Arc;
```

```
Draw_Circle: PROCEDURE (crt_num, radius, fill) EXTERNAL;
DECLARE (crt_num) BYTE,
        (radius, fill) WORD;
END Draw_Circle;
```

```
Draw_Abs_Elip_Arc: PROCEDURE (crt_num, x_radius, y_radius, angle1, angle2,
                               angle, num_sides, option) EXTERNAL;
DECLARE (crt_num) BYTE,
        (x_radius, y_radius, angle1, angle2, angle, num_sides, option) WORD;
END Draw_Abs_Elip_Arc;
```

```
Draw_Rel_Elip_Arc: PROCEDURE (crt_num, x_radius, y_radius, degrees,
                               angle, num_sides, option) EXTERNAL;
DECLARE (crt_num) BYTE,
        (x_radius, y_radius, degrees, angle, num_sides, option) WORD;
END Draw_Rel_Elip_Arc;
```

```
Draw_Ellipse: PROCEDURE (crt_num, x_radius, y_radius, angle,
                         num_sides, fill) EXTERNAL;
DECLARE (crt_num) BYTE,
        (x_radius, y_radius, angle, num_sides, fill) WORD;
END Draw_Ellipse;
```



## FILL PROCEDURES

```
Draw_Abs_Fill: PROCEDURE (crt_num, num_coord, coord_p) EXTERNAL;
DECLARE (crt_num) BYTE,
        (num_coord) WORD,
        (color_p, coord_p) POINTER;
END Draw_Abs_Fill;

Draw_Rel_Fill: PROCEDURE (crt_num, num_coord, coord_p) EXTERNAL;
DECLARE (crt_num) BYTE,
        (num_coord) WORD,
        (color_p, coord_p) POINTER;
END Draw_Rel_Fill;

Draw_Bounded_Fill: PROCEDURE (crt_num, pattern, background) EXTERNAL;
DECLARE (crt_num) BYTE,
        (pattern, background) WORD;
END Draw_Bounded_Fill;
```

## STANDARD SYMBOL PROCEDURES

```
Draw_Valve: PROCEDURE (crt_num, size, heading_angle) EXTERNAL;
DECLARE (crt_num, count) BYTE,
        (heading_angle, size) WORD;
END Draw_Valve;

Draw_Check_Valve: PROCEDURE (crt_num, size, heading_angle) EXTERNAL;
DECLARE (crt_num, count) BYTE,
        (heading_angle, size) WORD;
END Draw_Check_Valve;

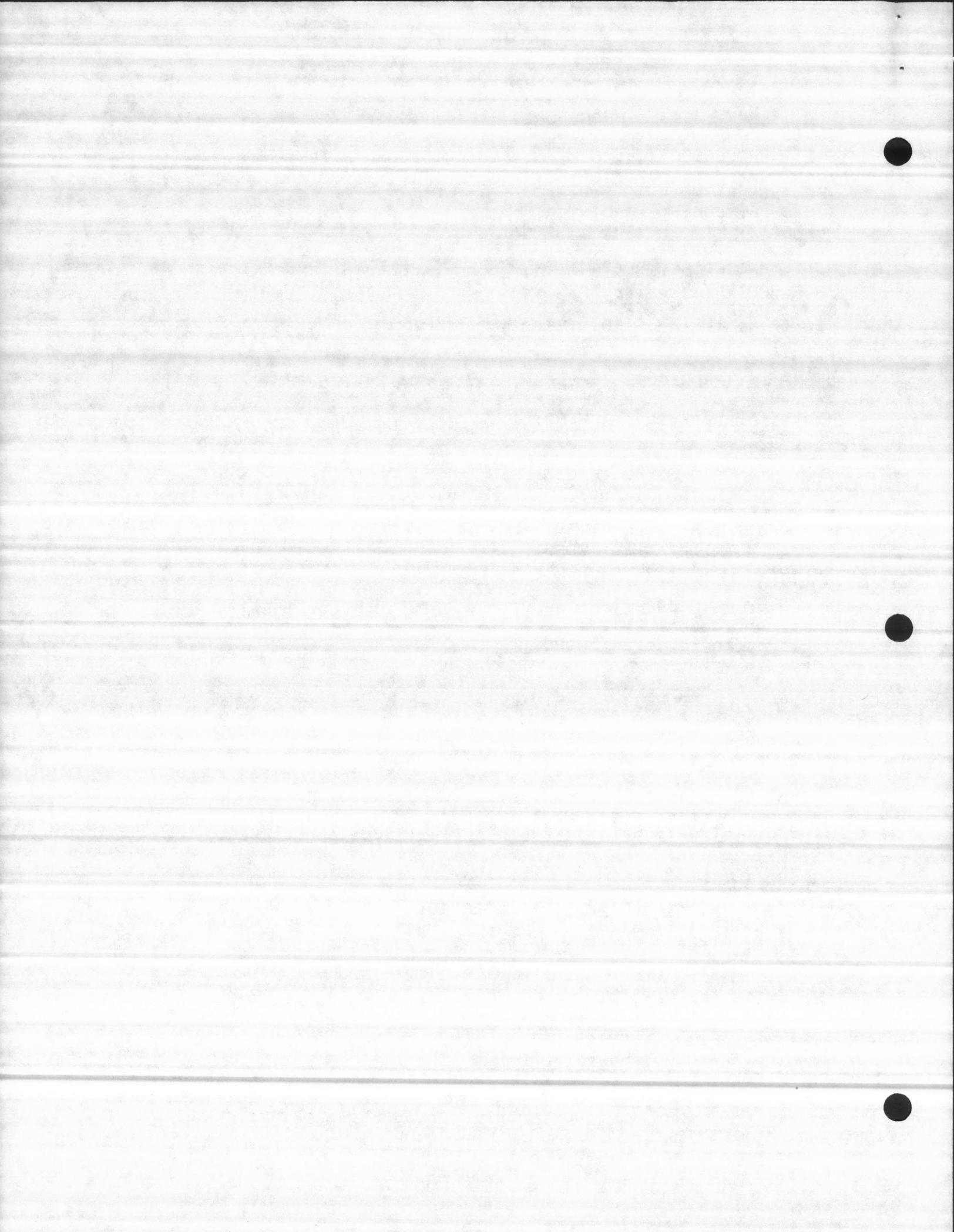
Draw_Ctrl_Valve: PROCEDURE (crt_num, size, heading_angle) EXTERNAL;
DECLARE (crt_num, count) BYTE,
        (heading_angle, size) WORD;
END Draw_Ctrl_Valve;

Draw_Motor: PROCEDURE (crt_num, radius, heading_angle) EXTERNAL;
DECLARE (crt_num, offset_sign) BYTE,
        (radius, offset, heading_angle, offset_edge_angle) WORD;
END Draw_Motor;

Draw_Pump: PROCEDURE (crt_num, radius, offset, offset_sign,
                      heading_angle) EXTERNAL;
DECLARE (crt_num, offset_sign) BYTE,
        (radius, offset, heading_angle, offset_edge_angle) WORD;
END Draw_Pump;

Draw_Screw_Pump: PROCEDURE (crt_num, blades, radius, heading_angle)
                           EXTERNAL;
DECLARE (crt_num, blades) BYTE,
        (radius, heading_angle) WORD;
END Draw_Screw_Pump;

Draw_ProPELLER: PROCEDURE (crt_num, radius, heading_angle) EXTERNAL;
DECLARE (crt_num) BYTE,
        (heading_angle, radius) WORD;
END Draw_ProPELLER;
```



```

Draw_Gate: PROCEDURE (crt_num, size, heading_angle) EXTERNAL;
DECLARE (crt_num, count) BYTE,
        (size, heading_angle) WORD,
END Draw_Gate;

Draw_Screen: PROCEDURE (crt_num, num_bars, size, distance,
                       heading_angle) EXTERNAL;
DECLARE (crt_num, num_bars, count) BYTE,
        (size, heading_angle, distance) WORD,
END Draw_Screen;

Draw_Graphic_Lite: PROCEDURE (crt_num, radius, heading_angle, num_sections,
                             color_list_p, status_p_list_p) EXTERNAL;
DECLARE (crt_num, num_sections, arc_fill, count) BYTE,
        (radius, heading_angle, arc_angle) WORD,
        (color_list_p, status_p_list_p, status_p) POINTER,
        (color_list BASED color_list_p) (6) BYTE,
        (status_p_list BASED status_p_list_p) (6) POINTER,
        (status BASED status_p) BYTE;
END Draw_Graphic_Lite;

Draw_Arrow_Head: PROCEDURE (crt_num, size, heading_angle) EXTERNAL;
DECLARE (crt_num, count) BYTE,
        (size, heading_angle) WORD;
END Draw_Arrow_Head;

Draw_Stick_Man: PROCEDURE (crt_num, size, heading_angle) EXTERNAL;
DECLARE (crt_num) BYTE,
        (size, heading_angle) WORD;
END Draw_Stick_Man;

```

#### GRAPH DRAWING PROCEDURES

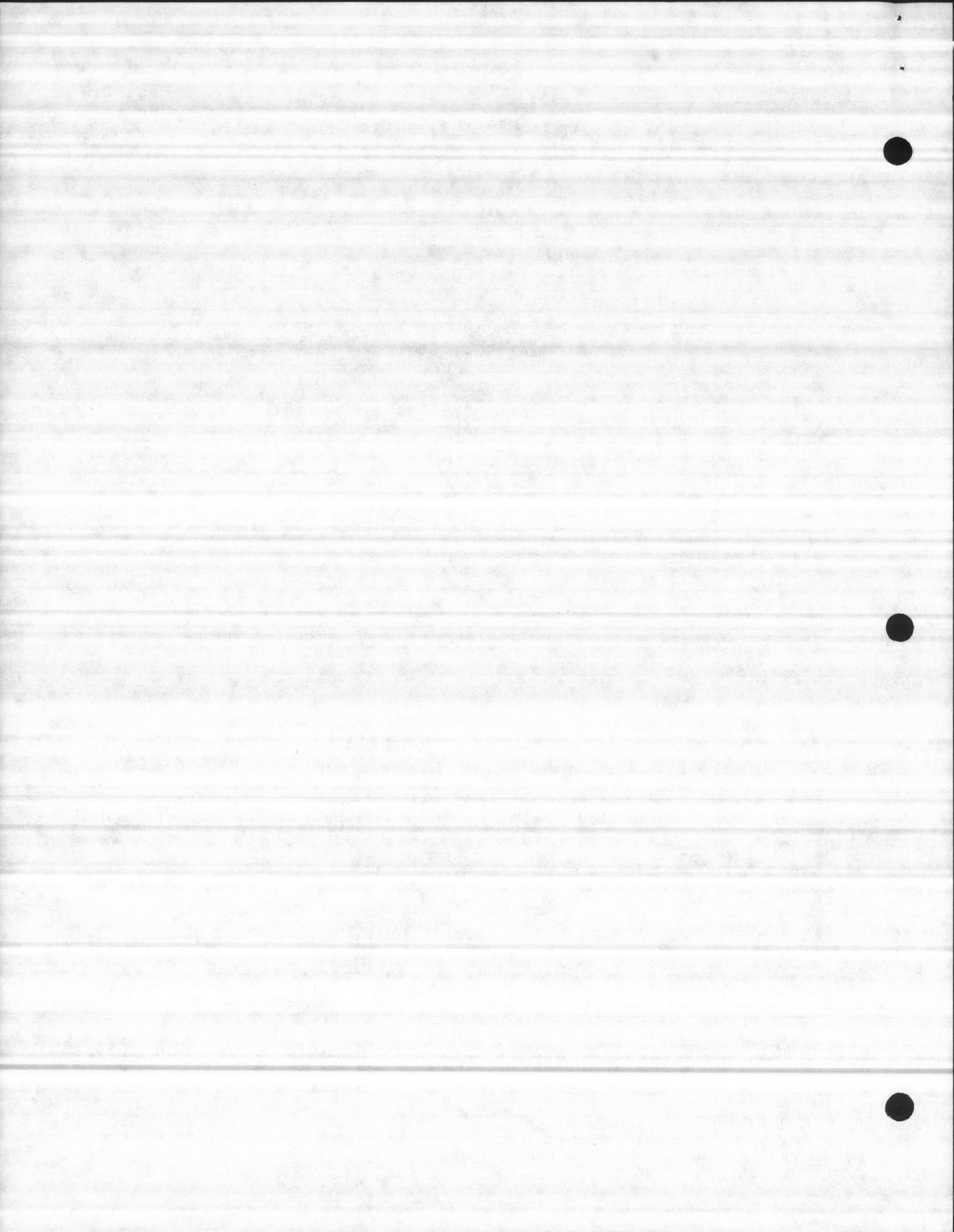
```

Draw_Bar_Graph:PROCEDURE (crt_num, color_index, width, height,
                          x_coord, y_coord, value, max_value, min_value,
                          scale, field_length) EXTERNAL;
DECLARE (crt_num, color_index, scale, field_length) BYTE,
        (width, height, x_coord, y_coord) WORD,
        (value, max_value, min_value) INTEGER;
END Draw_Bar_Graph;

Draw_X_Axis: PROCEDURE (crt_num, color_index, length, thickness, num_intervals,
                       origin_offset) EXTERNAL;
DECLARE (crt_num, color_index, thickness, origin_offset) BYTE,
        (length, num_intervals) WORD;
END Draw_X_Axis;

Draw_Y_Axis: PROCEDURE (crt_num, color_index, height, thickness, num_intervals,
                       origin_offset) EXTERNAL;
DECLARE (crt_num, color_index, num_intervals, thickness,
        origin_offset, count) BYTE,
        (height) WORD,
        (real_y_step, real_height) REAL;
END Draw_Y_Axis;

```



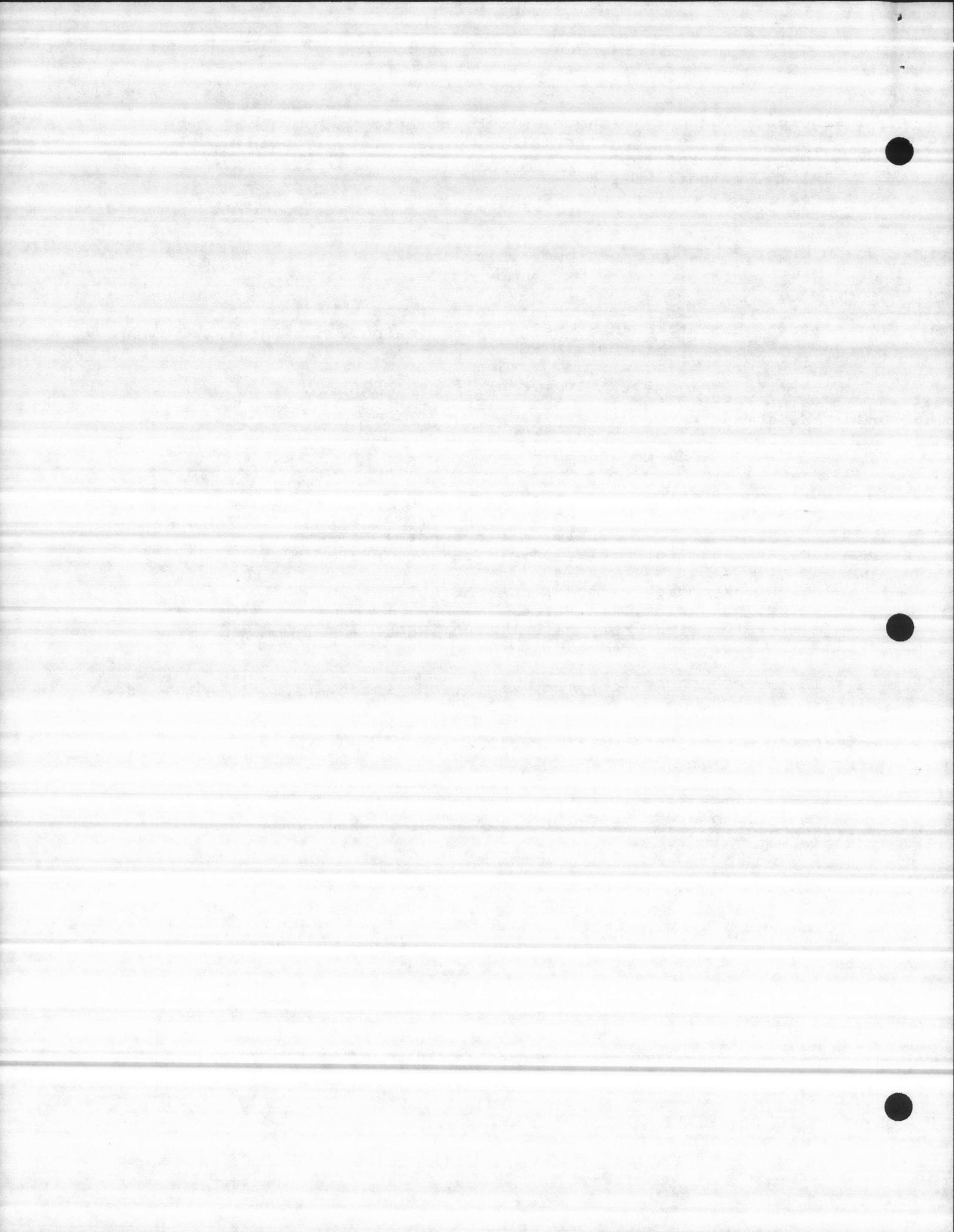
```
Draw_Line_Grid: PROCEDURE (crt_num, width, height,
                           x_intervals, y_intervals) EXTERNAL;
DECLARE (crt_num, x_intervals, y_intervals, count) BYTE,
        (width, height) WORD,
        (real_step, real_count) REAL,
END Draw_Line_Grid;

Draw_Graph_Line: PROCEDURE (crt_num, color_index, width, height,
                           value_p, max_value, min_value,
                           num_intervals) EXTERNAL;
DECLARE (crt_num, color_index, count, dash) BYTE,
        (width, height, current_height, prev_height, num_intervals) WORD,
        (max_value, min_value) INTEGER,
        (value_p) POINTER,
        (real_x_step, real_y_step, real_length) REAL,
        (value BASED value_p) (256) INTEGER;
END Draw_Graph_Line;

NON-GRAFIC PROCEDURES

Display_Y_Values: PROCEDURE (crt_num, height, num_intervals, origin_offset,
                             x_coord, y_coord, max_value, min_value,
                             scale, field_length) EXTERNAL;
DECLARE (crt_num, num_intervals, origin_offset, scale, field_length,
        count) BYTE,
        (height, x_coord, y_coord, current_height) WORD,
        (value, max_value, min_value) INTEGER,
        (real_value_step, real_height_step) REAL,
END Display_Y_Values;

Display_X_Times: PROCEDURE (crt_num, width, num_intervals, origin_offset,
                            x_coord, y_coord, start_time, time_interval,
                            time_str_index, field_length) EXTERNAL;
DECLARE (crt_num, num_intervals, origin_offset, time_str_index, field_length,
        count) BYTE,
        (width, x_coord, y_coord, current_width) WORD,
        (start_time, time_interval, current_time) DWORD,
        (real_width_step) REAL,
END Display_X_Times;
```



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 16

T R E N D  
S Y S T E M

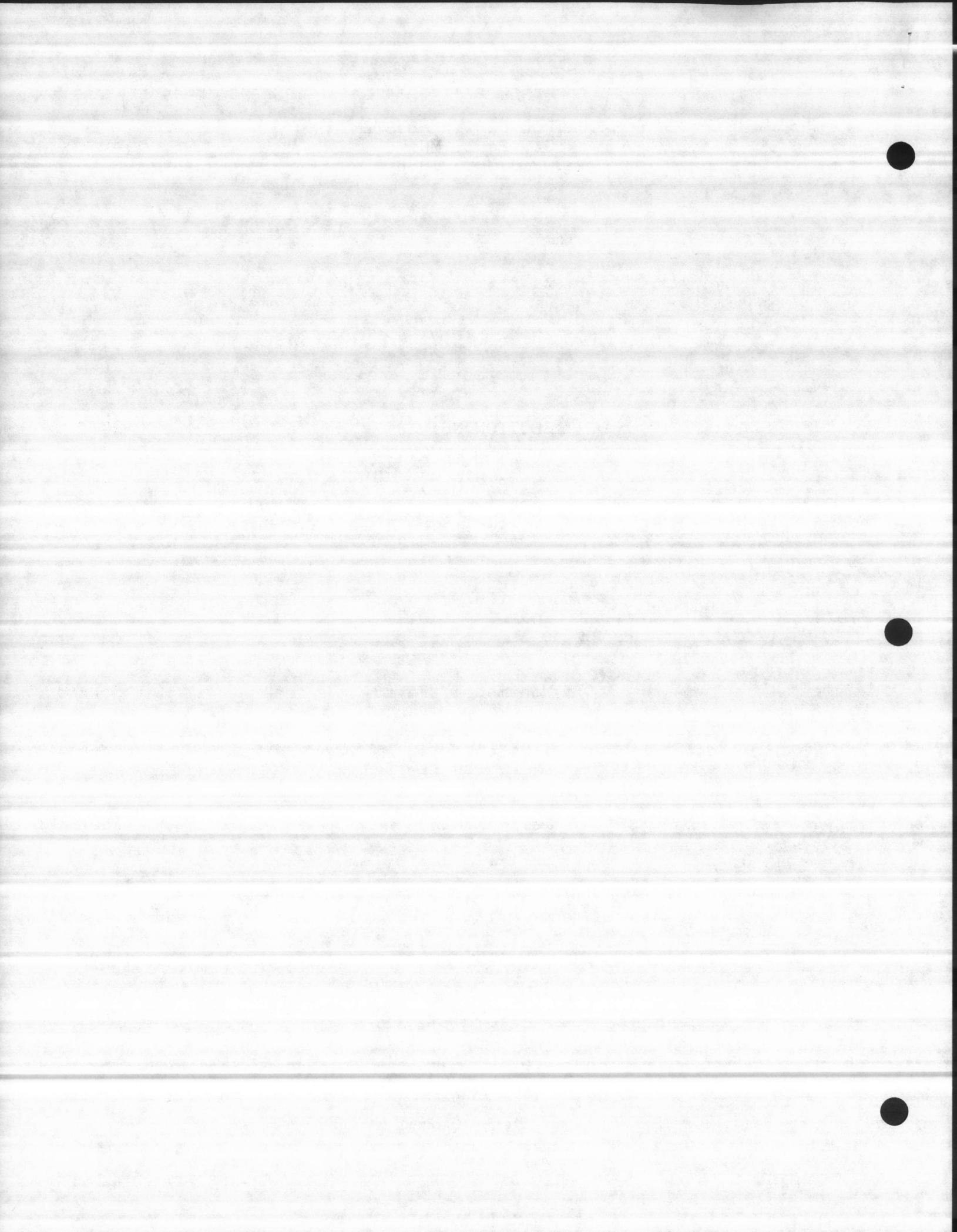
BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth



JOB # : C4758

-----  
TRND SYSTEM - Written by Peter Wollenzien, Bob Ryan, Greg Jensen.

(16.1) MODULE NAME : CxxxxTRND.Pyy

=====

DESCRIPTION :

=====

Used to initailize trend system on powerup and reset trend values at end of day. This procedure will support only the three specific trend data structures. A more generalized system could be written with more time.

This procedure should be called from the INIT procedure with the following calls :

```
CALL Init_Trend (0, 0, 1);
CALL Init_Trend (0, 1, 1);
CALL Init_Trend (1, 0, 1);
CALL Init_Trend (1, 1, 1);
CALL Init_Trend (2, 0, 1);
CALL Init_Trend (2, 1, 1);
CALL Init_Trend (3, 0, 1);
CALL Init_Trend (3, 1, 1);
CALL Init_Trend (4, 0, 1);
CALL Init_Trend (4, 1, 1);
```

At the end of day all CURR\_ files should be copied to a date encoded named file and over the PREV\_ files. Then the following calls should take place to re-initialize the CURR\_ time period files.

```
CALL Init_Trend (0, 0, 0);
CALL Init_Trend (1, 0, 0);
CALL Init_Trend (2, 0, 0);
CALL Init_Trend (3, 0, 0);
CALL Init_Trend (4, 0, 0);
```

Collect\_Trend :

This procedure is used to collect trends and save these trends on disk. The procedures written here will need to be fit into other system functions (probably in TOTL tasks).

Retrieve\_Trend :

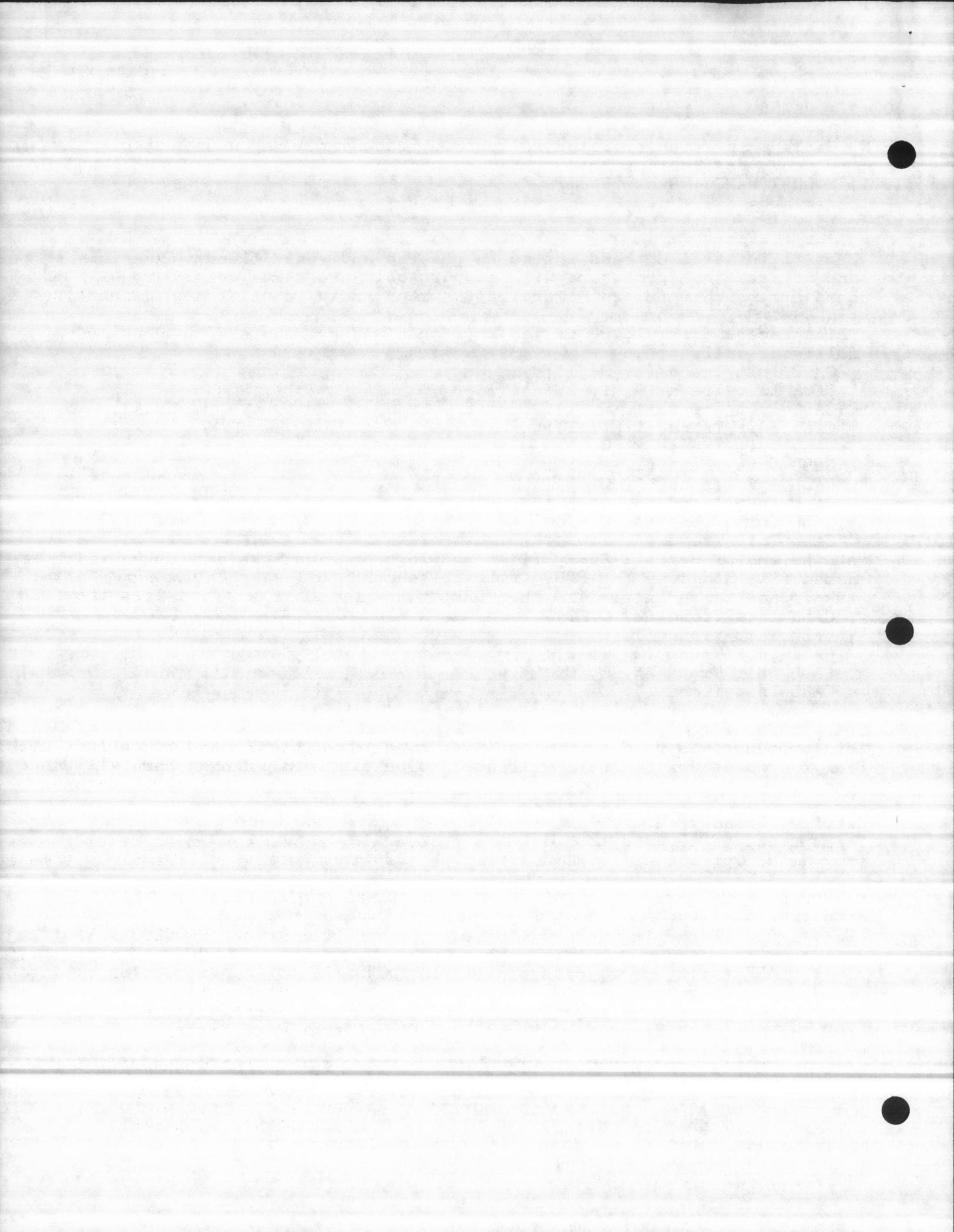
This procedure is used to bring a single analog's trend entry in from disk. Options will include time period windowing.

EXTERNAL PROCEDURES :

=====

```
Format_Disk_Error: PROCEDURE (buffer_t, message_str_p,
                               disk_error, disk_operation) EXTERNAL;
DECLARE (disk_error, disk_operation) WORD,
        (buffer_t)                      TOKEN,
        (message_str_p)                 POINTER;
END Format_Disk_Error;
```

```
Display_Table: PROCEDURE (table_p, param_p, table_reg_t,
                          buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p)      POINTER;
END Display_Table;
```



```
Prompt_Entry: PROCEDURE (max_count, crt_num) EXTERNAL;
    DECLARE (max_count, crt_num) BYTE;
END Prompt_Entry;
```

```
Clear_Screen: PROCEDURE (crt_num) EXTERNAL;
    DECLARE (crt_num) BYTE;
END Clear_Screen;
```

#### GLOBAL REFERENCE :

=====

See all the structure in CxxxxTPUB.Pyy

#### MODULE DECLARATION :

=====

```
rename_file_param BASED disk_param_seg_t STRUCTURE &
write_file_param  BASED disk_param_seg_t STRUCTURE &
write_block_param BASED disk_param_seg_t STRUCTURE &
read_block_param  BASED disk_param_seg_t STRUCTURE
(See DISK SYSTEM)
```

```
== Used for displaying the trend data value by value.
keyin_trend (3) STRUCTURE (index WORD,
                           interval BYTE,
                           min_max_avg BYTE,
                           window BYTE,
                           file_name (60) BYTE);
```

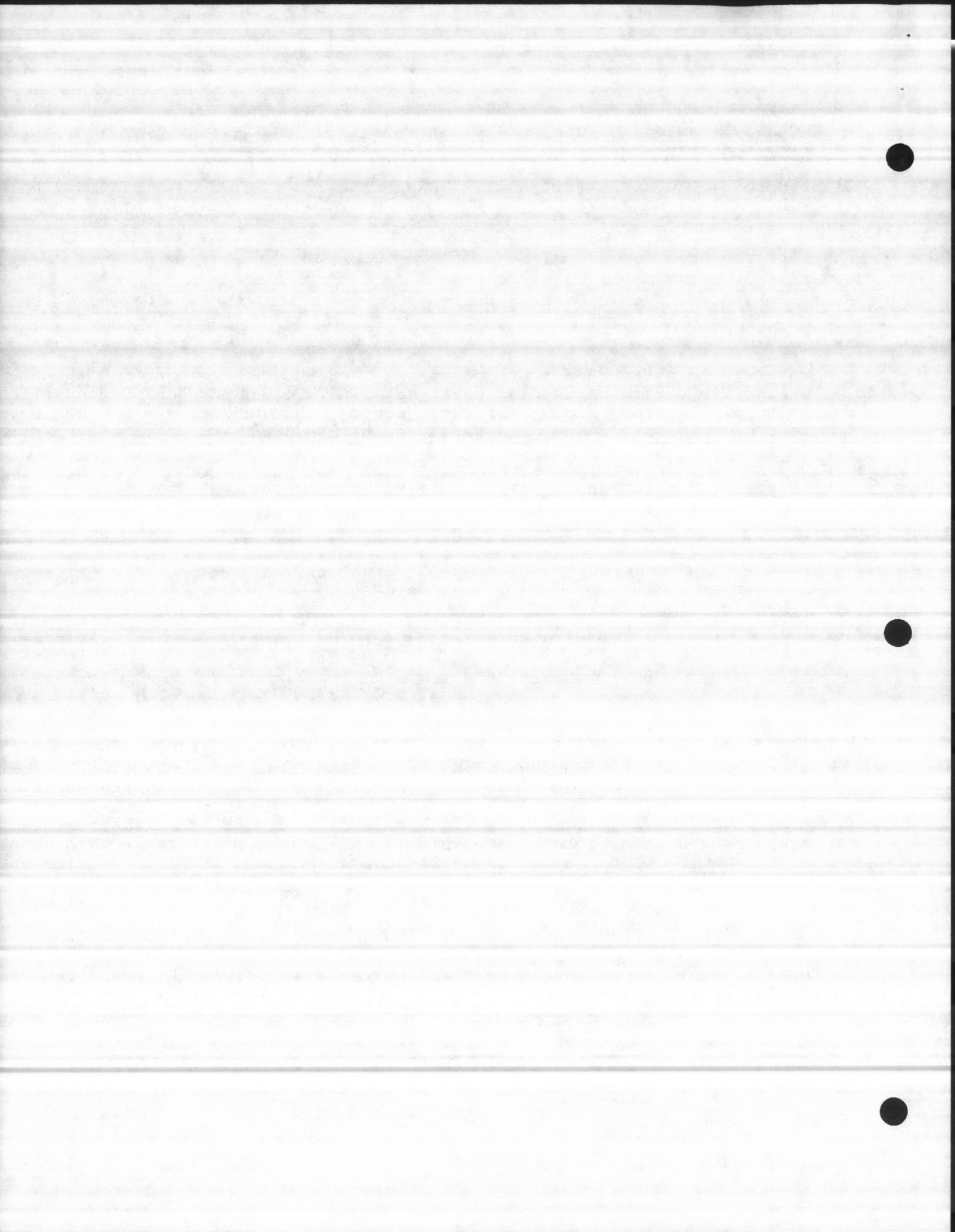
```
trend (3) STRUCTURE (index WORD,
                      interval BYTE,
                      min_max_avg BYTE,
                      window BYTE,
                      file_name (60) BYTE,
                      datta (721) WORD);
```

```
trend_edit_change (3) WORD;
time_table (3) STRUCTURE (datta (19) DWORD);
exception      => Three words, each holds the exception condition.
crt_update_sem_t => Three tokens, each used to create the crt update
                     semaphore.
trend_data_update_table => A pointer - the address to
                           Trend_Data_Update_Task
trend_data_delete_table => A pointer - the address to
                           Trend_Data_Delete_Task.
```

```
trend_crt_table (109) STRUCTURE :
```

-----

This structure is the trend crt display table which displays 108 entries of trend data.



LOCAL PROCEDURES :

=====

Retrieve\_Print\_Error:  
+++++  
Proc description :

-----

Prints the trending error messages.

END Retrieve\_Print\_Error;

PUBLIC PROCEDURES :

=====

Init\_Trend:  
+++++  
Input Parameter :

-----

trend\_type : value for specifically structured trend -  
0 = daily trend - 721 integer entries,  
1 = weekly trend - 255 (85 \* 3) integer entries,  
2 = monthly trend - 96 (32 \* 3) integer entries.  
3 = hourly average trend - 75 (25 \* 3) integer entries.  
4 = hourly total trend - 25 dword entries.

name\_type : flag to indicate CURR\_ or PREV\_ file -  
0 = CURR\_ file  
1 = PREV\_ file

existence\_check\_flag : byte for powerup initialization -  
if flag set and file exists, will not default initialize file

Output Parameters :

----- None.

File Name indications.

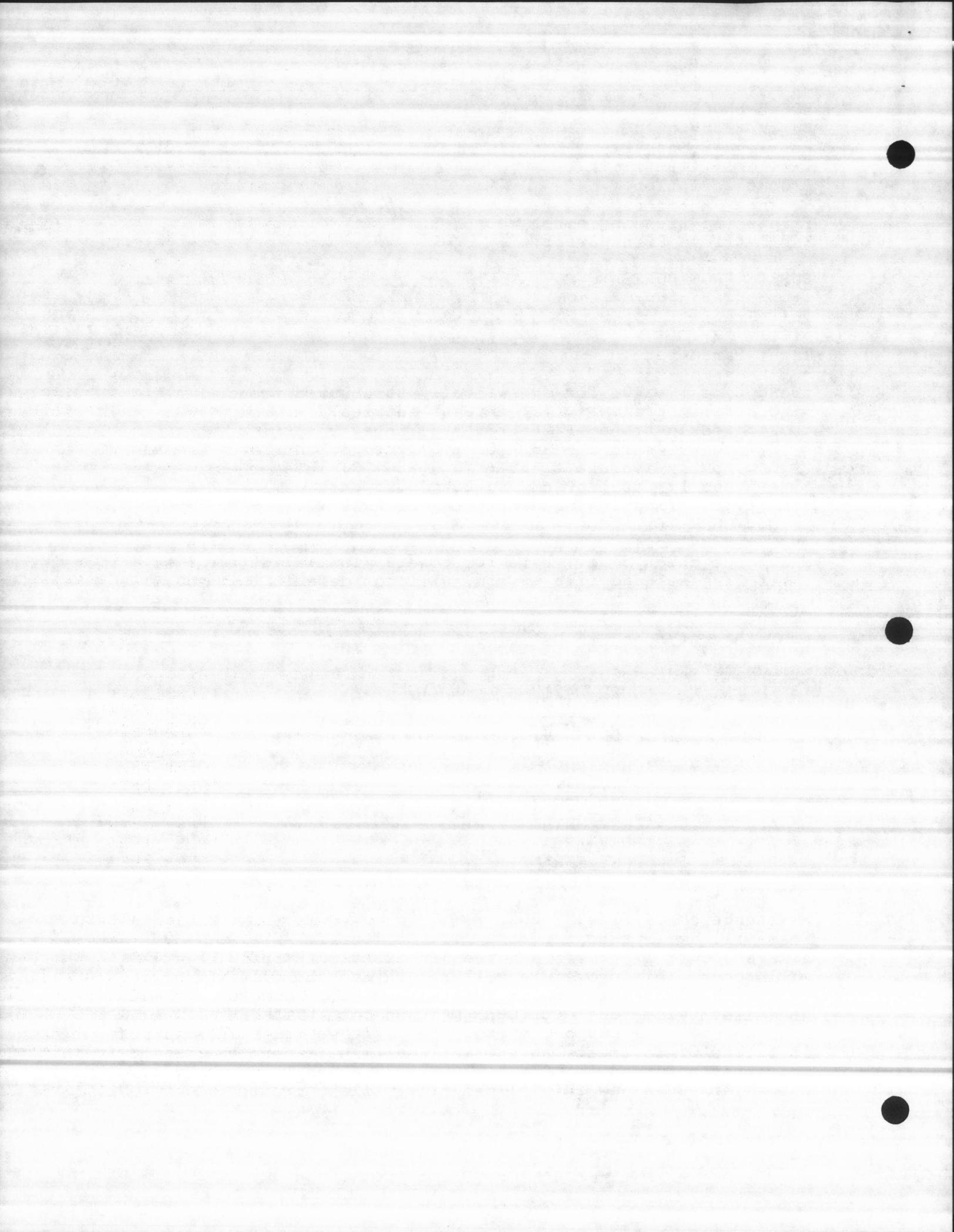
'CURR\_DAY'  
'CURR\_WEEK'  
'CURR\_MNTH'  
'CURR\_AHR'  
'CURR\_FHR'  
'PREV\_DAY'  
'PREV\_WEEK'  
'PREV\_MNTH'  
'PREV\_AHR'  
'PREV\_FHR'

WARNING :

=====

IF (existence\_check\_flag) THEN

This will attempt to rename a file to itself. An error will always occur. If the file currently exists a E\$FEXIST error will occur. This procedure will return and the file will not be initialized. This function will be used on powerup and will be called twice (one with CURR\_DAY and once with PREV\_DAY as file names). The purpose of this is to leave the file as is if it exists or to create an initialized file if none exists. If the error returned is E\$FNEXIST it is assumed that the file does not exist and a new initialized file will be created. \*/



## Proc description :

- 1. defines the file name pointer based on the trend type.
- 2. Initializes all of the trend data buffers.
- two\_minute trend data.
- one\_hour trend data.
- two\_hour trend data.
- 3. Renames the trend files based on the file name pointer.  
An error returns if the new file name exists (i.e. file exists).
- 4. Creates data segment for
  - a) the default zeroth element based on the trend\_type & the number of bytes per element.
- daily - bytes\_per\_element = 1442; (720 integer trend entries )  
 weekly - bytes\_per\_element = 510; (3 X 85 integer trend entries )  
 monthly - bytes\_per\_element = 192; (3 X 32 integer trend entries )  
 hourly Avr - bytes\_per\_element = 150; (3 X 25 integer trend entries )  
 hourly tot - bytes\_per\_element = 100; (25 dword trend entries )
- b) the header information structure.
- c) the header data structure.
- 5. Initializes
  - a) the default segment.
  - b) the integer arrays to 8000H
  - c) the dword arrays to 80000000H.
  - d) the header info & the header data structures.
- 6. Writes out the trend file.
- 7. Writes Øth entry of each element with last entry from previous day.
- 8. Deletes all segment created for initialization.

END Init\_Trend;

## Collect\_Trend:

++++++

## Input Parameters :

trend\_type : value for specifically structured trend -

|                        |                   |
|------------------------|-------------------|
| Ø = daily trend        | - name CURR_DAY,  |
| 1 = weekly trend       | - name CURR_WEEK, |
| 2 = monthly trend      | - name CURR_MNTH. |
| 3 = daily hourly trend | - name CURR_AHR.  |
| 4 = daily flow trend   | - name CURR_FHR.  |

time\_dword : the trend collect time.

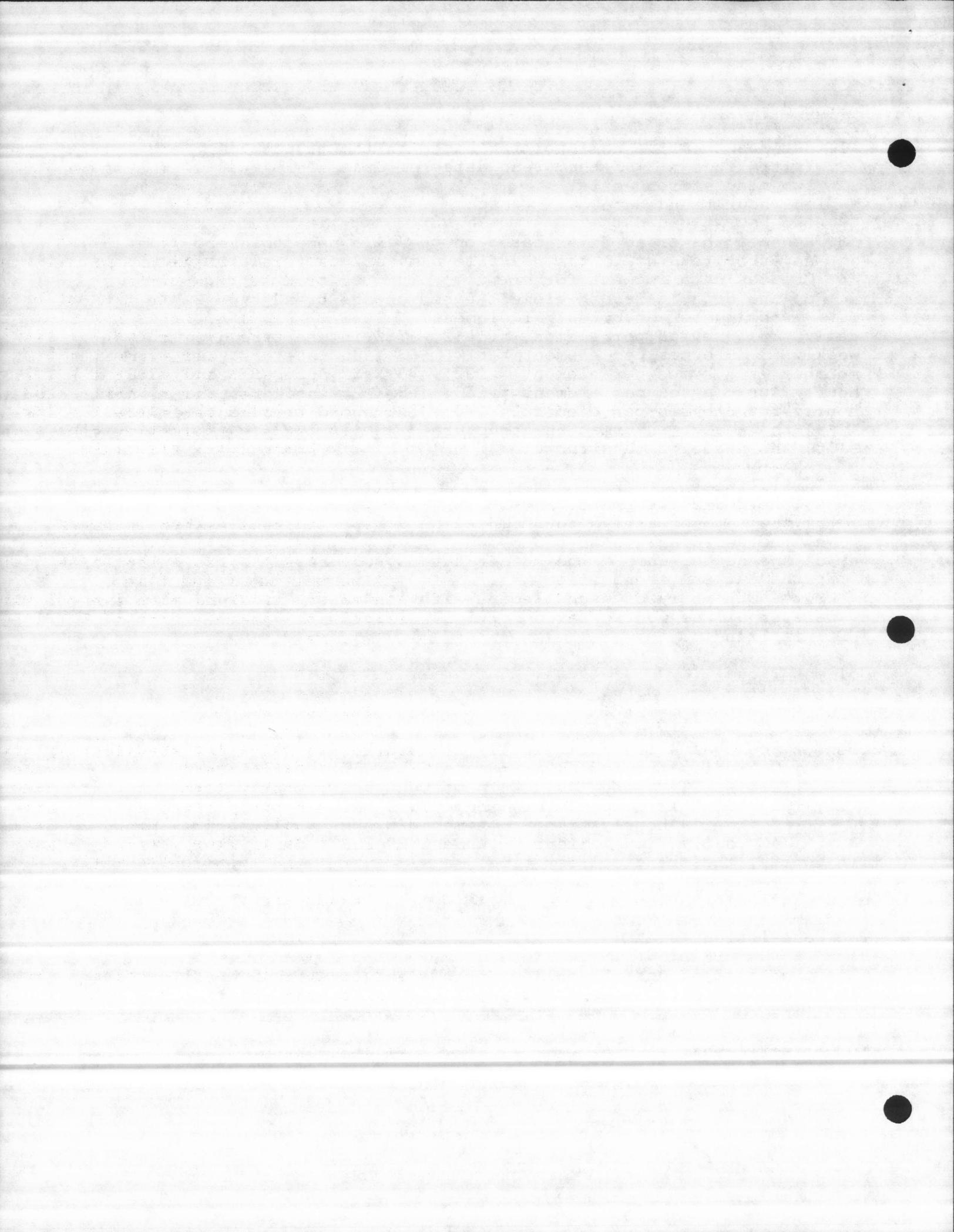
## Output Parameters :

----- None.

## Proc description :

This procedure is used to collect trends and save these trends on disk.  
The procedures written here will need to be fit into other system functions.

END Collect\_Trend;



Retrieve\_Trend:

++++++

Input Parameters :

trend\_type : value for specifically structured trend -

0 = daily trend - will return 721 integer entries,

1 = weekly trend - will return 255 (85 \* 3) integer entries

2 = monthly trend - will return 96 (32 \* 3) integer entries

3 = hourly average trend - returns 75 (25 \* 3) integer entries.

4 = hourly total trend - returns 25 dword entries.

pathname\_p : pointer to rmx string. If window\_flag set this name  
is ignored,

analog\_index : the one referenced index to analog trend needed. This  
index should correspond to the current system data base.

window\_flag : this flag determines whether a single file's data  
is loaded into the passed ram location or whether the time  
period "window" is loaded into ram by accessing CURR\_xx and  
PREV\_xx files. The purpose of the window feature is to allow  
a trend graphic screen to display the last 24 hours/ 7 days /  
31 days of data for current displays. The window feature will  
always use the CURR and PREV files. Historical data (indicated  
by name of file) cannot be windowed.

time\_dword : used only for window feature. This will be used to determine  
how much data is to be taken from the PREV\_ file and how much from  
the CURR\_ file.

data\_p : ram pointer which will hold the retrieved data.

Output Parameters :

----- None.

Proc description :

This procedure is used to bring a single analog's trend entry in  
from disk (from CURR\_ or PREV\_ file). Options will include time period  
windowing.

END Retrieve\_Trend;

Display\_Trend\_Feed\_Proc:

++++++

Input Parameters :

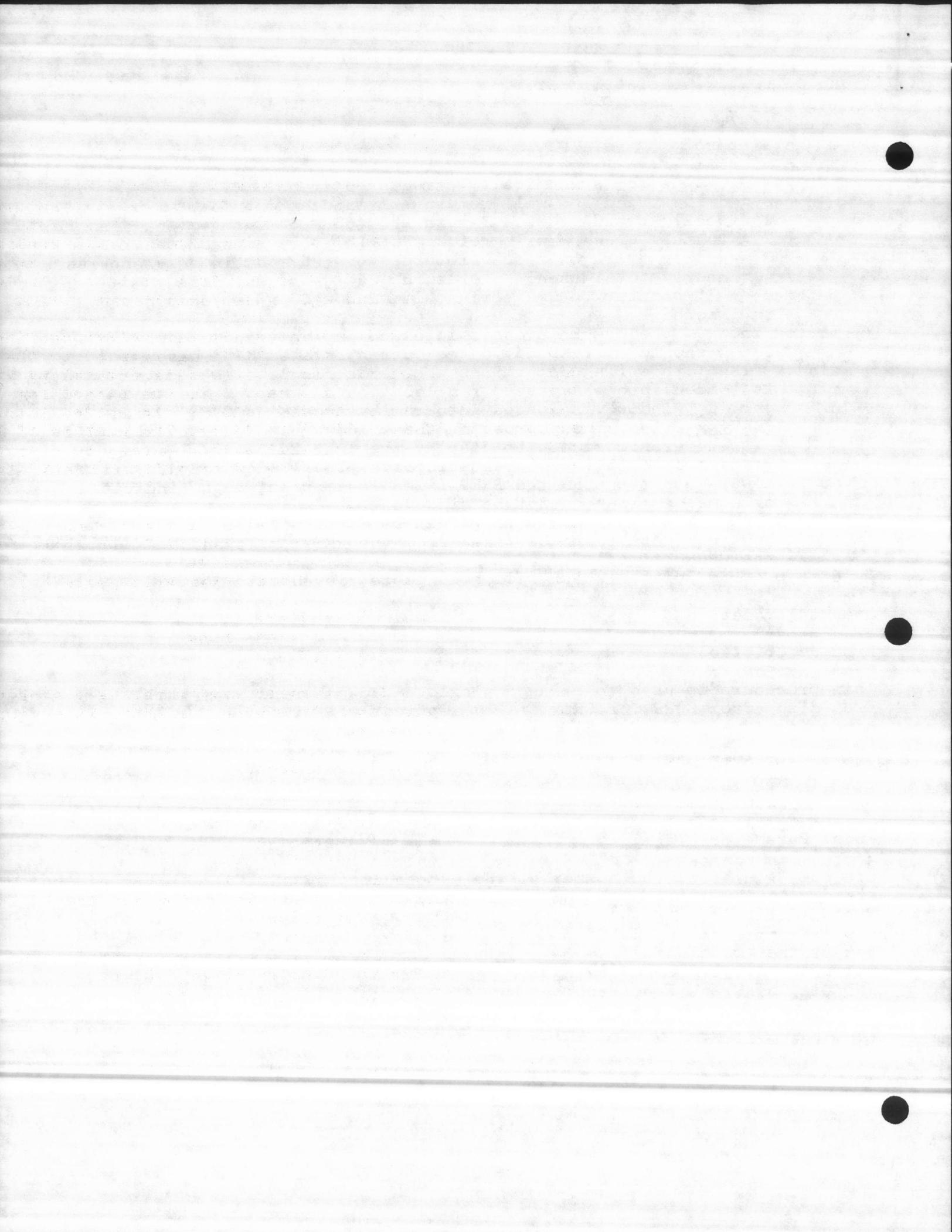
crt\_num => A byte, the number of the crt to display on.

trend\_p => A pointer - the address to the passed trend structure.

Proc description :

Passes the trend data from passed\_trend structure to keyin\_trend.  
Calls the display\_trend\_data\_proc.

END Display\_Trend\_Feed\_Proc;



---

**Display\_Trend\_Data\_Proc:****+++++****Proc description :**

- 
1. Receives control of the crt update region.
  2. Calls Stop\$Update - stop previous display update.
  3. Passes the keyin\_trend structure to the trend structure.
  4. Creates the data segment for
    - a) the trend crt table.
    - b) the crt parameters.
  5. Gets current time.
  6. Fixes current time and appends file name.
  7. Calls Retrieve\_Trend proc.
  8. Sets up display parameters.
    - a) trend heading pointer.
    - b) Paging system.
    - c) maximum lines to be displayed.
    - d) the number of entries in the crt display trend table.
    - e) the time span
    - f) the sample time.
    - g) the line time.
  9. Calls Clear\_Screen.
  10. Displays the trend heading.
  11. Displays the trend sub heading.
  12. Calls Set\$Update.
  13. Releases control of the crt update region.

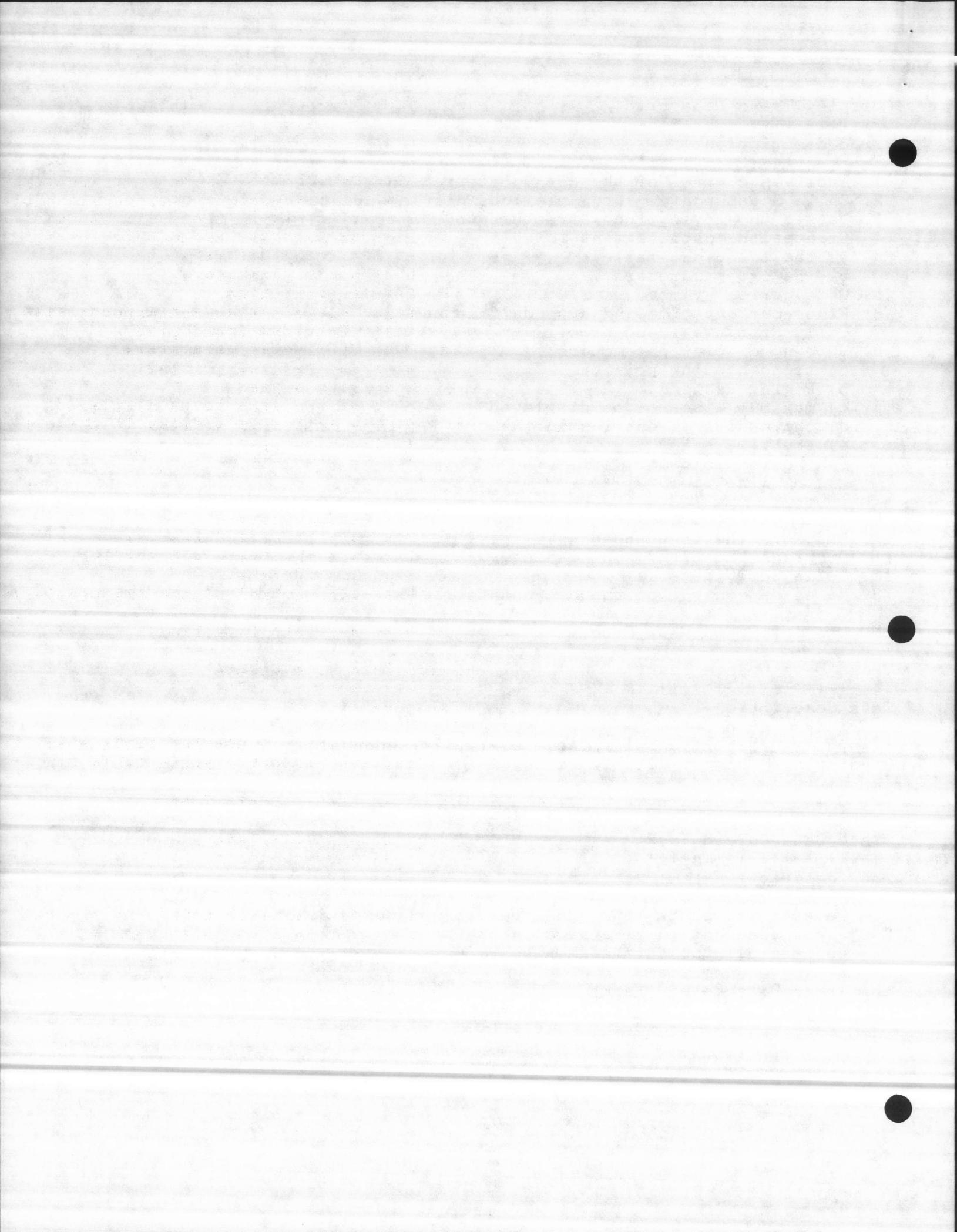
**END Display\_Trend\_Data\_Proc;****Trend\_Data\_Update\_Task:****+++++****Task description :**

- 
1. updates the crt display table every 120 seconds.
  2. displays the crt display table.

**END Trend\_Data\_Update\_Task;****Trend\_Data\_Delete\_Task:****+++++****Task description :**

- 
- 1 - Deletes the clock entry and the crt update semaphore.
  - 2 - Saves the edited file back on to the hard disk.
  - 3 - Deletes all the data segments.

**END Trend\_Data\_Delete\_Task;**



(16.2) MODULE NAME : CxxxxLINE.Pyy

=====

DESCRIPTION :

=====

EXTERNAL PROCEDURES :

=====

```
Retrieve_Trend: PROCEDURE (trend_type,
                           pathname_p,
                           analog_index,
                           window_flag,
                           min_max_avg_flag,
                           time_dword,
                           data_p)      EXTERNAL;
```

```
DECLARE (trend_type) BYTE,
        (analog_index, min_max_avg_flag, window_flag) WORD,
        (time_dword) DWORD,
        (pathname_p, data_p)      POINTER;
```

END Retrieve\_Trend;

```
Display_Table: PROCEDURE (table_p, param_p, table_reg_t,
                           buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p)      POINTER;
END Display_Table;
```

```
Prompt_Entry: PROCEDURE (entry_max, crt_num) EXTERNAL;
DECLARE (crt_num, entry_max) BYTE;
END Prompt_Entry;
```

GLOBAL REFERENCE :

=====

See CxxxxTPUB.Pyy.

MODULE DECLARATION :

=====

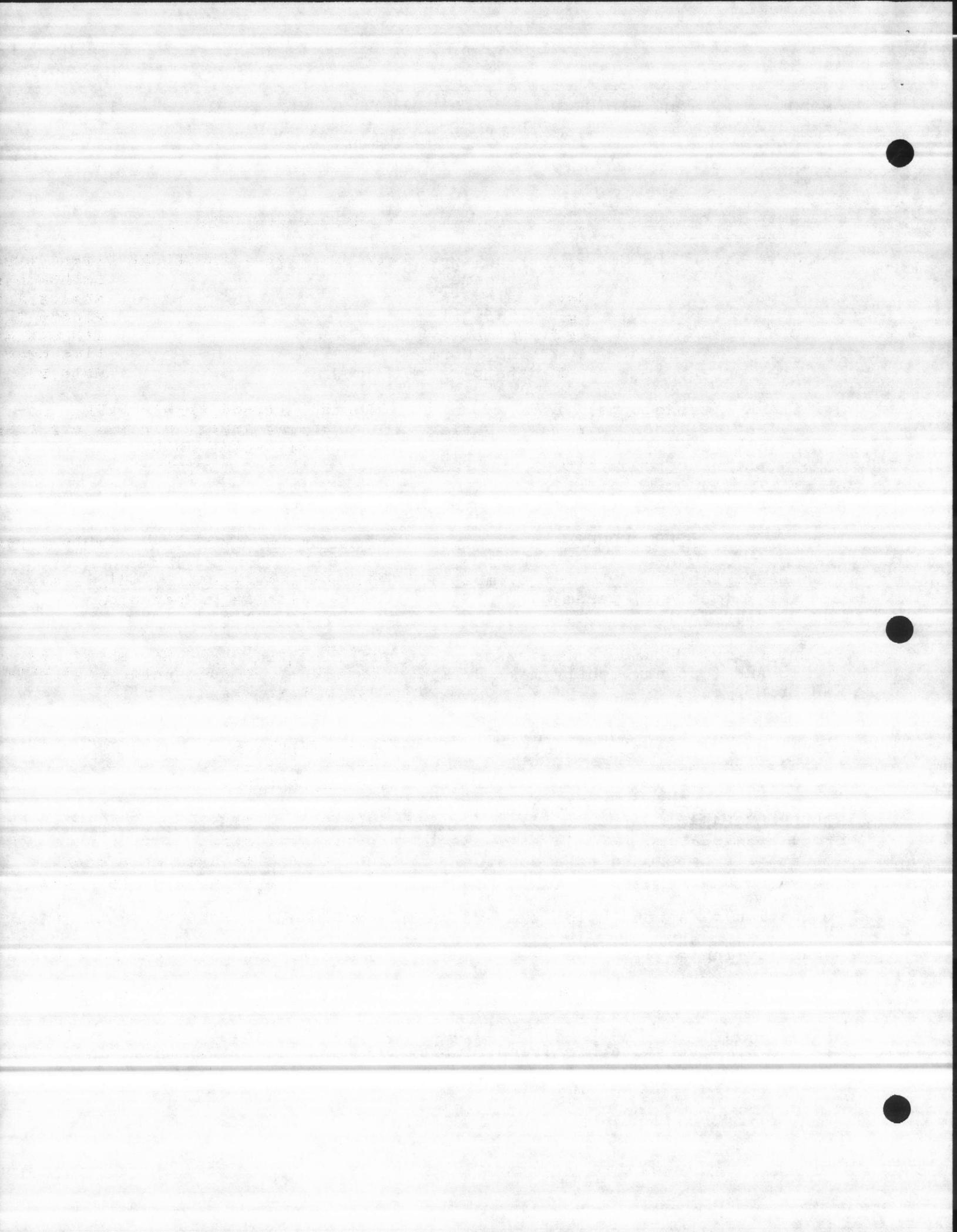
count => A byte used as a count.  
index => A byte used as an index.

exception => A word holds the exception condition.

space\_string => 34 bytes hold a space sting.

color\_list => Four bytes hold each hold a color index.

trnd\_head => Twenty two bytes hold the trend header  
'Graphic Trend Display'.



PUBLIC PROCEDURES :

=====

Display\_Line\_Desc:

+++++-----

Proc description :

-----  
Dipslays line description.

END Display\_Line\_Desc;

MODULE TASKS :

=====

Draw\_Trend\_Task:

+++++-----

Input Parameter :

-----

crt\_num => A byte holds the crt number to display on.

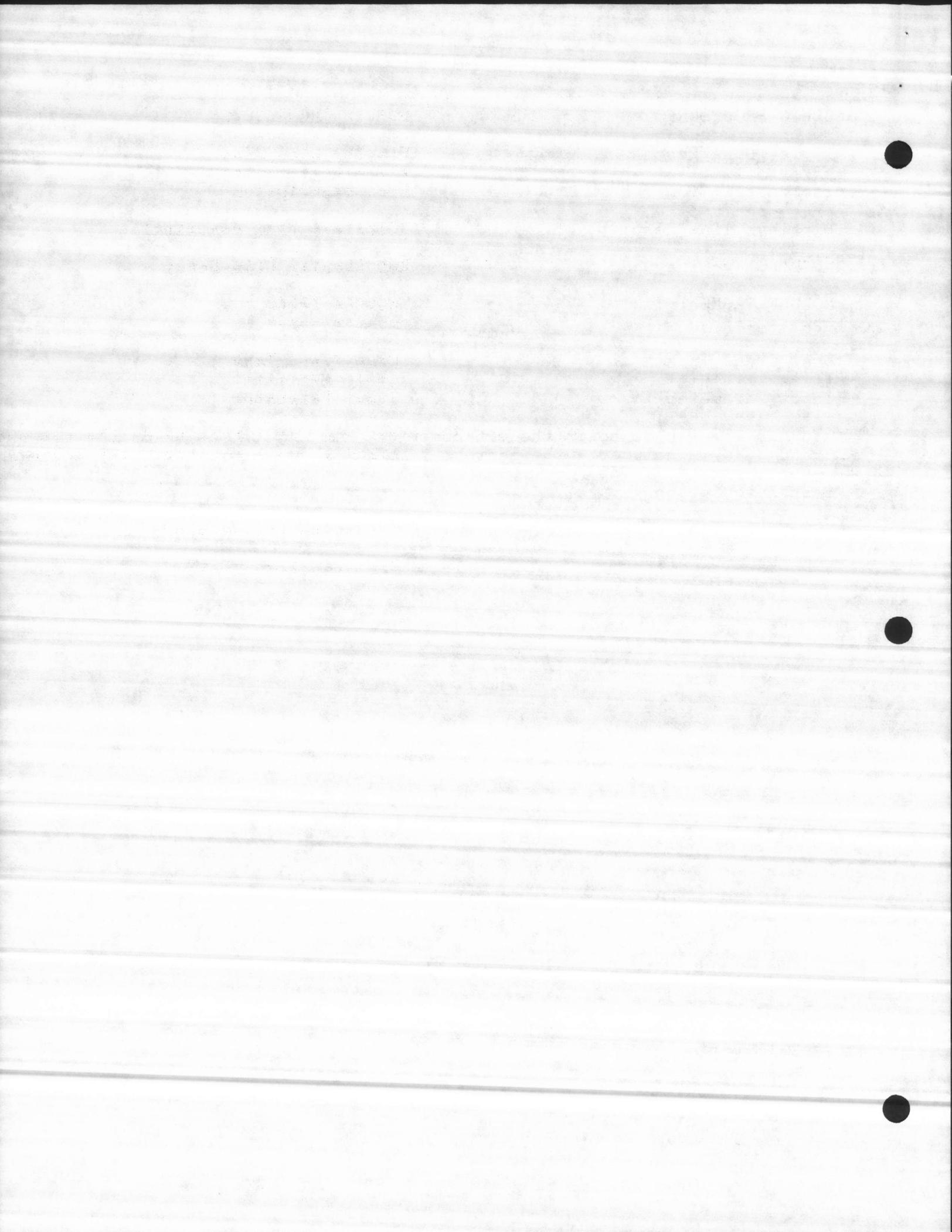
trend\_p => A pointer - the address to a trend structure.

Task description :

-----

Draws the line graph.

END Draw\_Trend\_Task;



(16.3) MODULE NAME : CxxxxLNGR.Pyy

=====

DESCRIPTION :

=====

Displays the line graph.

EXTERNAL PROCEDURES :

=====

```
Draw_Line_Graph: PROCEDURE (crt_num, trend_p) EXTERNAL;
  DECLARE (crt_num)      BYTE;
  DECLARE (trend_p)      POINTER;
END Draw_Line_Graph;
```

Table\_Fill:

```
  PROCEDURE (field_number, element, struc_type,
             table_p, param_p) WORD EXTERNAL;
  DECLARE (struc_type)      BYTE,
          (field_number, element) WORD,
          (table_p, param_p)     POINTER;
```

```
END Table_Fill;
```

Hist\_Table\_Fill:

```
  PROCEDURE (field_number, element, struc_type,
             table_p, param_p) WORD EXTERNAL;
  DECLARE (struc_type)      BYTE,
          (field_number, element) WORD,
          (table_p, param_p)     POINTER;
```

```
END Hist_Table_Fill;
```

```
Menu_Display: PROCEDURE (crt_num, result, off_set, start_id_p,
                           bytes_per_element, num_elements) WORD EXTERNAL;
  DECLARE (crt_num)      BYTE,
          (result, off_set) WORD,
          (bytes_per_element, num_elements) WORD,
          (start_id_p)       POINTER;
```

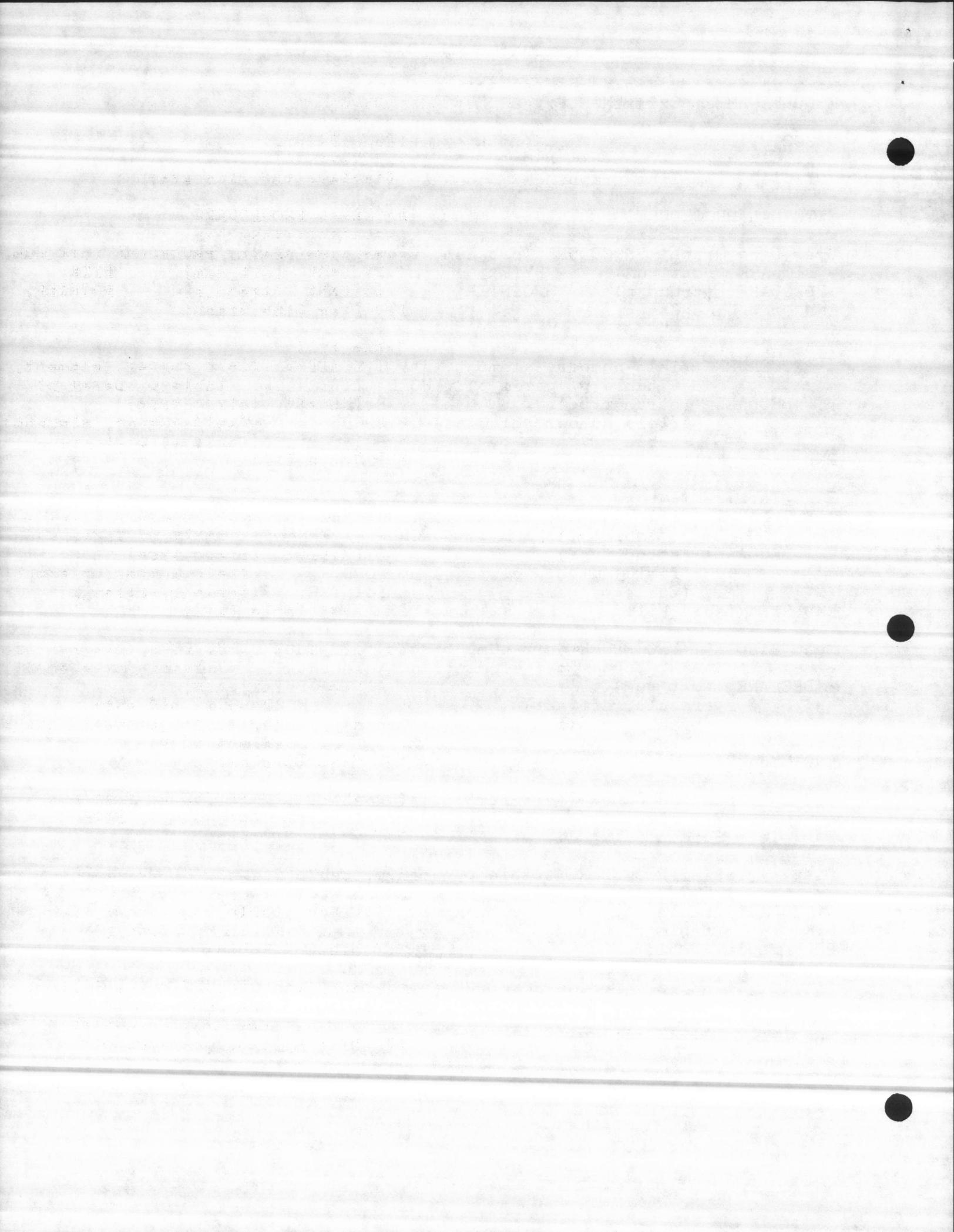
```
END Menu_Display;
```

```
Clear_Screen: PROCEDURE (crt_num) EXTERNAL;
  DECLARE (crt_num)      BYTE;
END Clear_Screen;
```

```
Display_Table: PROCEDURE (table_p, param_p, table_reg_t,
                           buf_t, mbx_t) EXTERNAL;
  DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
          (table_p, param_p)     POINTER;
```

```
END Display_Table;
```

```
Display_Trend_Feed_Proc: PROCEDURE (crt_num, trend_p) EXTERNAL;
  DECLARE (crt_num)      BYTE,
          (trend_p)      POINTER;
END Display_Trend_Feed_Proc;
```



## MODULE DECLARATION :

```
=====

```

**== Used for displaying the trend data value by value .**

keyin\_trend (3) STRUCTURE &

trend (3) STRUCTURE &

numeric\_trend (3) STRUCTURE :

See CxxxxTPUB.Pyy

line\_graph\_update\_table => A pointer - the address to the  
Line\_Graph\_Update\_Task

line\_graph\_delete\_table => A pointer - the address to the  
Line\_Graph\_Delete\_Task.

## CURRENT TREND HEADING :

trend\_data\_heading0 => 23 bytes hold the 'Daily Two Minute Trend'.

trend\_data\_heading1 => 22 bytes hold the 'Weekly Two Hour Trend'

trend\_data\_heading2 => 22 bytes hold the 'Monthly One Day Trend'

trend\_data\_heading3 => 14 bytes hold the 'Hourly Trends'

## HISTORICAL TREND HEADING :

h\_trend\_data\_heading0 (34 bytes)'Historical Daily Two Minute Trend'

h\_trend\_data\_heading1 (33 bytes)'Historical Weekly Two Hour Trend'

h\_trend\_data\_heading2 (33 bytes)'Historical Monthly One Day Trend'

h\_trend\_data\_heading3 (24 bytes)'Historical Hourly Trends'

exception => Three words, each holds the exception condition.

crt\_update\_sem\_t => Three tokens used to create the crt update  
semaphore.

heading\_p => Three pointers - the address to the trend heading.  
For the X reference structures:

field => A byte holds the field number.

row => A byte holds the row number.

column => A byte holds the column number.

## trend\_data0\_xref (9) STRUCTURE :

DATA /\* --- label ----- field row col \*/

/\* null \*/ Ø, Ø, Ø,

/\* low range \*/ 18, 37, 16,

/\* high range \*/ 19, 37, 38,

/\* current \*/ 3, 37, 62,

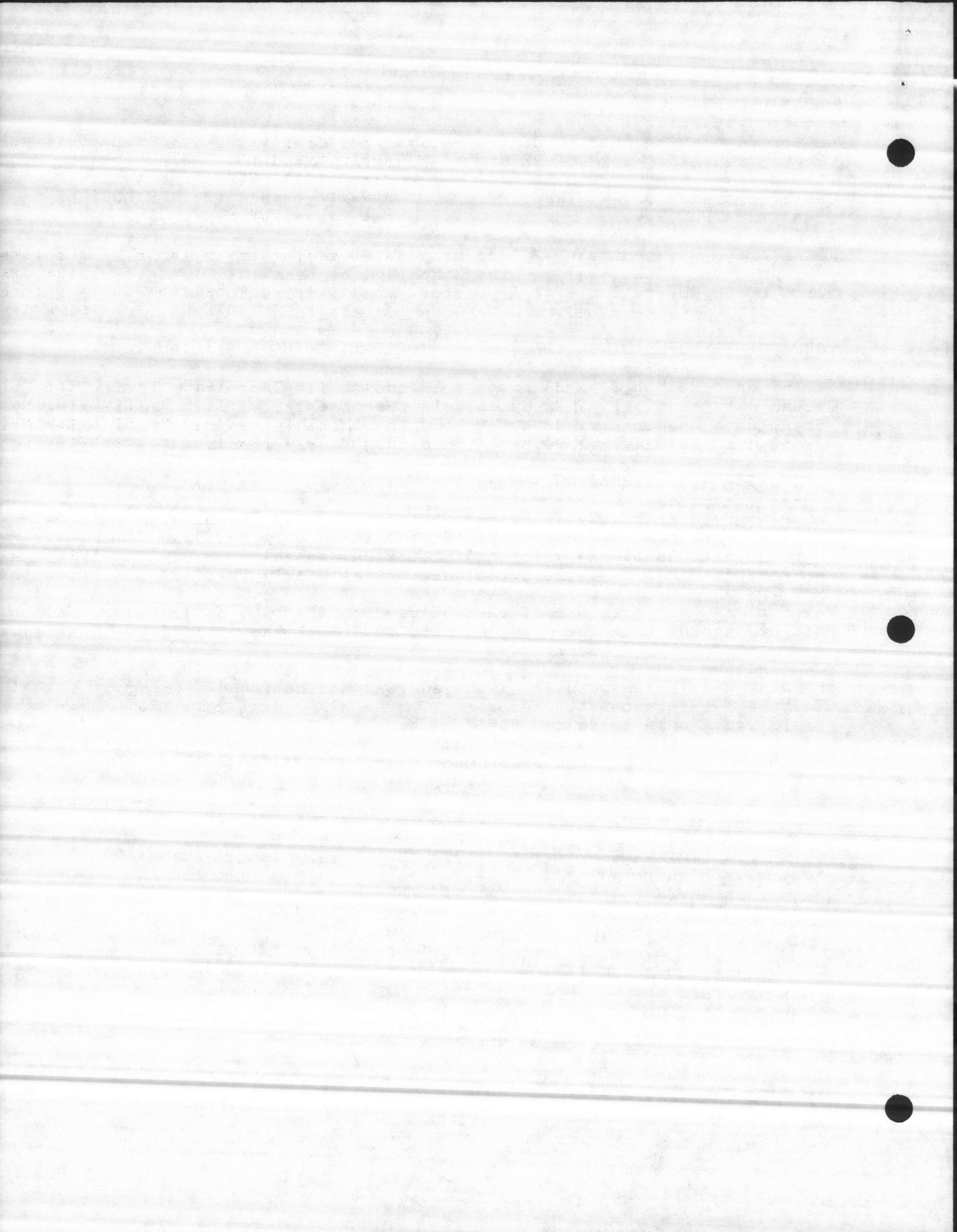
/\* eng units \*/ 21, 37, 71,

/\* low low alarm \*/ 23, 38, 12,

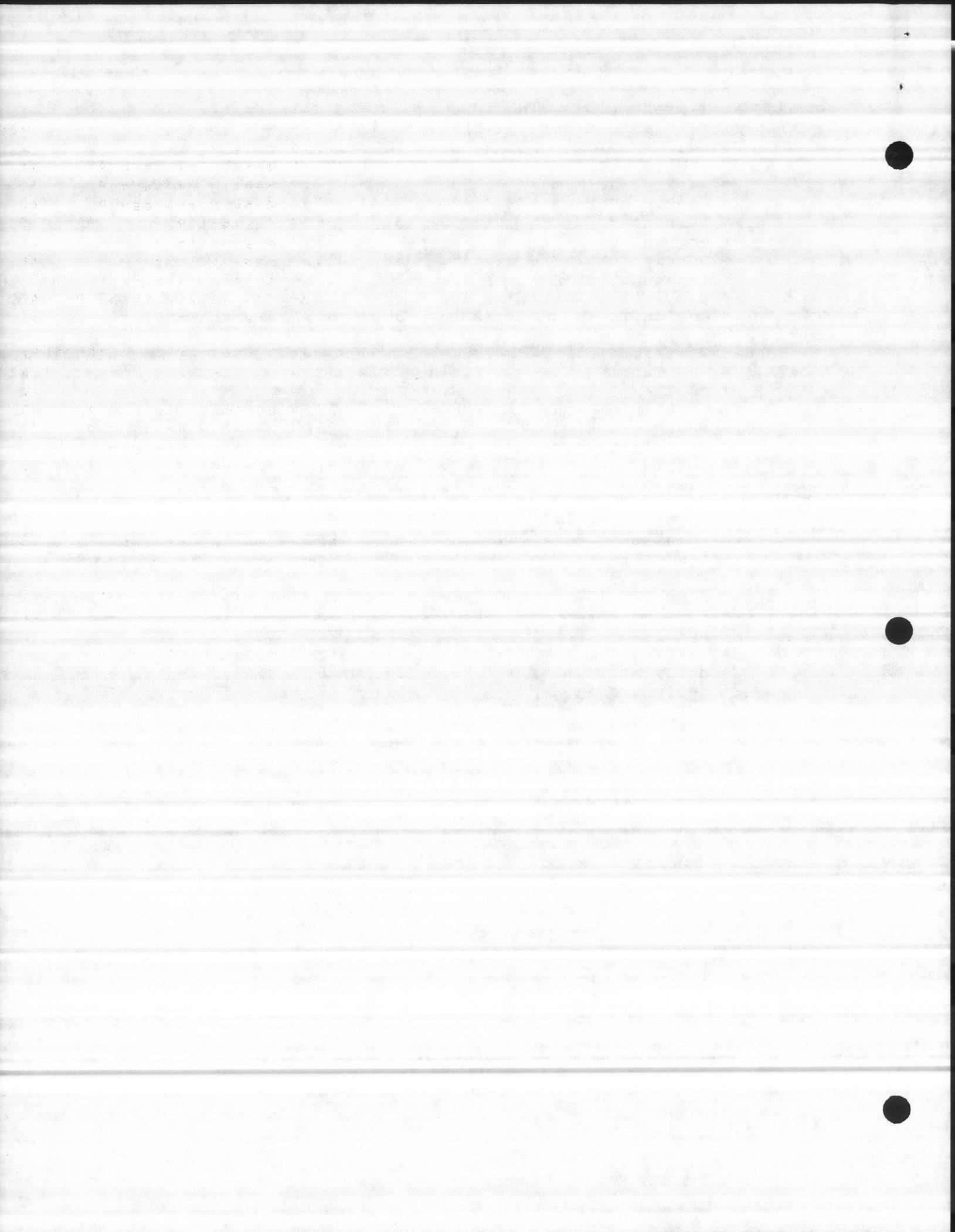
/\* low alarm \*/ 24, 38, 29,

/\* high alarm \*/ 25, 38, 47,

/\* high high alarm \*/ 26, 38, 69);



```
trend_data1_xref (10) STRUCTURE :  
DATA /* --- label ----- field row col */  
/* null */ Ø, Ø, Ø,  
/* weekly minimum */ 31, 37, 12,  
/* weekly maximum */ 33, 37, 29,  
/* weekly average */ 42, 37, 47,  
/* current */ 3, 37, 64,  
/* eng units */ 21, 37, 73,  
/* weekly total */ 46, 38, 21,  
/* total units */ 21, 38, 33,  
/* cumulative total */ 48, 38, 52,  
/* eng units */ 21, 38, 64);  
  
trend_data2_xref (10) STRUCTURE :  
DATA /* --- label ----- field row col */  
/* null */ Ø, Ø, Ø,  
/* monthly minimum */ 35, 37, 12,  
/* monthly maximum */ 37, 37, 29,  
/* monthly average */ 43, 37, 47,  
/* current */ 3, 37, 64,  
/* eng units */ 21, 37, 73,  
/* monthly total */ 47, 38, 21,  
/* total units */ 21, 38, 33,  
/* cumulative total */ 48, 38, 52,  
/* eng units */ 21, 38, 64);  
  
trend_data3_xref (10) STRUCTURE :  
DATA /* --- label ----- field row col */  
/* null */ Ø, Ø, Ø,  
/* daily minimum */ 27, 37, 12,  
/* daily maximum */ 29, 37, 29,  
/* daily average */ 41, 37, 47,  
/* current */ 3, 37, 64,  
/* eng units */ 21, 37, 73,  
/* daily total */ 45, 38, 21,  
/* total units */ 21, 38, 33,  
/* cumulative total */ 48, 38, 52,  
/* eng units */ 21, 38, 64);
```



**PUBLIC PROCEDURES :****=====****Display\_Line\_Graph\_Proc:****+++++****Proc description :****-----**

1. calls stop\$update.
2. creates a data segment for the crt table.
3. Sets the crt display parameters.
4. initializes the zeroth entry of the crt table.
5. Fills the crt display table with the proper data.
6. Calls clear\_screen.
7. displays the proper trend heading and sub heading.
8. Sets up the historical file names.
9. Calls Draw\_Line\_Graph.
10. Calls Set\$Update.

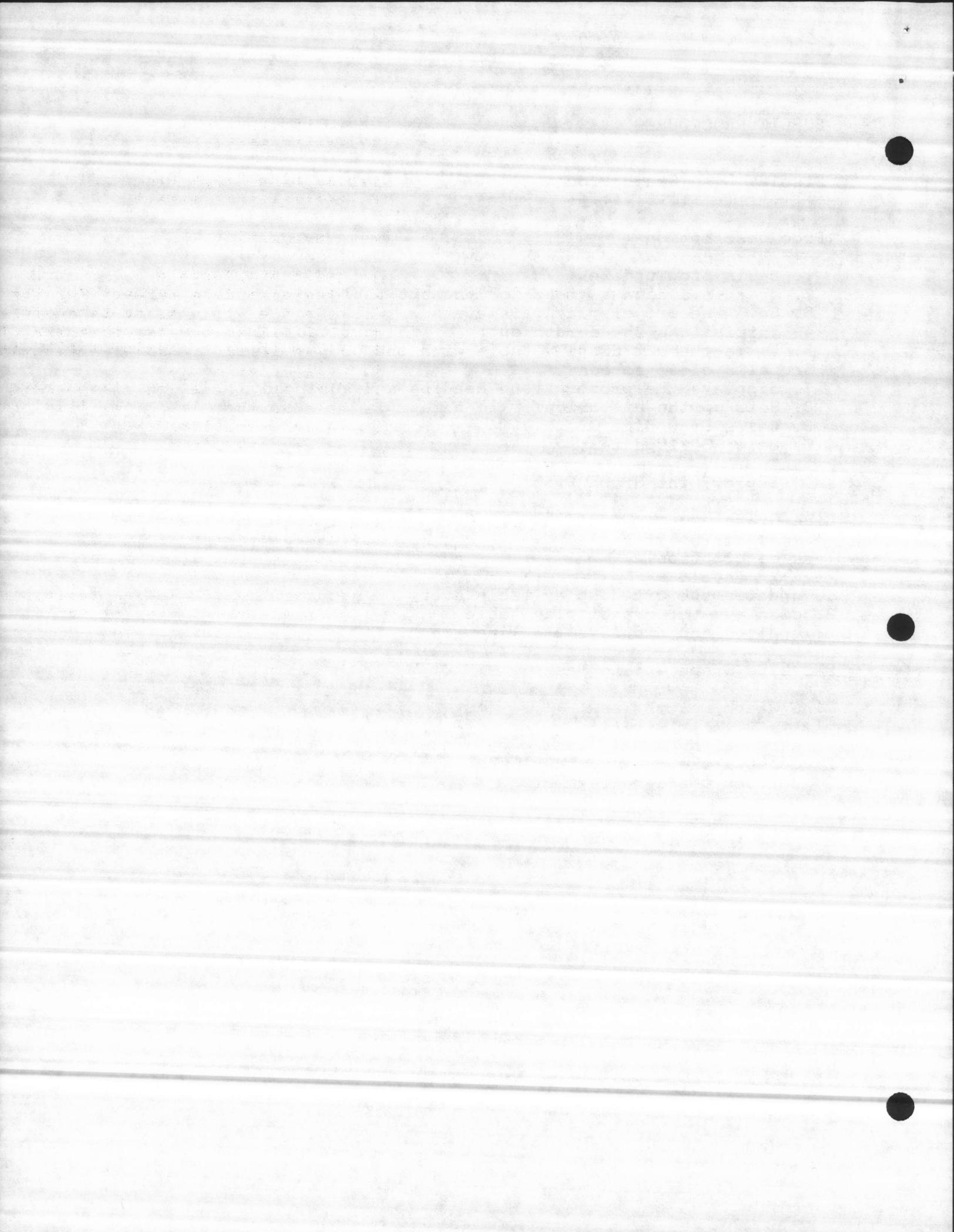
**END Display\_Line\_Graph\_Proc;****Line\_Graph\_Update\_Task:****+++++****Task description :****-----**

1. updates the crt display table every 10 seconds.
2. displays the crt display table.
3. updates and displays the line graph.

**END Line\_Graph\_Update\_Task;****Line\_Graph\_Delete\_Task:****+++++**

1. deletes the update task clock entry.
2. deletes the crt update semaphore.
3. deletes the data segment.

**END Line\_Graph\_Delete\_Task;**



---

---

=====  
Graphs\_Display\_Menu  
=====

GENERAL DECLARATION :

---

graph\_command\_count => A byte holds the value of 4.  
graph\_menu\_prompt\_asc (4) STRUCTURE:

menu\_asc => Ten bytes hold the trending sub command string.  
Daily = Selects daily trend for report  
Weekly = Selects weekly trend for report  
Monthly = Selects monthly trend for report  
Hourly = Selects hourly trend for report

graph\_menu\_help\_tbl => Four pointers - Four addresses to the graph menu help page.

type\_command\_count => A byte holds the value of 2.

type\_menu\_prompt\_asc (2) STRUCTURE :

menu\_asc => Ten bytes hold the sub sub command string  
Graphic = Selects graphic display  
Numeric = Selects numeric display

type\_menu\_help\_tbl => Two pointer - the addresses to the help page.

mma\_command\_count => A byte holds the value of 3.

mma\_menu\_prompt\_asc (3) STRCUTURE :

menu\_asc => 10 bytes hold the sub sub sub command string.  
Minimum = Selects minimum display.  
Maximum = Selects maximum display.  
Average = Selects average display.

mma\_menu\_help\_tbl => Three pointers - the addresses to the help page.

Graphs\_Display\_Menu:

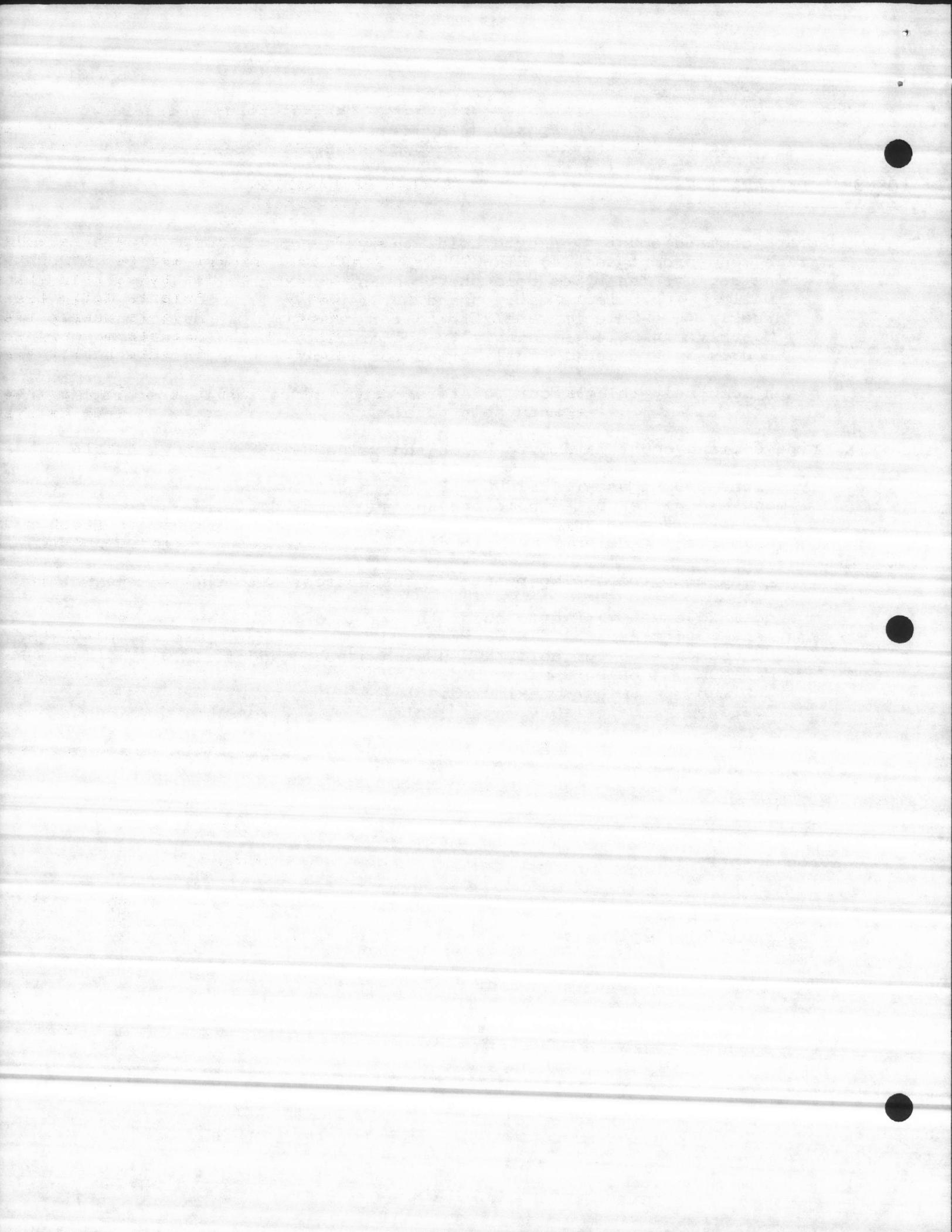
+++++++

Proc description :

---

Displays the trending menu and sub menus.

END Graphs\_Display\_Menu;



(16.4) MODULE NAME : CxxxxDLGR.Pyy

=====

DESCRIPTION : This module draws the line graph.

=====

EXTERANAL PROCEDURES :

=====

Retrieve\_Trend: PROCEDURE (trend\_type,  
                          pathname\_p,  
                          analog\_index,  
                          window\_flag,  
                          min\_max\_avg\_flag,  
                          time\_dword,  
                          data\_p)       EXTERNAL;

DECLARE (trend\_type)                           BYTE,  
                         (analog\_index,  
                         min\_max\_avg\_flag,  
                         window\_flag)             WORD,  
                         (time\_dword)              DWORD,  
                         (pathname\_p, data\_p)    POINTER;

END Retrieve\_Trend;

Display\_Table: PROCEDURE (table\_p,param\_p,table\_reg\_t,  
                                                         buf\_t, mbx\_t) EXTERNAL;

DECLARE (table\_reg\_t, buf\_t, mbx\_t)   TOKEN,  
                                                         (table\_p, param\_p)    POINTER;  
END Display\_Table;

PUBLIC PROCEDURES :

=====

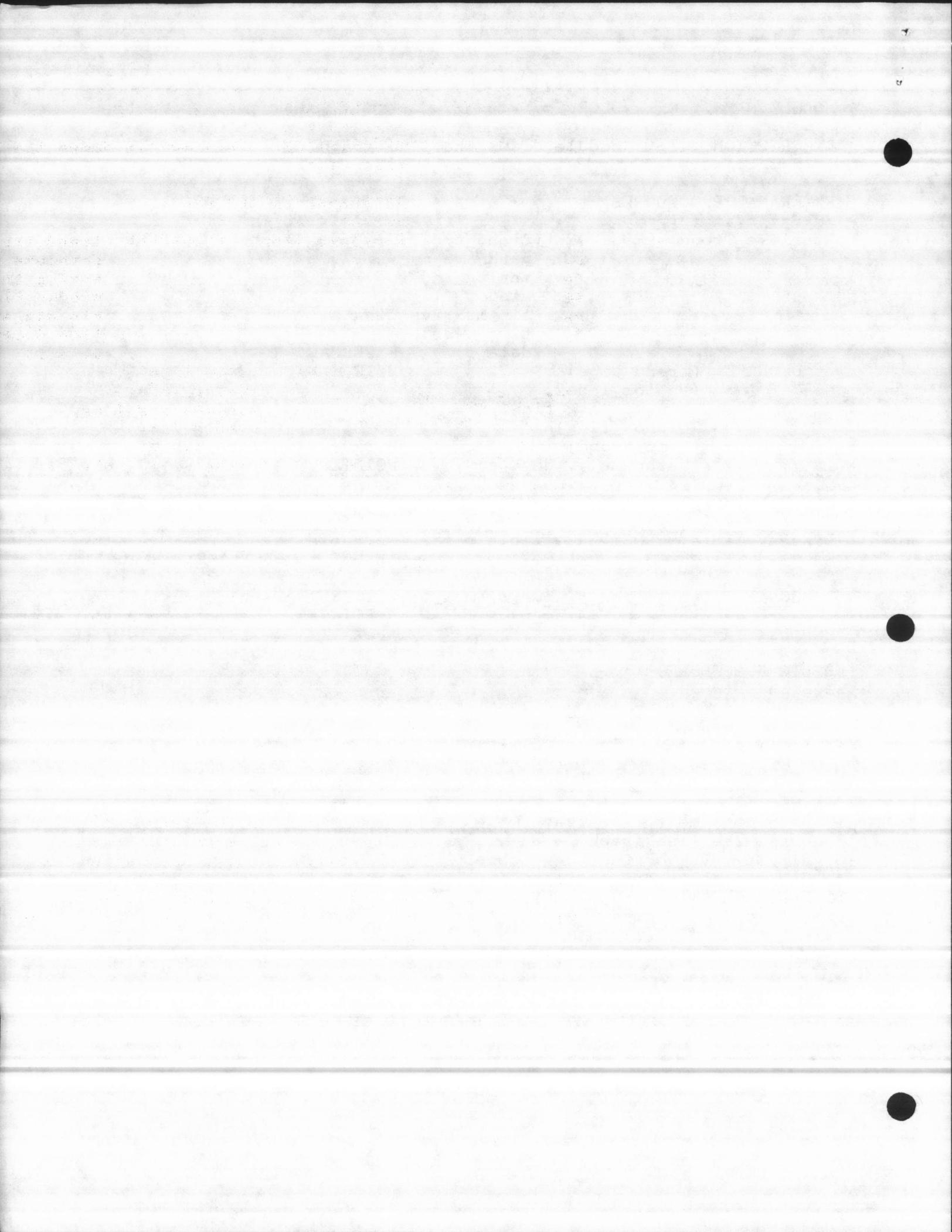
Draw\_Line\_Graph:

++++++

Proc description :

- 
1. Sets up time for retrieve.
  2. Draws the line graph.
  3. Displays the crt display table.
  4. Creates necessary segments for line graph.
  5. Draws the line graph again.
  6. Deletes the segments of the line graph.

END Draw\_Line\_Graph;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 17

R E P O R T  
G E N E R A T O R

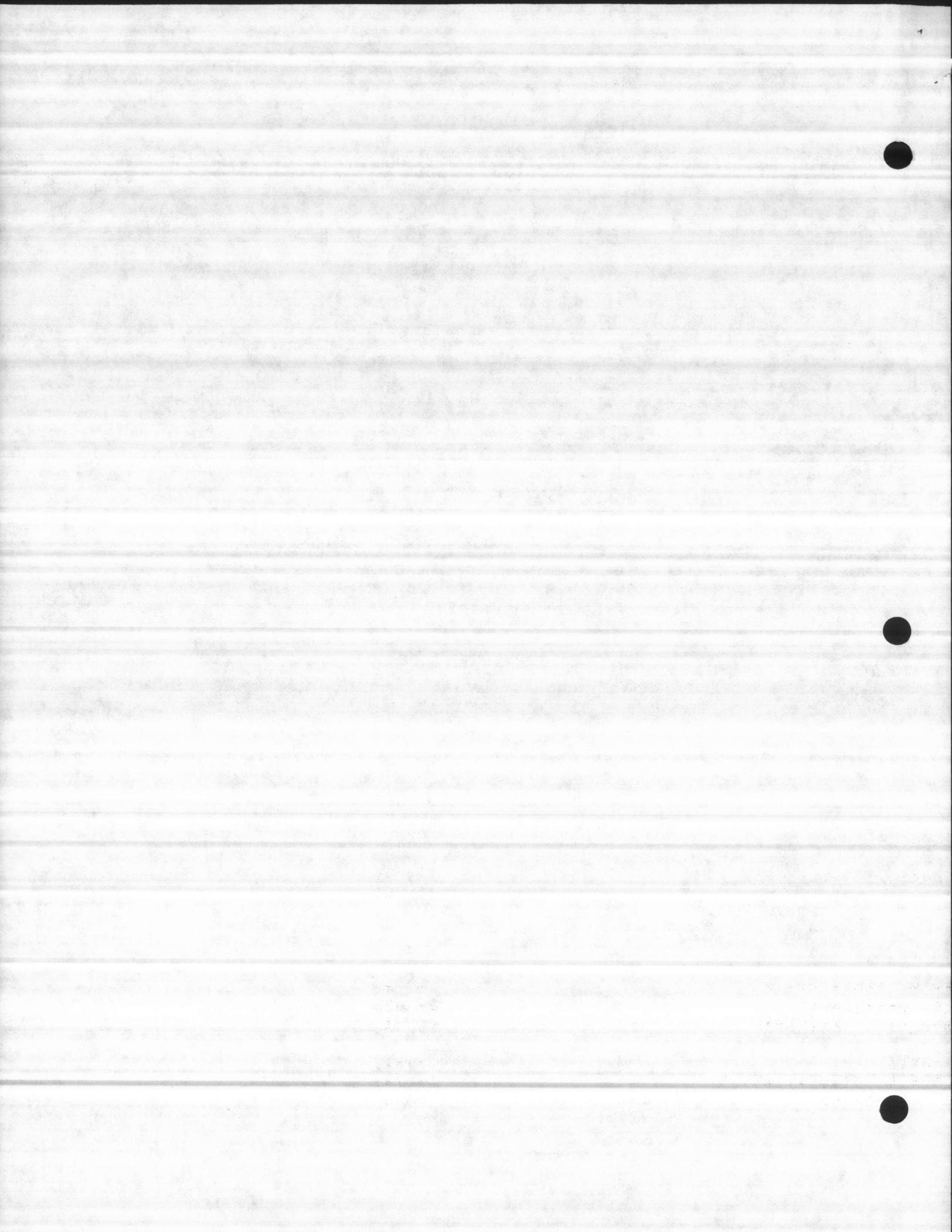
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

REPORT GENERATOR - written by Bob Ryan, Mohamed Fayad.

(17.1) MODULE NAME : CxxxxREPG.Pyy

=====

DESCRIPTION :

=====

This module constructs daily, weekly, monthly, yearly and periodic reports.

EXTERNAL PROCEDURES :

=====

Menu\_Display:

```
PROCEDURE (crt_num, result, off_set,
           start_id_p, bytes_per_element, num_elements) WORD EXTERNAL;
DECLARE (crt_num) BYTE,
        (result, off_set) WORD,
        (bytes_per_element, num_elements) WORD,
        (start_id_p) POINTER;
```

END Menu\_Display;

```
Format_Disk_Error: PROCEDURE (buffer_t, message_str_p, disk_error,
                               disk_operation) EXTERNAL;
DECLARE (disk_error, disk_operation) WORD,
        (buffer_t) TOKEN,
        (message_str_p) POINTER;
```

END Format\_Disk\_Error;

```
Display_Table: PROCEDURE(table_p, param_p, table_reg_t,
                           buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p) POINTER;
```

END Display\_Table;

Get\_Password:

```
PROCEDURE (crt_num, table_p, table_reg_t,
           display_buf_t, display_mbx_t,
           ci_conn_t, ci_mbx_t) BYTE EXTERNAL;
DECLARE (crt_num) BYTE,
        (table_reg_t, display_buf_t, display_mbx_t,
         ci_conn_t, ci_mbx_t) TOKEN,
        (table_p) POINTER;
```

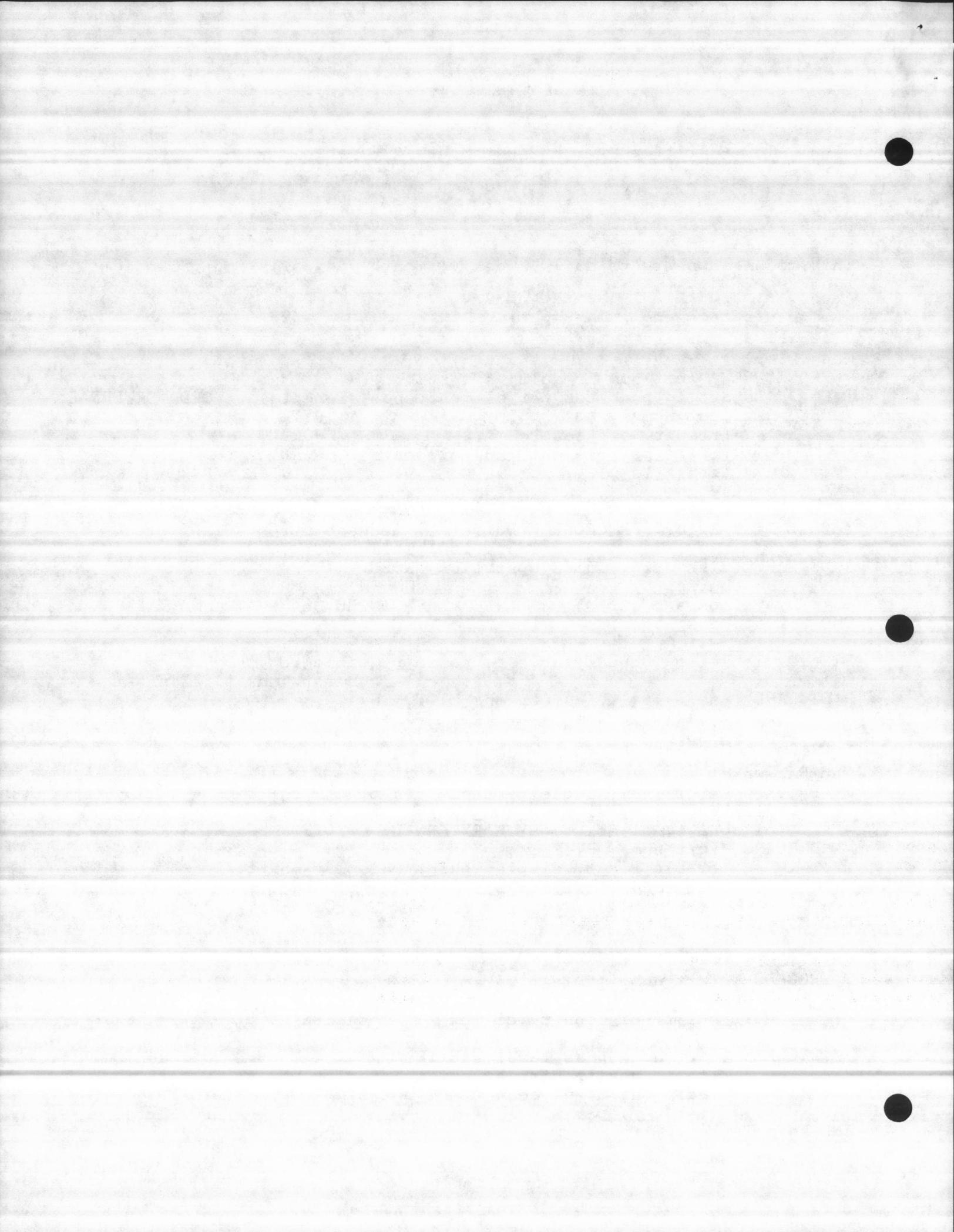
END Get\_Password;

Edit\_Table:

```
PROCEDURE (crt_num, table_p, param_p, table_reg_t,
           display_buf_t, display_mbx_t, pass_level,
           edit_buf_t, ci_conn_t, ci_mbx_t) EXTERNAL;
```

```
DECLARE (crt_num, pass_level) BYTE,
        (table_reg_t, display_buf_t, display_mbx_t,
         edit_buf_t, ci_conn_t, ci_mbx_t)
        (table_p, param_p) TOKEN,
```

END Edit\_Table;



```
Display_Report_Static:
  PROCEDURE (crt_num, max_display_size, max_page_size,
             view_offset, col_view_offset, static_asc_p,
             display_buf_t, display_mbx_t) EXTERNAL;

  DECLARE (crt_num)           BYTE,
          (max_display_size, max_page_size, view_offset,
           col_view_offset) WORD,
          (display_buf_t, display_mbx_t)      TOKEN,
          (static_asc_p)    POINTER;
END Display_Report_Static;

Edit_Report_Screen:
  PROCEDURE (crt_num, max_display_size, max_page_size, static_asc_p,
             display_buf_t, edit_buf_t, display_mbx_t,
             ci_conn_t, ci_mbx_t) EXTERNAL;

  DECLARE (crt_num)   BYTE,
          (max_display_size,max_page_size)        WORD,
          (display_buf_t, display_mbx_t,edit_buf_t,
           ci_conn_t, ci_mbx_t)                  TOKEN,
          (static_asc_p)    POINTER;
END Edit_Report_Screen;

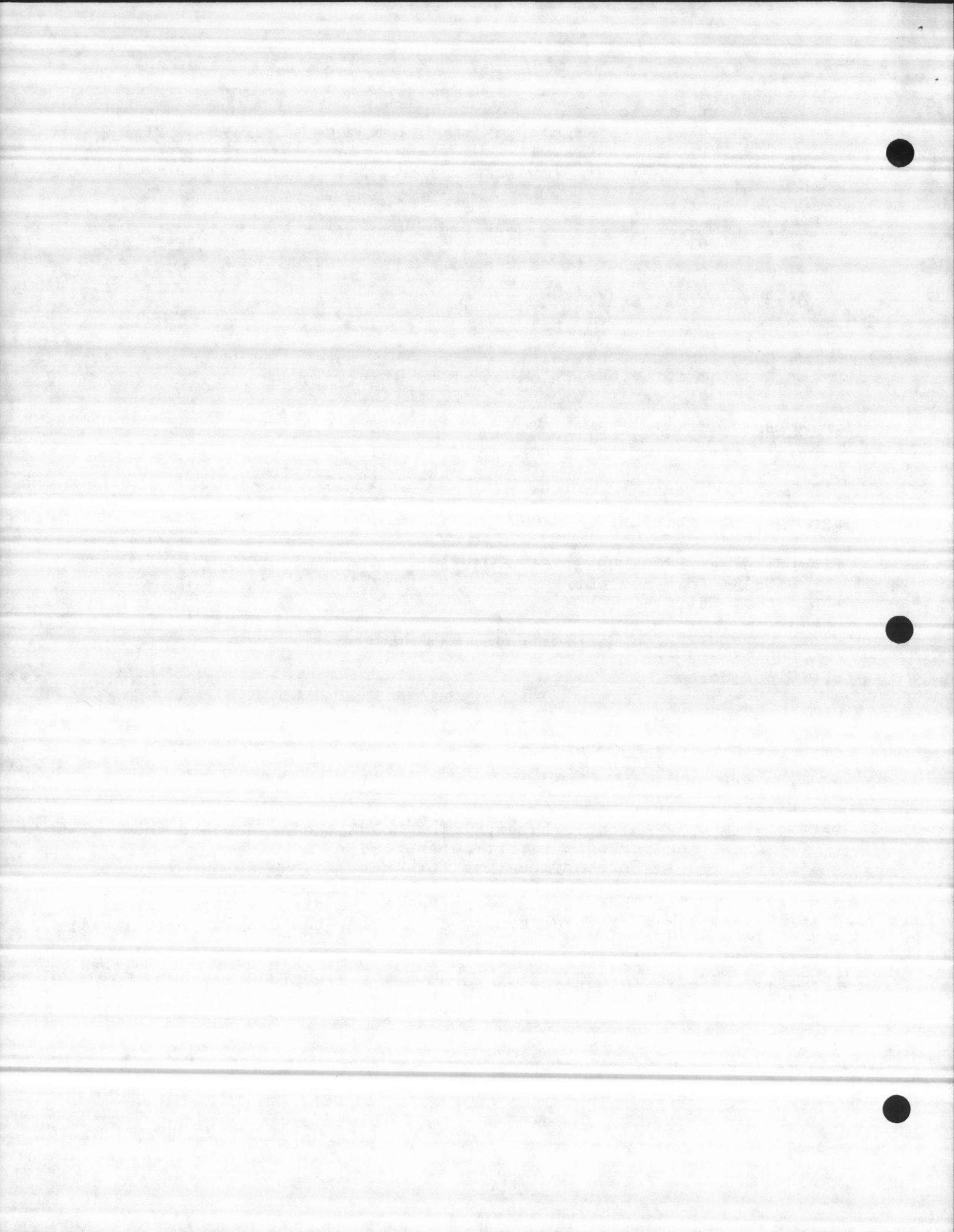
Prompt_Entry: PROCEDURE (entry_max_count,crt_num) EXTERNAL;
  DECLARE (crt_num,entry_max_count) BYTE;
END Prompt_Entry;

Clear_Screen: PROCEDURE (crt_num) EXTERNAL;
  DECLARE crt_num BYTE;
END Clear_Screen;

GLOBAL REFERNCE :
=====
disp_report (See CxxxxGLOB.Pyy)

MODULE DECLARATIONS :
=====
parameter_asc => An ascii bytes string holds the header and the
                  periodic report parameters.
edit_flag       => Three bytes (one for each crt) holds the value of
                  the edit flag.
display_page    => Three bytes, each holds the display page number.
display_page_limit => Three bytes, each holds the display page limit.

rpt_file_name => Fifteen bytes hold the report file name.
read_file_parameters BASED disk_param_seg_t STRUCTURE &
write_file_parameters BASED disk_param_seg_t STRUCTURE
(See disk system)
```



header\_info (4) STRUCTURE : holds the elements of the report file  
 header\_data (4) STRUCTURE : holds the elements of the report file data  
 exception => Three words, each holds the exception condition.  
 crt\_update\_sem\_t => Three crt update semaphore tokens, each used to  
                     create the update task semaphore.  
 dummy\_color => A word holds the index to the white color.  
 report\_disp\_heading => 29 bytes hold the report display header.  
                     'Construct Report Page Display'

## PUBLIC PROCEDURES :

=====

## Keyin\_Construct\_Report\_Sub\_Menu:

+++++

crt\_num => A byte holds the number of the crt to display on.  
 old\_echo\_count => A byte holds the old\_echo\_count.  
 result => A word used as an index.

## Proc description :

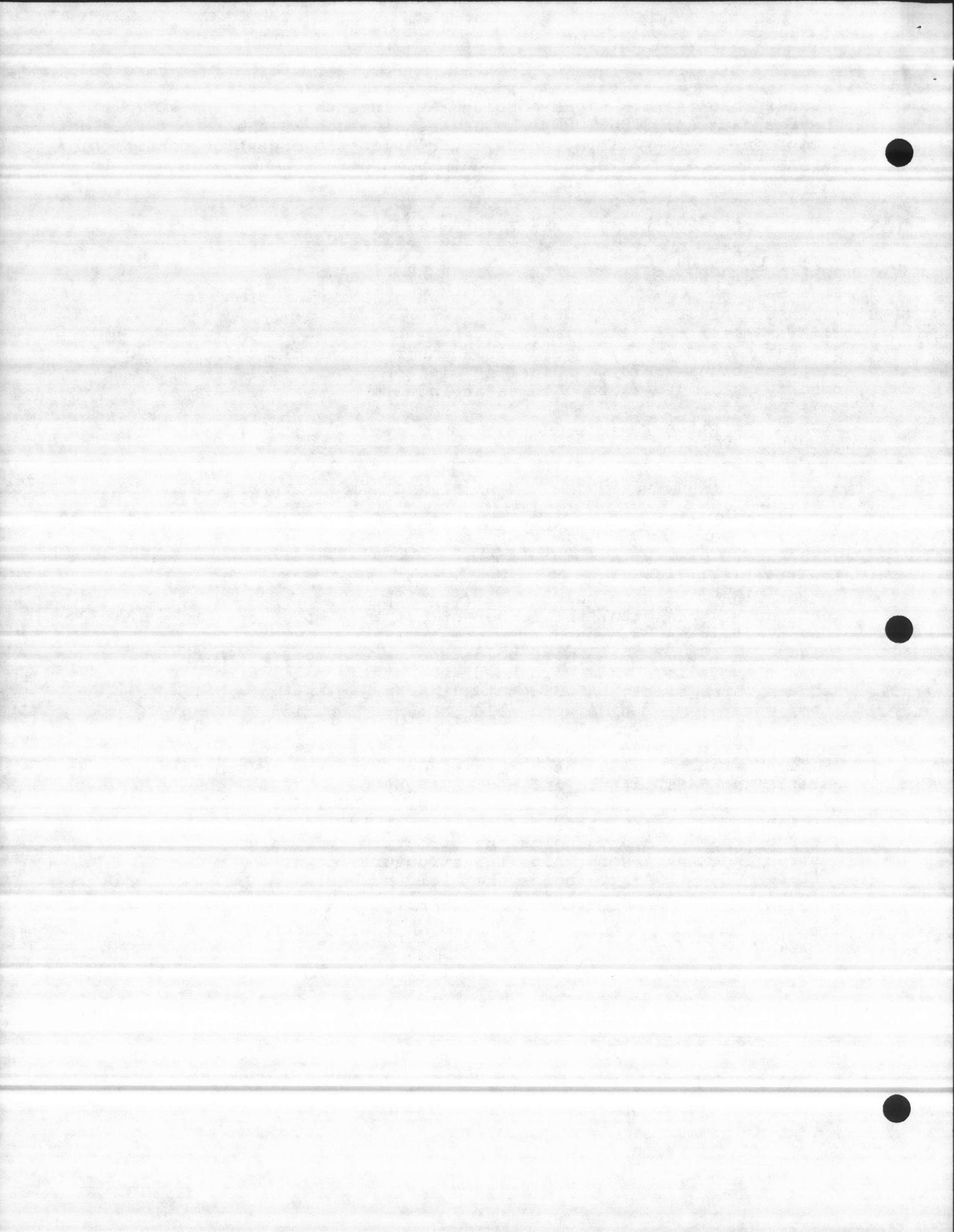
- 1. Checks for the supervisor password.  
 2. displays the reports sub menu.

END Keyin\_Construct\_Report\_Sub\_Menu;

## Read\_Report\_Disk\_File:

+++++

error\_flag => A byte holds an error\_flag value.  
 crt\_num => A byte, the number of crt to display on.  
 count => A byte used as a count.  
 page => A word holds the page number.  
 static\_asc\_size => A word holds the static ascii size.  
 screen\_param\_size => A word holds the screen parameter size.  
 temp\_word => A word holds a temp. location.  
 edit\_level\_p => A pointer - the address to the edit\_level  
 static\_asc\_p => A pointer - the address to static ascii.  
 screen\_param\_p => A pointer - the address to the screen parameter.  
 static\_asc BASED static\_asc\_p => 8713 bytes.  
 screen\_param BASED screen\_param\_p (601) STRUCTURE:  
 point\_id => 10 bytes hold the report id.  
 struc\_type => A byte holds the structure type value.  
 struc\_index => A word holds the structure index.  
 field => A byte holds the field number.  
 level => A byte holds the level number.



Proc description :

-----  
Reads the report disk file which is in the format REPOR000X.PARM  
END Read\_Report\_Disk\_File;

Save\_Disk\_File:

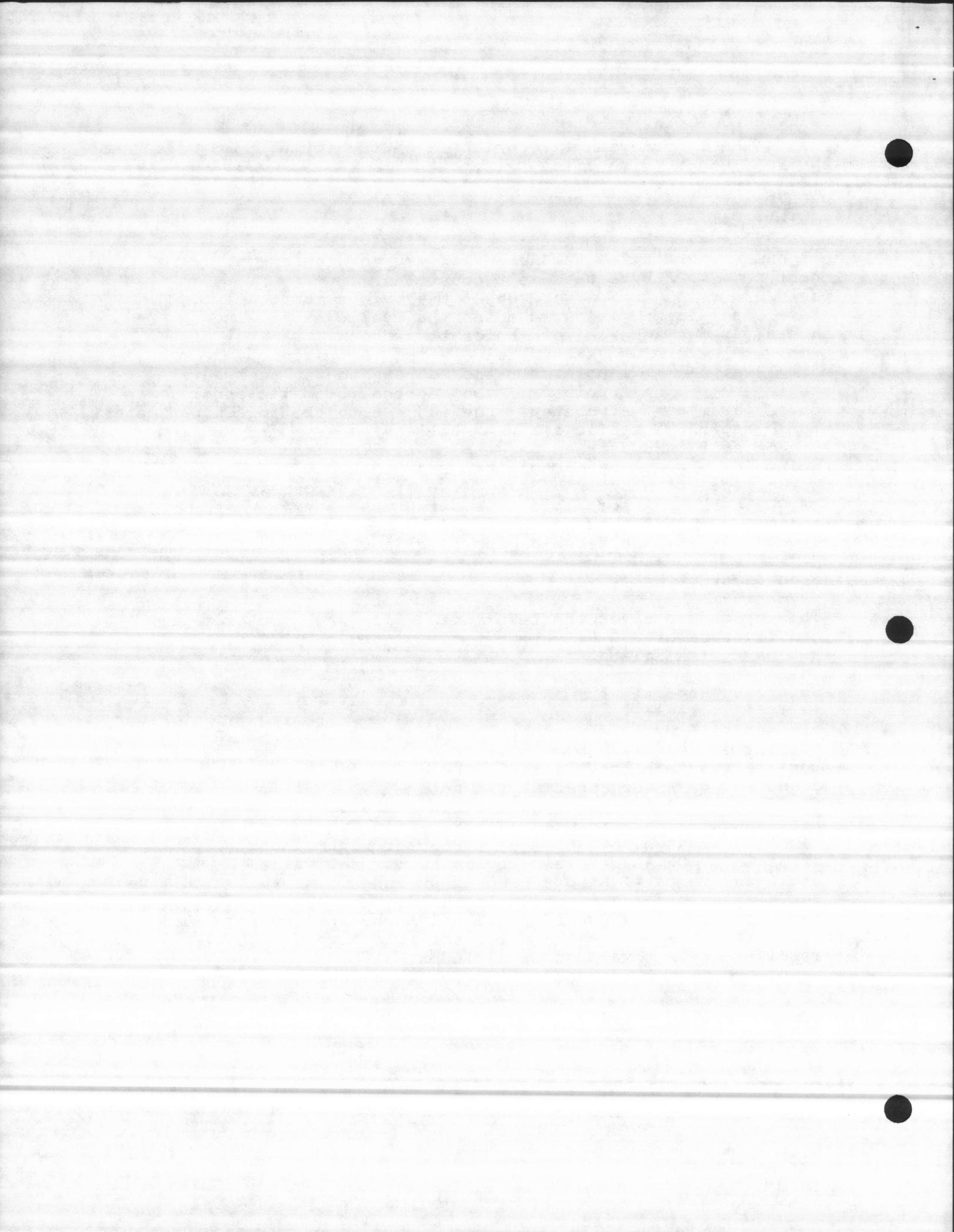
++++++  
error\_flag => A byte holds an error\_flag value.  
crt\_num => A byte, the number of crt to display on.  
count => A byte used as a count.  
page => A word holds the page number.  
static\_asc\_size => A word holds the static ascii size.  
screen\_param\_size => A word holds the screen parameter size.  
temp\_word => A word holds a temp. location.  
edit\_level\_p => A pointer - the address to the edit\_level  
static\_asc\_p => A pointer - the address to static ascii.  
screen\_param\_p => A pointer - the address to the screen parameter.  
static\_asc BASED static\_asc\_p => 8713 bytes.  
screen\_param BASED screen\_param\_p (601) STRUCTURE:  
point\_id => 10 bytes hold the report id.  
struc\_type => A byte holds the structure type value.  
struc\_index => A word holds the structure index.  
field => A byte holds the field number.  
level => A byte holds the level number.

Proc description :

-----  
This procedure saves the report disk file when you exit from  
construct report.  
END Save\_Disk\_File;

Construct\_Report:

++++++  
crt\_num => A byte, the number of the crt to display on.  
disk\_error\_flag => A byte holds the disk\_error\_flag value.  
row => A byte holds the row number.  
count => A word used as a count.  
field => A word used as a field number.  
left\_index => A word used as parameter left index.  
right\_index => A word used as parameter right index.  
off\_set => A word value holds an off\_set value.  
remaining\_units => A word used as an index.  
max\_display\_size => A word value holds the maximum display size.  
max\_page\_size => A word value holds the maximum page size.  
file\_size => A word value holds the file size.  
result => A word used as an index.  
temp\_word => A word used as a temp. location.



field\_asc => Three bytes hold the field ascii.  
 static\_disk\_buffer\_seg\_t => A token used to create the static disk buffer segment.  
 fore\_color\_buffer\_seg\_t => A token used to create fore color buffer segment.  
 back\_color\_buffer\_seg\_t => A token used to create the back color buffer segment.  
 edit\_level => Two bytes hold the edit level  
 static\_asc BASED static\_disk\_buffer\_seg\_t => 8713 bytes.  
 table\_p => A pointer - the address to the crt display table.  
 crt\_table\_seg\_t => A token used to create the crt table segment.  
 screen\_param\_seg\_t => A token used to create the screen parameter segment.  
 screen\_param BASED screen\_param\_p (613) STRUCTURE:  
 point\_id => 10 bytes hold the report id.  
 struc\_type => A byte holds the structure type value.  
 struc\_index => A word holds the structure index.  
 field => A byte holds the field number.  
 level => A byte holds the level number.

#### Proc description :

- 
- 1. displays construct\_report.
- 2. calls stop\$update.
- 3. initializes the parameter crt table and creates a segment for it.
- 4. paging defination.
- 5. Creates the data segment.
- 6. defines the file\_size & max\_page\_size equal to 8713.
- 7. creates the screen parameter segment, the static\_disk\_buffer\_segment,
- 8. checks for report disk file if it exists then read it otherwize creates a new report file.
- 9. displays the construct menu driver.

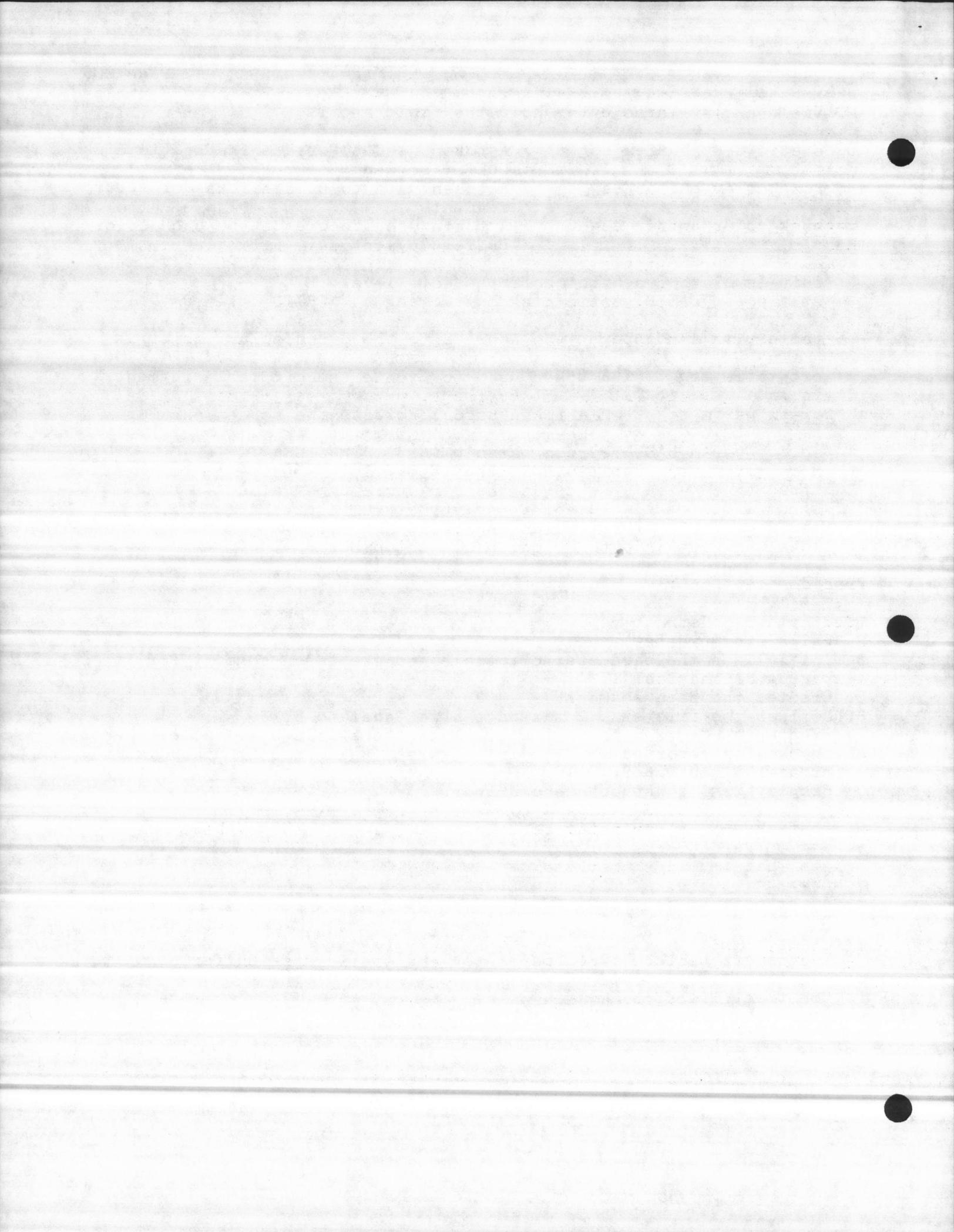
|           |           |      |      |
|-----------|-----------|------|------|
| configure | parameter | Edit | Next |
|-----------|-----------|------|------|

#### Note That :

=====

After edit an auto save of the disk file will take place.

- 10. deletes crt table, screen parameter and static disk buffer segments
- END Construct\_Report;



(17.2) MODULE NAME : CxxxxPRTG.Pyy

=====

DESCRIPTION :

===== This module prints the constructed reports.

EXTERNAL PROCEDURES :

=====

```
Menu_Display: PROCEDURE (crt_num, result, off_set, start_id_p,
                         bytes_per_element, num_elements) WORD EXTERNAL;
DECLARE (crt_num) BYTE,
        (result, off_set) WORD,
        (bytes_per_element, num_elements) WORD,
        (start_id_p) POINTER;
```

END Menu\_Display;

```
Format_Disk_Error: PROCEDURE (buffer_t, message_str_p, disk_error,
                               disk_operation) EXTERNAL;
DECLARE (disk_error, disk_operation) WORD,
        (buffer_t) TOKEN,
        (message_str_p) POINTER;
```

END Format\_Disk\_Error;

```
Display_Case: PROCEDURE(table_element_p, param_element_p, buf_t) EXTERNAL;
DECLARE buf_t TOKEN,
      (table_element_p, param_element_p) POINTER;
```

END Display\_Case;

Get\_Password:

```
PROCEDURE (crt_num, table_p, table_reg_t,
          display_buf_t, display_mbx_t,
          ci_conn_t, ci_mbx_t) BYTE EXTERNAL;
DECLARE (crt_num) BYTE,
        (table_reg_t, display_buf_t, display_mbx_t,
         ci_conn_t, ci_mbx_t) TOKEN,
        (table_p) POINTER;
```

END Get\_Password;

```
Table_Fill: PROCEDURE(field_number, element,
                      struc_type, table_p, param_p) WORD EXTERNAL;
DECLARE (table_p, param_p) POINTER,
        (field_number, element) WORD,
        (struc_type) BYTE;
```

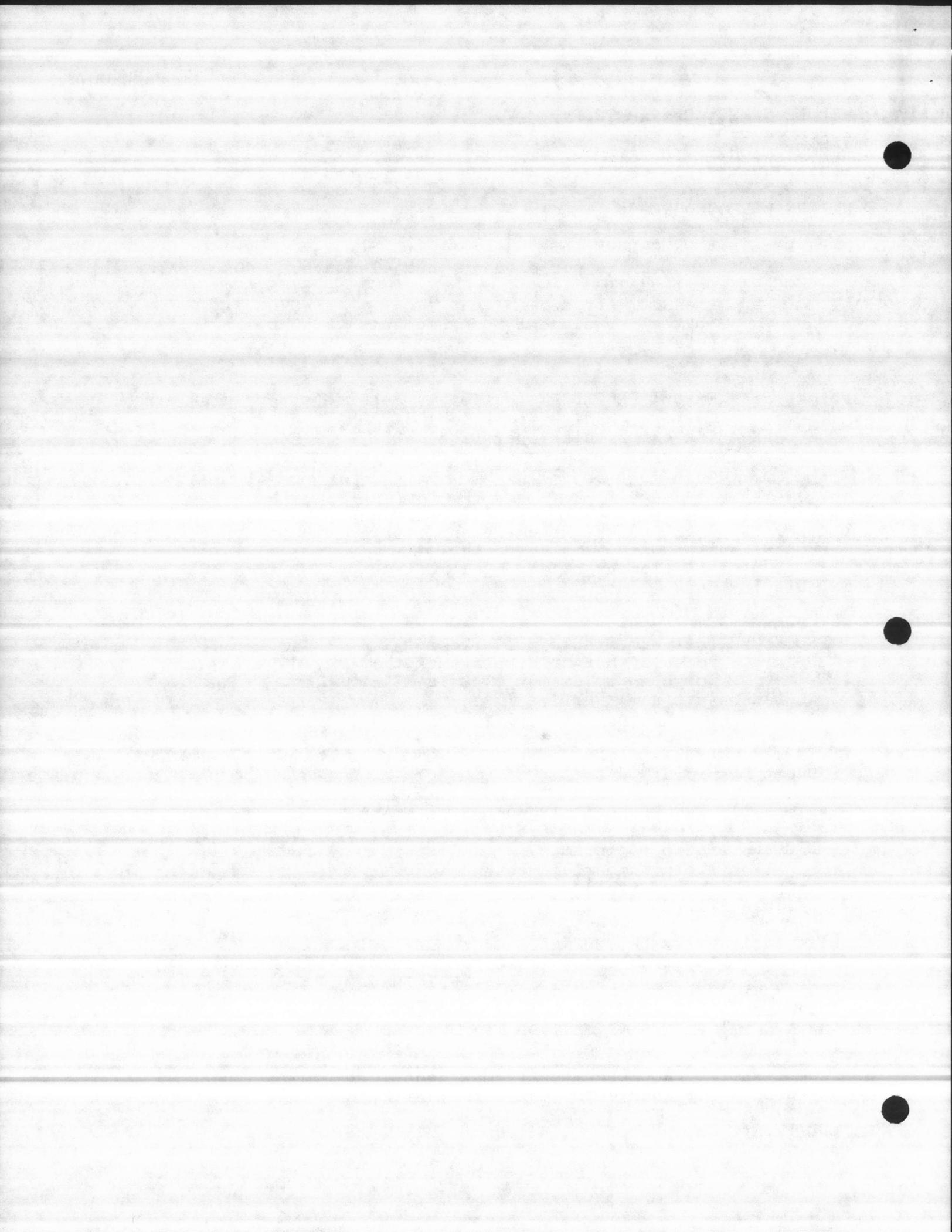
END Table\_Fill;

```
Hist_table_fill: PROCEDURE(field_number, element,
                           struc_type, table_p, param_p) WORD EXTERNAL;
DECLARE (table_p, param_p) POINTER,
        (field_number, element) WORD,
        (struc_type) BYTE;
```

END Hist\_table\_fill;

```
Prompt_Entry: PROCEDURE (entry_max_count, crt_num) EXTERNAL;
DECLARE (crt_num, entry_max_count) BYTE;
```

END Prompt\_Entry;



```
Read_Report_Disk_File: PROCEDURE (page,crt_num,edit_level_p,
    static_asc_p,static_asc_size,
    screen_param_p, screen_param_size) BYTE EXTERNAL;
DECLARE (crt_num) BYTE,
    (page,static_asc_size,screen_param_size) WORD,
    (edit_level_p, static_asc_p,screen_param_p) POINTER;
END Read_Report_Disk_File;
```

GLOBAL REFERENCE :

=====

```
data base & disp_report structrues (See CxxxxGLOB.Pyy)
```

MODULE DECLARATION :

=====

```
read_file_param BASED disk_param_seg_t STRUCTURE &
write_file_parameters BASED disk_param_seg_t STRUCTURE
(See disk system)
```

```
header_info (4) STRUCTURE &
header_data (4) STRUCTURE
(See 17.1)
```

```
exception      => Three words, each holds the exception condition.
crt_update_sem_t => Three crt update semaphore tokens, each used to
create the update task semaphore.
```

PUBLIC PROCEDURES :

=====

Keyin\_Print\_Report\_Sub\_Menu:

+++++-----+

```
crt_num  => A byte, the number of the crt to display on.
```

```
result   => A word used as an index.
```

```
param_seg_t=> A token used to create report param segment.
```

```
report_print_param BASED param_seg_t STRUCTURE :
```

```
file_name    -> Twenty bytes hold the report file name
                  file_name(Ø) = Ø for current.
```

```
start_index  -> A word used as the start index.
```

```
error        -> A word value holds the error index.
```

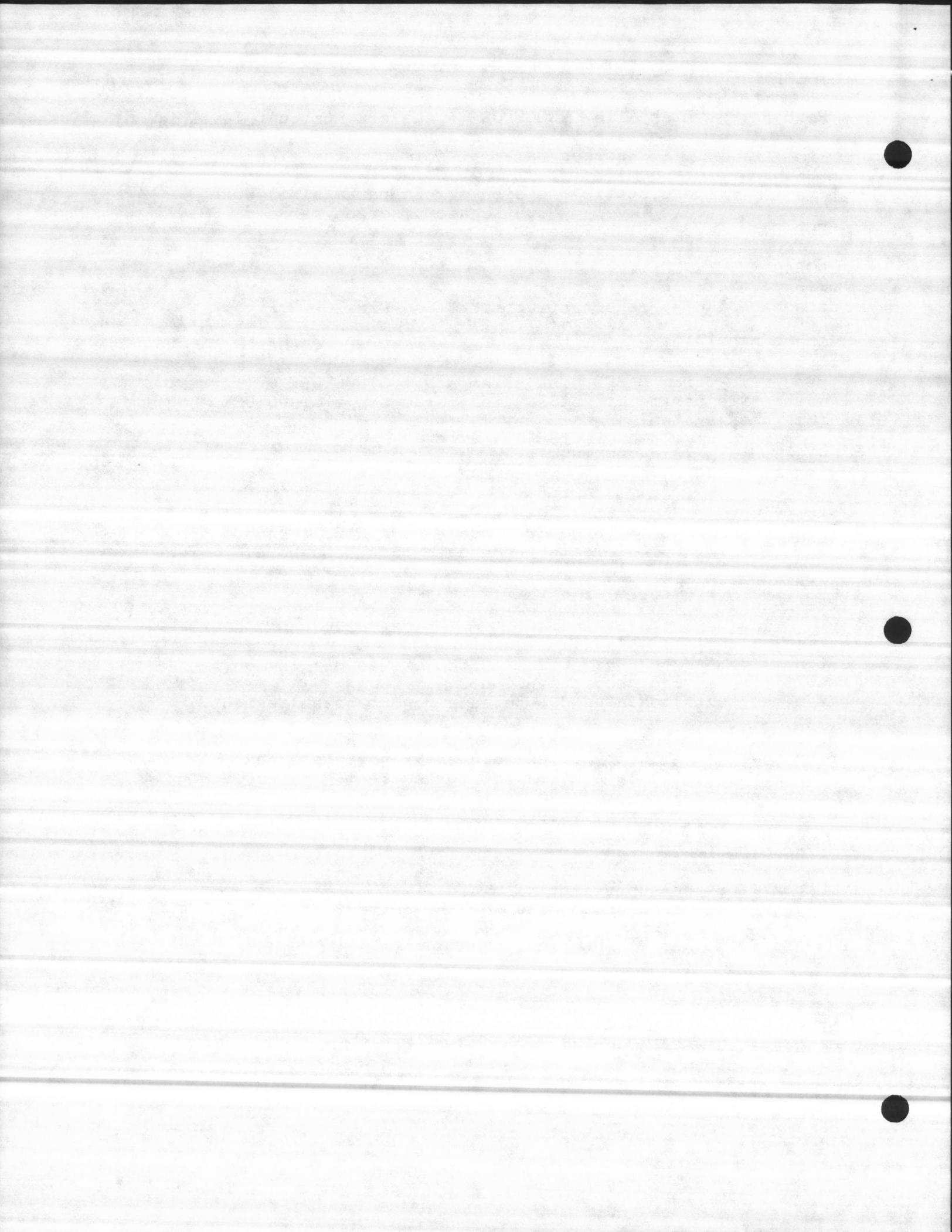
Proc description :

-----

1. Checks for the supervisor password.

2. Displays the gen. print menu for the current data.

```
END Keyin_Print_Report_Sub_Menu;
```



**Hist\_Print\_Report\_Sub\_Menu:**

++++++  
crt\_num => A byte, the number of the crt to display on.  
result => A word used as an index.  
param\_seg\_t=> A token used to create report param segment.

report\_print\_param BASED param\_seg\_t STRUCTURE :  
file\_name -> Twenty bytes hold the report file name.  
                  file\_name (Ø) = 1 for hist.  
start\_index -> A word used as the start index.  
error         -> A word value holds the error index.

**Proc description :**

- 1. Checks for the supervisor password.  
2. Displays the gen. print menu for the historical data.

END Hist\_Print\_Report\_Sub\_Menu;

**Expand\_Date:**

++++++

**Proc description :**

-----  
generates the expanded data for printing.

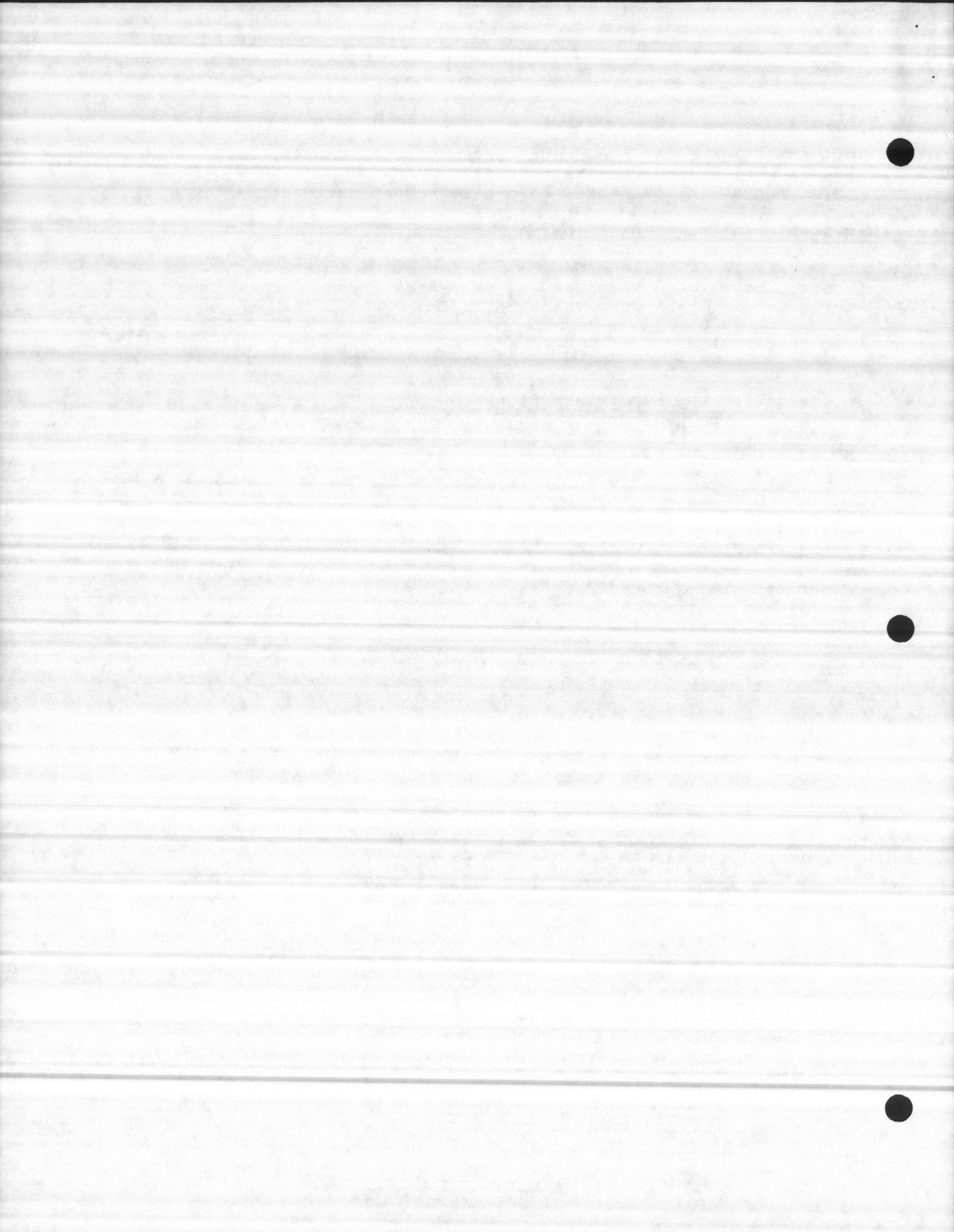
END Expand\_Date;

**Print\_Report\_Task:**

++++++

crt\_num => A byte, the number of the crt to display on.  
disk\_error\_flag => A byte holds the disk\_error\_flag value.  
end\_of\_report => A byte holds the end of report mark.  
found\_flag    => A byte holds the found\_flag value.

count       => A word used as a count.  
field       => A word used as a field number.  
remaining\_units => A word used as an index.  
result      => A word used as an index.  
temp\_word => A word used as a temp. location.



param\_seg\_t => A token used to create report param segment.

report\_print\_param BASED param\_seg\_t STRUCTURE :  
file\_name -> Twenty bytes hold the report file name.  
                  file\_name (Ø) = 1 for hist.

start\_index -> A word used as the start index.

error -> A word value holds the error index.

start\_tag\_index => A word holds the index of the first tag name.

link => A word holds the value of the link.

exception => A word holds the exception code number.

tag\_number=> A word holds the tag number.

num        => A word used as an index.

print\_time => A dword holds the print time.

dummy\_dword=> A dword used as a temp. location.

edit\_level => Two bytes hold the edit level

static\_asc BASED static\_disk\_buffer\_seg\_t => 8713 bytes.

table\_p => A pointer - the address to the crt display table.

crt\_table\_seg\_t => A token used to create the crt table segment.

screen\_param\_seg\_t => A token used to create the screen parameter segment.

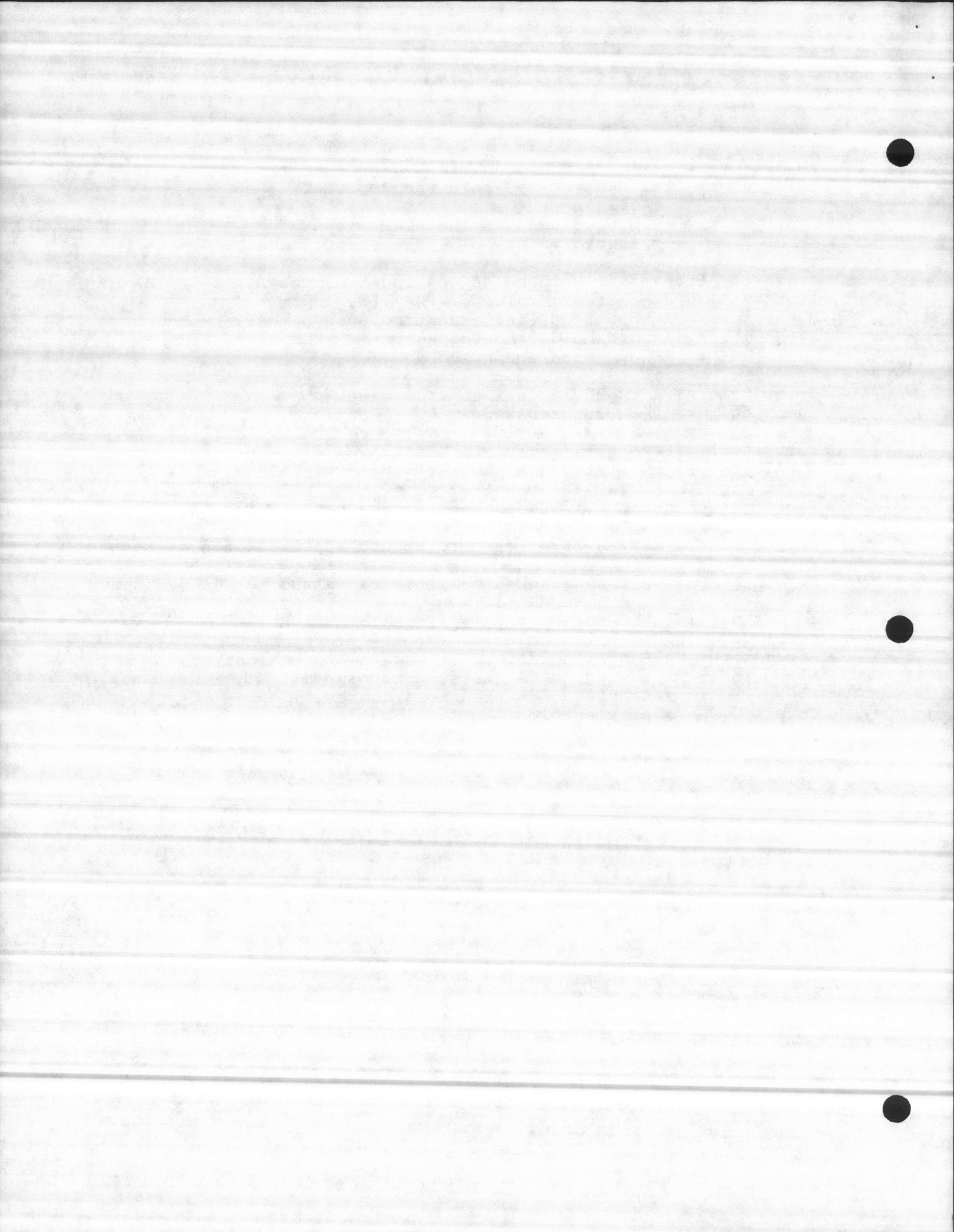
screen\_param BASED screen\_param\_p (613) STRUCTURE:  
point\_id   => 10 bytes hold the report id.

struc\_type => A byte holds the structure type value.

struc\_index => A word holds the structure index.

field      => A byte holds the field number.

level      => A byte holds the level number.



Task Algorithm :

-----

Set the exception handler.

DO FOREVER;

    wait for mailbox  
    create the crt table segment.  
    get current time and deposit it into the print time.

    If the first byte of the file name greater than zero  
        then deposit the historical time into print time.

    link = report\_print\_param.start\_index;  
    end\_of\_report = Ø;

    Create the data segments for screen parameters & static ascii

    Get control of the line printer region.

DO WHILE end\_of\_report <> ØFFH;

    Check for Disk File

    IF disk\_error\_flag <> Ø THEN  
        end\_of\_report = ØFFH;  
    ELSE  
        DO;  
            new for FF in the begining of the report because of only  
            one lp used in this job  
        END;

    Print the report

    skip past blank lines

    Check for start of a parameter

    Write the time and date

    Print Actual Data

END;                         End Of Report While

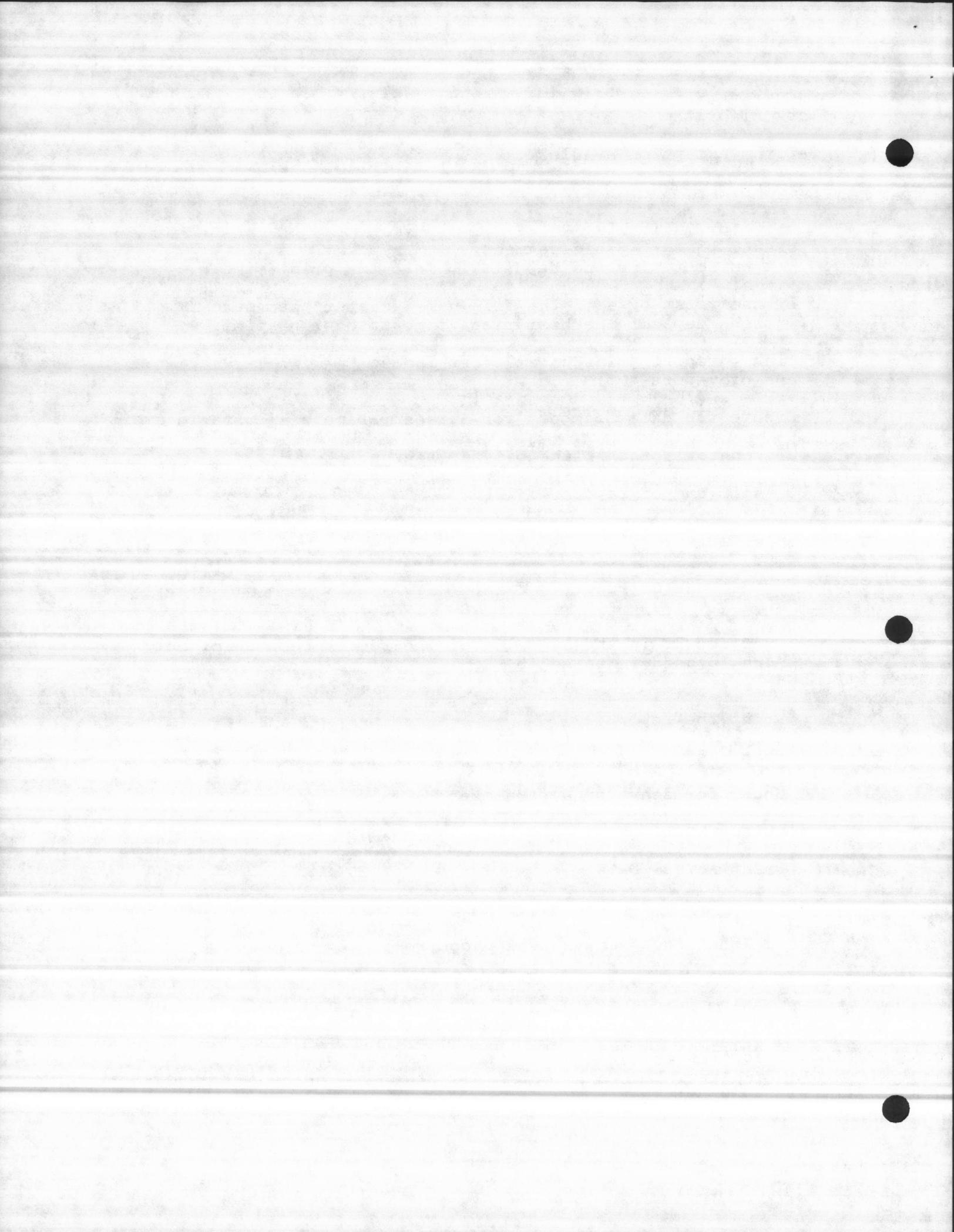
release the line printer region control.

delete the data segments.

Send message that report is complete

End of Forever

END Print\_Report\_Task;



(17.3) MODULE NAME : CxxxxERPT.Pyy

=====

DESCRIPTION :

=====

This module is a full screen editor used to construct a report.

EXTERNAL PROCEDURES :

=====

Format\_Cursor\_And\_Color:

```
PROCEDURE (display_buf_t, row, column, color_index) EXTERNAL;
DECLARE (row, column) BYTE,
        (display_buf_t) TOKEN,
        (color_index) BYTE;
```

END Format\_Cursor\_And\_Color;

Enter\_Password\_And\_Verify: PROCEDURE(crt\_num) WORD EXTERNAL;

returns :

Ø - password entered is incorrect or display invoked,  
1 to 3 : password entered has matched level

(equal passwords will return highest level).

display invoked by entering PATIODOOR as password  
default password is PASSWORD \*/

          (crt\_num) BYTE;

END Enter\_Password\_And\_Verify;

Prompt\_Display:

```
PROCEDURE (crt_num, display_buf_t, crt_out_mbx_t,
          row, column, display_str_p) EXTERNAL;
```

```
          (row, column, crt_num)                   BYTE,
          (display_buf_t, crt_out_mbx_t)           TOKEN,
          (display_str_p)                            POINTER;
```

END Prompt\_Display;

GLOBAL REFERENCE :

=====

See CxxxxGLOB.Pyy

See CxxxxEDIT.Pyy

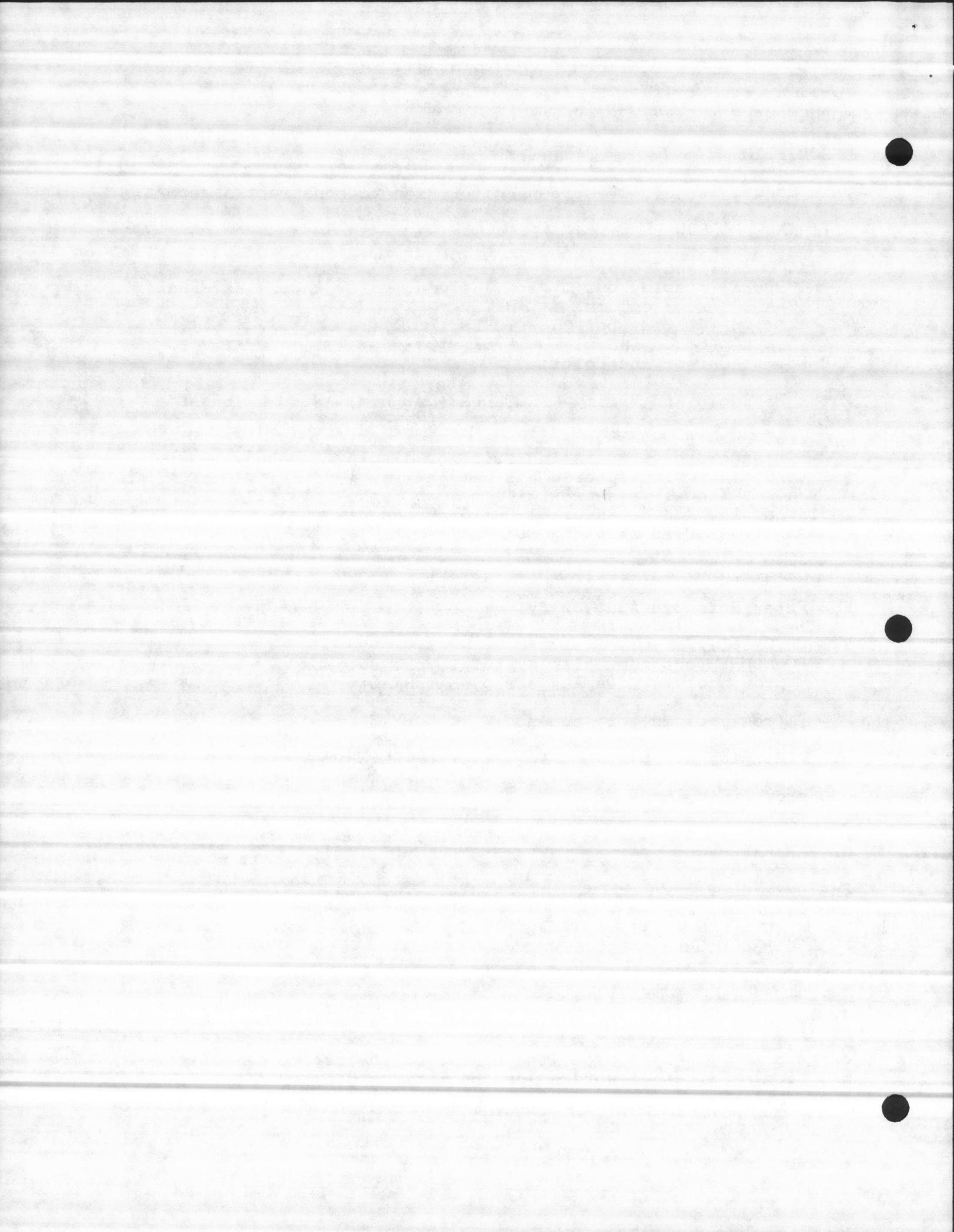
PUBLIC PROCEDURES :

=====

Edit\_Report\_Screen:

+++++  
+++++

crt\_num               => A byte, the number of the crt to display on.  
max\_display\_size=> A word holds the maximum display size.



display\_buf\_t => A display buffer token.  
display\_mbx\_t => A display mailbox token.  
edit\_buf\_t => An edit buffer token.  
ci\_conn\_t => A CI connection token.  
ci\_mbx\_t => A CI mailbox token.  
static\_asc\_p => A pointer - the address of the static ascii.

char => A byte holds a character.  
char\_case => A byte used as an index.  
char\_index => A word used as an index.  
current\_table\_index => A word used as an index to the current crt table.  
previous\_table\_index => A word used as an index to the previous table.  
view\_offset => A word holds the view\_offset value.  
col\_view\_offset => A word holds the column view offset.  
old\_view\_offset => A word holds the old view offset.  
old\_col\_view\_offset => A word holds the old column view offset.

Maximum of a 43 X 80 screen

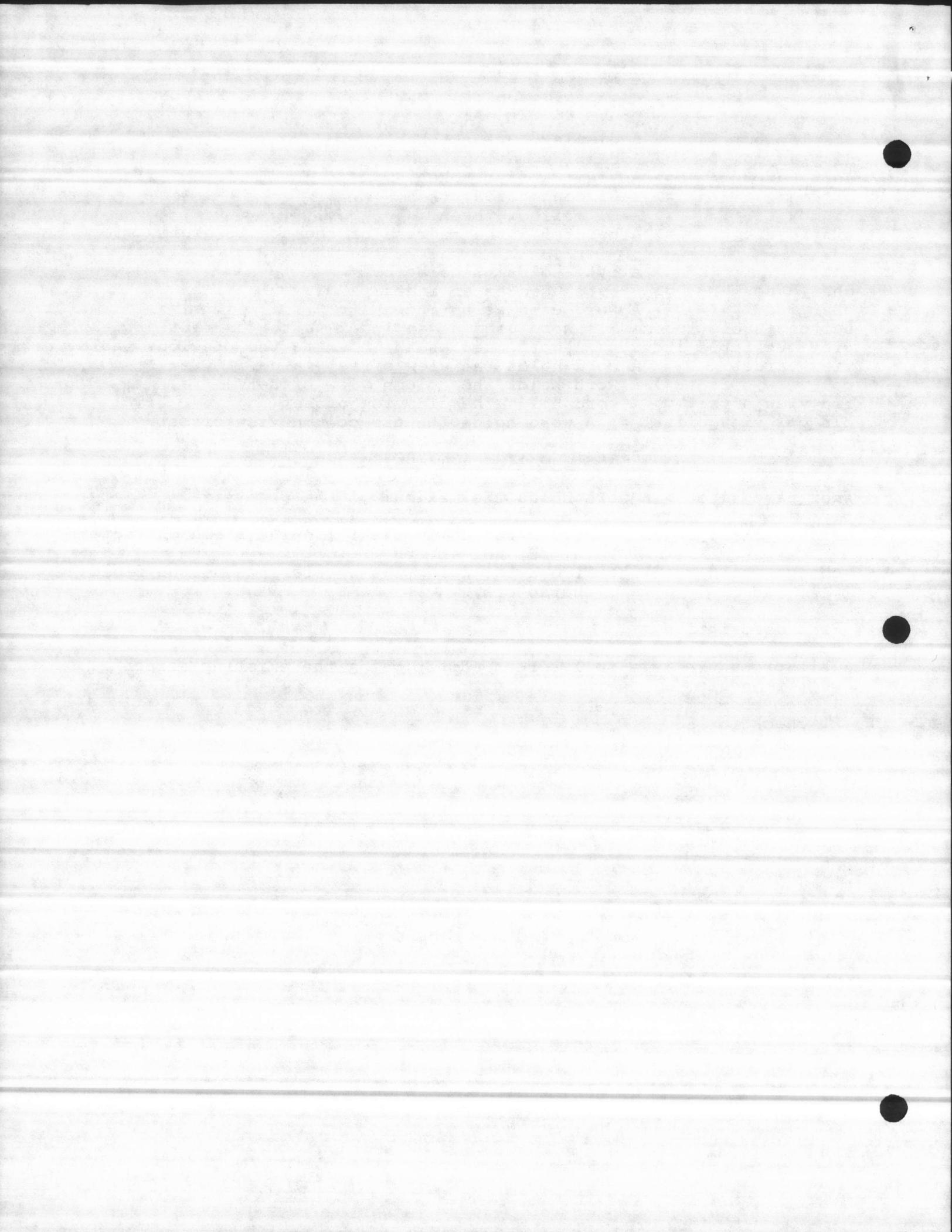
max\_page\_size => A word holds the maximum of the page size.  
change\_field\_chars => Five bytes holds the following field characters  
1H, 2H, 3H, 4H, 5H  
change\_field\_xref => Five bytes holds the x\_reference to the field  
characters.

#### Proc description :

This procedure is a full screen editor using the existing screen editor module.

1. get character.
2. check for ESC.
3. enter character into the crt table.
4. cursor movement and cursor control  
invalid input, bell sound.  
cursor forward.  
cursor backward  
cursor up.  
cursor down.  
Home.

END Edit\_Report\_Screen;



(17.4) MODULE NAME : CxxxxRGEN.Pyy

=====

DESCRIPTION :

=====

This module displays the generated report static.

PUBLIC PROCEDURES :

=====

Display\_Report\_Static:

+++++-----+

crt\_num => A byte, the number of the crt to display on.

max\_display\_size=> A word holds the maximum display size.

display\_buf\_t => A display buffer token.

display\_mbx\_t => A display mailbox token.

static\_asc\_p => A pointer - the address of the static ascii.

char => A byte holds a character.

view\_offset => A word holds the view\_offset value.

col\_view\_offset => A word holds the column view offset.

Maximum of a 43 X 80 screen

max\_page\_size => A word holds the maximum of the page size.

count => A word holds a count value.

line => A word holds the line number.

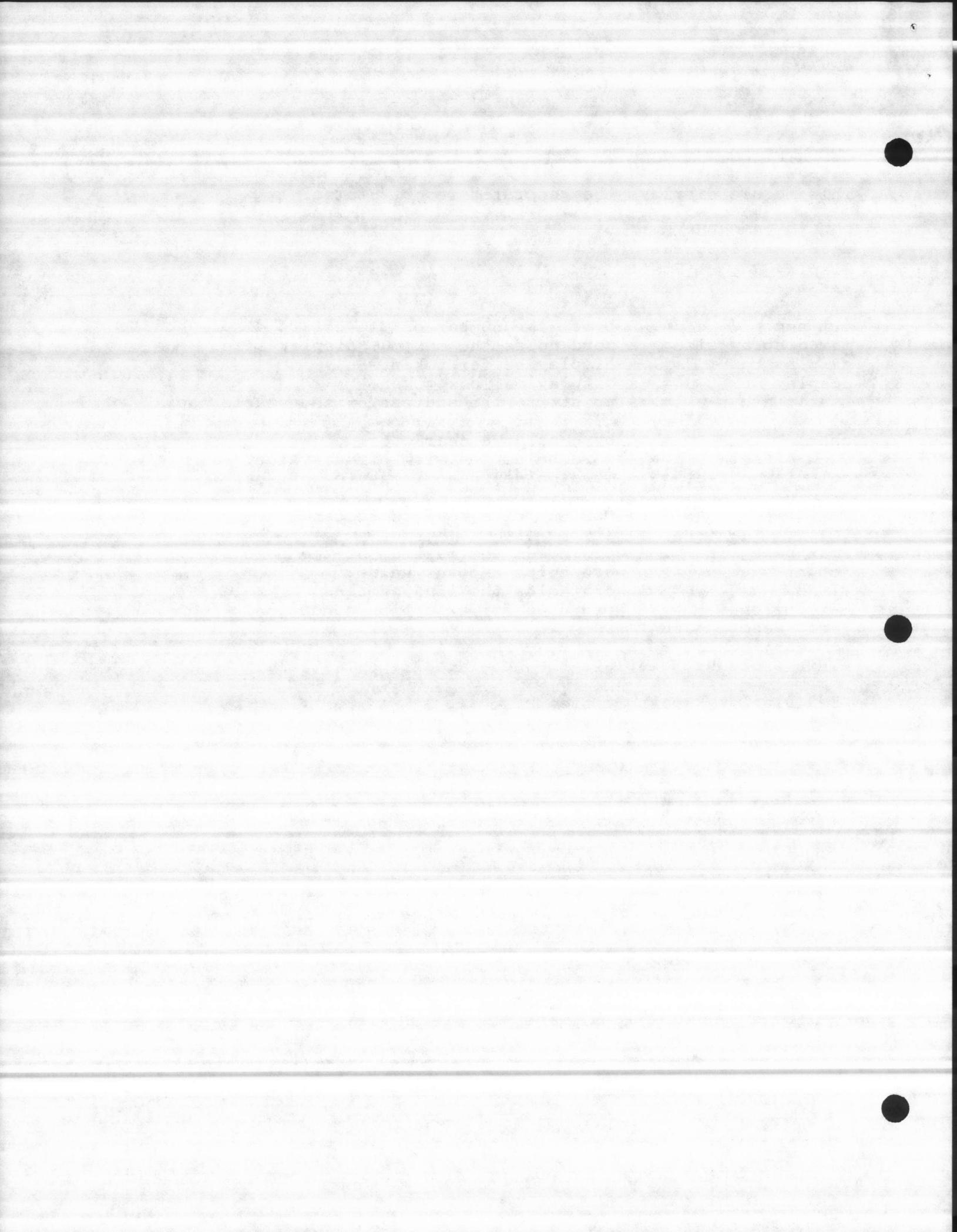
exception => A word used for exception condition.

Proc description :

-----

This procedure displays the generated report static.

END Display\_Report\_Static;



(17.5) MODULE NAME : CxxxxPERD.Pyy

=====

DESCRIPTION :

=====

This module monitors the periodic report setup.

GLOBAL REFERENCES :

=====

disp\_report structure (See CxxxxGLOB.Pyy).

PUBLIC PROCEDURES :

=====

Start\_Period\_Report\_Link:

++++++

index => A word value used as an index.

report\_type => A word value holds the report type.

Proc Description :

-----

This procedure goes through all the available reports, and finds out which periodic report to be printed on auto periodic setup.

End Start\_Period\_Report\_Link;

MODULE TASKS :

=====

Period\_Report\_Task:

++++++

count => A word used as a count.

exception => A word used for the exception condition.

remaining\_units => A word value used as an index.

temp\_word => A word value used as a temp. location.

current\_time => A dword holds the current time.

scan\_sem\_t => A scan semaphore token used to create a scan semaphore.

report\_param\_seg\_t=> A token used to create report param segment.

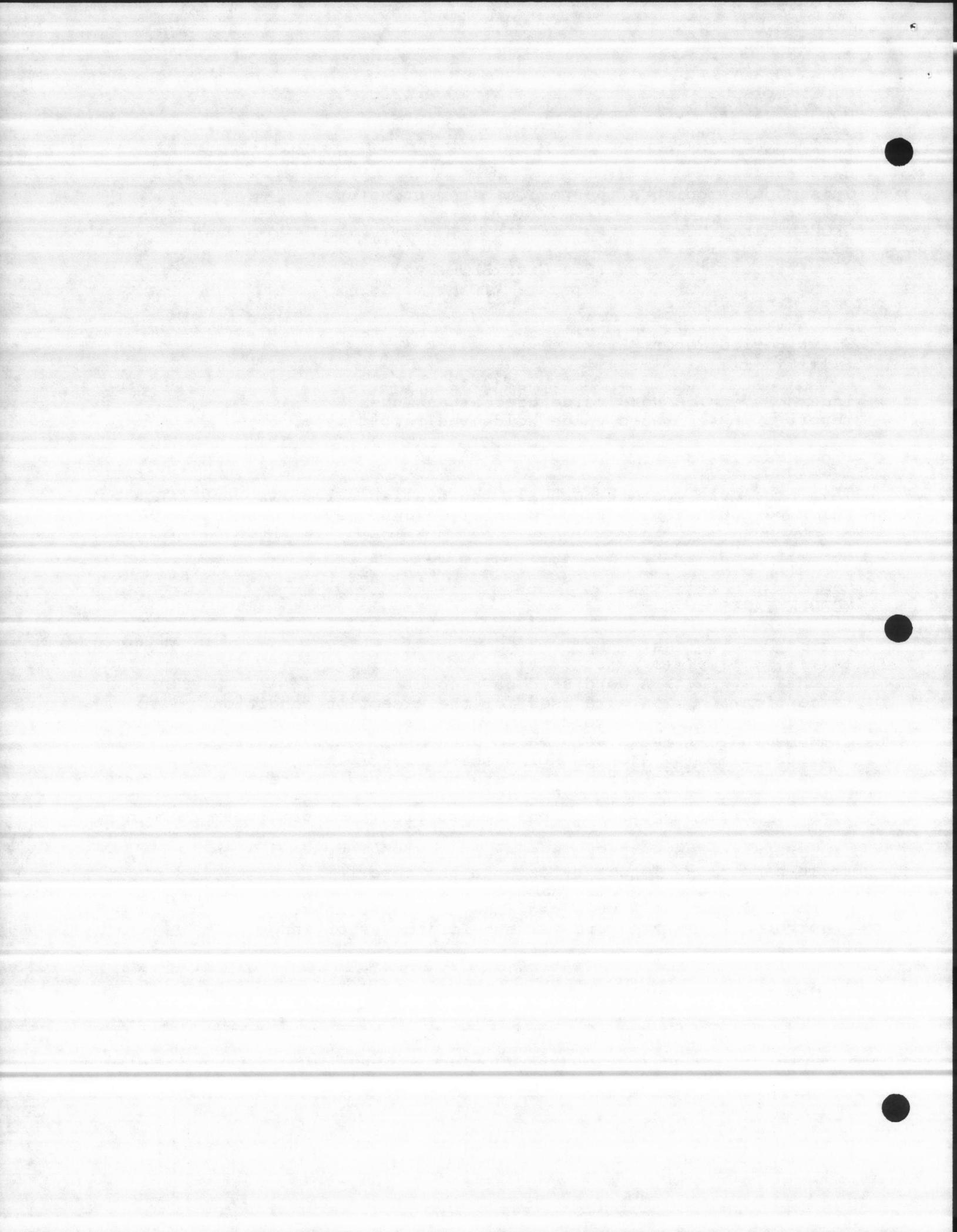
report\_print\_param BASED param\_seg\_t STRUCTURE :

file\_name -> Twenty bytes hold the report file name.

the file\_name(Ø) = Ø.

start\_index -> A word used as the start index.

error -> A word value holds the error index.



Task Algorithm :

Set the exception handler.

Create the scan semaphore.

Create a clock entry with 10 second scan time.

Create the report parameters segment.

DO FOREVER;

    wait until time up

    get current time.

    Loop1 go through all the reports

        if the frequency equal to periodic then  
            do;

            check the periodic report setup. if true then  
                adjust the periodic setup  
            end;

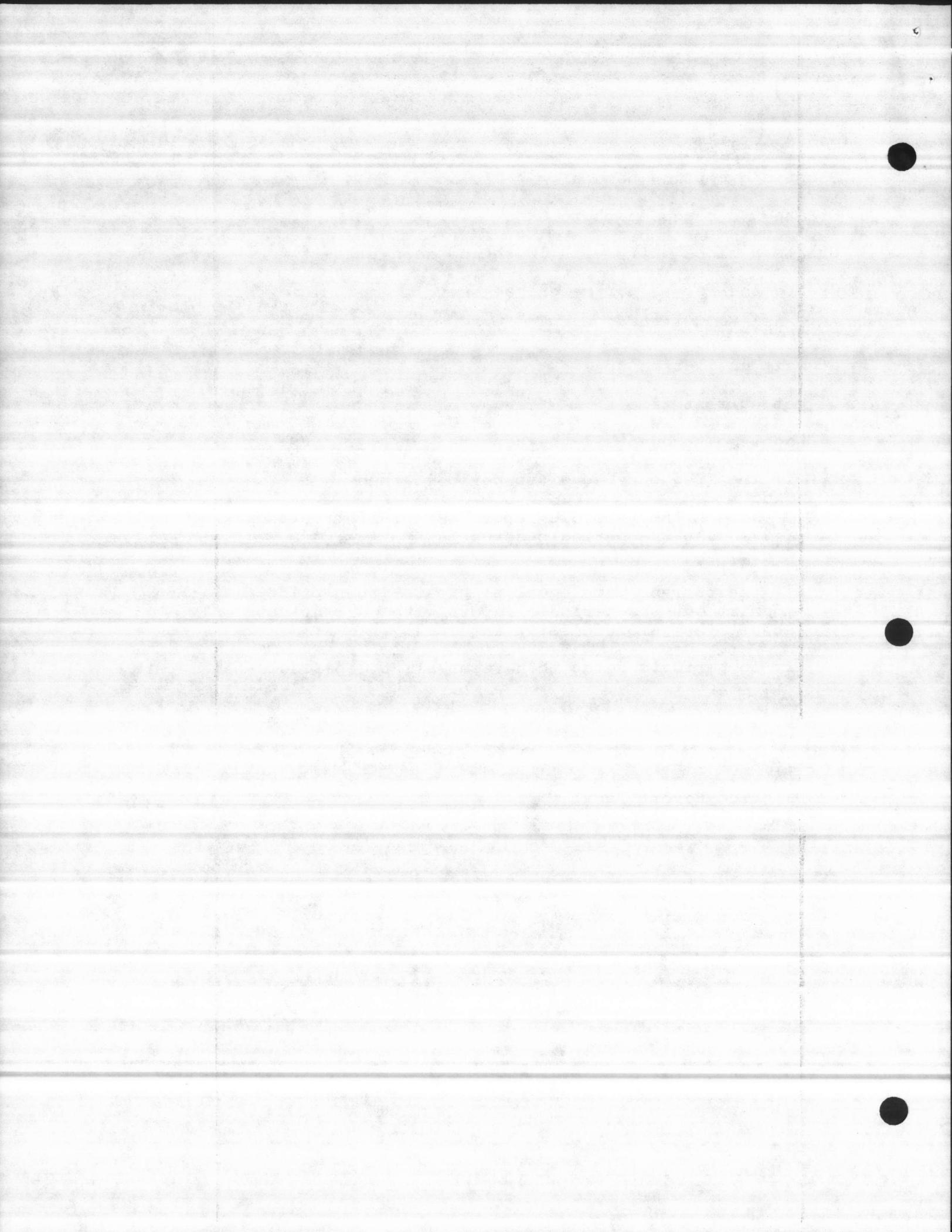
        If the current time equal or greater then print time then  
            print out the periodic report with current data only.  
            adjust the peroidic setup internally.

        If no periodic setup then reset all the periodic parameters  
            to zeros

    END; loop1

END; do forever

END Period\_Report\_Task;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 18

T E S T

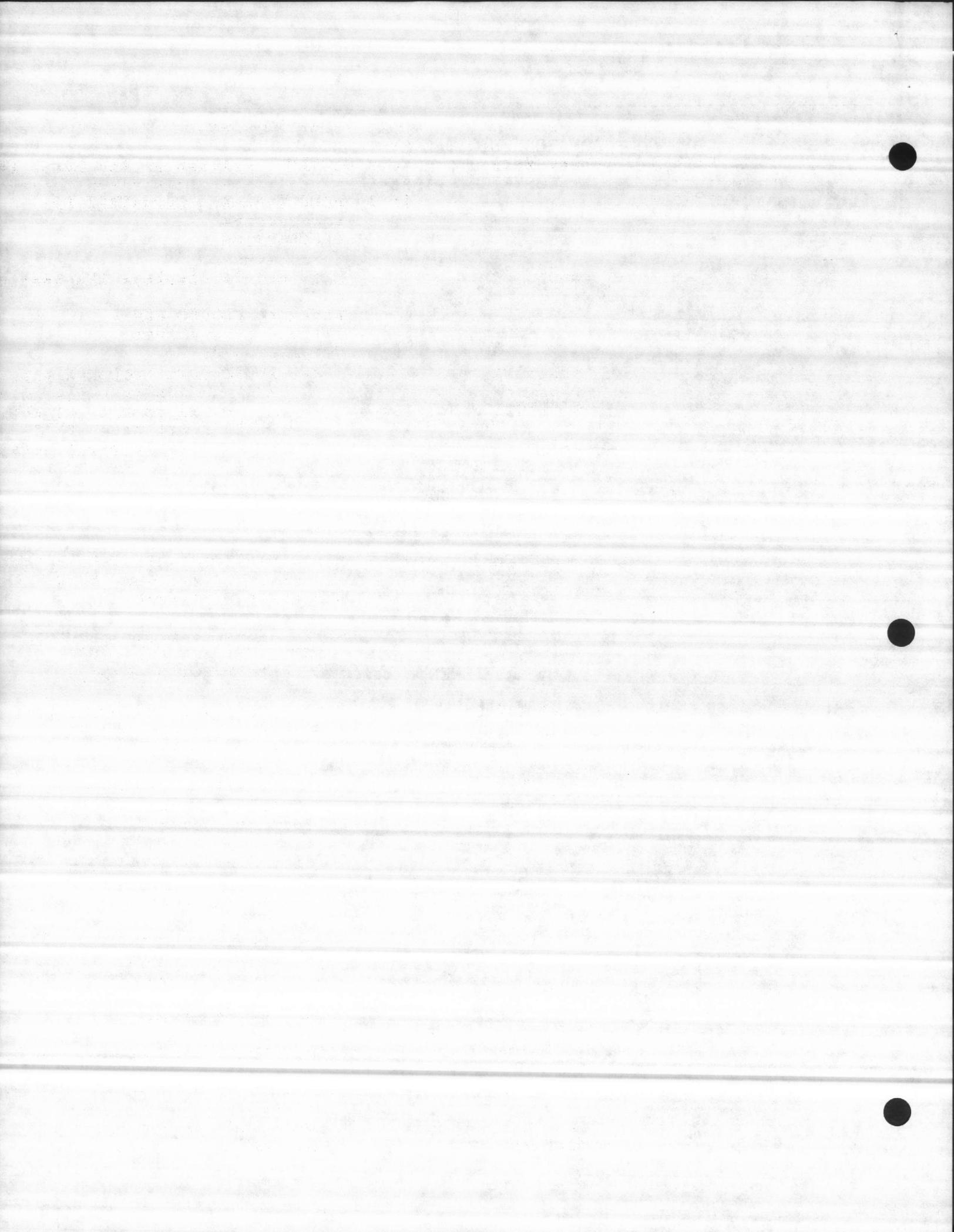
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

TEST SYSTEM - written by Bob Ryan, Peter Wollenzien.

(18.1) MODULE NAME : CxxxxTEST.Pyy

=====

DESCRIPTION :

=====

This module displays the test menu.

EXTERNAL PROCEDURES :

=====

Get\_Password:

```
PROCEDURE (crt_num, table_p, table_reg_t,
           display_buf_t, display_mbx_t,
           ci_conn_t, ci_mbx_t) BYTE EXTERNAL;
DECLARE   (crt_num) BYTE,
          (table_reg_t, display_buf_t,
           display_mbx_t,
           ci_conn_t, ci_mbx_t) TOKEN,
          (table_p)             POINTER;
```

END Get\_Password;

```
Keyin_Next_Proc: PROCEDURE (crt_num) EXTERNAL;
DECLARE crt_num BYTE;
END Keyin_Next_Proc;
```

```
Keyin_Previous_Proc: PROCEDURE (crt_num) EXTERNAL;
DECLARE crt_num BYTE;
END Keyin_Previous_Proc;
```

```
Remote_Display: PROCEDURE (crt_num) EXTERNAL;
DECLARE crt_num BYTE;
END Remote_Display;
```

```
Master_Display: PROCEDURE (crt_num) EXTERNAL;
DECLARE crt_num BYTE;
END Master_Display;
```

```
Io_Rack_Display: PROCEDURE (crt_num) EXTERNAL;
DECLARE crt_num BYTE;
END Io_Rack_Display;
```

```
Keyin_Edit_Sub_Menu_Proc: PROCEDURE (crt_num) EXTERNAL;
DECLARE crt_num BYTE;
END Keyin_Edit_Sub_Menu_Proc;
```

MODULE DECLARATION :

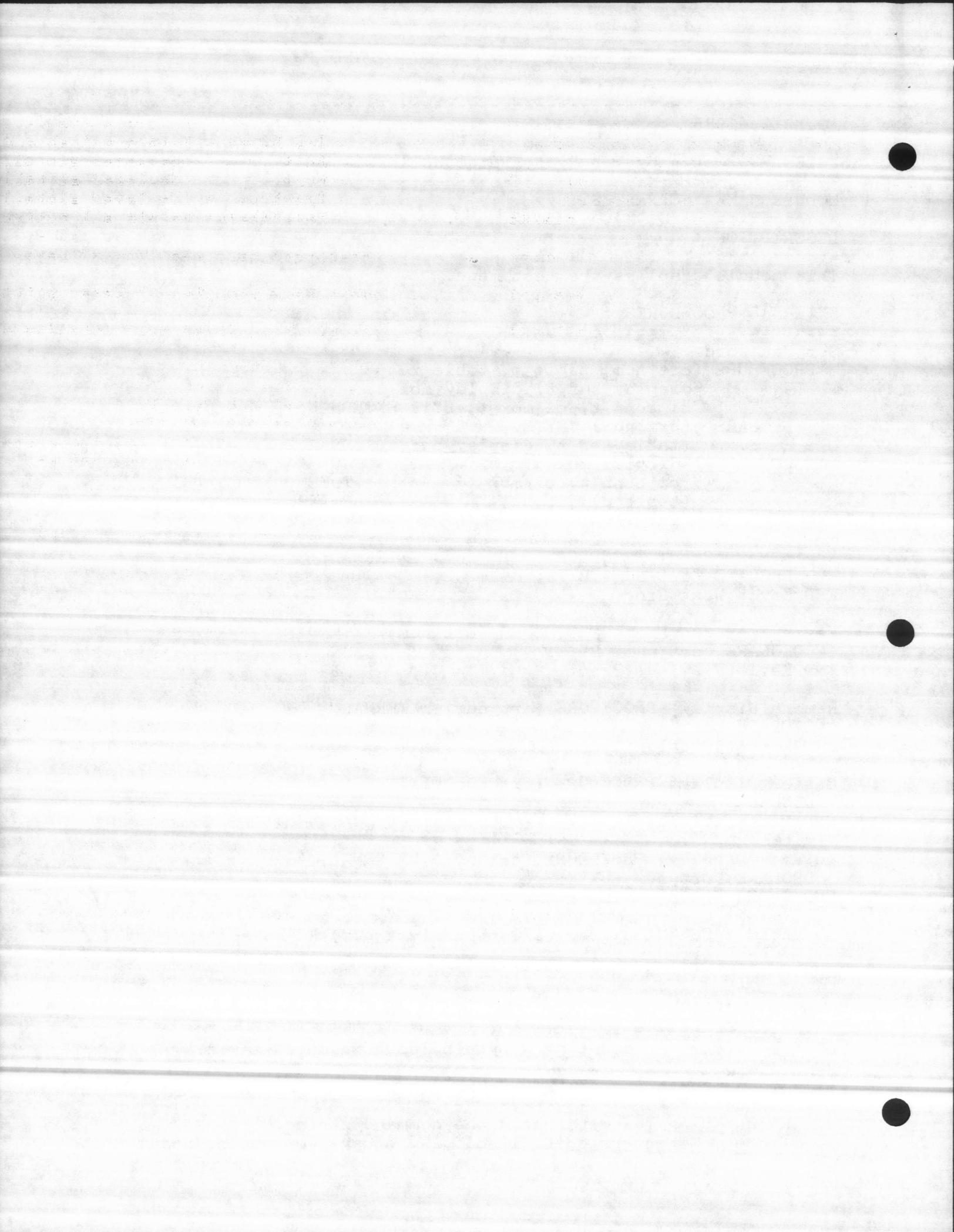
=====

main\_menu\_command\_count => A byte holds a value of seven.

main\_menu\_prompt\_asc (7) STRUCTURE :

-----  
menu\_asc => Ten bytes hold the command string.

Remote = Enters editable display of system remote data



Master = Enters editable display of system master data  
I/O Racks = Enters editable display of system i/o rack data  
Edit = (declared globally)  
Next = (declared globally)  
Previous = (declared globally)  
ResetAlarm= Resets all active alarms.

main\_menu\_help\_desc\_tbl => Seven pointers - points to the help table.

main\_menu\_proc\_tbl => Seven pointers points to the procedure table.

exception => Three word (one for each crt), used for exception condition.

PUBLIC PROCEDURES :

=====

Keyin\_Test\_Sub\_Menu\_Proc:

+++++  
crt\_num => A byte, the number of the crt to display on.  
result\_word => A word used as an index.

Proc description :

-----  
This procedure displays the test menu.

END Keyin\_Test\_Sub\_Menu\_Proc;

Keyin\_Remote\_Display:  
+++++  
crt\_num => A byte, the number of the crt to display on.

Proc description :

-----  
This procedure calls Remote\_Display Procedure.

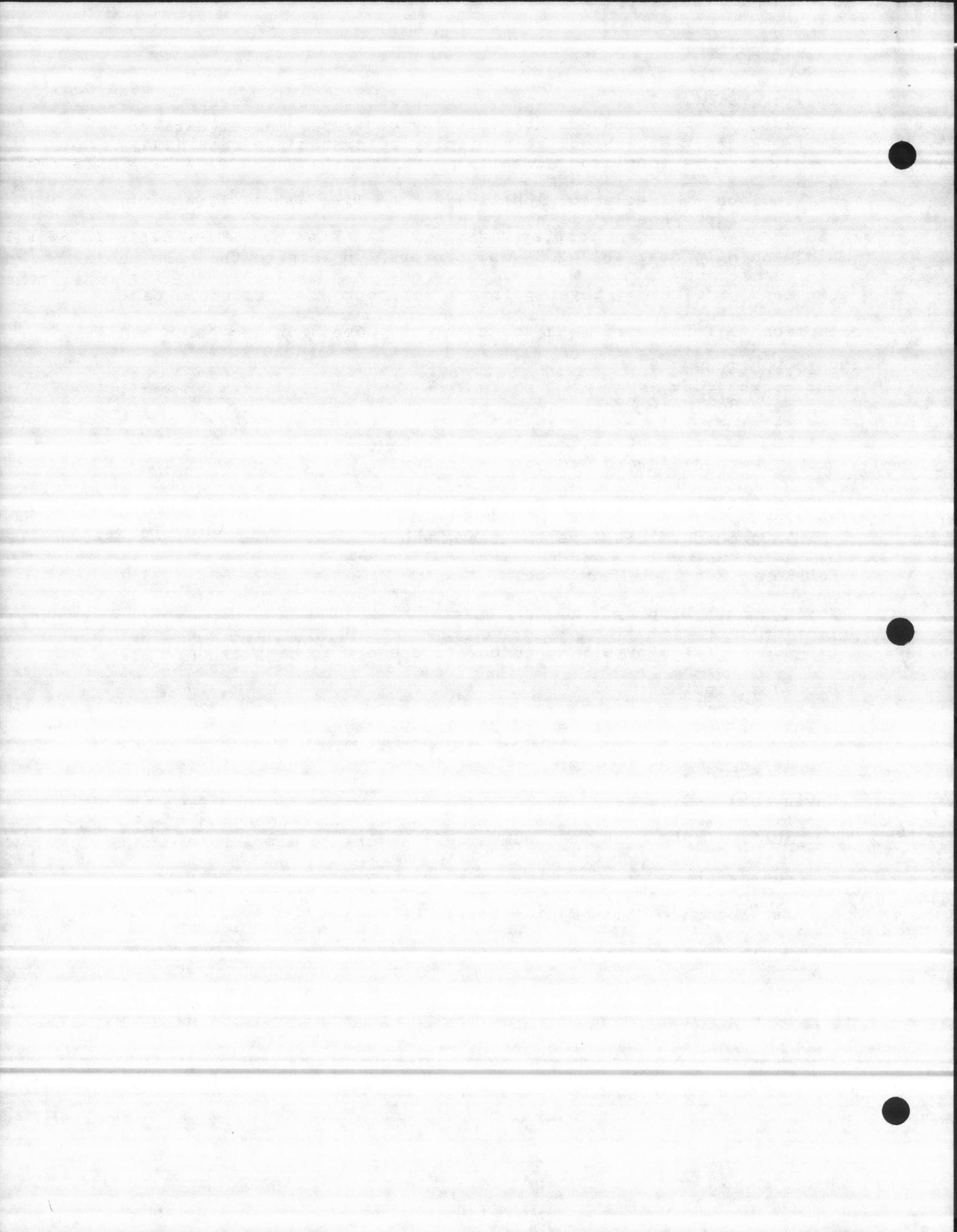
END Keyin\_Remote\_Display;

Keyin\_Master\_Display:  
+++++  
crt\_num => A byte, the number of the crt to display on.

Proc description :

-----  
This procedure calls Master\_Display Procedure.

END Keyin\_Master\_Display;



Keyin\_Io\_Rack\_Display:

+++++  
+++++  
+++++

crt\_num => A byte, the number of the crt to display on.

Proc description :

-----  
This procedure calls Io\_Rack\_Display Procedure.

END Keyin\_Io\_Rack\_Display;

Reset\_Alarms\_Proc:

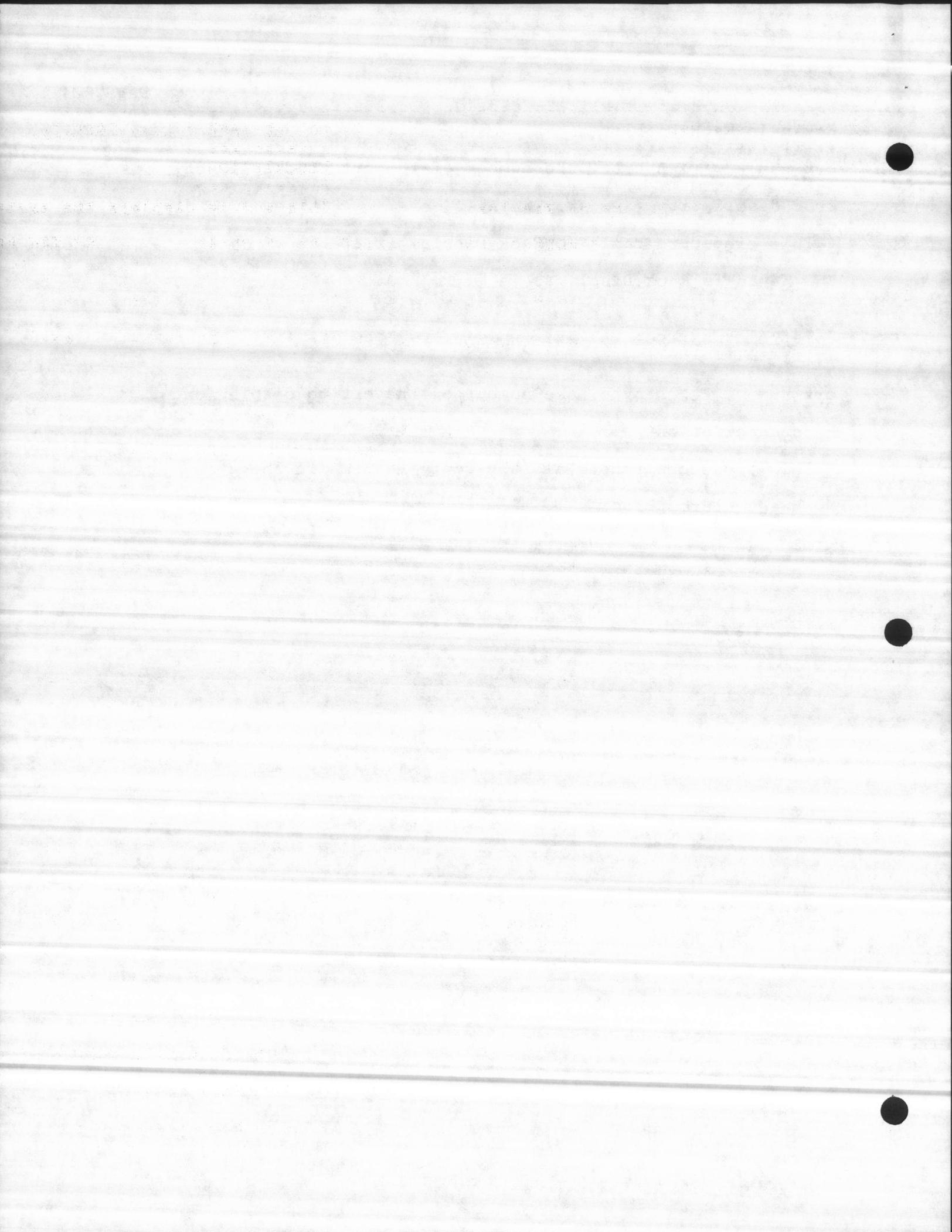
+++++  
+++++  
+++++

crt\_num => A byte, the number of the crt to display on.

Proc description :

-----  
This procedure resets all the active alarms in the system.

END Reset\_Alarms\_Proc;



(18.2) MODULE NAME : CxxxxDREM.Pyy

=====

DESCRIPTION :

=====

This module displays the system remotes

111/07/85 12:05:46

## SYSTEM REMOTES

Page 1 of x

|    | 1           | 2   | 3    | 4   | 5    | 6   | 7   | 8    | 9    |
|----|-------------|-----|------|-----|------|-----|-----|------|------|
| 2  | ---         | --- | ---  | --- | ---  | --- | --- | ---  | ---  |
| 3  | xxxC        | xxx | xxxC | xxx | xxxC | xxx | xxx | xxxC | xxxC |
| 4  |             |     |      |     |      |     |     |      |      |
| 5  |             |     |      |     |      |     |     |      |      |
| 6  | Rem Ø1 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 7  |             |     |      |     |      |     |     |      |      |
| 8  |             |     |      |     |      |     |     |      |      |
| 9  | Rem Ø2 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 10 |             |     |      |     |      |     |     |      |      |
| 11 |             |     |      |     |      |     |     |      |      |
| 12 | Rem Ø3 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 13 |             |     |      |     |      |     |     |      |      |
| 14 |             |     |      |     |      |     |     |      |      |
| 15 | Rem Ø4 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 16 |             |     |      |     |      |     |     |      |      |
| 17 |             |     |      |     |      |     |     |      |      |
| 18 | Rem Ø5 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 19 |             |     |      |     |      |     |     |      |      |
| 20 |             |     |      |     |      |     |     |      |      |
| 21 | Rem Ø6 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 22 |             |     |      |     |      |     |     |      |      |
| 23 |             |     |      |     |      |     |     |      |      |
| 24 | Rem Ø7 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 25 |             |     |      |     |      |     |     |      |      |
| 26 |             |     |      |     |      |     |     |      |      |
| 27 | Rem Ø8 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 28 |             |     |      |     |      |     |     |      |      |
| 29 |             |     |      |     |      |     |     |      |      |
| 30 | Rem Ø9 <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |
| 31 |             |     |      |     |      |     |     |      |      |
| 32 |             |     |      |     |      |     |     |      |      |
| 33 | Rem 1Ø <DF> | xxx | xxx  | xxx | xxx  | xxx | xxx | xxx  | xxx  |

EXTERNAL PROCEDURES :

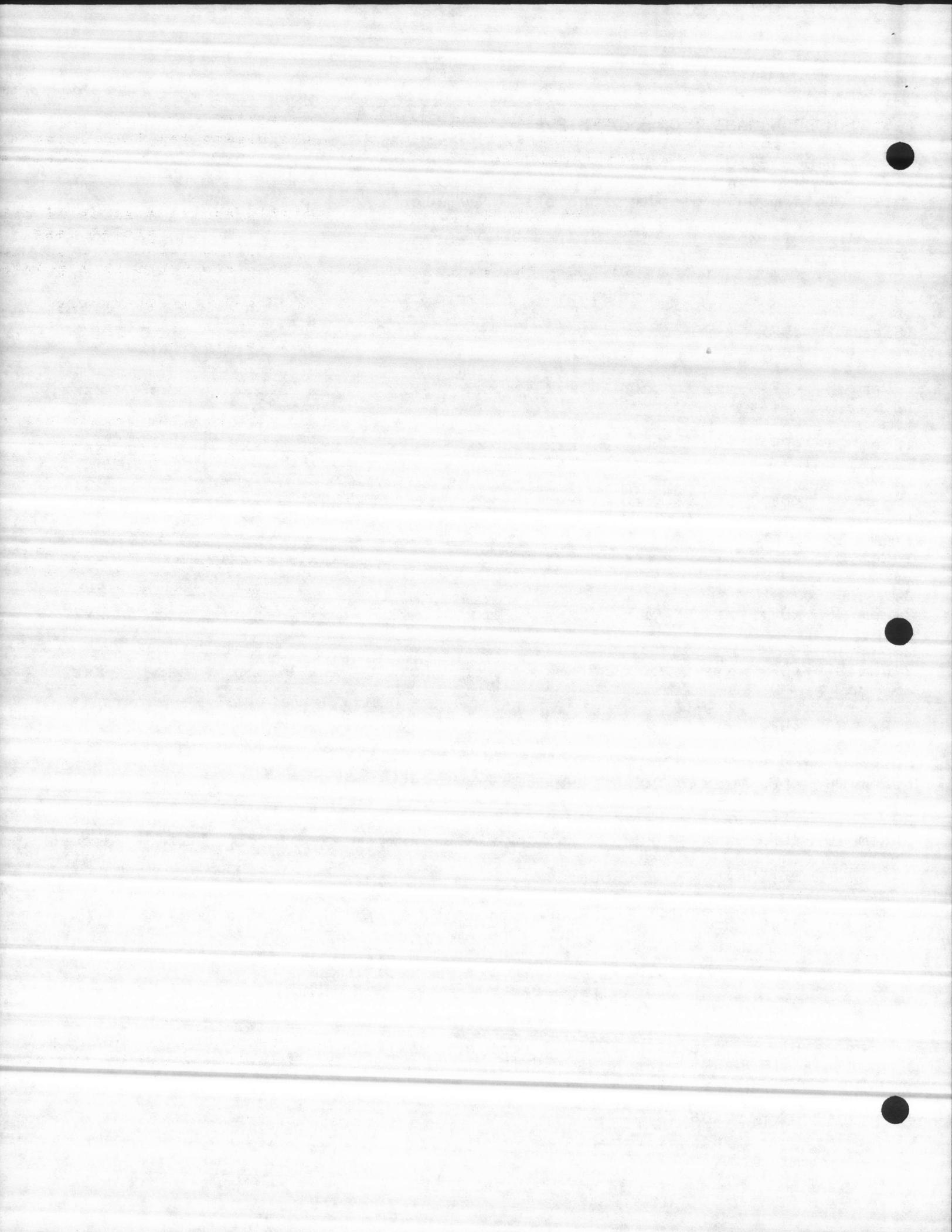
=====

```
Display_Table: PROCEDURE (table_p,
                           param_p,
                           table_reg_t, buf_t, mbx_t) EXTERNAL;
    DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
            (table_p, param_p)           POINTER;
End Display_Table;
```

```
Clear_Screen: PROCEDURE (crt_num) EXTERNAL;
    DECLARE (crt_num)   BYTE;
END Clear_Screen;
```

MODULE DECLARATION :

=====



exception => Three word each holds the exception condition.  
crt\_update\_sem\_t => Three tokens each used to create the crt update semaphore.

static\_asc => A group of bytes hold the sub header of each remote system page.  
crt\_update\_table => A pointer - the address of the Remote\_Update  
crt\_delete\_table => A pointer - the address of the Remote\_Delete.

MODULE PUBLIC PROCEDURES :

=====

Remote\_Display:

++++++

Proc description :

- 1. displays the remote system.
- 2. calls Stop\$Update proc.
- 3. define the number of pages needed.
- 4. creates the crt table segment and the crt parameter segment.
- 5. initializes the crt\_param structure.
- 6. displays the header of each page.
- 7. calls clear screen.
- 8. displays the static\_asc of the sub heading.
- 9. biuld the crt table & fill each entry of the crt table with the remote data.
- 10. Calls set\$update proc.

End Remote\_Display;

Remote\_Update:

++++++

crt\_num => A byte, the nubmer of the crt to display on.  
remaining\_units => A word used as an index.  
result => A word used as an index.

Proc description :

- 1. updates the crt table every 6 second.
- 2. displays the crt table.

END Remote\_Update;

Remote\_Delete:

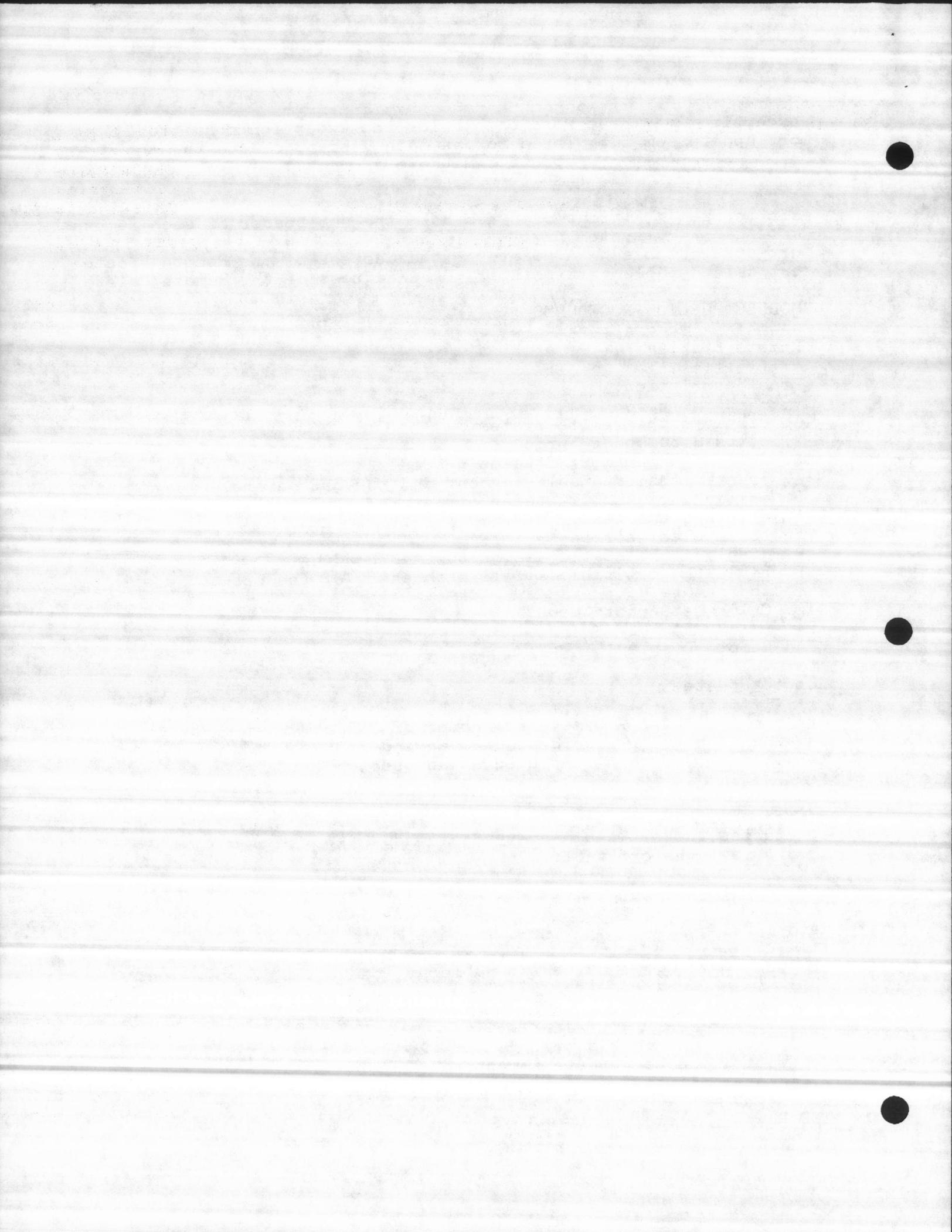
++++++

crt\_num => A byte, the nubmer of the crt to display on.

Proc description :

- 1. deletes the clock\$entry.
- 2. deletes the crt update semaphore.
- 4. deletes all the segments created

END Remote\_Delete;



(18.3) MODULE NAME : CxxxxDMST.Pyy

=====

DESCRIPTION :

=====

This module displays the system Master.

111/07/85 12:05:46

## SYSTEM MASTER

Page 1 of x

|    |        |      | Word 1 |       | Word 2 |       |
|----|--------|------|--------|-------|--------|-------|
|    |        |      | chng   | data  | chng   | data  |
| 2  |        | data | -----  | ----- | -----  | ----- |
| 3  |        | fail | ----   | ----  | ----   | ----  |
| 4  | Master | Ø1   | <DF>   | C     | dddd   | C     |
| 5  |        |      |        |       | ddd    | ddd   |
| 6  |        |      |        |       |        |       |
| 7  | Master | Ø2   | <DF>   | C     | ddd    | C     |
| 8  |        |      |        |       | dd     | dd    |
| 9  |        |      |        |       |        |       |
| 10 | Master | Ø3   | <DF>   | C     | ddd    | C     |
| 11 |        |      |        |       | dd     | dd    |
| 12 |        |      |        |       |        |       |
| 13 | Master | Ø4   | <DF>   | C     | ddd    | C     |
| 14 |        |      |        |       | dd     | dd    |
| 15 |        |      |        |       |        |       |
| 16 | Master | Ø5   | <DF>   | C     | ddd    | C     |
| 17 |        |      |        |       | dd     | dd    |
| 18 |        |      |        |       |        |       |
| 19 | Master | Ø6   | <DF>   | C     | ddd    | C     |
| 20 |        |      |        |       | dd     | dd    |
| 21 |        |      |        |       |        |       |
| 22 | Master | Ø7   | <DF>   | C     | ddd    | C     |
| 23 |        |      |        |       | dd     | dd    |
| 24 |        |      |        |       |        |       |
| 25 | Master | Ø8   | <DF>   | C     | ddd    | C     |
| 26 |        |      |        |       | dd     | dd    |
| 27 |        |      |        |       |        |       |
| 28 | Master | Ø9   | <DF>   | C     | ddd    | C     |
| 29 |        |      |        |       | dd     | dd    |
| 30 |        |      |        |       |        |       |
| 31 | Master | 10   | <DF>   | C     | ddd    | C     |
| 32 |        |      |        |       | dd     | dd    |
| 33 |        |      |        |       |        |       |
| 34 |        |      |        |       |        |       |
| 35 |        |      |        |       |        |       |

## EXTERNAL PROCEDURES :

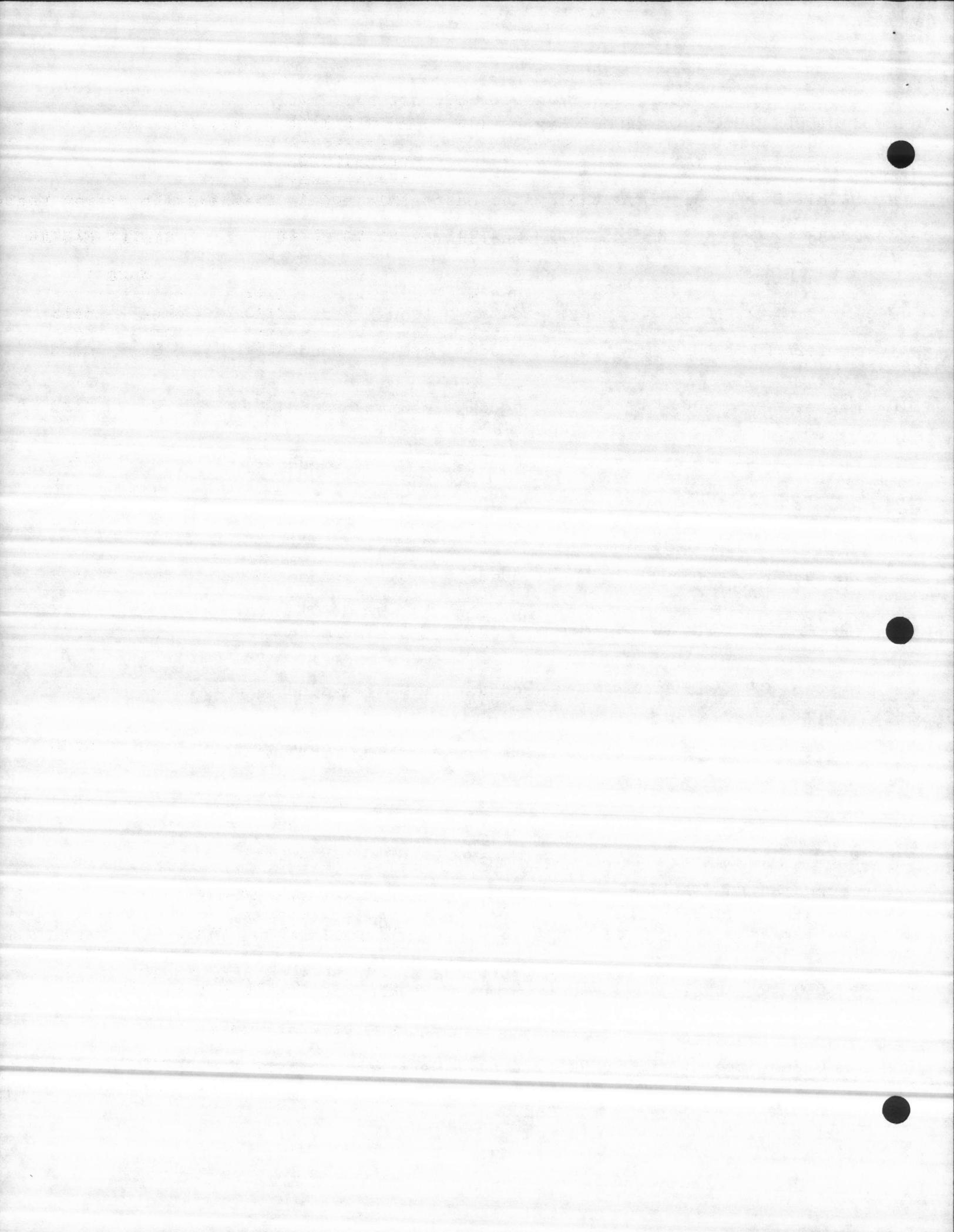
=====

```
Display_Table: PROCEDURE (table_p,
                           param_p,
                           table_reg_t, buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p)           POINTER;
End Display_Table;
```

```
Clear_Screen: PROCEDURE (crt_num) EXTERNAL;
```

```
DECLARE (crt_num) BYTE;
```

```
END Clear_Screen;
```



**MODULE DECLARATION :**

```
=====
exception  => Three word each holds the exception condition.
crt_update_sem_t => Three tokens each used to create the crt
                     update semaphore.
static_asc => A group of bytes hold the sub header of each
               system master page.
crt_update_table => A pointer - the address of the Master_Update
crt_delete_table => A pointer - the address of the Master_Delete.
```

**MODULE PUBLIC PROCEDURES :**

```
=====
```

**Master\_Display:**

```
++++++
```

**Proc description :**

- ```
-----
1. displays the system master.
2. calls Stop$Update proc.
3. define the number of pages needed.
4. creates the crt table segment and the crt parameter segment.
5. initializes the crt_param structure.
6. displays the header of each page.
7. calls clear screen.
8. displays the static_asc of the sub heading.
9. biuld the crt table & fill each entry of the crt table with
   the master data.
10. Calls set$update proc.
```

```
End Master_Display;
```

**Master\_Update:**

```
++++++
```

```
crt_num      => A byte, the nubmer of the crt to display on.
remaining_units => A word used as an index.
result       => A word used as an index.
```

**Proc description :**

- ```
-----
1. updates the crt table every 6 second.
2. displays the crt table.
```

```
END Master_Update;
```

**Master\_Delete:**

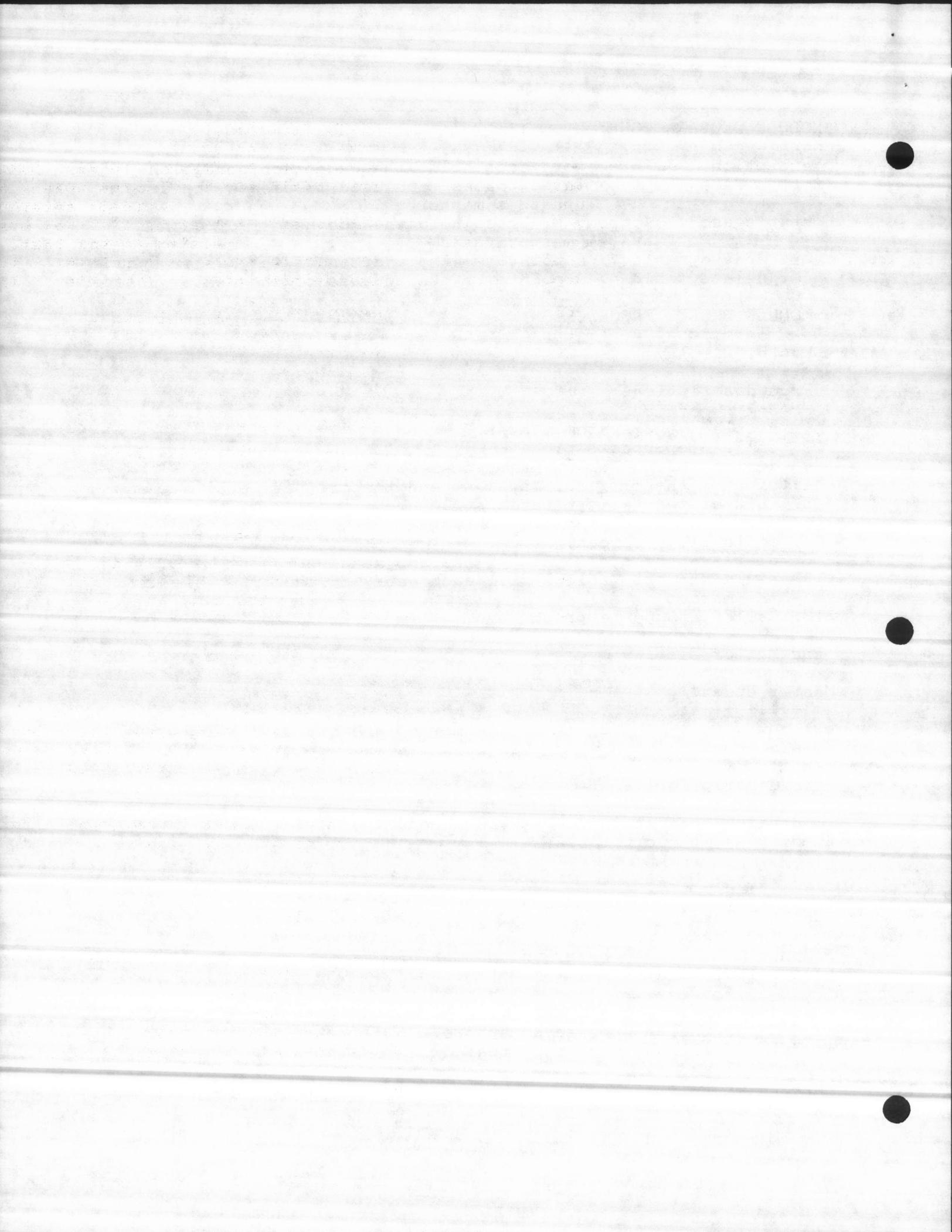
```
++++++
```

```
crt_num      => A byte, the nubmer of the crt to display on.
```

**Proc description :**

- ```
-----
1. deletes the clock$entry.
2. deletes the crt update semaphore.
4. deletes all the segments created
```

```
END Master_Delete;
```



(18.4) MODULE NAME : CxxxxDIOR.Pyy

=====

DESCRIPTION :

=====

This module displays the local IO Rack.

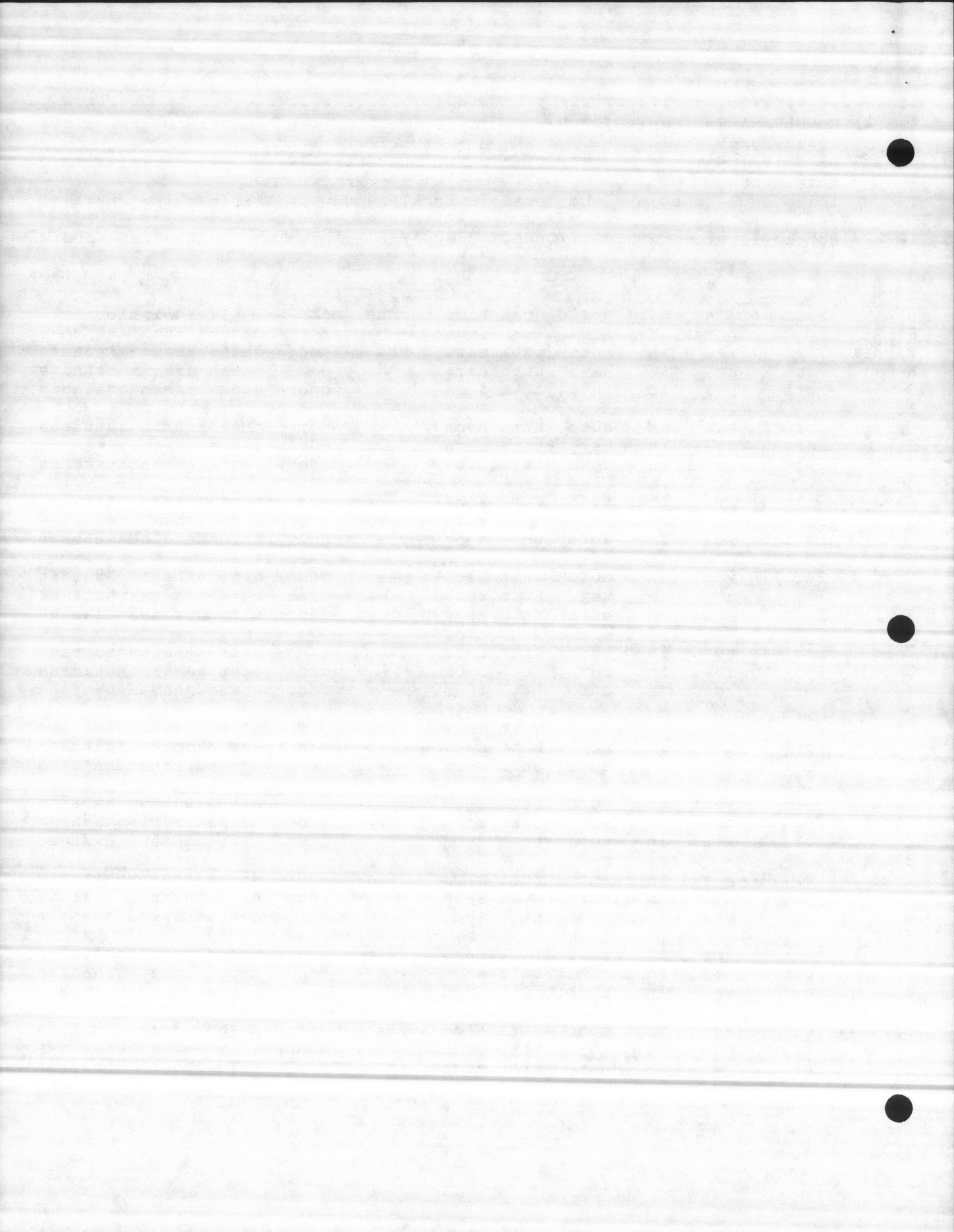
10/22/1985 13:14:57

## I/O RACK DISPLAY

Page 1 of 2

2  
3 Card Rack x  
4

		online	dir	ctrl	scan time	chng	data	description
		-----	---	----	-----	----	----	-----
5	Card 1	Port 1	yes	in	auto	tttt secs	yes	ddd aaaaaaaaaaaaaaaaaaaaaaa
8		Port 2	no	in	key	tttt secs		ddd aaaaaaaaaaaaaaaaaaaaaaa
9		Port 3	yes	in	auto	tttt secs		ddd aaaaaaaaaaaaaaaaaaaaaaa
10								
2	Card 2	Port 1	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
9		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
20		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
3								
5	Card 3	Port 1	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
6		Port 2	no	in	key	tttt secs	yes	ddd aaaaaaaaaaaaaaaaaaaaaaa
7		Port 3	yes	in	auto	tttt secs	yes	ddd aaaaaaaaaaaaaaaaaaaaaaa
9								
20	Card 4	Port 1	yes	in	auto	tttt secs	yes	ddd aaaaaaaaaaaaaaaaaaaaaaa
2		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
20		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
5								
3	Card 5	Port 1	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
4		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
5		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
7								
8	Card 6	Port 1	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
9		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
9		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
10								
2	Card 7	Port 1	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
3		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
3		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
3								
5	Card 8	Port 1	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
6		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
7		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
7								
9	Card 9	Port 1	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
9		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
40		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
1								
3	Card 10	Port 1	yes	in	auto	tttt secs	yes	ddd aaaaaaaaaaaaaaaaaaaaaaa
4		Port 2	no	in	key	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa
4		Port 3	yes	in	auto	tttt secs	ddd	aaaaaaaaaaaaaaaaaaaaaa

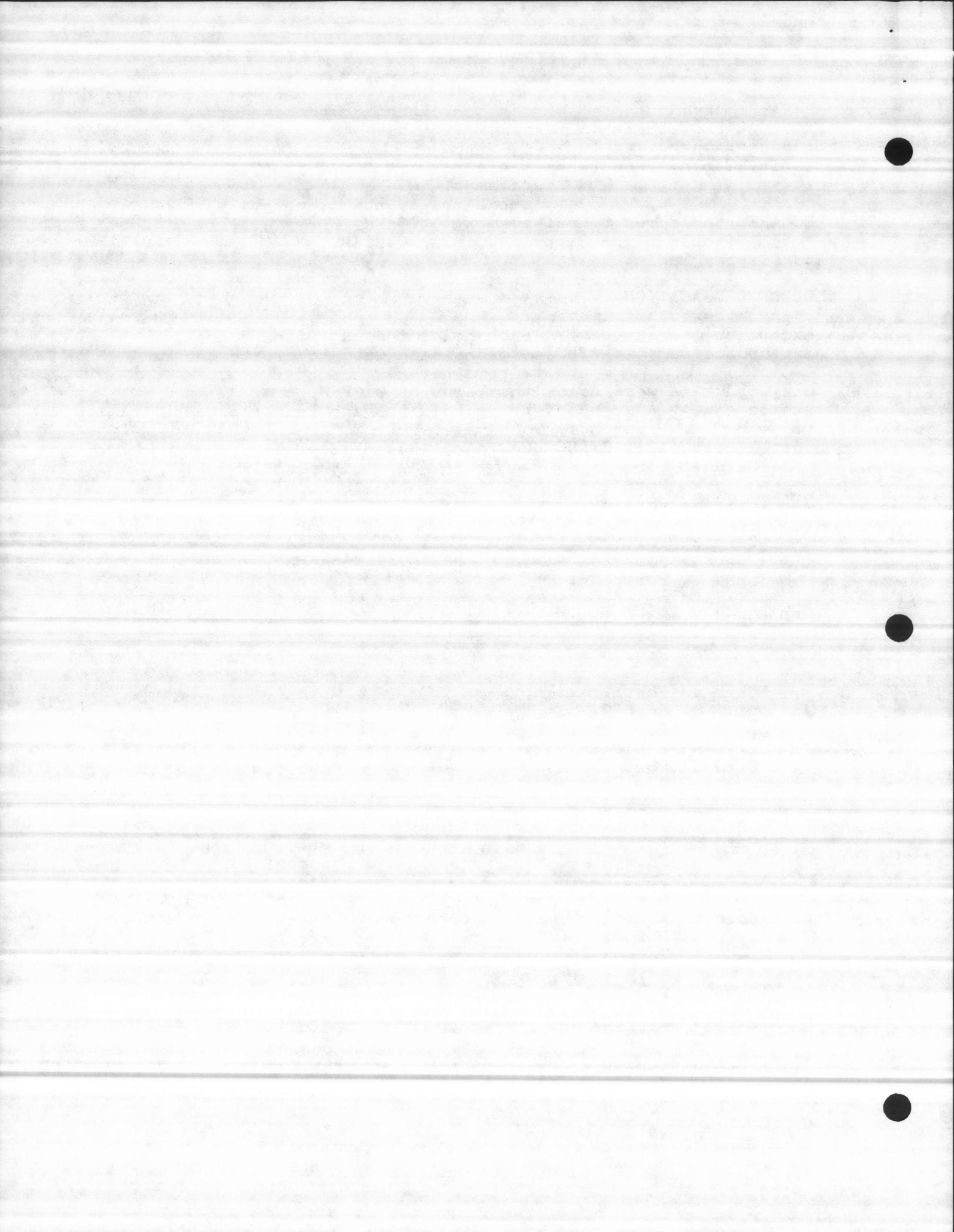


## EXTERNAL PROCEDURES :

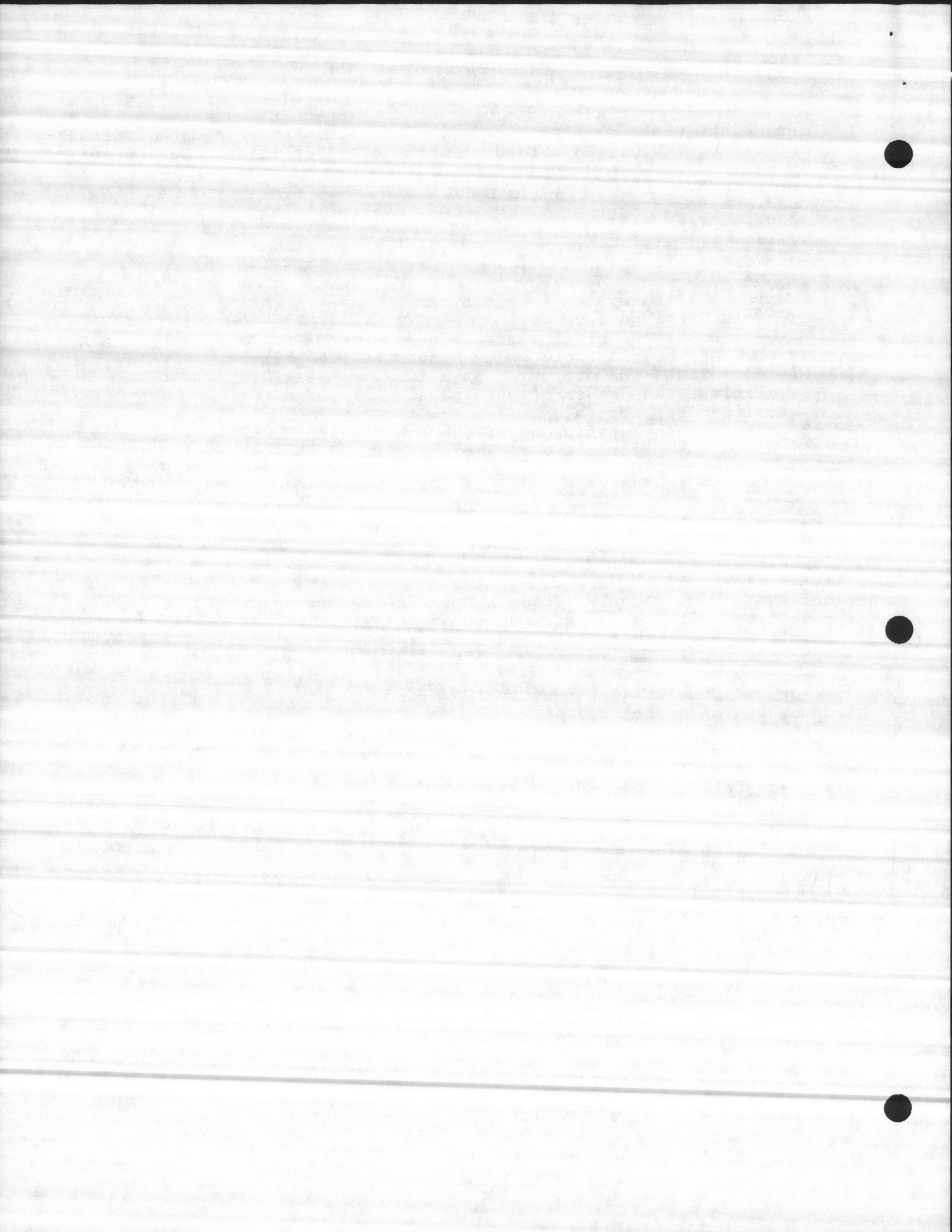
```
=====
Display_Table: PROCEDURE (table_p,
                          param_p,
                          table_reg_t, buf_t, mbx_t) EXTERNAL;
DECLARE (table_reg_t, buf_t, mbx_t) TOKEN,
        (table_p, param_p)           POINTER;
End Display_Table;
```

## MODULE DECLARATION :

```
=====
exception  => Three word each holds the exception condition.
crt_update_sem_t => Three tokens each used to create the crt
                      update semaphore.
max_word => A word holds the value of $FFFFH.
static_asc => A group of bytes hold the sub header of each
                 system io rack page.
port_labels (10) STRUCTURE (asc (133) BYTE) DATA
  'Card 1  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 2  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 3  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 4  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 5  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 6  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 7  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 8  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 9  Port 1                  secs',
  ,          Port 2                  secs',
  ,          Port 3                  secs',
  'Card 10 Port 1                  secs',
  ,          Port 2                  secs',
```



```
crt_update_table => A pointer - the address of the Io_Rack_Update  
crt_delete_table => A pointer - the address of the Io_Rack_Delete.  
  
MODULE PUBLIC PROCEDURES :  
=====  
Io_Rack_Display:  
+++++  
Proc description :  
-----  
1. displays the system local io rack.  
2. calls Stop$Update proc.  
3. define the number of pages needed.  
4. creates the crt table segment and the crt parameter segment.  
5. initializes the crt_param structure.  
6. displays the header of each page.  
7. calls clear screen.  
8. displays the static_asc of the sub heading.  
9. biuld the crt table & fill each entry of the crt table with  
the io rack data.  
10. Calls set$update proc.  
  
End Io_Rack_Display;  
  
Io_Rack_Update:  
+++++  
crt_num      => A byte, the nubmer of the crt to display on.  
remaining_units => A word used as an index.  
result        => A word used as an index.  
Proc description :  
-----  
1. updates the crt table every 10 second.  
2. displays the crt table.  
  
END Io_Rack_Update;  
  
Io_Rack_Delete:  
+++++  
crt_num      => A byte, the nubmer of the crt to display on.  
Proc description :  
-----  
1. deletes the clock$entry.  
2. deletes the crt update semaphore.  
4. deletes all the segments created  
  
END Io_Rack_Delete;
```



(18.5) MODULE NAME : CxxxxIORK.Pyy

=====

DESCRIPTION :

=====

This module monitors the local input & the local output.

MODULE TASKS :

=====

Input\_Rack\_Task:

+++++-----

count => A word used as count for the input ports.

temp\_word => A word used as an index.

exception => A word used for exception condition.

scan\_sem\_t => A scan semaphore token used to create the task semaphore.

Task Algorithm :

-----

Set the exception handler.

create input scan semaphore

setup clock entry for defined scan time with on second.

DO FOREVER;

    wait at the clock entry semaphore

    get control of io rack region

    Loop 1 go through all input ports

        check io port mode for port existance.

        increment scan time count by task time interval

        check io port mode for input port

        if scan time met scan time count.

            read io port

            check the port value.

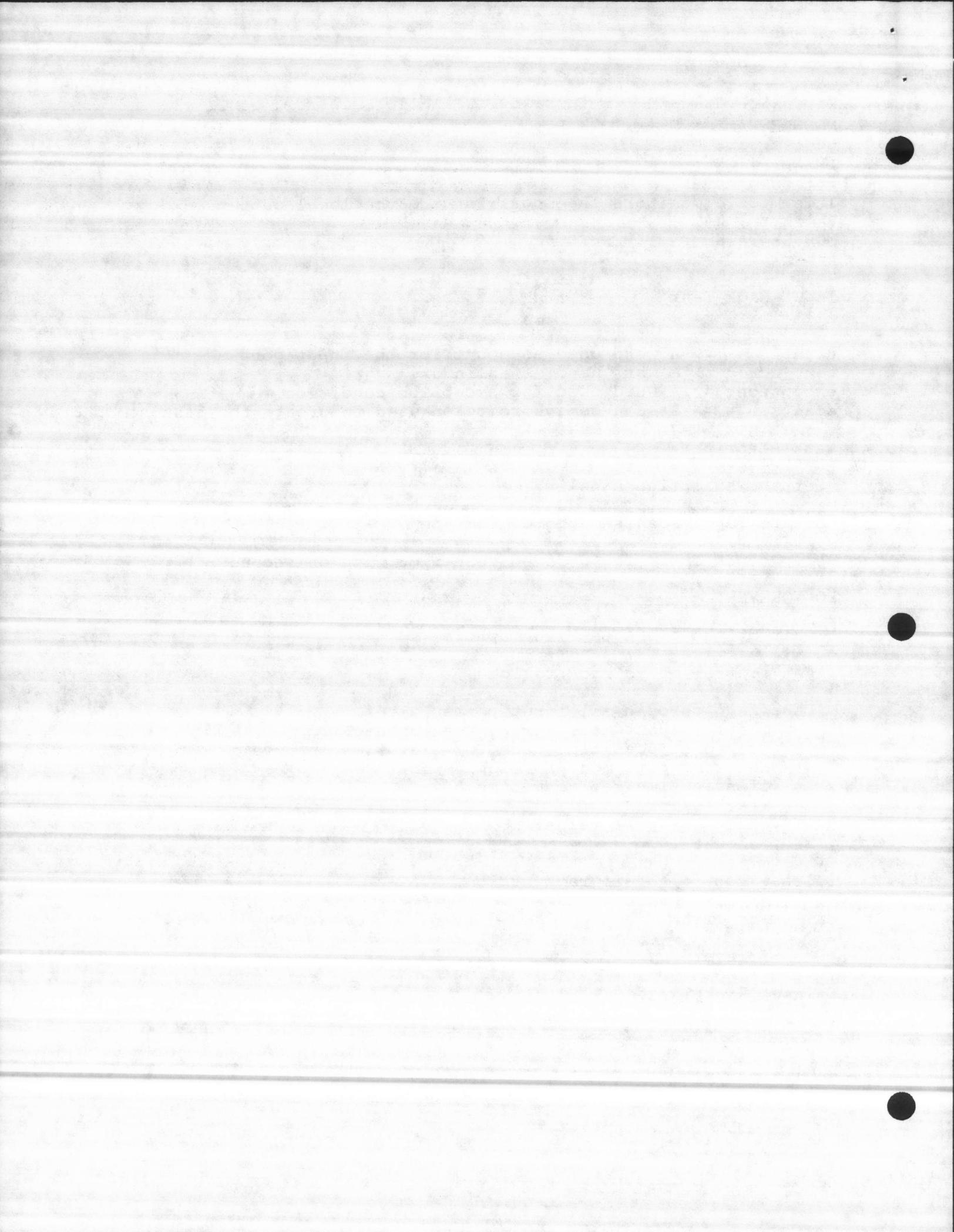
    END;    count loop

    release control of io rack region

    call send list

END;    forever

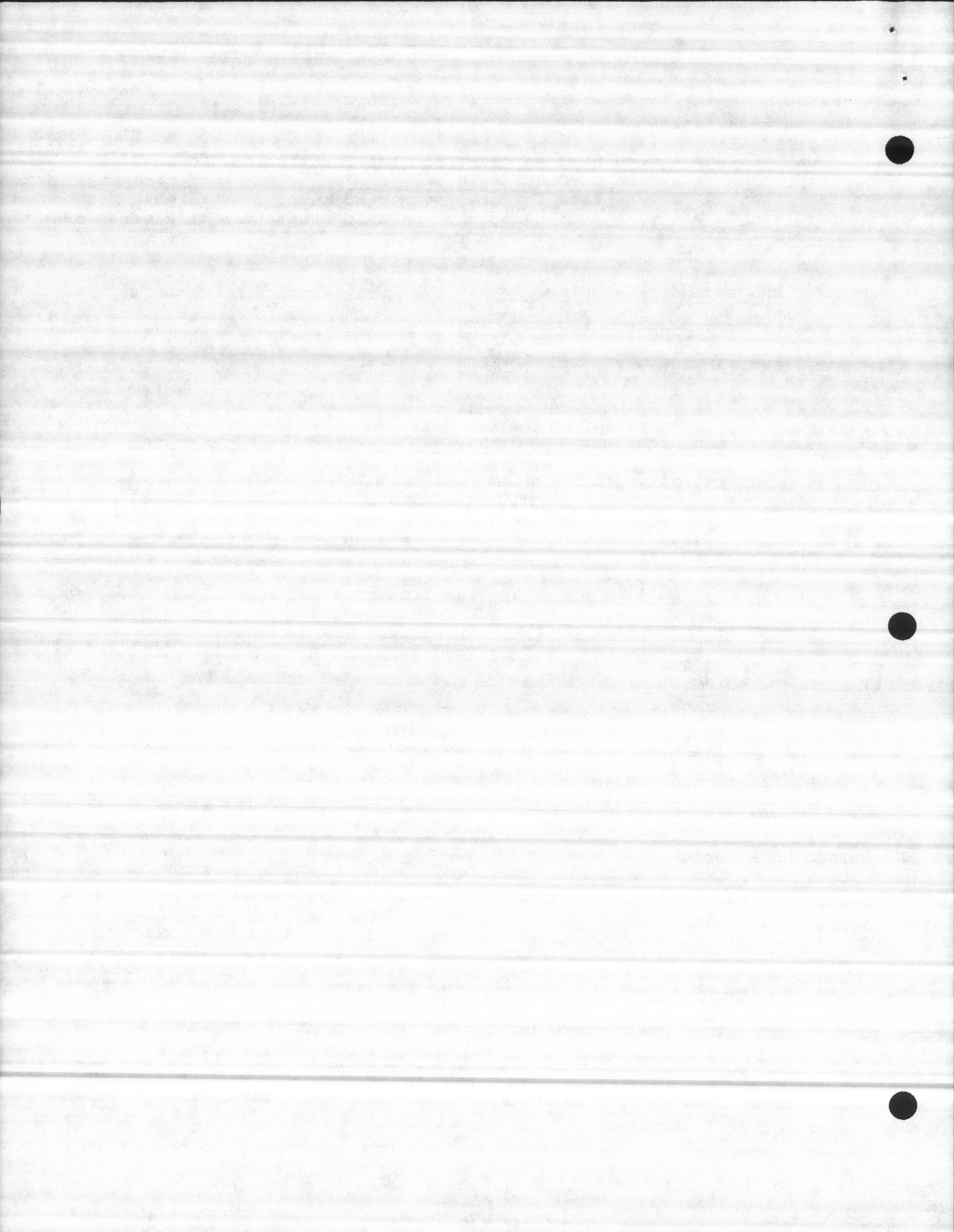
END Input\_Rack\_Task;



```
Output_Rack_Task:  
+++++  
count      => A word used as count for the input ports.  
temp_word  => A word used as an index.  
exception   => A word used for exception condition.  
scan_sem_t => A scan semaphore token used to create the task semaphore.
```

Task Algorithm :

```
-----  
Set the exception handler.  
  
create ouput scan semaphore  
  
Create list entry for the defined scan time.  
  
DO FOREVER;  
  
    wait at the clock entry semaphore  
  
    get control of io rack region  
  
    Loop1 go through all output ports  
  
    check the io_ports mode for port existance.  
  
    increment scan time count by task time interval  
  
    check the io_ports mode for output port.  
  
    test if scan time up  
  
    write io rack.  
  
    END;  count loop  
  
    release control of io rack region  
  
END;  forever  
  
END Output_Rack_Task;
```



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 19

P R I N T

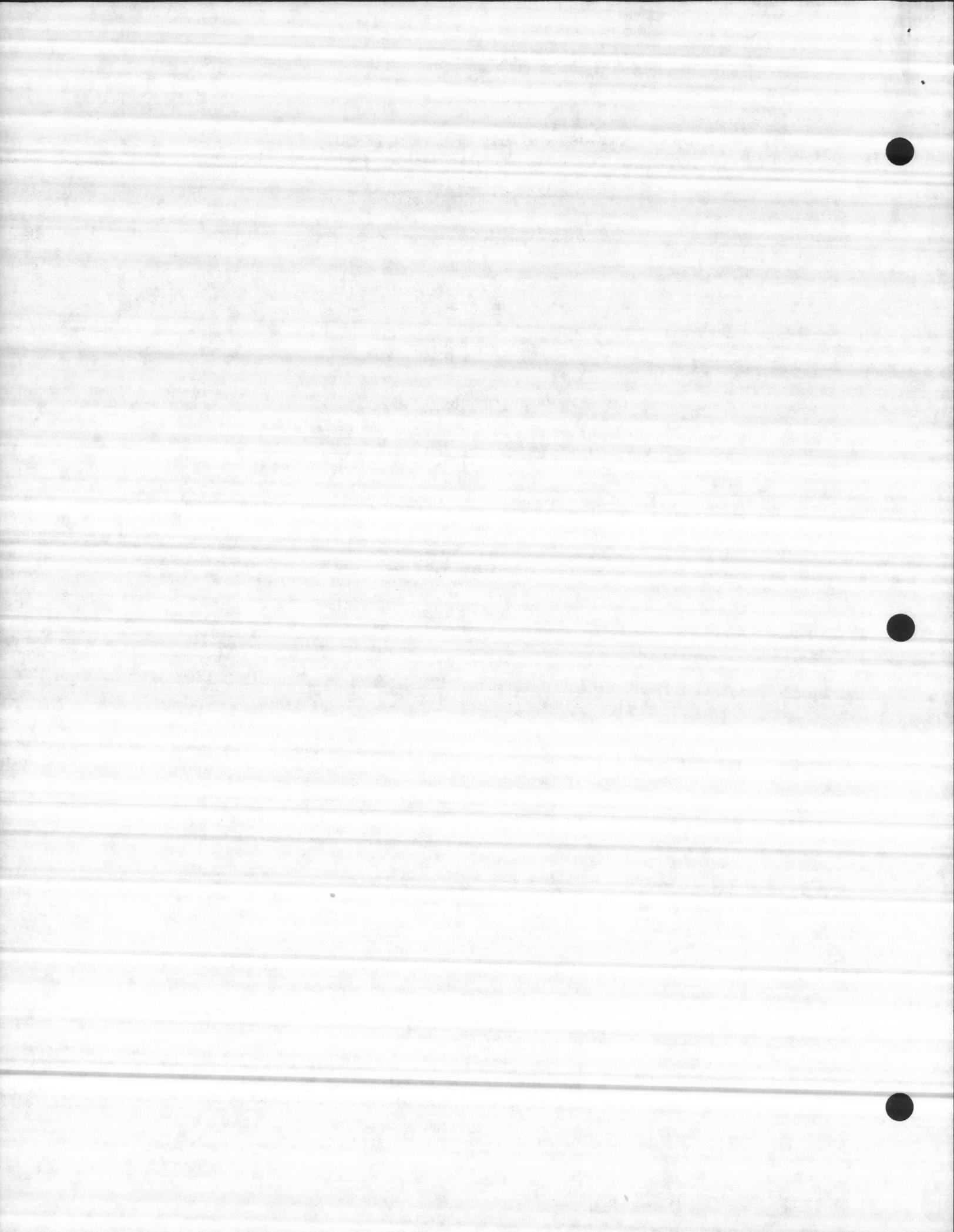
BY

Mohamed Fayad

REVIEWED

BY

Jerry Knoth



DATE : Oct 10, 1985

PRNT - written by Bob Ryan, Peter Wollenzien, Greg Jensen, M Fayad

(19.1) MODULE NAME : CxxxxPRNT.Pyy

=====

DESCRIPTION :

=====

1. Displays the menu for print command
2. Calls print current alarm log
3. Calls print current data base.

EXTERNAL PROCEDURES:

=====

```
Print_Alarmlog: PROCEDURE (crt_num) EXTERNAL;
  DECLARE crt_num BYTE;
END Print_Alarmlog;
```

```
Keyin_Print_Data_Base_Proc: PROCEDURE (crt_num) EXTERNAL;
  DECLARE crt_num BYTE;
END Keyin_Print_Data_Base_Proc;
```

```
Menu_Display: PROCEDURE (crt_num, result, off_set, start_id_p,
                         bytes_per_element, num_elements) WORD EXTERNAL;
  DECLARE (crt_num)                      BYTE,
          (result, off_set)           WORD,
          (bytes_per_element, num_elements) WORD,
          (start_id_p)                POINTER;
END Menu_Display;
```

MODULE DECLARATION :

=====

exception => Three words, each holds the exception condition number.

main\_menu\_command\_count => A byte contains a value of two.

main\_menu\_prompt\_asc (2) STRUCTURE :

menu\_asc => Ten bytes contains the command string.  
Alarmlog = Prints report of current alarm log  
Data Base = Enters sub-menu for printing data base setup  
main\_menu\_help\_desc\_tbl => Two pointers point to the help descriptions table.

MODULE PROCEDURES :

=====

Keyin\_Print\_Sub\_Menu\_Proc:

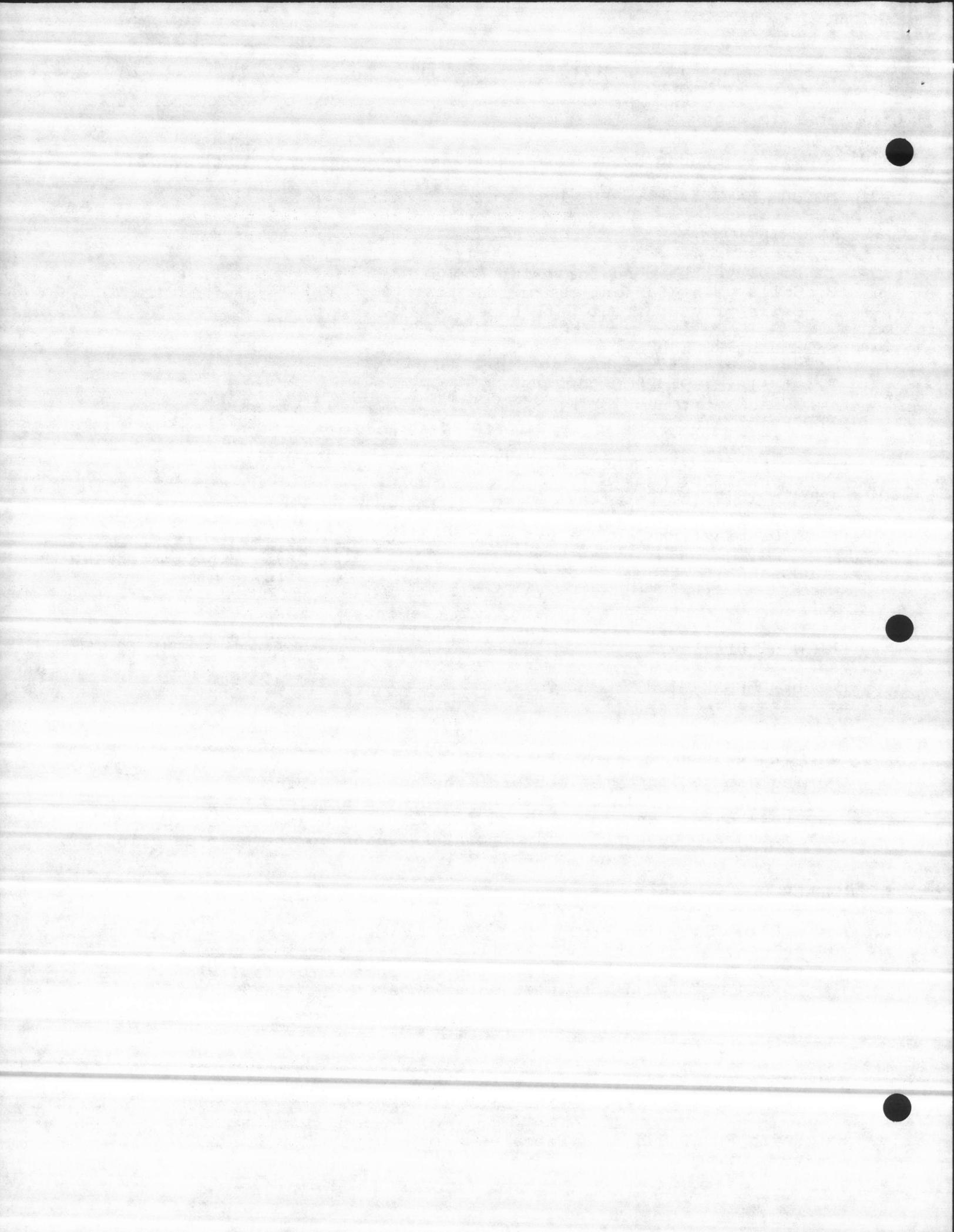
+++++

crt\_num => A byte holds the crt number to display on.  
result\_word => A word used as an index.  
temp\_word => A word used as temp. location.

Proc description :

This procedures displays the print menu and calls  
Print\_Alarmlog, Keyin\_Print\_Data\_Base\_Proc.

END Keyin\_Print\_Sub\_Menu\_Proc;



(19.2) MODULE NAME : CxxxxPBAS.Pyy

=====

DESCRIPTION :

=====

1. displays print\_data\_base menu for current data.
2. contains mechanisms to print the entire data base or a point from it.
3. contains mechanisms to fill the data base with current data or historical data.
4. displays print\_data\_base menu for historical data.
5. prints the historical data base (all or single point).

EXTERNAL PROCEDURES :

=====

```
Display_Case: PROCEDURE (table_element_p, param_p, buf_t) EXTERNAL;
    DECLARE buf_t TOKEN,
            (table_element_p, param_p) POINTER;
END Display_Case;
```

```
Table_Fill: PROCEDURE (field_number, element,
                        struc_type, table_p, param_p) WORD EXTERNAL;
    DECLARE (table_p, param_p) POINTER,
            (field_number, element) WORD,
            (struc_type) BYTE;
END Table_Fill;
```

```
Hist_Table_Fill: PROCEDURE (field_number, element,
                            struc_type, table_p, param_p) WORD EXTERNAL;
    DECLARE (table_p, param_p) POINTER,
            (field_number, element) WORD,
            (struc_type) BYTE;
END Hist_Table_Fill;
```

```
Menu_Display: PROCEDURE (crt_num, result, off_set, start_id_p,
                         bytes_per_element, num_elements) WORD EXTERNAL;
    DECLARE (crt_num) BYTE,
            (result, off_set) WORD,
            (bytes_per_element, num_elements) WORD,
            (start_id_p) POINTER;
END Menu_Display;
```

struc\_type\_string (7) STRUCTURE :

-----

count => A byte contains a value of 15.

asc => Fifteen bytes contains data base string.

main\_menu\_command\_count => A byte contains a value of seven.

main\_menu\_prompt\_asc (7) STRUCTURE:

-----

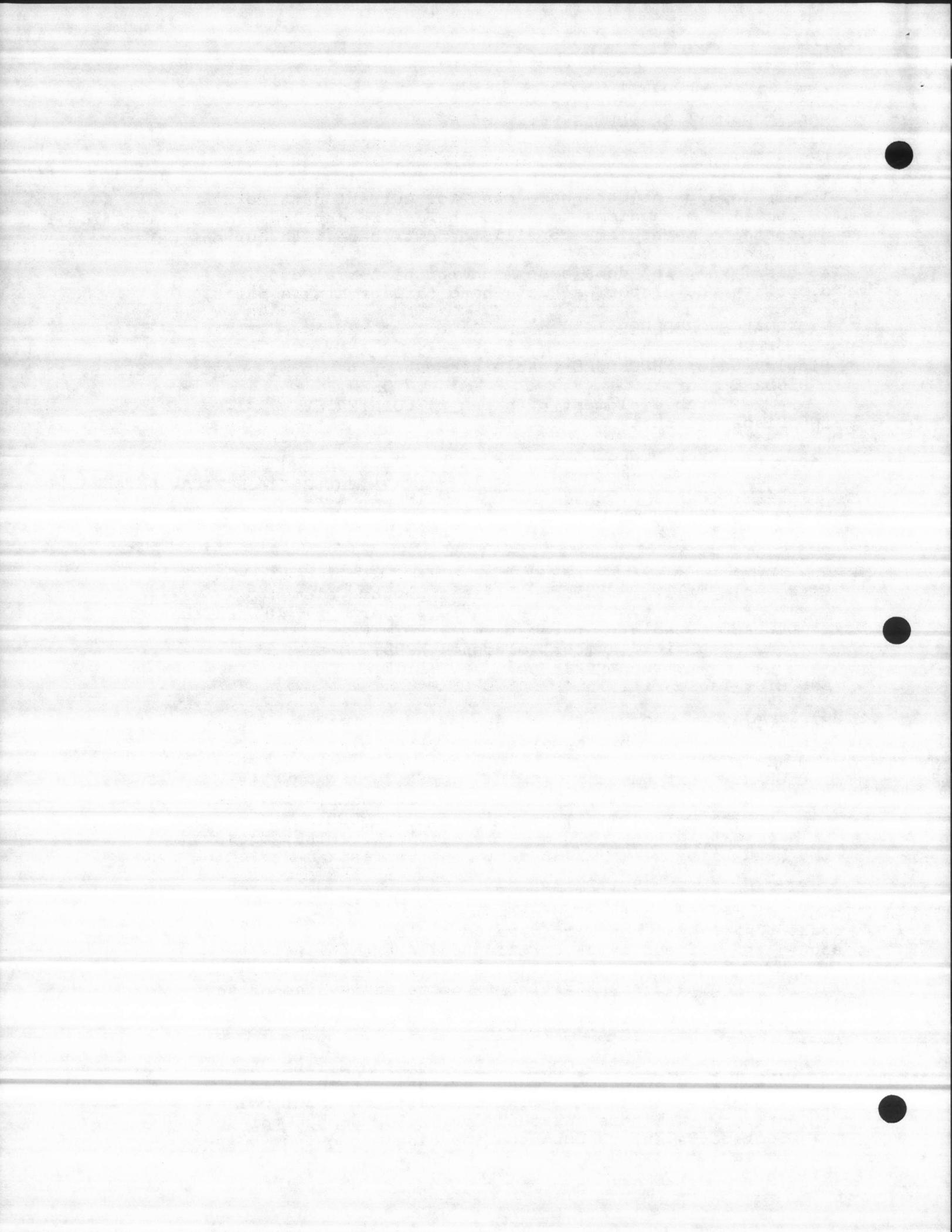
menu\_asc => Ten bytes contain the command string.

Analog-I = Enters sub-menu for analog input points

Discrete-I= Enters sub-menu for discrete input points

Analog-O = Enters sub-menu for analog output points

Discrete-O= Enters sub-menu for discrete output points



Year Trend= Enters sub-menu for yearly analog trend points  
Level Ctrl= Enters sub-menu for level control points  
Pump Cntrl= Enters sub-menu for pump control points

main\_menu\_help\_desc\_tbl => Seven pointers point to the help description table.

sub\_menu\_command\_count => A byte contains a value of two.

sub\_menu\_prompt\_asc (2) STRUCTURE:

menu\_asc => Ten bytes contains the sub menu string.

    All       = Generates complete report

    Select     = Enters sub-menu specific report selection

sub\_menu\_help\_desc\_tbl (2) POINTER DATA

exception       => Three words used for exception condition.

crt\_update\_sem\_t => Three tokens (one for each crt), used to create the crt update semaphore.

PUBLIC PROCEDURES :

=====

Keyin\_Print\_Data\_Base\_Proc:

+++++

result\_word => A word used as an index.

crt\_num      => A byte, the number of the crt to display on.

print\_param\_seg\_t => A token used to create a segment for print\_parameters.

print\_parameters BASED print\_param\_seg\_t STRUCTURE :

max\_field   => A byte holds the max\_field value.

struc\_type  => A byte holds the structure type value.

    analog\_in struc\_type = 0;

    discrete\_in struc\_type = 1; and so on.

start\_index => A word used as the start index.

end\_index   => A word used as the end index.

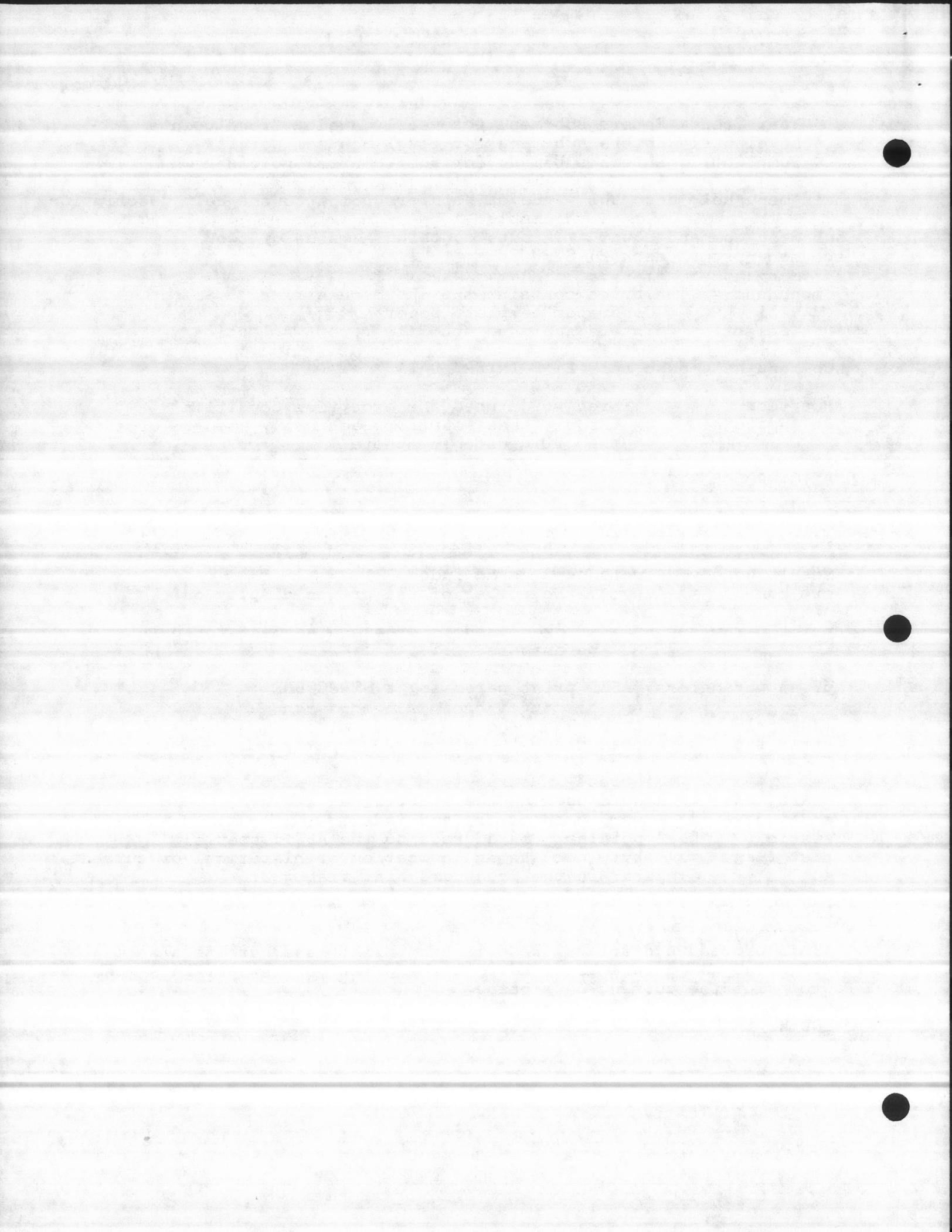
static\_p     => A pointer - an address to the ASCII static

hist\_flag   => A byte used as an indication of historical or current data. In the procedure hist\_flag = 0.

Proc description :

This procedure displays the data base print menu, create a segment of print\_parameters size and fill the print parameters. This menu used to print the current data base.

END Keyin\_Print\_Data\_Pase\_Proc;



```
Hist_Print_Data_Base_Proc:  
+++++  
result_word => A word used as an index.  
crt_num      => A byte, the number of the crt to display on.  
  
print_param_seg_t => A token used to create a segment for print_ parameters.  
  
print_parameters BASED print_param_seg_t STRUCTURE :  
-----  
max_field    => A byte holds the max_field value.  
struc_type   => A byte holds the structure type value.  
            analog_in struc_type = 0;  
            discrete_in struc_type = 1; and so on.  
start_index  => A word used as the start index.  
end_index    => A word used as the end index.  
static_p      => A pointer - an address to the ASCII static  
hist_flag     => A byte used as an indication of historical or current  
                  data. In the procedure the hist_flag = OFFH.
```

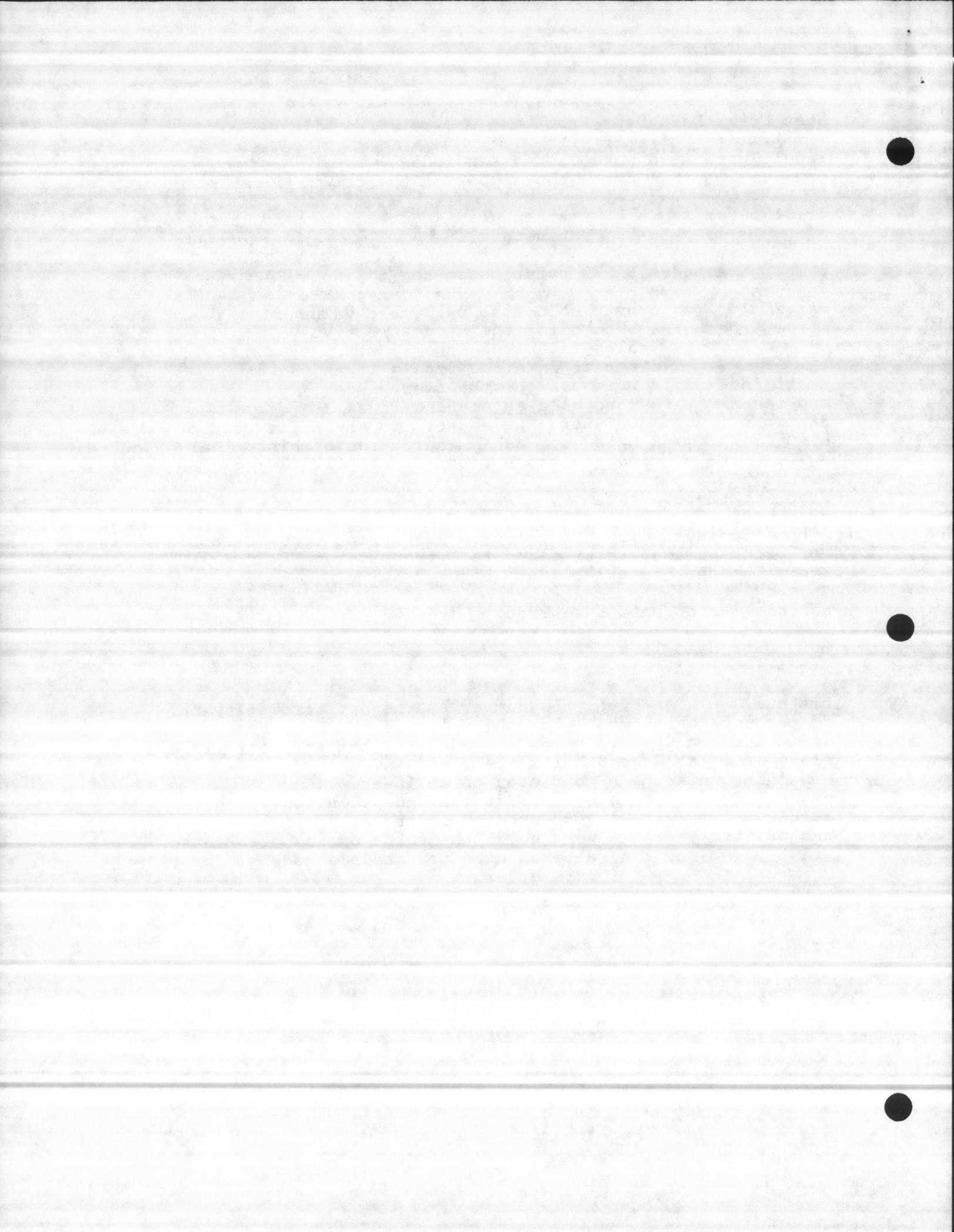
Proc description :

This procedure displays the data base print menu, create a segment of print\_parameters size and fill the print parameters. This menu used to print the historical data base.

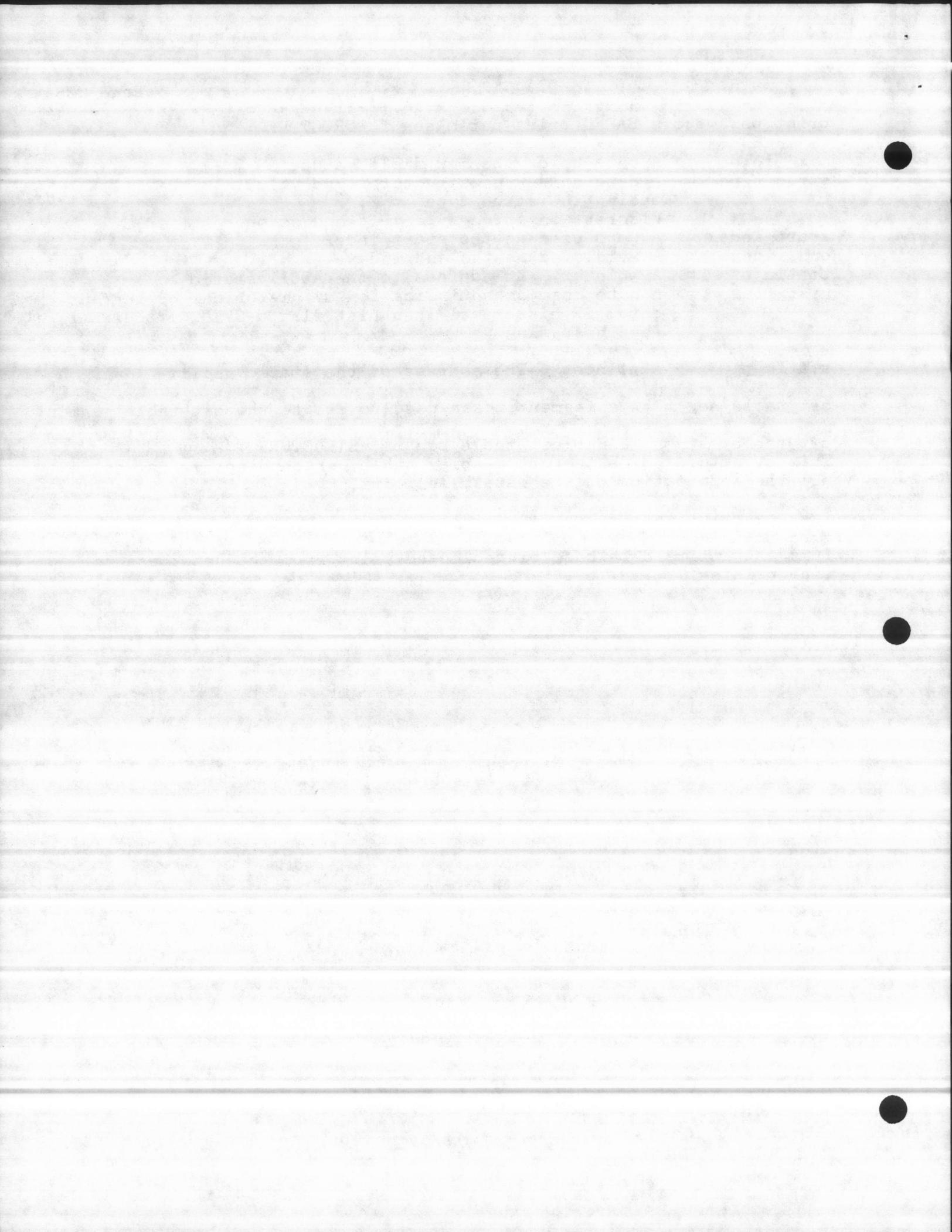
END Hist\_Print\_Data\_Base\_Proc;

Dump\_Data\_Base\_Task:

```
+++++  
crt_num      => A byte, the number of the crt to display on.  
ff_per_page  => A byte holds the value of the Form Feed  
struc_index  => A word holds an index for the structure in use.  
field        => A word holds the field index.  
remaining_units => A word used as an index.  
exception     => A word holds the exception condition.  
result        => A word used as an index.  
num_printed   => A word holds the value of number printed.  
print_param_seg_t => A token used to create a segment for print_ parameters.
```



```
print_parameters BASED print_param_seg_t STRUCTURE :  
-----  
max_field    => A byte holds the max_field value.  
struc_type   => A byte holds the structure type value.  
                analog_in struc_type = Ø;  
                discrete_in struc_type = 1; and so on.  
start_index  => A word used as the start index.  
end_index    => A word used as the end index.  
static_p      => A pointer - an address to the ASCII static  
hist_flag     => A byte used as an indication of historical or current  
                  data. IF historical Then hist_flag = ØFFH, If current  
                  Then hist_flag = Ø.  
  
static_p      => A pointer - The address of the static ascii.  
  
crt_table_seg_t => A segment token used to create the crt table segment.  
  
crt_table      BASED crt_table_seg_t      STRUCTURE :  
This structure holds the crt display table, which displays and holds  
all the information of the data base point.  
  
Task description :  
-----  
Dump the entire data base or selected point from the data base to  
line printer.
```



**Task Algorithm:**

-----  
Set the exception handler.

DO FOREVER;

wait for mailbox.

Initialize the crt table and create the crt table segment.

get control of the line printer region.

determine number of formfeeds per page

DO CASE (ff\_per\_page);

; already set up to 66 lines per ff

num\_printed = Ø;

static\_p = print\_parameters.static\_p;

struc\_index = print\_parameters.start\_index;

DO WHILE (struc\_index <= print\_parameters.end\_index);

IF hist\_flag THEN

    fill the crt table with historical data.

ELSE

    fill the crt table with current data.

will not print "spares" unless request or Øth element

create a segment for the print\_parameters.

Print the report

determine how many ff's are needed to set correct top of form

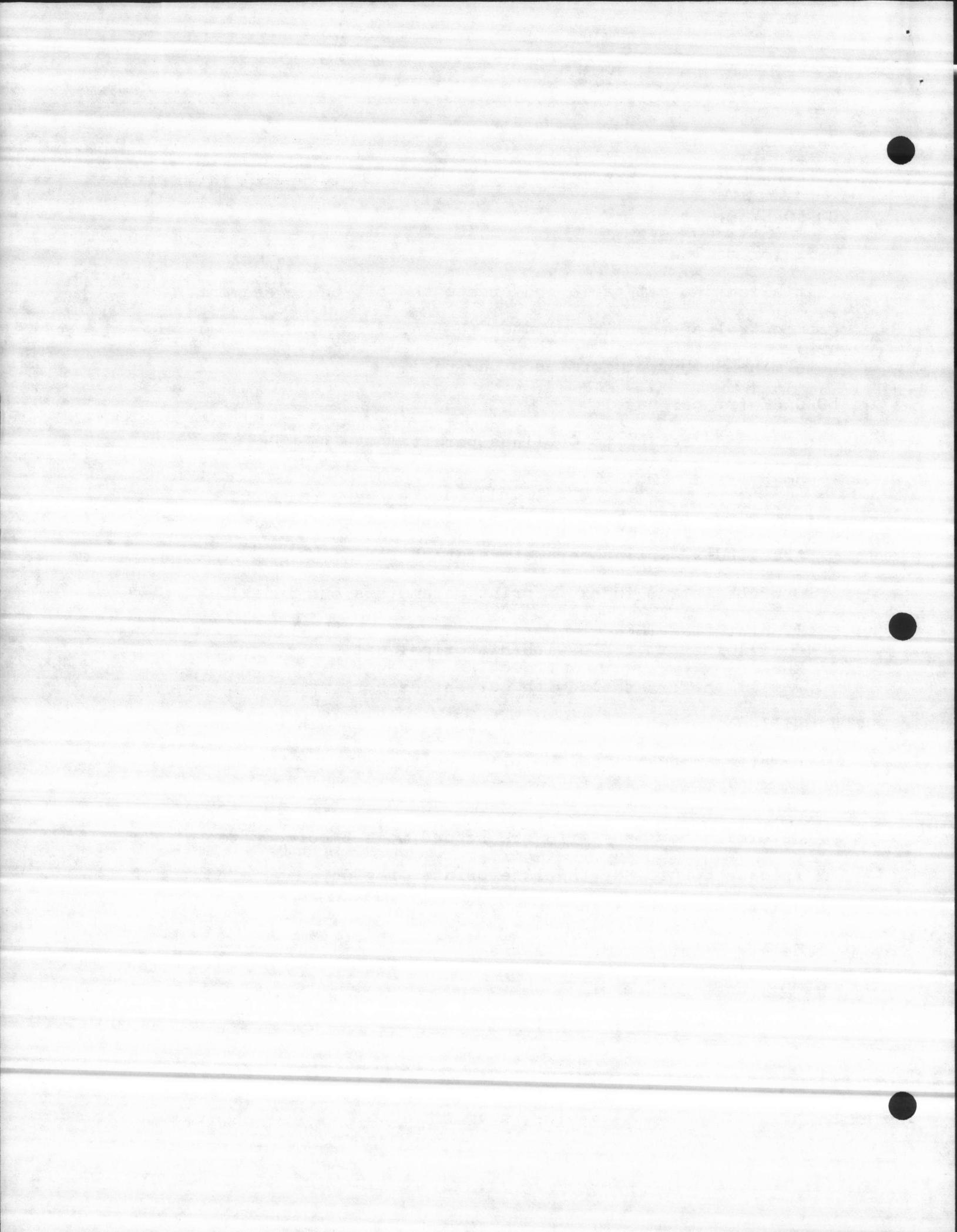
release control of the line printer region.

delete crt table segment.

delete print parameter segment.

END; End of Forever

END Dump\_Data\_Base\_Task;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

SECTION No. 20

HISTORICAL DATA

BY

Mohamed E. Fayad

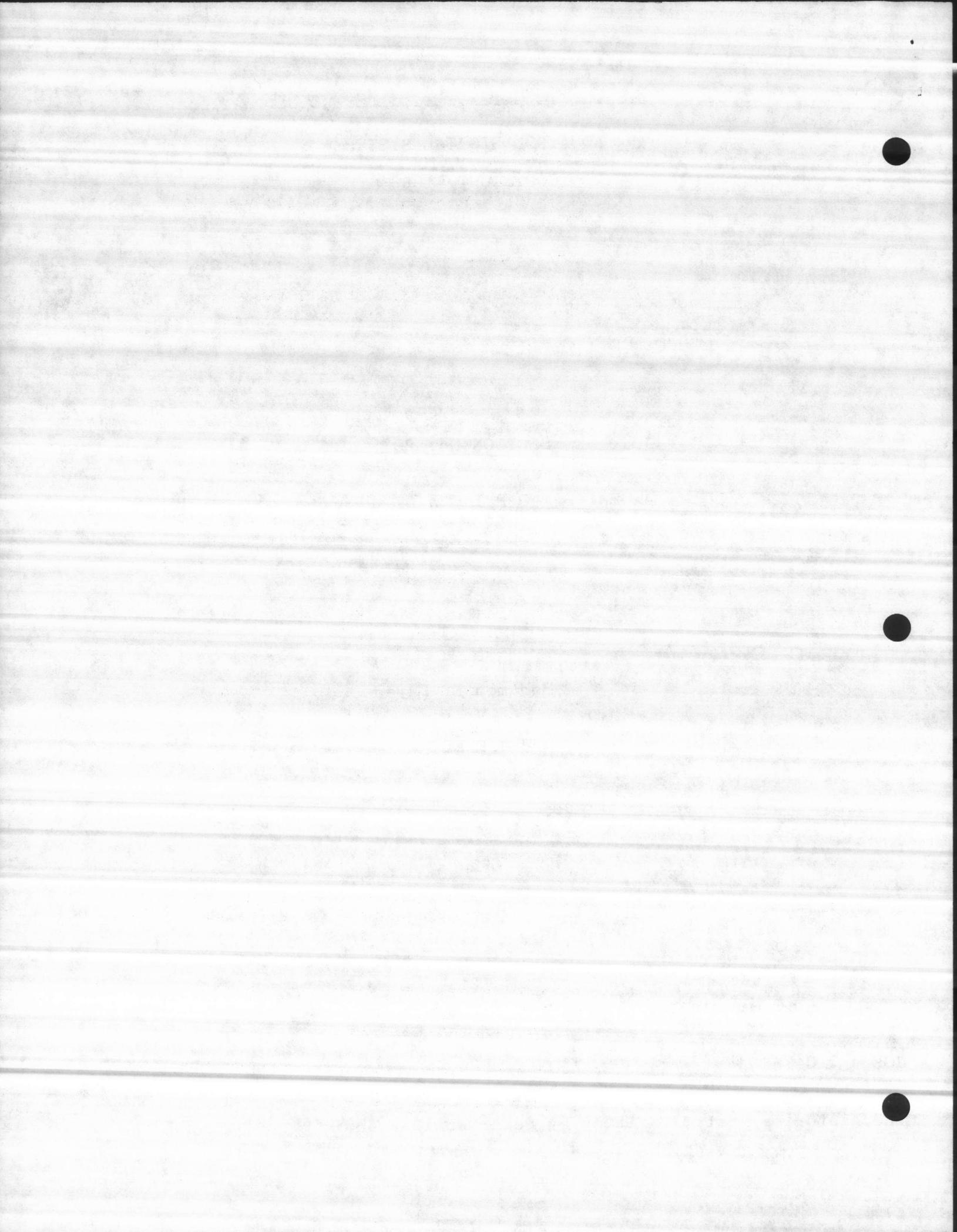
REVIEWED

BY

Jerry Knoth

JOB # : C4758

The historical data - Written by Peter Wollenzien.



(20.1) MODULE NAME : CxxxxHSTM.Pyy

=====

DESCRIPTION :

=====

This procedure displays and prints the historical data of the loaded file.

EXTERNAL PROCEDURES :

=====

Menu\_Display: PROCEDURE (crt\_num, result, off\_set, start\_id\_p,  
bytes\_per\_element, num\_elements) WORD EXTERNAL;

DECLARE (crt\_num) BYTE,  
(result, off\_set) WORD,  
(bytes\_per\_element, num\_elements) WORD,  
(start\_id\_p) POINTER;

END Menu\_Display;

Historical\_Load\_Command: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;  
END Historical\_Load\_Command;

Keyin\_Next\_Proc: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;  
END Keyin\_Next\_Proc;

Keyin\_Previous\_Proc: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;  
END Keyin\_Previous\_Proc;

Keyin\_Page\_Proc: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;  
END Keyin\_Page\_Proc;

Keyin\_Edit\_Sub\_Menu\_Proc: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;  
END Keyin\_Edit\_Sub\_Menu\_Proc;

Prompt\_Display:

PROCEDURE (crt\_num, display\_buf\_t, crt\_out\_mbx\_t,  
row, column, display\_str\_p) EXTERNAL;  
DECLARE (row, column, crt\_num) BYTE,  
(display\_buf\_t, crt\_out\_mbx\_t) TOKEN,  
(display\_str\_p) POINTER;

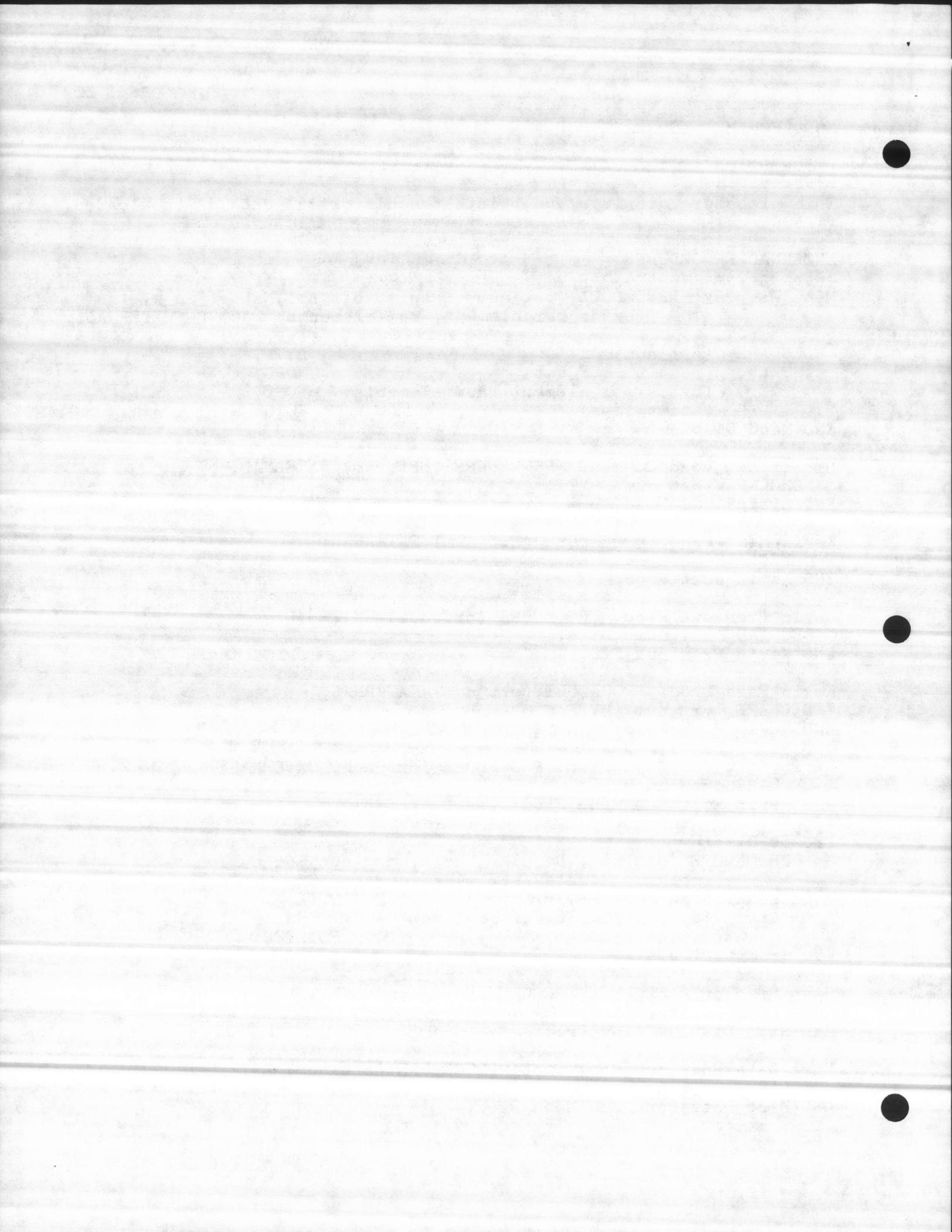
END Prompt\_Display;

Hist\_Display\_Alog\_Proc: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;  
END Hist\_Display\_Alog\_Proc;

Hist\_Data\_Base\_Sub\_Menu\_Proc: PROCEDURE (crt\_num) EXTERNAL;

DECLARE crt\_num BYTE;  
END Hist\_Data\_Base\_Sub\_Menu\_Proc;



```
Hist_Display_Menu: PROCEDURE (crt_num) EXTERNAL;
    DECLARE crt_num BYTE;
END Hist_Display_Menu;

Graphs_Display_Menu: PROCEDURE (crt_num, hist_select) EXTERNAL;
    DECLARE (crt_num) BYTE,
            (hist_select) WORD;
END Graphs_Display_Menu;

Hist_Print_Alog: PROCEDURE (crt_num) EXTERNAL;
    DECLARE crt_num BYTE;
END Hist_Print_Alog;

Hist_Print_Data_Base_Proc: PROCEDURE (crt_num) EXTERNAL;
    DECLARE crt_num BYTE;
END Hist_Print_Data_Base_Proc;

Hist_Print_Report_Sub_Menu: PROCEDURE (crt_num) EXTERNAL;
    DECLARE crt_num BYTE;
END Hist_Print_Report_Sub_Menu;
```

MODULE DECLARATION :  
=====

main\_menu\_command\_count => A byte holds a value of seven.

main\_menu\_prompt\_asc (7) STRUCTURE

menu\_asc => Ten bytes holds the command string.  
Load = Loads specified archive file into memory  
Display = Enters sub-menu for historical display  
Print = Enters sub-menu for historical reports  
Edit = (Declared Globally)  
Next = (Declared Globally)  
Page = (Declared Globally)  
Previous = (Declared Globally)

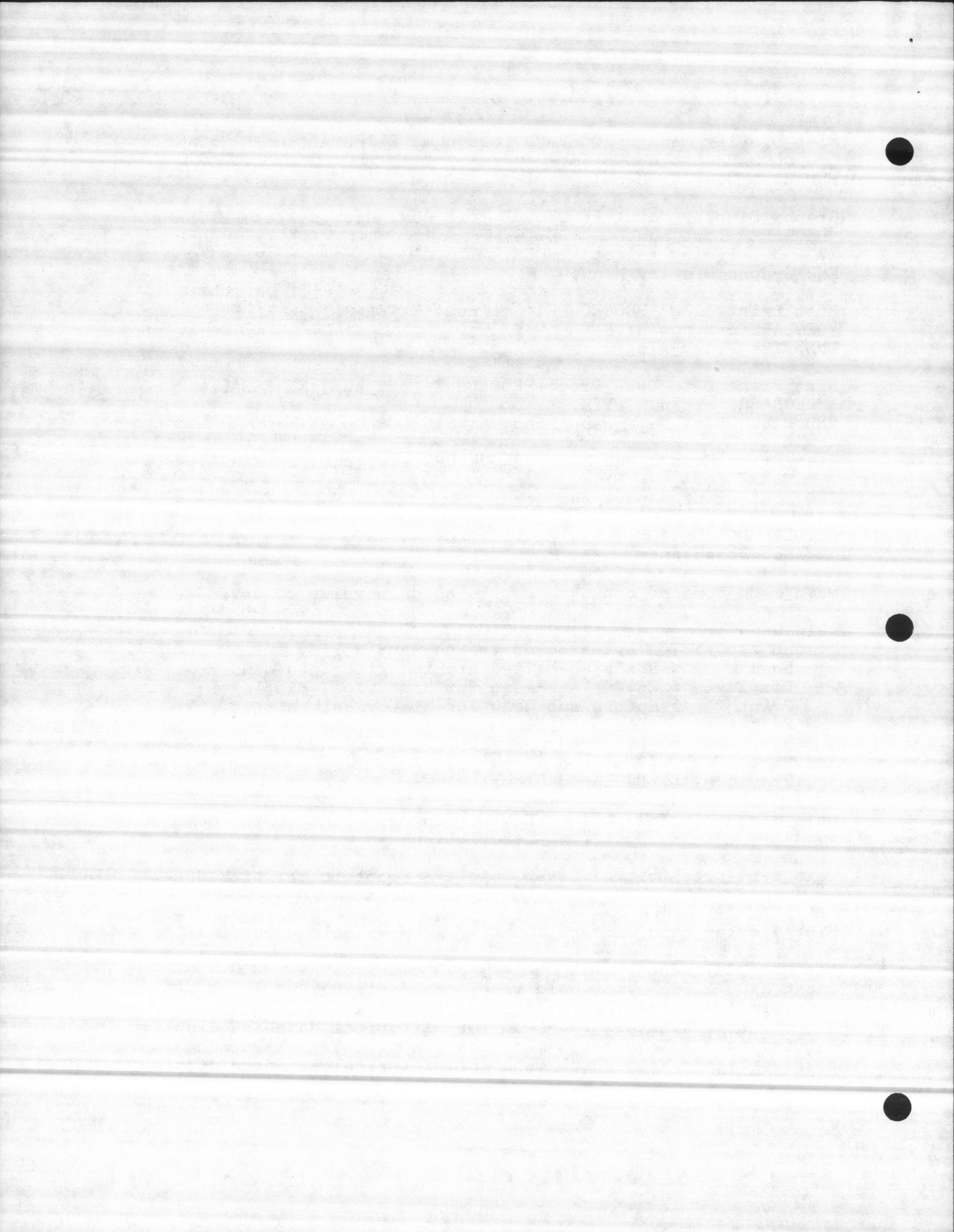
main\_menu\_help\_desc\_tbl => Seven pointers point to the command descriptions.

main\_menu\_proc\_tbl => Seven pointers point to the procedures table.

hist\_display\_menu\_command\_count => A bytes hold a value of four.

hist\_display\_menu\_prompt\_asc (4) STRUCTURE:

menu\_asc => Ten bytes hold the command string.  
Alarmlog = Displays historical alarmlog  
Data Base = Enters sub-menu for historical database display  
Display = Enters sub-menu for historical constructed display  
Disp Trend= Enters sub-menu for historical standard display



hist\_display\_menu\_help\_desc\_tbl => four pointers point to help description table.  
hist\_display\_menu\_proc\_tbl => Four pointers point to the procedure table.

hist\_print\_menu\_command\_count => A byte holds a value of three.  
hist\_print\_menu\_prompt\_asc (3) STRUCTURE:

menu\_asc => 10 bytes hold the command string.

    Alarmlog = Prints historical alarmlog report

    Data Base = Enters sub-menu for historical database report

    Print Cnst= Enters sub-menu for historical report

hist\_print\_menu\_help\_desc\_tbl => Three pointers point to the help descriptions table.

hist\_print\_menu\_proc\_tbl => Three pointers point to the procedures table.

exception => Three words, each used for the exception condition number.

#### PUBLIC PROCEDURES :

=====

Historical\_Menu:

++++++

crt\_num => A byte holds the number of the crt to display on.  
result\_word => A word used as an index.

Proc description :

-----

This procedure displays the historical menu.

Load      Display      Print      Edit      Next      Page      Previous

END Historical\_Menu;

Hist\_Display\_Sub\_Menu:

++++++

crt\_num => A byte holds the number of the crt to display on.  
result\_word => A word used as an index.

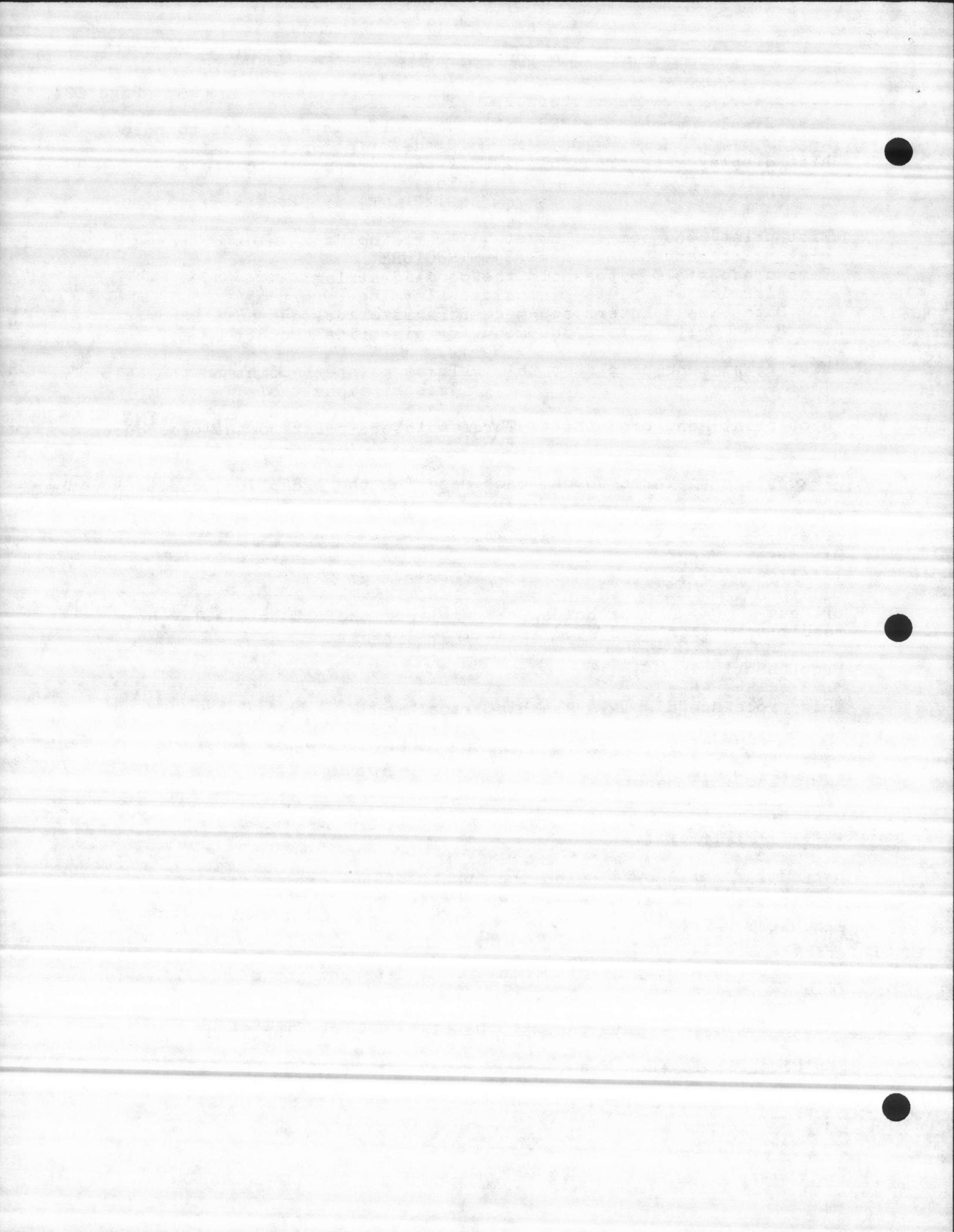
Proc description :

-----

This procedure displays the historical data and its menu of the loaded file.

Alarm log      Data base      Display      Display Trend

END Hist\_Display\_Sub\_Menu;



Keyin\_Hist\_Display\_Alog:  
++++++

crt\_num => A byte holds the number of the crt to display on.

Proc description :

This procedure displays historical alarm log.

END Keyin\_Hist\_Display\_Alog;

Hist\_Print\_Sub\_Menu:  
++++++

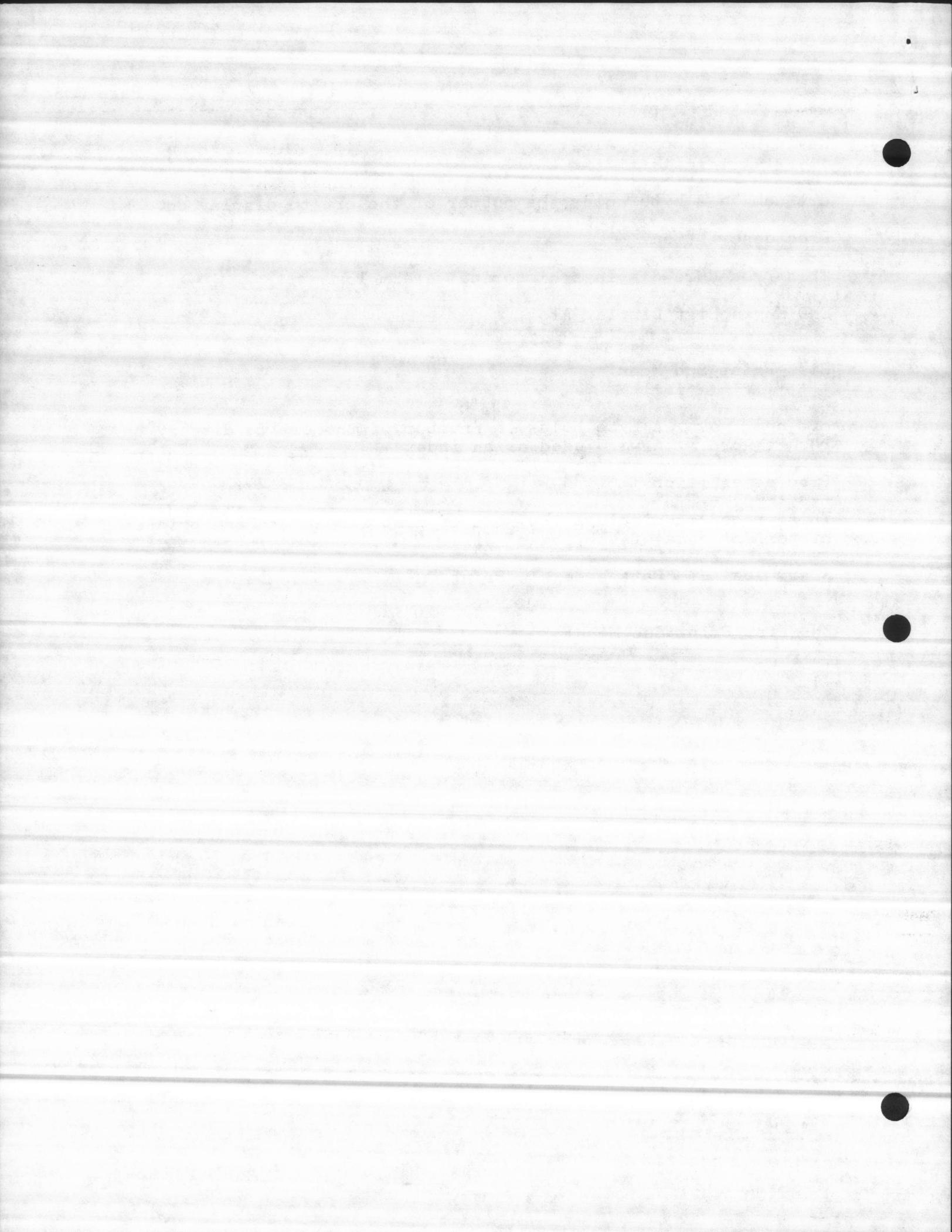
crt\_num => A byte holds the number of the crt to display on.  
result\_word => A word used as an index.

Proc description :

This procedure displays the menu of historical data and prints historical data of the loaded file.

Alarm log      Data base      Print Cnst

END Hist\_Print\_Sub\_Menu;



AQUATROL DIGITAL SYSTEMS  
St. Paul, MN.  
COPYRIGHT 1986

---

SECTION No. 23

A P P E N D I X E S

---

BY

Mohamed E. Fayad

REVIEWED

BY

Jerry Knoth

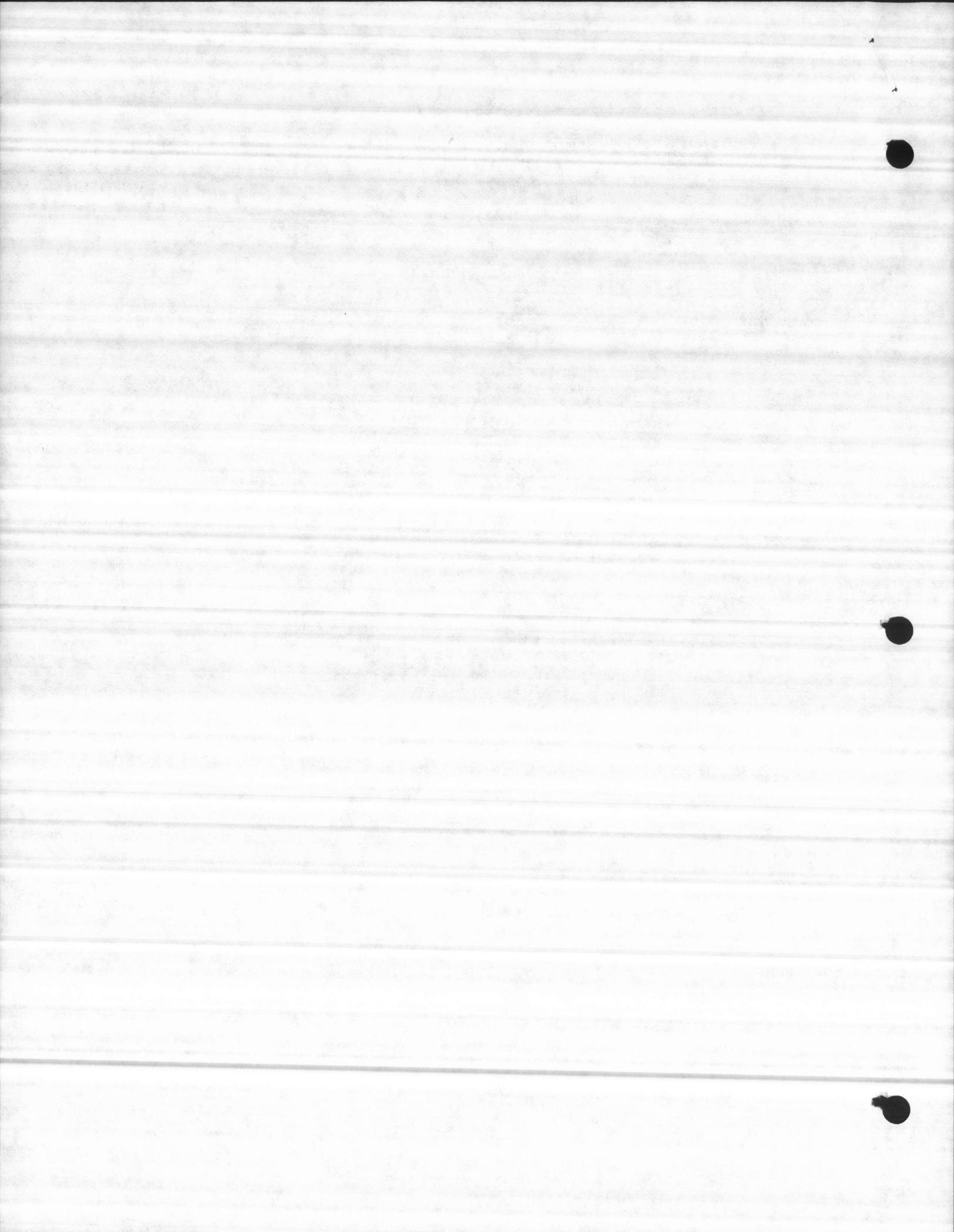
DATE : Aug 13, 1986

---

JOB # : C4758

---

HOLCOMB BLVD, W.W.T.P CAMP LEJEUNE, NC



## APPENDIX - A : ANALOG INPUT POINTS :

\*\* TOTAL REMOTE ANALOG INPUT POINTS = 24

\*\* TOTAL LOCAL ANALOG INPUT POINTS = 9

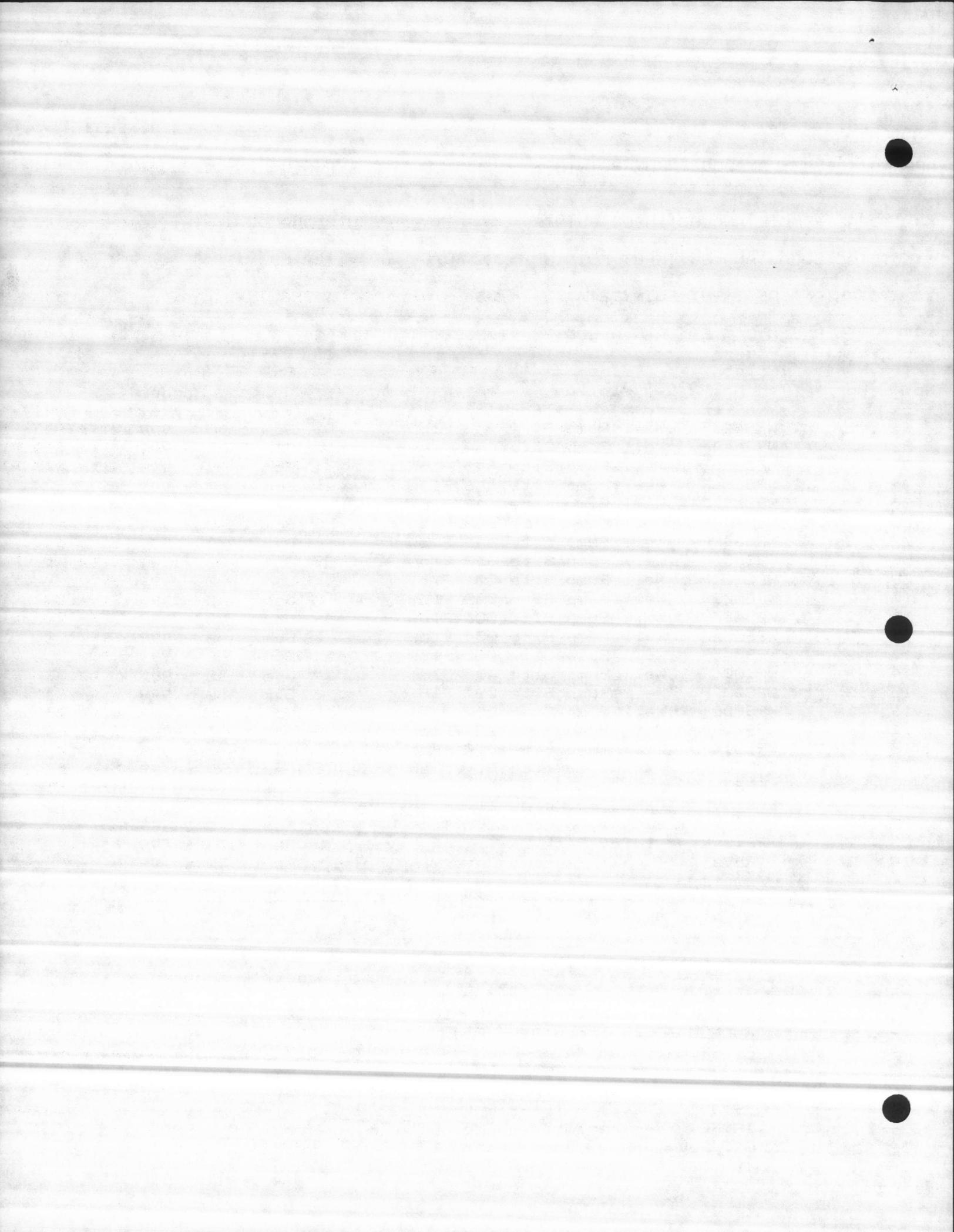
\*\* TOTAL ANALOG INPUT POINYTS = 33

## R E M O T E   A N A L O G   I N P U T   P O I N T S

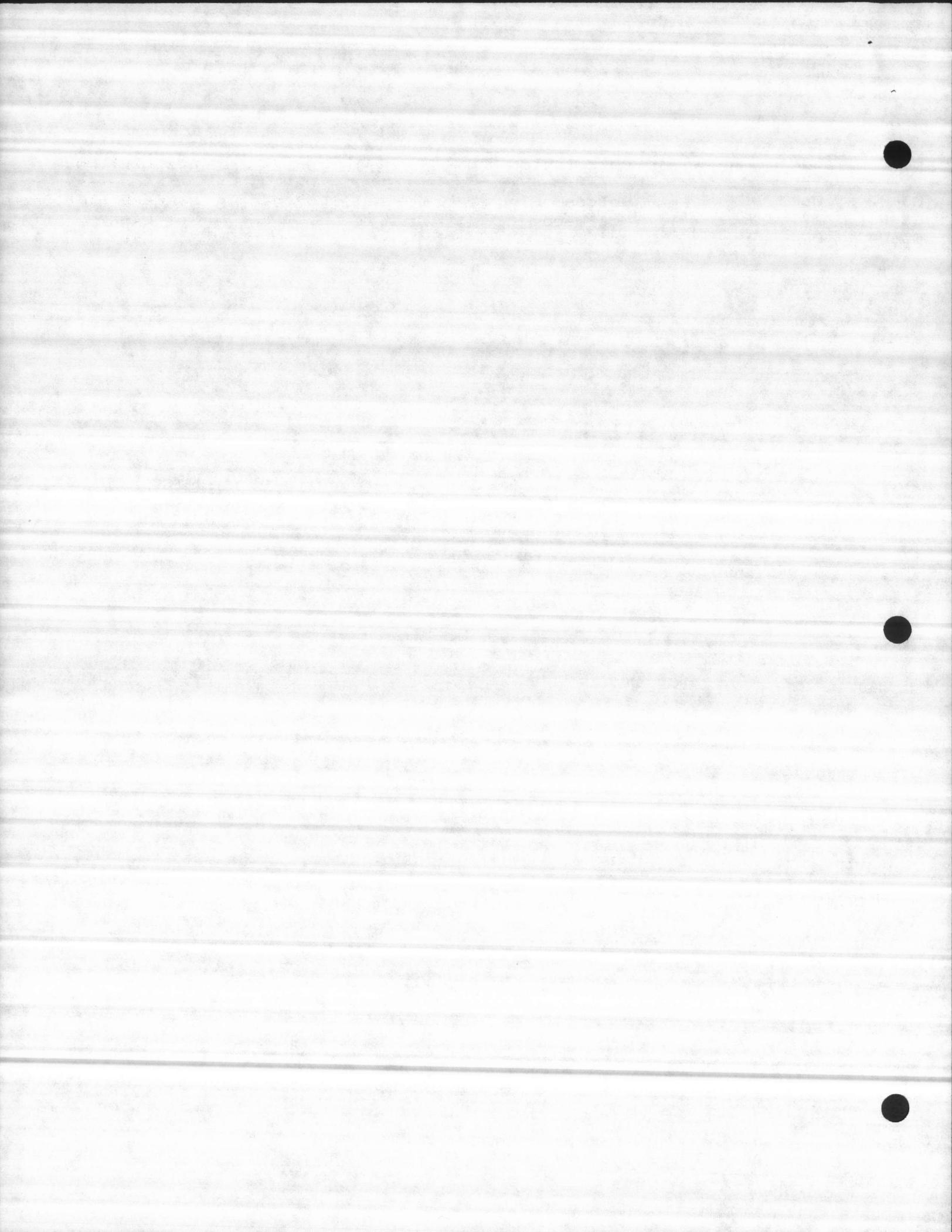
RMT No.	Recv. Word#	Point Tag No.	Point Description	Point Units	Point Ranges
01	2	WPS643F	Well Pump Site 643 Flow	Gpm	0 - 999
02	2	WPS644F	Well Pump Site 644 Flow	Gpm	0 - 999
03	2	WPS645F	Well Pump Site 645 Flow	Gpm	0 - 999
04	2	WPS646F	Well Pump Site 646 Flow	Gpm	0 - 999
05	2	WPS647F	Well Pump Site 647 Flow	Gpm	0 - 999
06	2	WPS648F	Well Pump Site 648 Flow	Gpm	0 - 999
07	2	WPS649F	Well Pump Site 649 Flow	Gpm	0 - 999
08	2	WPS650F	Well Pump Site 650 Flow	Gpm	0 - 999
09	2	WPS001F	Well Pump Site 001 Flow	Gpm	0 - 999
10	2	WPS002F	Well Pump Site 002 Flow	Gpm	0 - 999
11	2	WPS003F	Well Pump Site 003 Flow	Gpm	0 - 999
	2	WPS004F	Well Pump Site 004 Flow	Gpm	0 - 999
13	2	WPS005F	Well Pump Site 005 Flow	Gpm	0 - 999
14	2	WPS006F	Well Pump Site 006 Flow	Gpm	0 - 999
15	2	WPS007F	Well Pump Site 007 Flow	Gpm	0 - 999
16	2	WPS008F	Well Pump Site 008 Flow	Gpm	0 - 999
17	2	WPS009F	Well Pump Site 009 Flow	Gpm	0 - 999
18	2	WPS010F	Well Pump Site 010 Flow	Gpm	0 - 999
19	2	TTETLVL	Tarawa Terrace Elv. Tank Level	Feet	0 to 99.9
20	2	MFPETLVL	Montford Point Elv. Tank Level	Feet	0 to 99.9
21	2	PPETLVL	Paradise Point Elv. Tank Level	Feet	0 to 99.9
22	2	BMETLVL	Berkley Manor Elv. Tank Level	Feet	0 to 99.9
23	2	MPETLVL	Midway Park Elv. Tank Level	Feet	0 to 99.9
24	2	TTRSVL	Tarawa Terrace Reservoir Level	Feet	0 to 99.9

## L O C A L   A N A L O G   I N P U T   P O I N T S

port	addr	Tag No.	description	unit	range
C1-P2	00	PINFLW	Plant Influent Flow	MGD	0 - 9.99
C1-P3	01	HBDFLW	Holcomb Blvd Flow	Gpm	0 - 999
C1-P4	02	RWTRFLW	Raw Water Flow	Gpm	0 - 999
port	addr	Tag No.	description	unit	range
C2-P2	03	TRMFLW	Transmission Main Flow	Gpm	0 - 999
C2-P3	04	RWTRESV	Raw Water Reservoir Level	Feet	0 - 99.9
C2-P4	05	TRMWRSV	Trans. Main Fnsh Wtr Rsrv Lvl	Feet	0 - 99.9



port	addr	Tag No.	description	unit	range
C3-P2	06	HBFWRL	Holcomb Blvd Fnsh Wtr Rsvr Lvl Feet	0 - 99.9	
C3-P3	07	-----	Not Used		
C3-P4	08	-----	Not Used		



## APPENDIX - B : DISCRETE INPUT POINTS :

\*\* TOTAL REMOTE DISCRETE INPUT POINTS = 193  
 \*\* TOTAL LOCAL DISCRETE INPUT POINTS = 72  
 \*\* TOTAL ANALOG ALARMS (HIGH/LOW) = 62  
 \*\* TOTAL MANUAL FAIL SIGNALS = 25  
 \*\* MORE ANALOG ALARMS (LOW LOW ALARMS)= 8  
 \*\* TOTAL DISCRETE INPUT POINYTS = 360

## REMOTE DISCRETE INPUT POINTS

Remote 01 - Well Site No. 643.

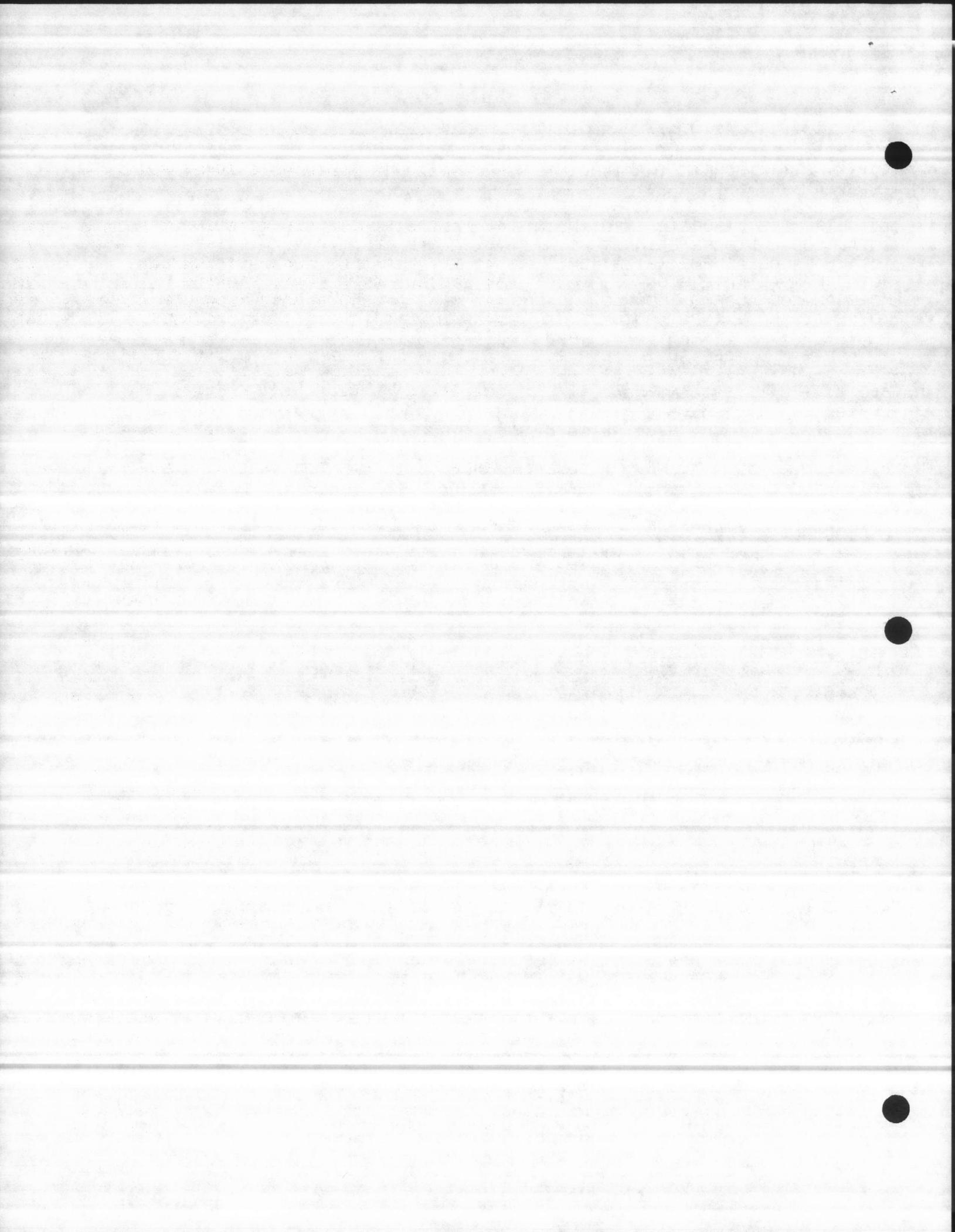
## Receive Word 1

RMT#	bit	Tag No.	description
01	1	WP643NA	Well Pump 643 Not Auto
	2	PWR643F	Power Fail
	4	WP643R	Well Pump 643 Run
	8	WP643F	Well Pump 643 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
 .			
8000	R1DF	Remote 1 Data Fail	

Remote 02 - Well Site No. 644.

## Receive Word 1

RMT#	bit	Tag No.	description
02	1	WP644NA	Well Pump 644 Not Auto
	2	PWR644F	Power Fail
	4	WP644R	Well Pump 644 Run
	8	WP644F	Well Pump 644 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
 .			
8000	R2DF	Remote 2 Data Fail	



Remote 03 - Well Site No. 645.

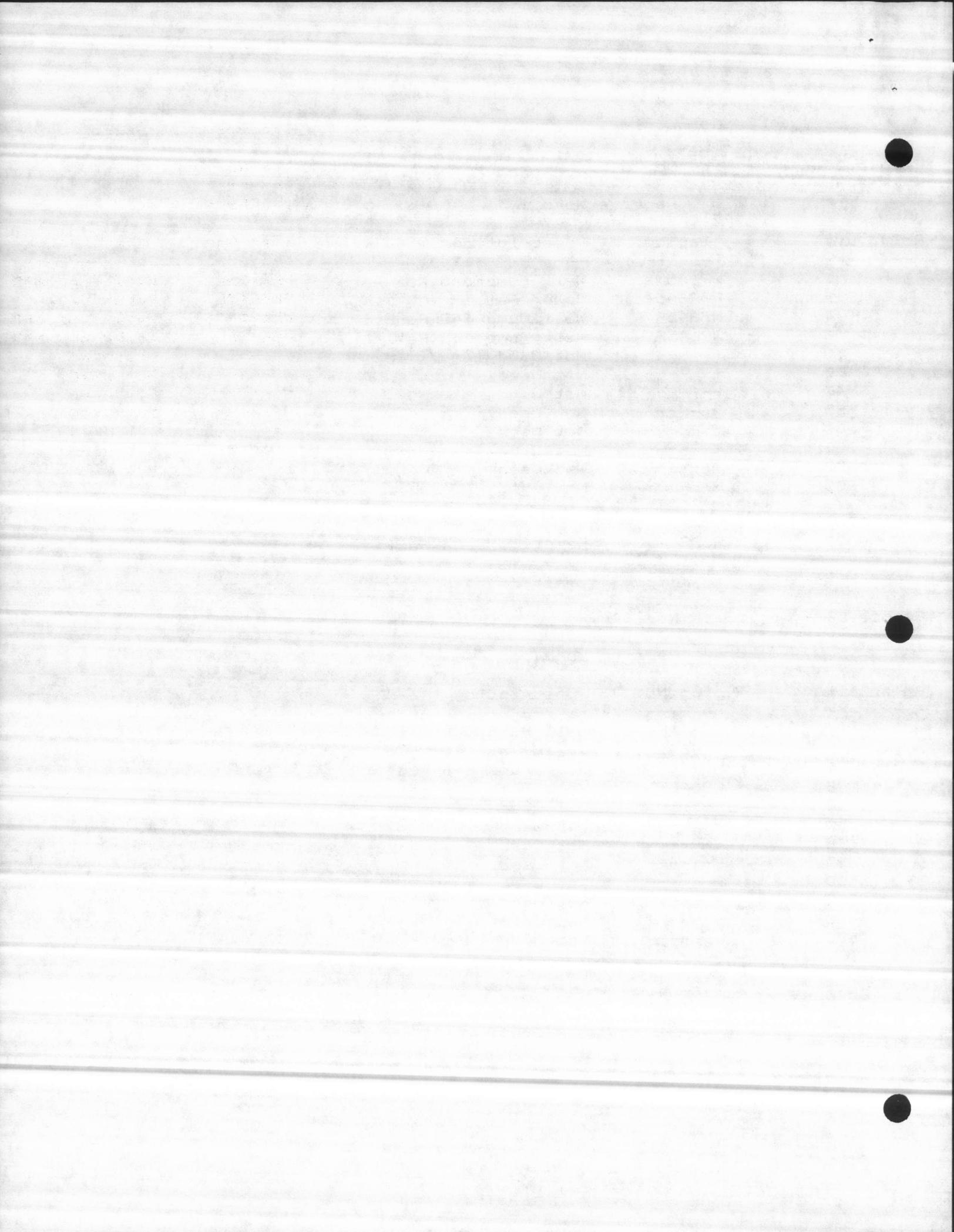
Receive Word 1

RMT#	bit	Tag No.	description
03	1	WP645NA	Well Pump 645 Not Auto
	2	PWR645F	Power Fail
	4	WP645R	Well Pump 645 Run
	8	WP645F	Well Pump 645 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
8000		R3DF	Remote 3 Data Fail

Remote 04 - Well Site No. 646.

Receive Word 1

RMT#	bit	Tag No.	description
04	1	WP646NA	Well Pump 646 Not Auto
	2	PWR646F	Power Fail
	4	WP646R	Well Pump 646 Run
	8	WP646F	Well Pump 646 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
8000		R4DF	Remote 4 Data Fail



Remote 05 - Well Site No. 647.

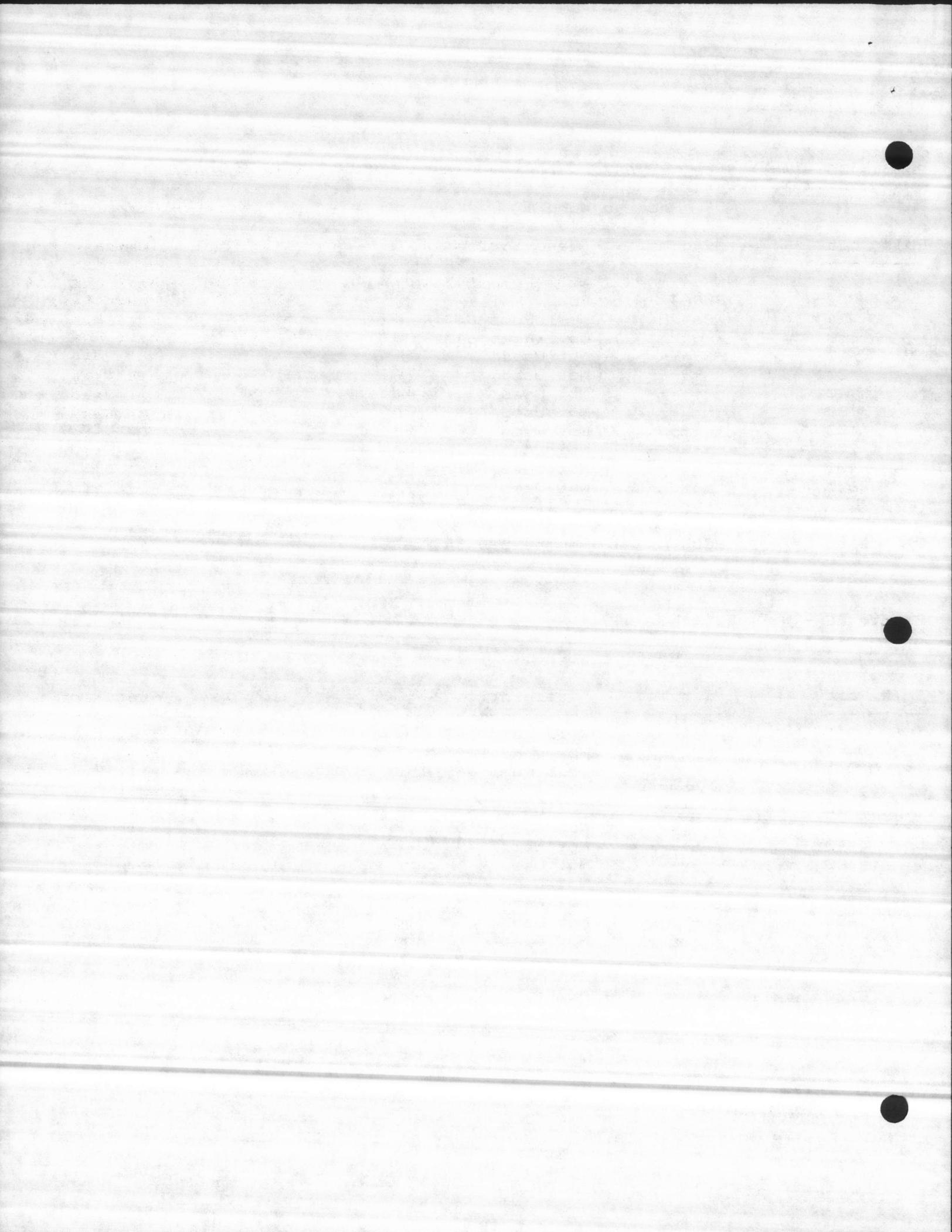
Receive Word 1

RMT#	bit	Tag No.	description
05	1	WP647NA	Well Pump 647 Not Auto
	2	PWR647F	Power Fail
	4	WP647R	Well Pump 647 Run
	8	WP647F	Well Pump 647 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
	.		
	.		
	8000	R5DF	Remote 5 Data Fail

Remote 06 - Well Site No. 648.

Receive Word 1

RMT#	bit	Tag No.	description
06	1	WP648NA	Well Pump 648 Not Auto
	2	PWR648F	Power Fail
	4	WP648R	Well Pump 648 Run
	8	WP648F	Well Pump 648 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
	.		
	.		
	8000	R6DF	Remote 6 Data Fail



Remote 07 - Well Site No. 649.

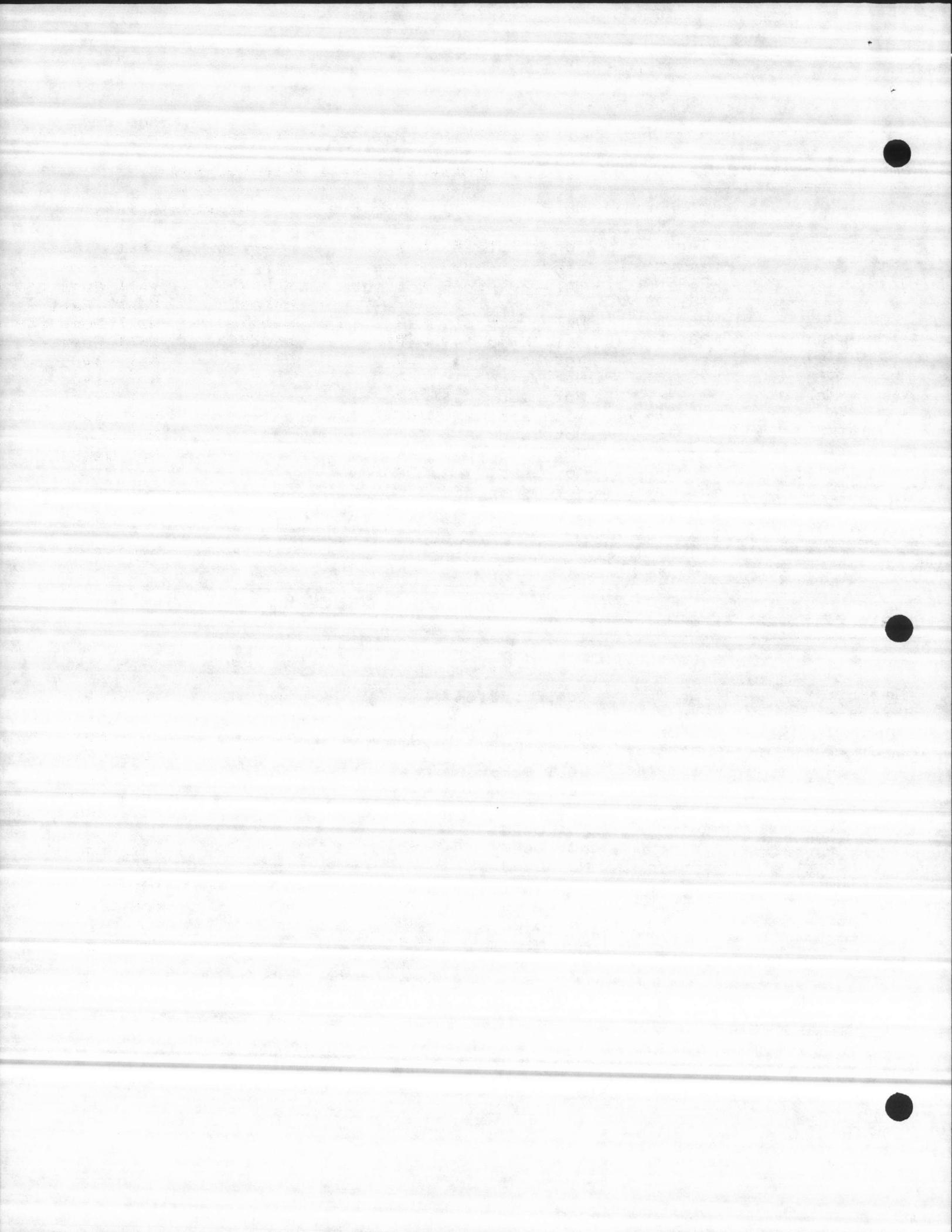
Receive Word 1

RMT#	bit	Tag No.	description
07	1	WP649NA	Well Pump 649 Not Auto
	2	PWR649F	Power Fail
	4	WP649R	Well Pump 649 Run
	8	WP649F	Well Pump 649 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
.	.		
8000		R7DF	Remote 7 Data Fail

Remote 08 - Well Site No. 650.

Receive Word 1

RMT#	bit	Tag No.	description
08	1	WP650NA	Well Pump 650 Not Auto
	2	PWR650F	Power Fail
	4	WP650R	Well Pump 650 Run
	8	WP650F	Well Pump 650 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
.	.		
8000		R8DF	Remote 8 Data Fail



Remote 09 - Well Site No. 001.

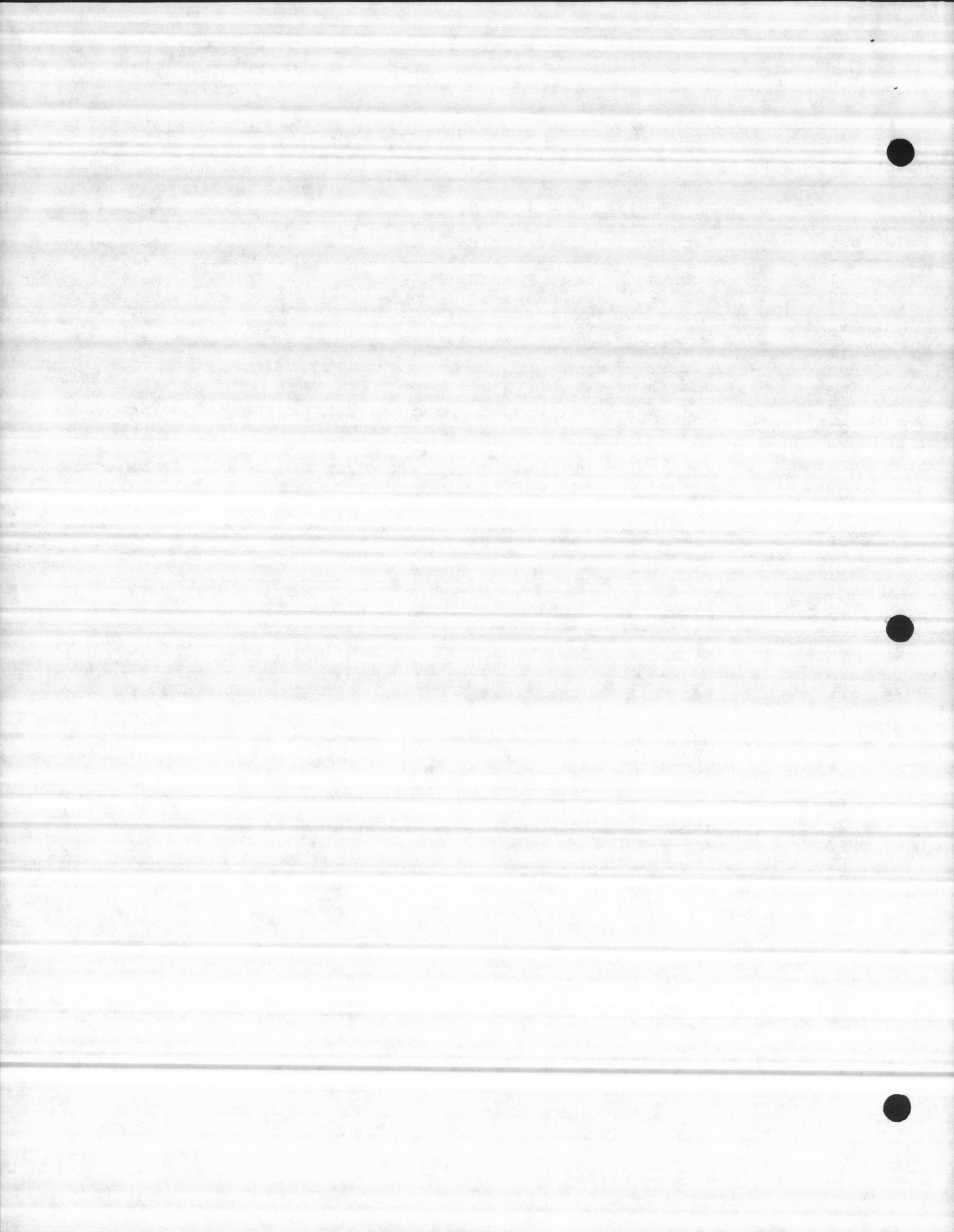
Receive Word 1

RMT#	bit	Tag No.	description
09	1	WP001NA	Well Pump 001 Not Auto
	2	PWR001F	Power Fail
	4	WP001R	Well Pump 001 Run
	8	WP001F	Well Pump 001 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
8000		R9DF	Remote 9 Data Fail

Remote 10 - Well Site No. 002.

Receive Word 1

RMT#	bit	Tag No.	description
10	1	WP002NA	Well Pump 002 Not Auto
	2	PWR002F	Power Fail
	4	WP002R	Well Pump 002 Run
	8	WP002F	Well Pump 002 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
8000		R10DF	Remote 10 Data Fail



Remote 11 - Well Site No. 003.

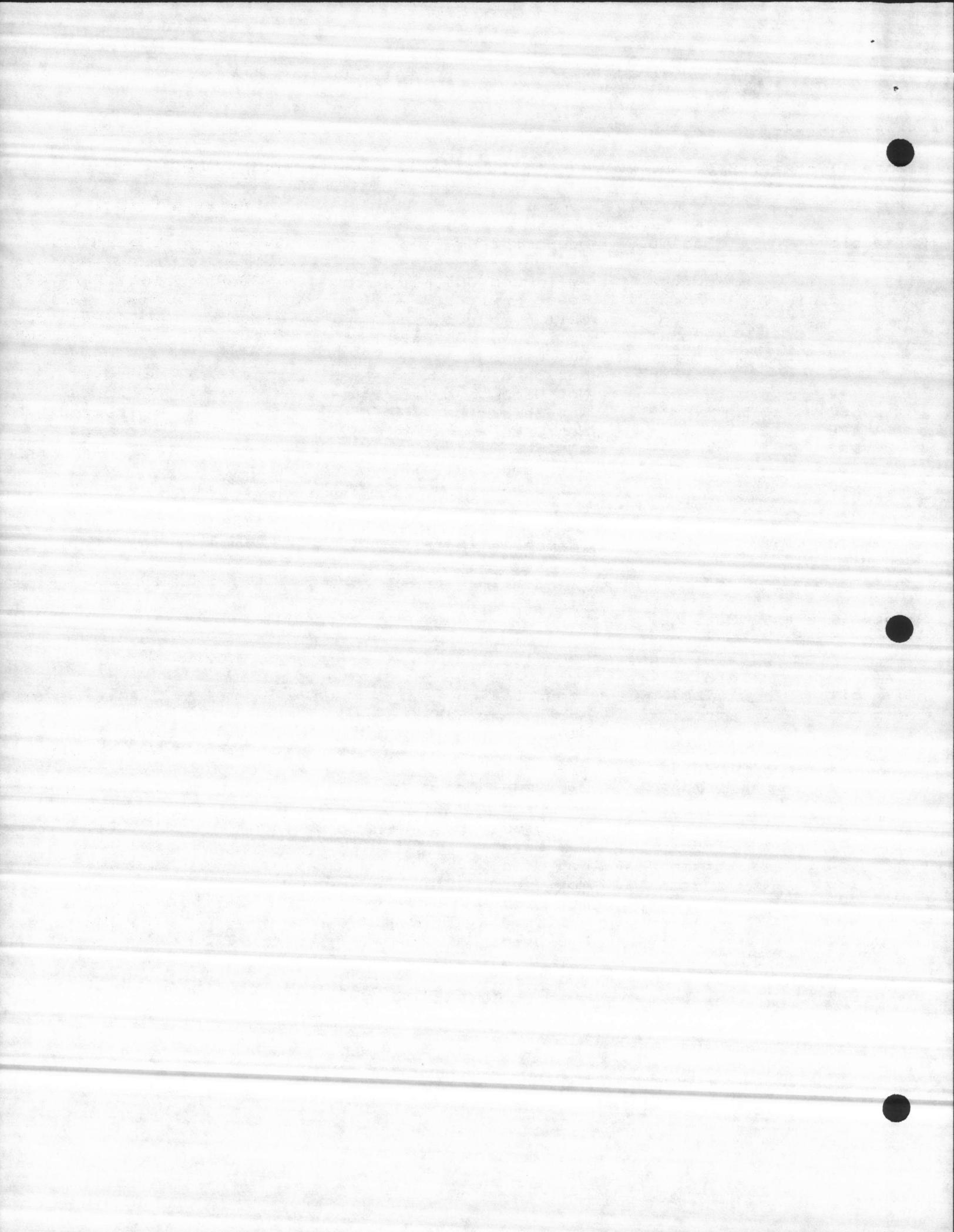
Receive Word 1

RMT#	bit	Tag No.	description
11	1	WP003NA	Well Pump 003 Not Auto
	2	PWR003F	Power Fail
	4	WP003R	Well Pump 003 Run
	8	WP003F	Well Pump 003 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
.			
8000		R11DF	Remote 11 Data Fail

Remote 12 - Well Site No. 004.

Receive Word 1

RMT#	bit	Tag No.	description
12	1	WP004NA	Well Pump 004 Not Auto
	2	PWR004F	Power Fail
	4	WP004R	Well Pump 004 Run
	8	WP004F	Well Pump 004 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
.			
8000		R12DF	Remote 12 Data Fail



Remote 13 - Well Site No. 005.

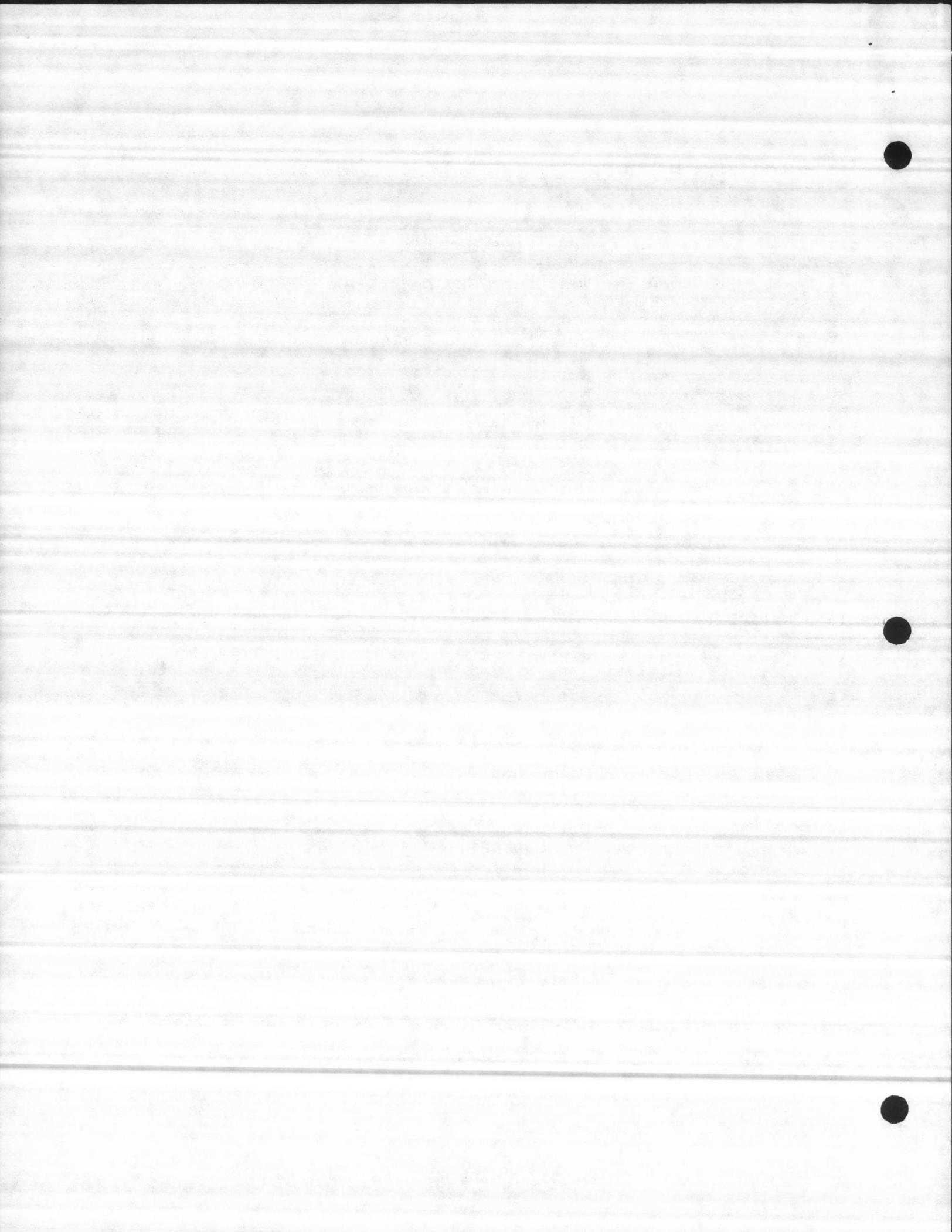
Receive Word 1

RMT#	bit	Tag No.	description
13	1	WP005NA	Well Pump 005 Not Auto
	2	PWR005F	Power Fail
	4	WP005R	Well Pump 005 Run
	8	WP005F	Well Pump 005 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
8000		R13DF	Remote 13 Data Fail

Remote 14 - Well Site No. 006.

Receive Word 1

RMT#	bit	Tag No.	description
14	1	WP006NA	Well Pump 006 Not Auto
	2	PWR006F	Power Fail
	4	WP006R	Well Pump 006 Run
	8	WP006F	Well Pump 006 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
8000		R14DF	Remote 14 Data Fail



Remote 15 - Well Site No. 007.

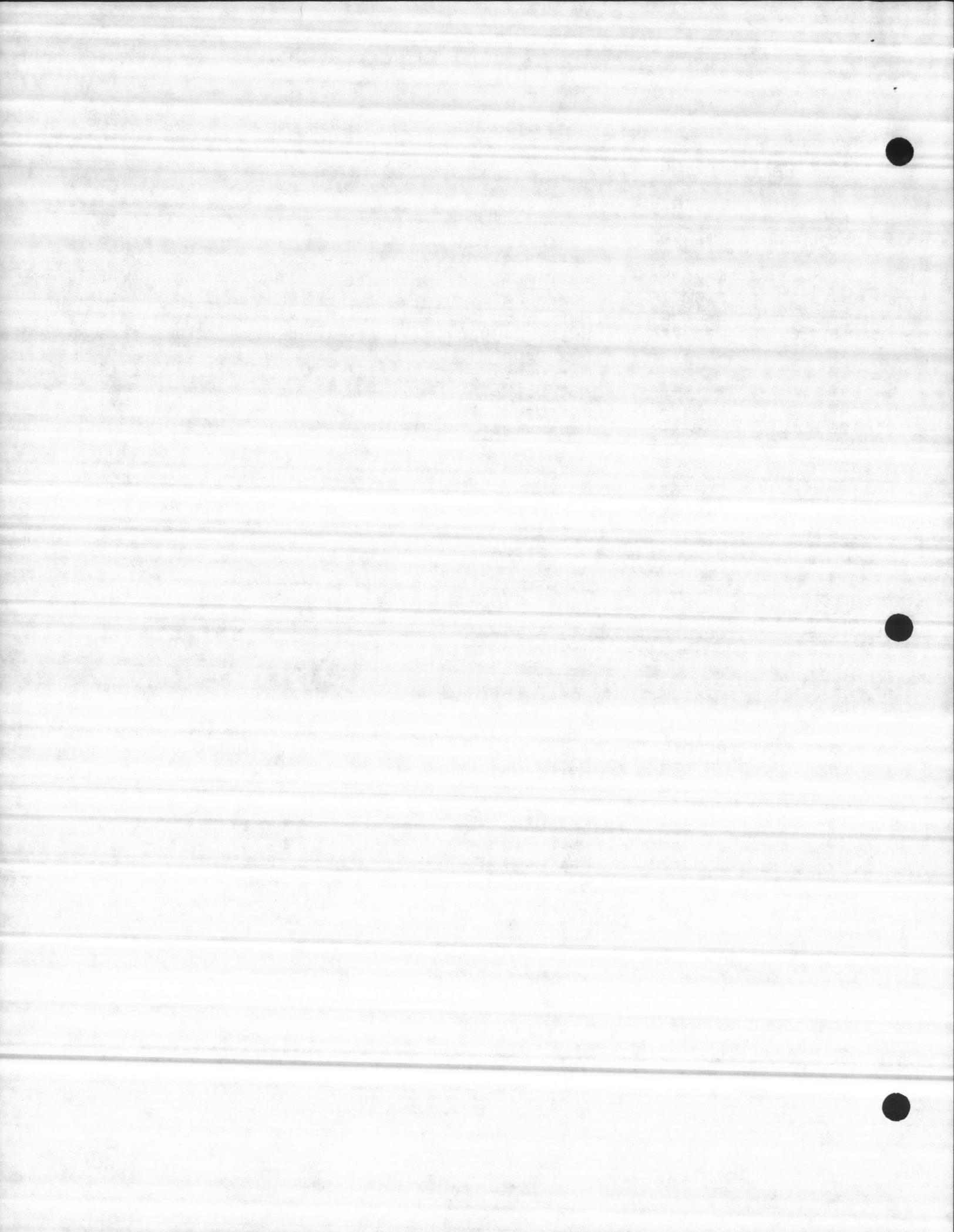
Receive Word 1

RMT#	bit	Tag No.	description
15	1	WP007NA	Well Pump 007 Not Auto
	2	PWR007F	Power Fail
	4	WP007R	Well Pump 007 Run
	8	WP007F	Well Pump 007 Fail
10	-----		Not Used
20	-----		Not Used
40	-----		Not Used
80	-----		Not Used
100	-----		Not Used
200	-----		Not Used
400	-----		Not Used
800	-----		Not Used
.			
.			
8000	R15DF		Remote 15 Data Fail

Remote 16 - Well Site No. 008.

Receive Word 1

RMT#	bit	Tag No.	description
16	1	WP008NA	Well Pump 008 Not Auto
	2	PWR008F	Power Fail
	4	WP008R	Well Pump 008 Run
	8	WP008F	Well Pump 008 Fail
10	-----		Not Used
20	-----		Not Used
40	-----		Not Used
80	-----		Not Used
100	-----		Not Used
200	-----		Not Used
400	-----		Not Used
800	-----		Not Used
.			
.			
8000	R16DF		Remote 16 Data Fail



Remote 17 - Well Site No. 009.

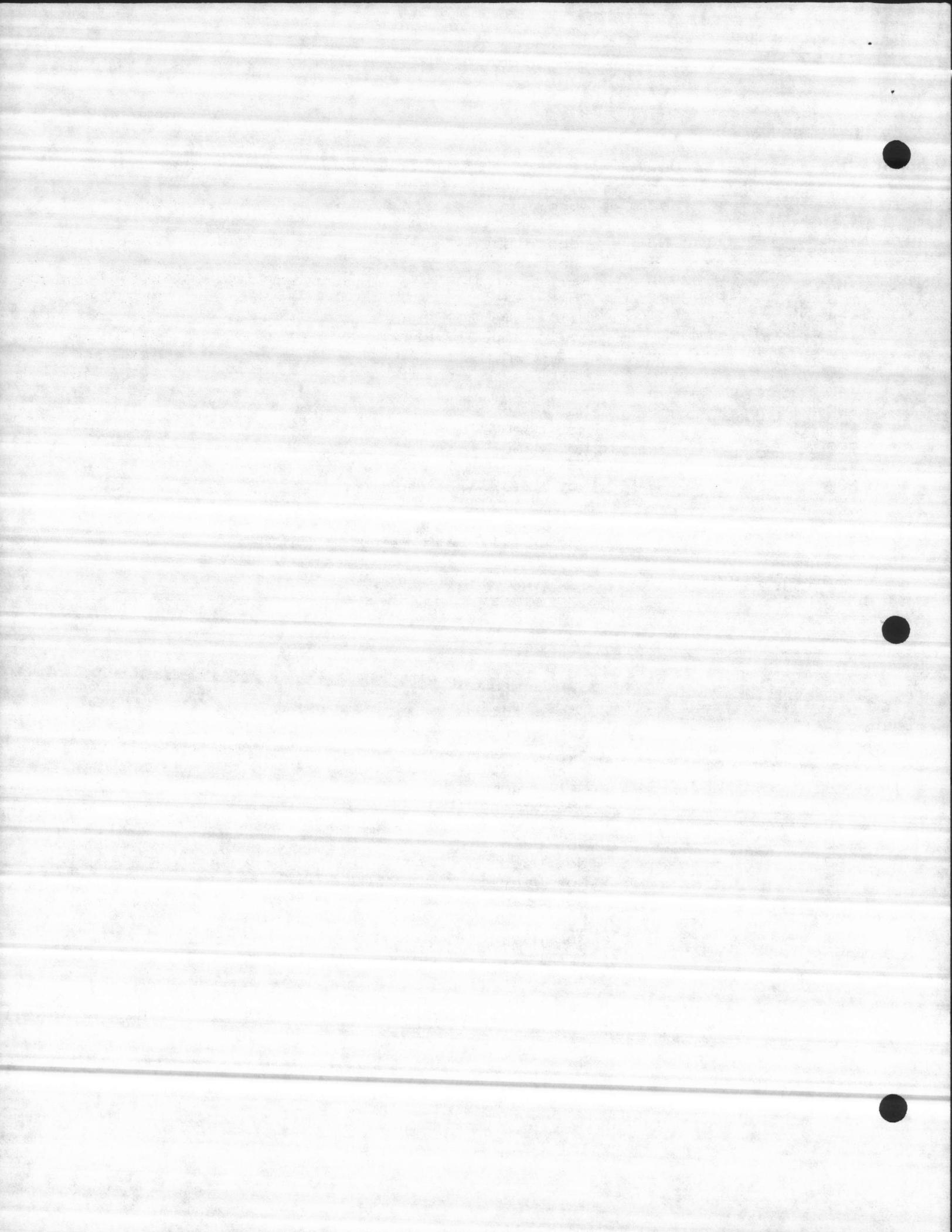
Receive Word 1

RMT#	bit	Tag No.	description
17	1	WP009NA	Well Pump 009 Not Auto
	2	PWR009F	Power Fail
	4	WP009R	Well Pump 009 Run
	8	WP009F	Well Pump 009 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
	8000	R17DF	Remote 17 Data Fail

Remote 18 - Well Site No. 010.

Receive Word 1

RMT#	bit	Tag No.	description
18	1	WP010NA	Well Pump 010 Not Auto
	2	PWR010F	Power Fail
	4	WP010R	Well Pump 010 Run
	8	WP010F	Well Pump 010 Fail
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
	8000	R18DF	Remote 18 Data Fail



-----  
Remote 19 - Tarawa Terrace Elevated Tank  
-----

## Receive Word 1

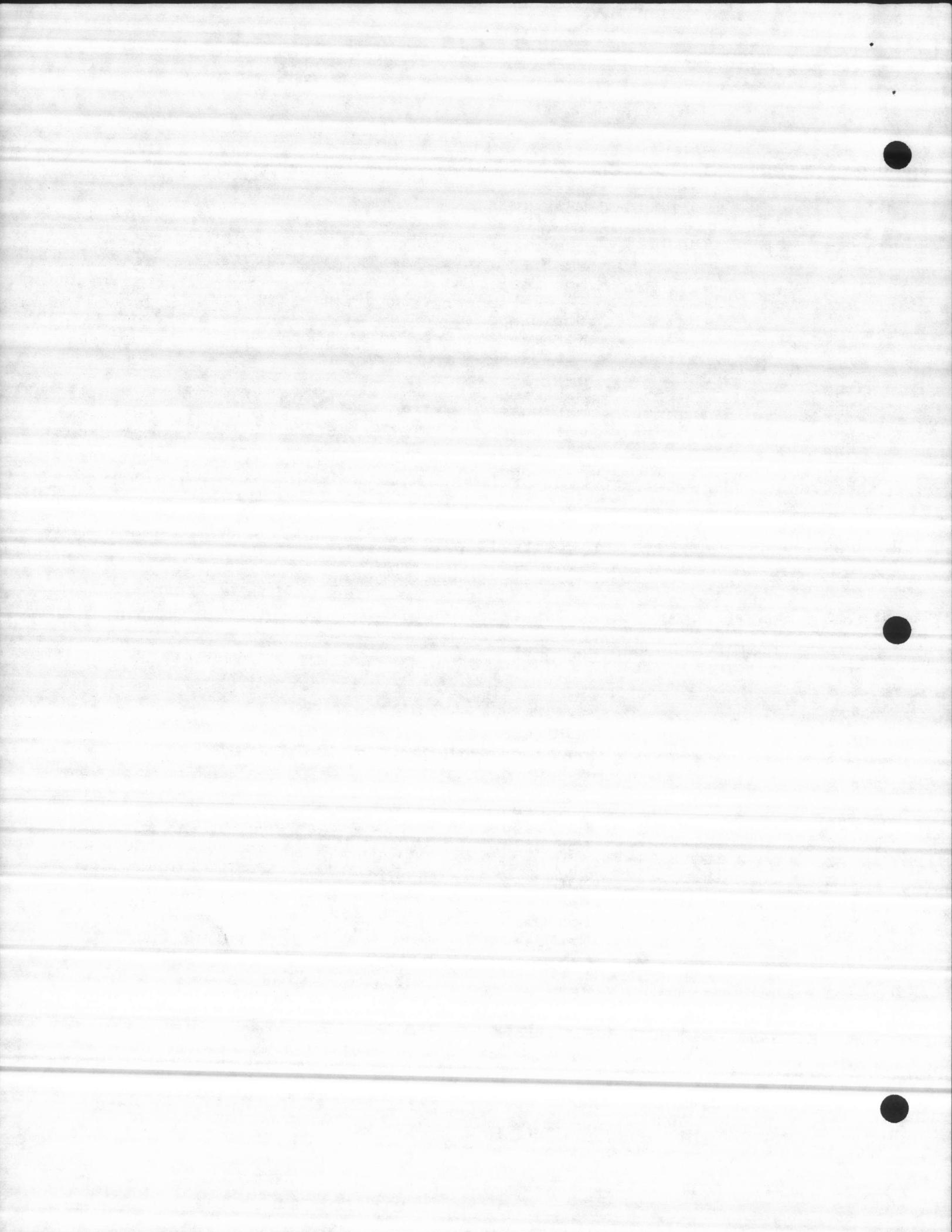
RMT#	bit	Tag No.	description
19	1	-----	Not Used
	2	TTEPWRF	Tarawa Terrace Elv. Tank Power Fail
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
			.
			.
8000		R19DF	Remote 19 Data Fail

----------  
Remote 20 - Montford Point Elevated Tank  
-----

## Receive Word 1

RMT#	bit	Tag No.	description
20	1	-----	Not Used
	2	MPEPWRF	Montford Point Elv. Tank Power Fail
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
			.
8000		R20DF	Remote 20 Data Fail

-----



## Remote 21 - Paradise Point Elevated Tank

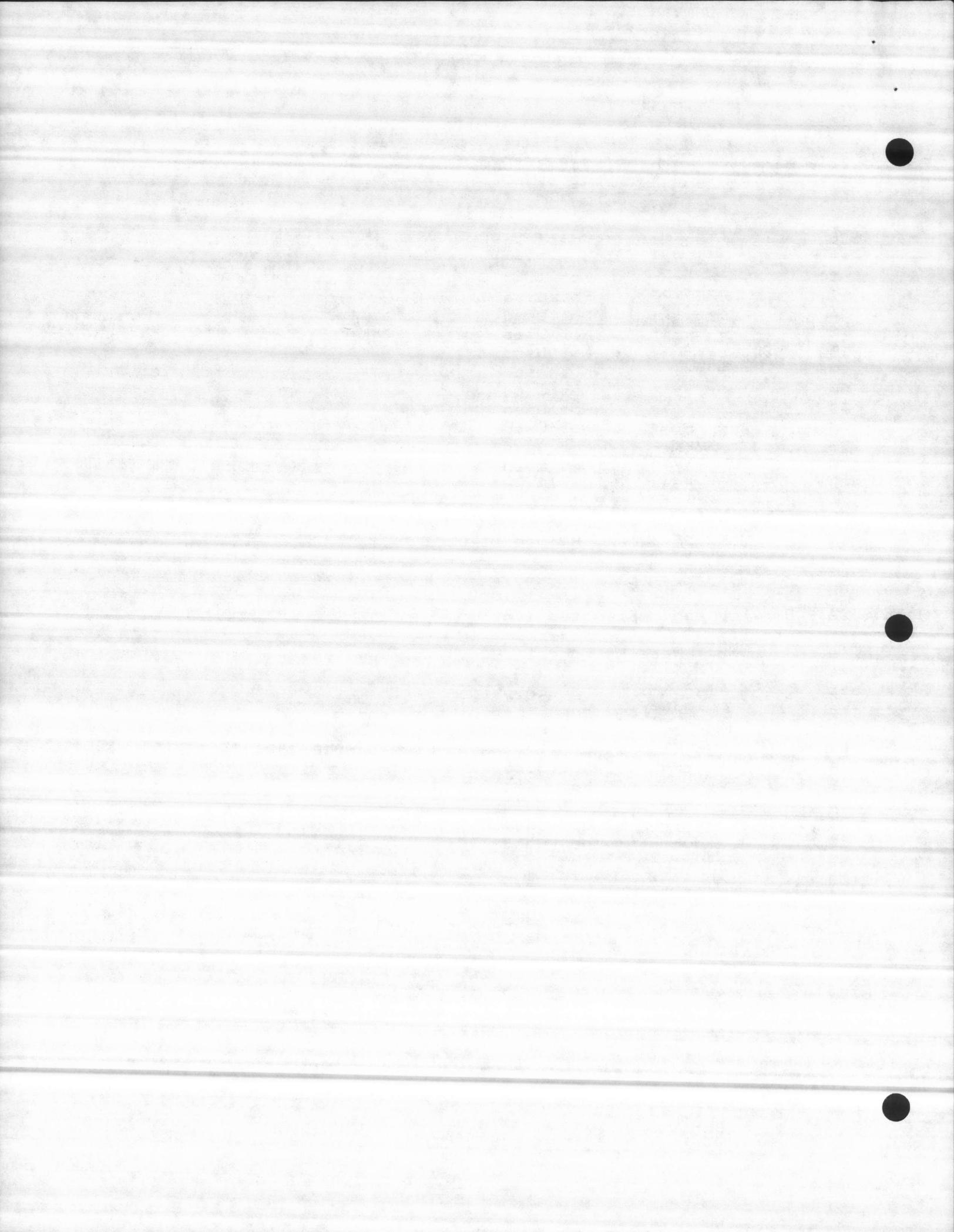
## Receive Word 1

RMT#	bit	Tag No.	description
21	1	-----	Not Used
	2	PPEPWRF	Paradise Point Elv. Tank Power Fail
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
	.		
	.		
	8000	R21DF	Remote 21 Data Fail

## Remote 22 - Berkley Manor Elevated Tank

## Receive Word 1

RMT#	bit	Tag No.	description
22	1	-----	Not Used
	2	BMEPWRF	Berkley Manor Elv. Tank Power Fail
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
	.		
	.		
	8000	R22DF	Remote 22 Data Fail



## Remote 23 - Midway Park Elevated Tank

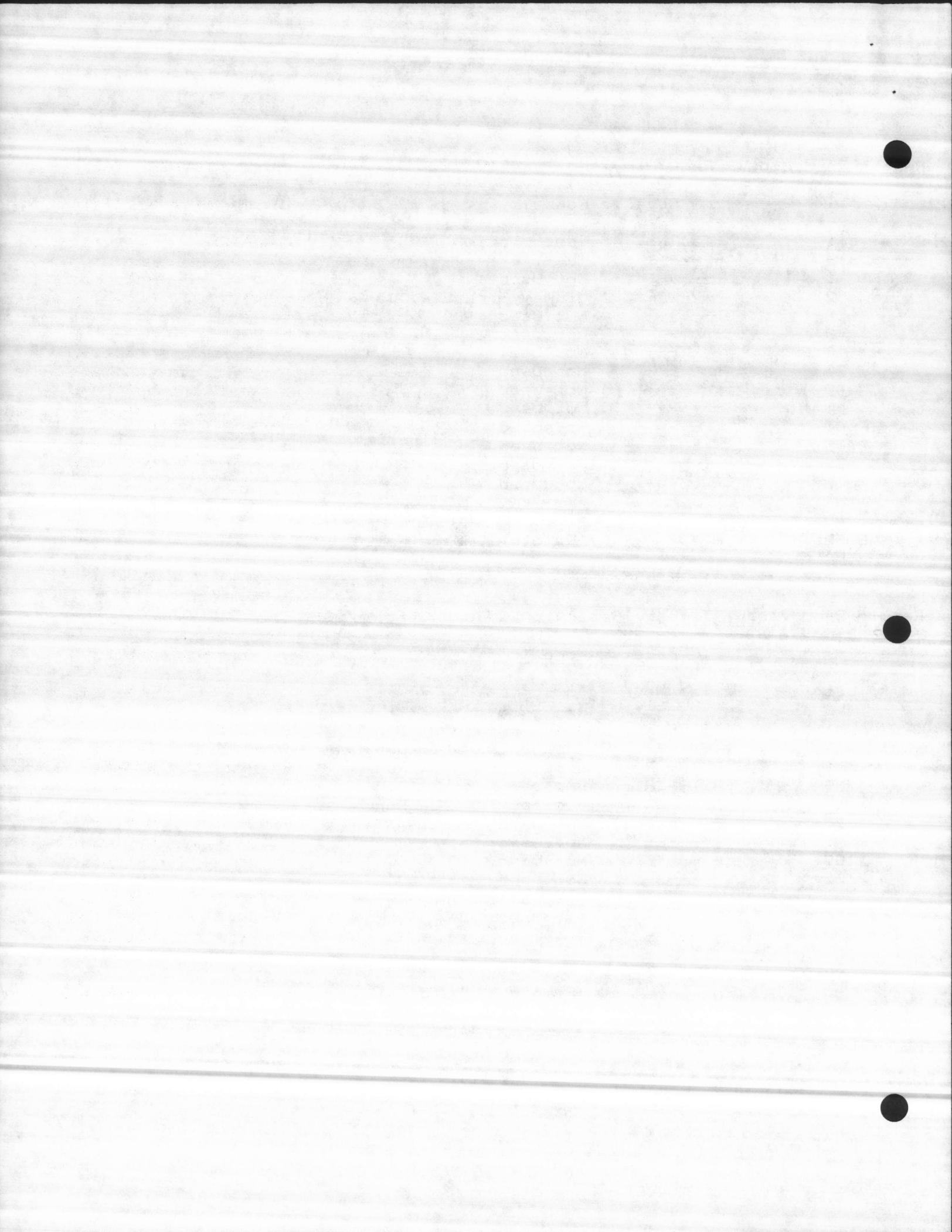
## Receive Word 1

RMT#	bit	Tag No.	description
23	1	-----	Not Used
	2	MWYPWRF	Midway Park Elv. Tank Power Fail
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
.	.		
8000		R23DF	Remote 23 Data Fail

## Remote 24 - Tarawa Terrace Reservoir &amp; Pump Station

## Receive Word 1

RMT#	bit	Tag No.	description
24	1	BSTRP1R	Booster Pump 01 Run
	2	TPSPWRF	Tarawa Terrace Pump Sta. Power Fail
	4	BSTRP2R	Booster Pump 02 Run
	8	BSTRP3R	Booster Pump 03 Run
	10	BSTRP4R	Booster Pump 04 Run
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used
.	.		
8000		R24DF	Remote 24 Data Fail



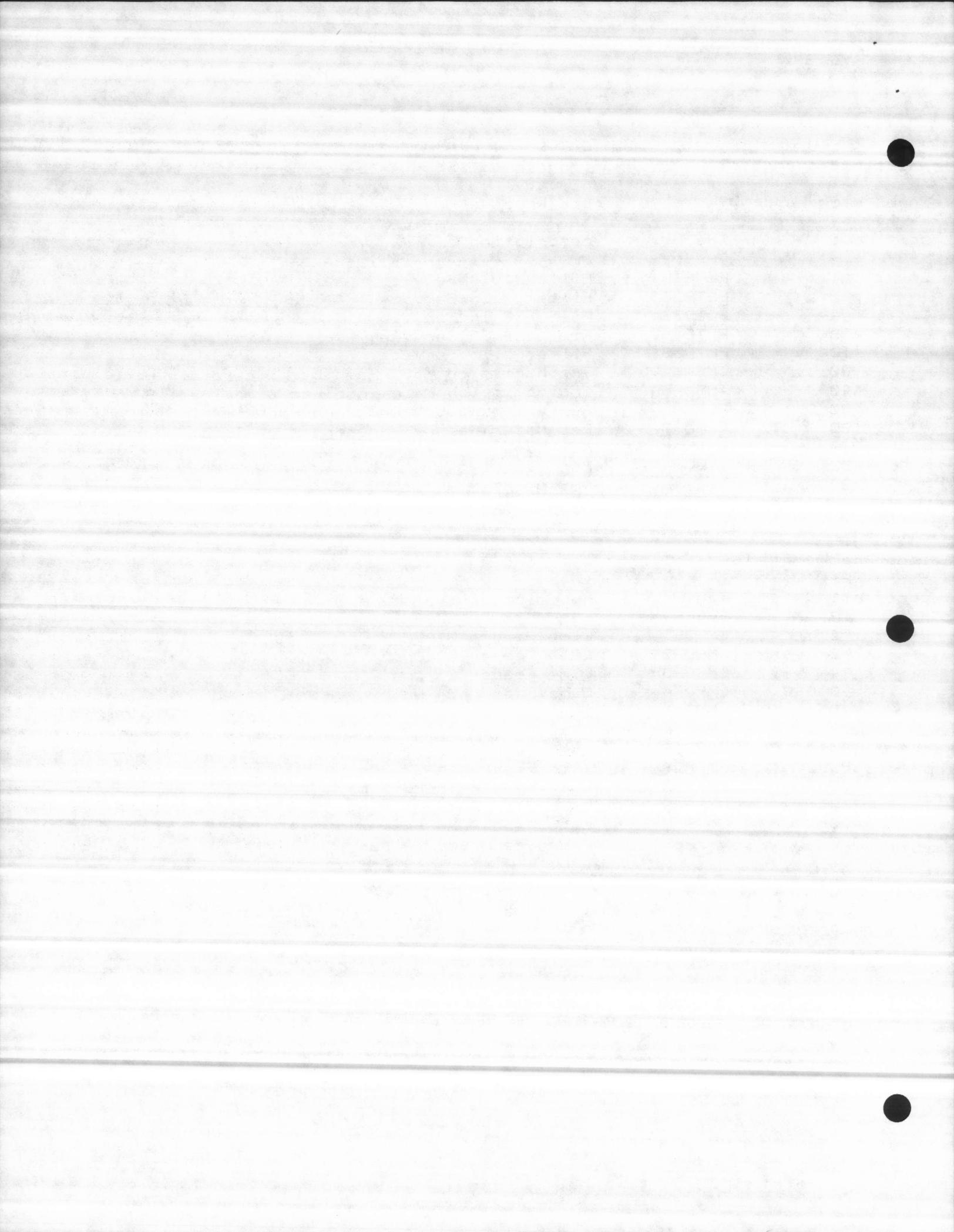
## LOCAL DISCRETE INPUT POINTS

input rack addr 09 lcl\_discrete word #

port	bit	Tag No.	description
C4-P1	1	FLTR1S	Filter 1 In Service
	2	FLTR2S	Filter 2 In Service
	4	FLTR3S	Filter 3 In Service
	8	FLTR4S	Filter 4 In Service
	10	FLTR5S	Filter 5 In Service
	20	TMFWHLVL	Trainsmission Main Fnsh Wtr Underdrain H. L
	40	TMFWHSA	Trainsmission Main Fnsh Wtr pumps heat S A.
	80	RWRUHL	Raw Water Reserv Underdrn High Level.
	100	RWRUHT	Raw Water Reserv Underdrn High Temp.
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used

input rack addr 10 lcl\_discrete word #

port	bit	Tag No.	description
C4	2	LMFDR1	Lime Feeder 01 On
	2	LMFDR2	Lime Feeder 02 On
	4	LMFDR3	Lime Feeder 03 On
	8	LMFDR4	Lime Feeder 04 On
	10	LMFDR5	Lime Feeder 05 On
	20	LMFDR6	Lime Feeder 06 On
	40	LMFP1R	Lime Feed Pump 01 Run
	80	LMFP2R	Lime Feed Pump 02 Run
	100	LMFP3R	Lime Feed Pump 03 Run
	200	LMFP4R	Lime Feed Pump 04 Run
	400	LMFP5R	Lime Feed Pump 05 Run
	800	LMFP6R	Lime Feed Pump 06 Run

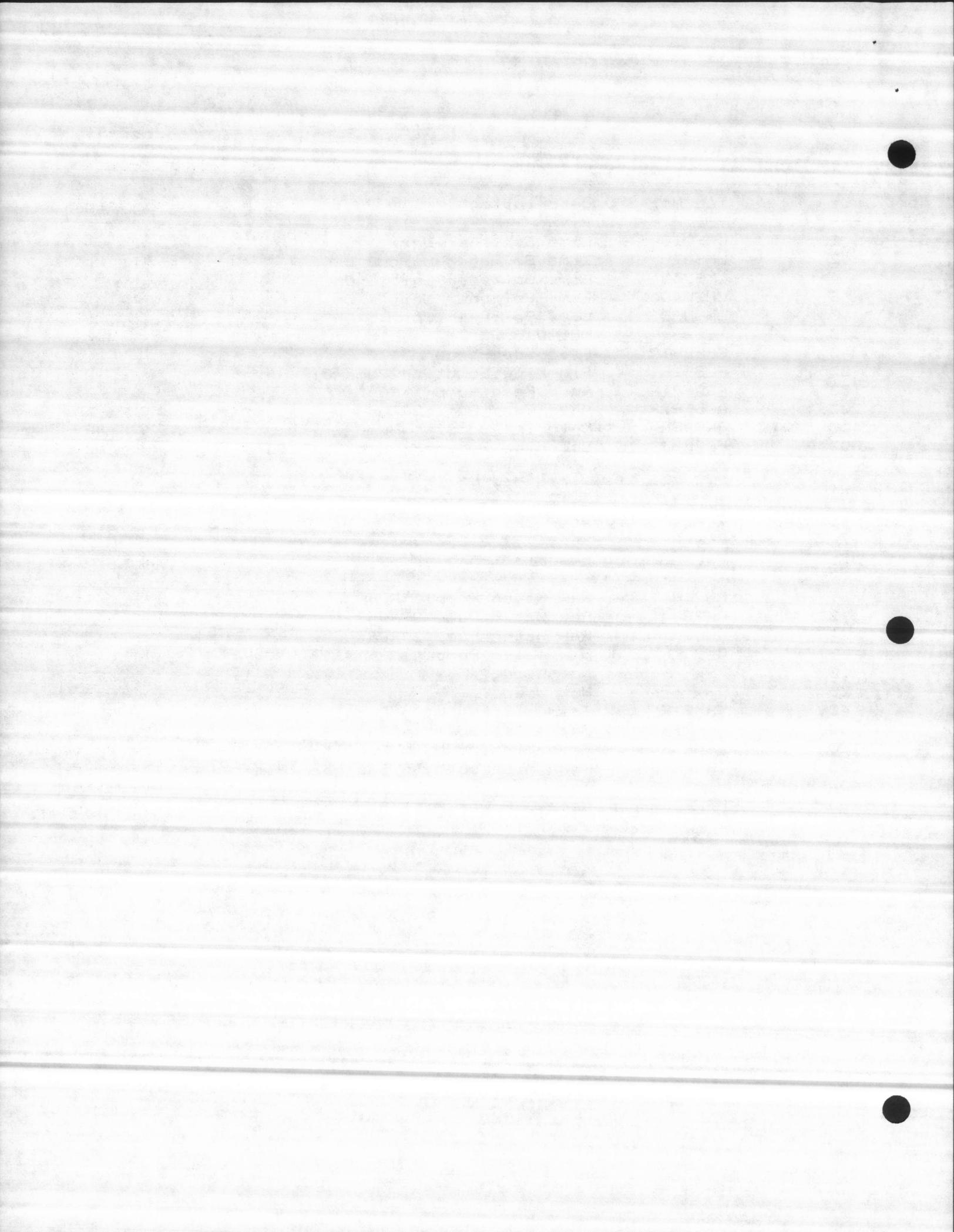


input rack addr 11 lcl\_discrete word #

port	bit	Tag No.	description
C4-P3	1	LMSU10	Lime Softener Unit 1 On
	2	LMSU20	Lime Softener Unit 2 On
	4	LMSU30	Lime Softener Unit 3 On
	8	LMSU40	Lime Softener Unit 4 On
	10	LMSU50	Lime Softener Unit 5 On
	20	-----	Not Used
	40	FLFDR1R	Fluoride Feeder 01 On
	80	FLSP1R	Fluoride Solution Pump 01 Run
	100	ACDFP1R	Acid Feed Pump 1 Run
	200	ACDFP2R	Acid Feed Pump 2 Run
	400	RWCBPR	Chlorine Booster Pump 1
	800	PEFLBPR	Chlorine Booster Pump 2 (Future)

input rack addr 12 lcl\_discrete word #

port	bit	Tag No.	description
CS-P1	1	RWP1R	Raw Water Pump 1 Run
	2	RWP2R	Raw Water Pump 2 Run
	4	RWP3R	Raw Water Pump 3 Run
	8	RWP4R	Raw Water Pump 4 Run
	10	-----	Not Used
	20	-----	Not Used
	40	TRMP1R	Transmission Main Pump 1 Run
	80	TRMP2R	Transmission Main Pump 2 Run
	100	TRMP3R	Transmission Main Pump 3 Run
	200	TRMBWPR	Transmission Main Backwash Pump Run
	400	-----	Not Used
	800	NPFGO	Normal Power Fail/Generator Operating

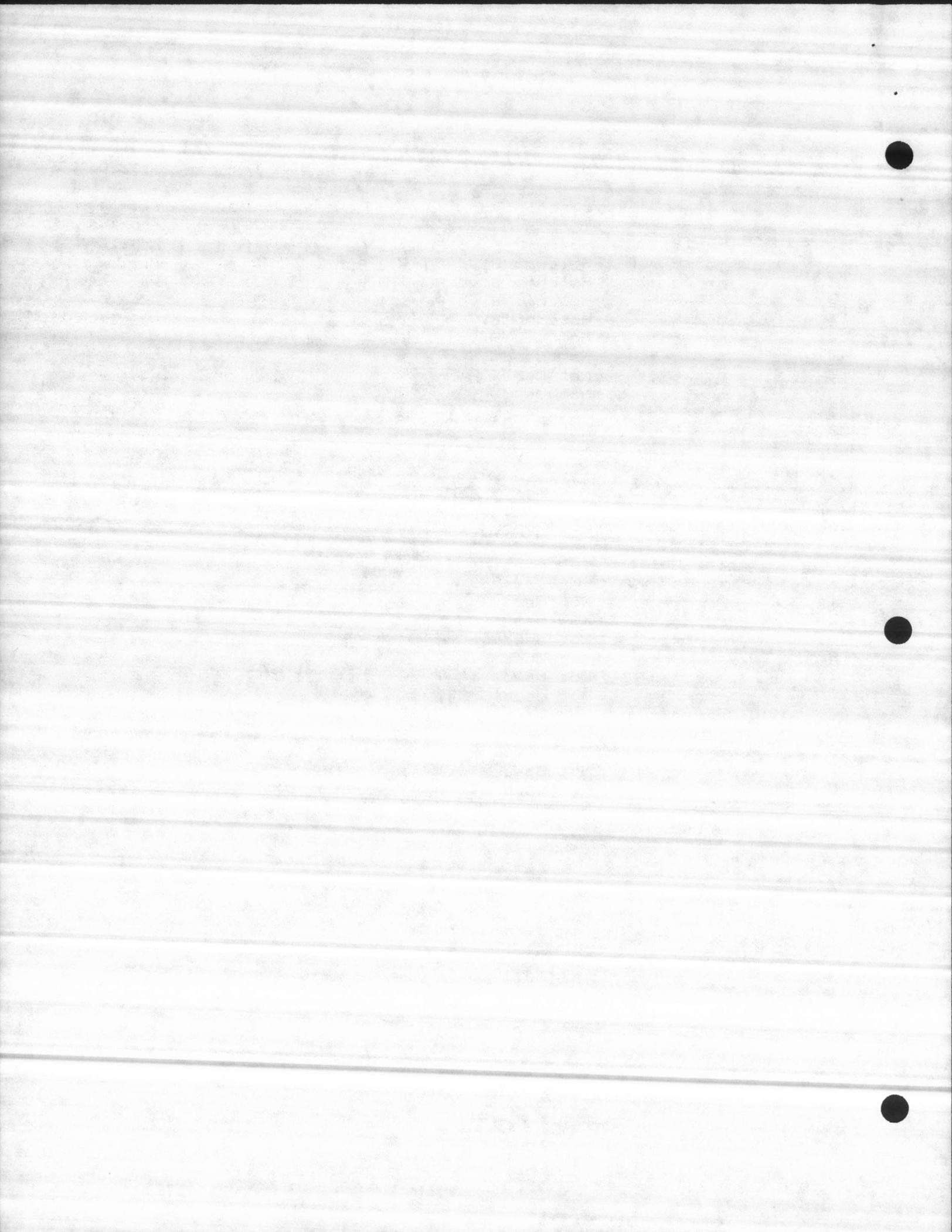


input rack addr 13 lcl\_discrete word #

port	bit	Tag No.	description
C5-P2	1	HBP1R	Holcomb Blvd Pump 1 Run
	2	HBP2R	Holcomb Blvd Pump 2 Run
	4	HBP3R	Holcomb Blvd Pump 3 Run
	8	HBP4R	Holcomb Blvd Pump 4 Run
	10	HBP5R	Holcomb Blvd Pump 5 Run
	20	HBKWPR	Holcomb Blvd Backwash Pump Run
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used

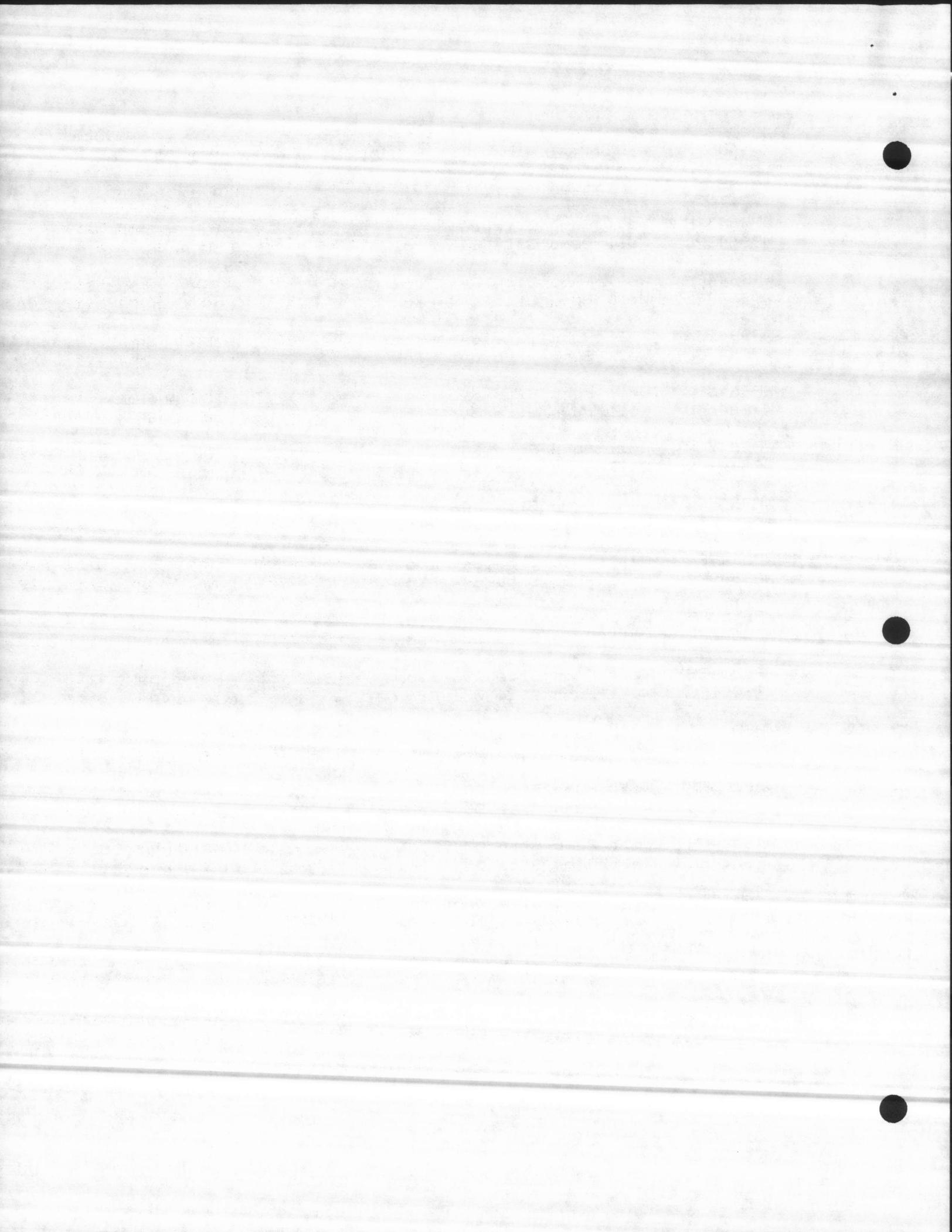
input rack addr 14 lcl\_discrete word #

port	bit	Tag No.	description
C5-P3	1	-----	Not Used
	2	-----	Not Used
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used

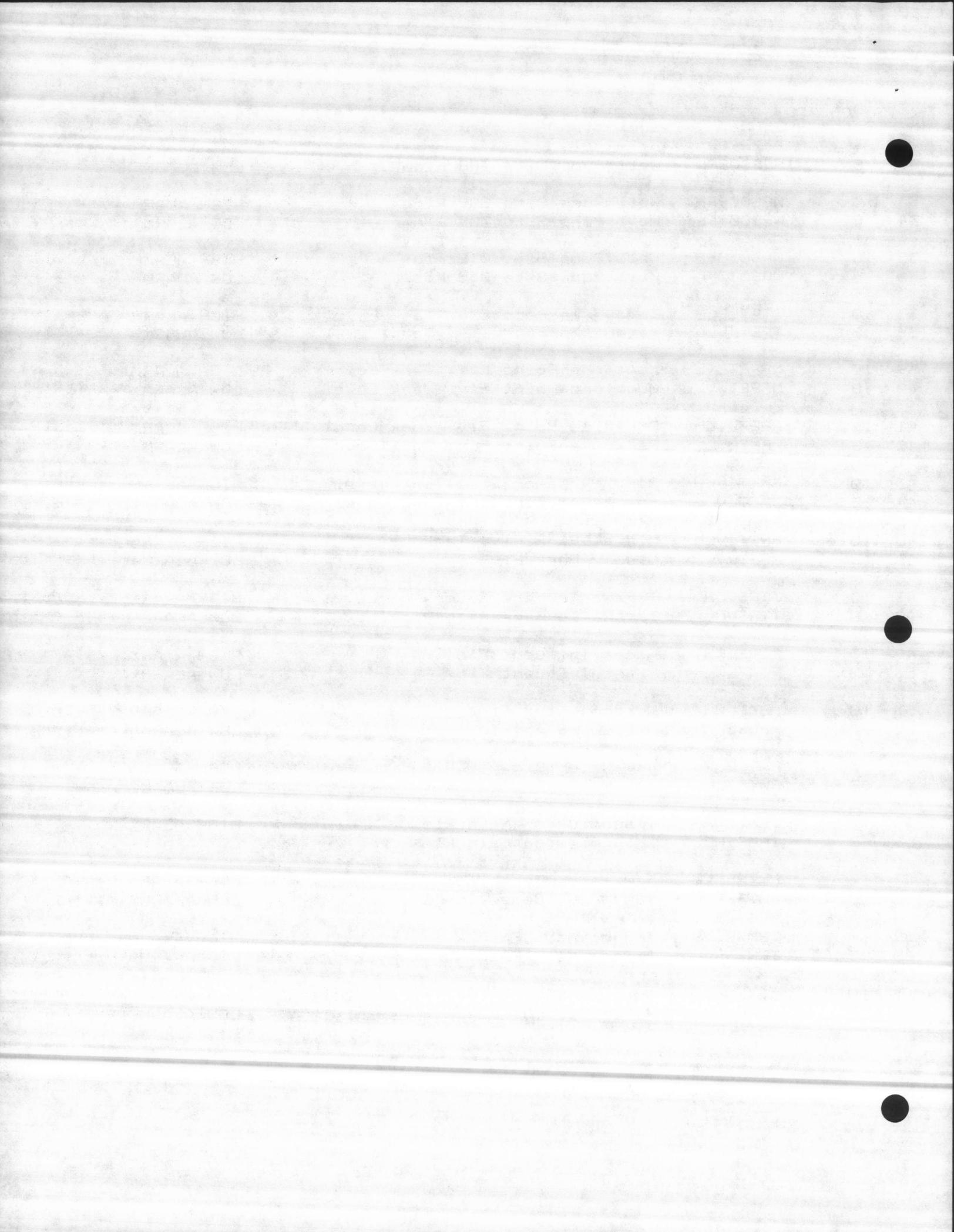


## A N A L O G   I N P U T   A L A R M S

indx	Bit #	Tag No.	Description	ALARM STATUS
01	10	WPS643FH	Well Pump Site 643 Flow	HIGH ALARM
02	8	WPS643FL	Well Pump Site 643 Flow	LOW ALARM
03	10	WPS644FH	Well Pump Site 644 Flow	HIGH ALARM
04	8	WPS644FL	Well Pump Site 644 Flow	LOW ALARM
05	10	WPS645FH	Well Pump Site 645 Flow	HIGH ALARM
06	8	WPS645FL	Well Pump Site 645 Flow	LOW ALARM
07	10	WPS646FH	Well Pump Site 646 Flow	HIGH ALARM
08	8	WPS646FL	Well Pump Site 646 Flow	LOW ALARM
09	10	WPS647FH	Well Pump Site 647 Flow	HIGH ALARM
10	8	WPS647FL	Well Pump Site 647 Flow	LOW ALARM
11	10	WPS648FH	Well Pump Site 648 Flow	HIGH ALARM
12	8	WPS648FL	Well Pump Site 648 Flow	LOW ALARM
13	10	WPS649FH	Well Pump Site 649 Flow	HIGH ALARM
14	8	WPS649FL	Well Pump Site 649 Flow	LOW ALARM
15	10	WPS650FH	Well Pump Site 650 Flow	HIGH ALARM
	8	WPS650FL	Well Pump Site 650 Flow	LOW ALARM
17	10	WPS001FH	Well Pump Site 001 Flow	HIGH ALARM
18	8	WPS001FL	Well Pump Site 001 Flow	LOW ALARM
19	10	WPS002FH	Well Pump Site 002 Flow	HIGH ALARM
20	8	WPS002FL	Well Pump Site 002 Flow	LOW ALARM
21	10	WPS003FH	Well Pump Site 003 Flow	HIGH ALARM
22	8	WPS003FL	Well Pump Site 003 Flow	LOW ALARM
23	10	WPS004FH	Well Pump Site 004 Flow	HIGH ALARM
24	8	WPS004FL	Well Pump Site 004 Flow	LOW ALARM
25	10	WPS005FH	Well Pump Site 005 Flow	HIGH ALARM
26	8	WPS005FL	Well Pump Site 005 Flow	LOW ALARM
27	10	WPS006FH	Well Pump Site 006 Flow	HIGH ALARM
28	8	WPS006FL	Well Pump Site 006 Flow	LOW ALARM
29	10	WPS007FH	Well Pump Site 007 Flow	HIGH ALARM
30	8	WPS007FL	Well Pump Site 007 Flow	LOW ALARM
31	10	WPS008FH	Well Pump Site 008 Flow	HIGH ALARM

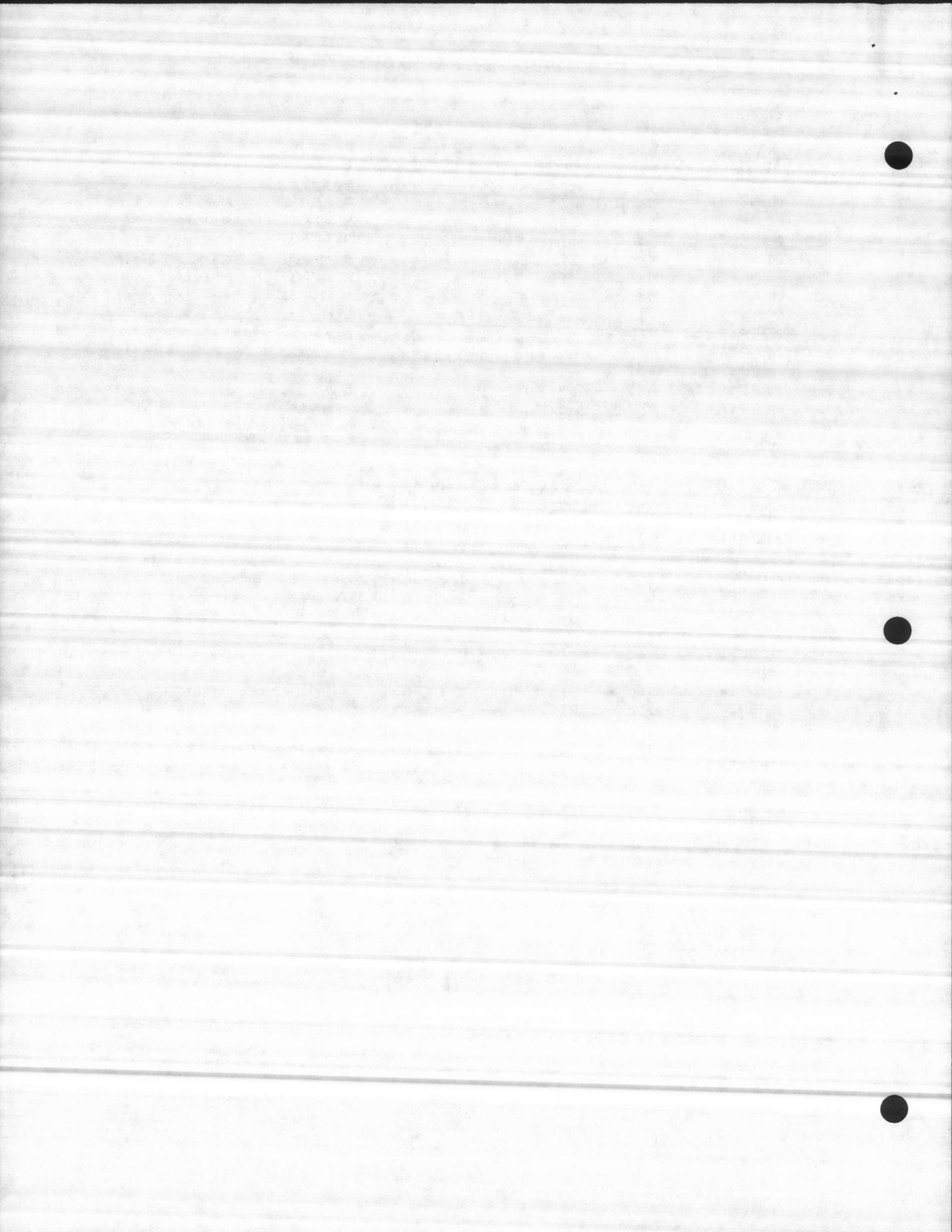


indx	Bit #	Tag No.	Description	ALARM STATUS
32	8	WPS008FL	Well Pump Site 008 Flow	LOW ALARM
33	10	WPS009FH	Well Pump Site 009 Flow	HIGH ALARM
34	8	WPS009FL	Well Pump Site 009 Flow	LOW ALARM
35	10	WPS010FH	Well Pump Site 010 Flow	HIGH ALARM
36	8	WPS010FL	Well Pump Site 010 Flow	LOW ALARM
37	10	TTETLVLH	Tarawa Terrace Elv. Tank Level	HIGH ALARM
38	8	TTETLVLLL	Tarawa Terrace Elv. Tank Level	LOW ALARM
39	10	MFPETLVLH	Montford Point Elv. Tank Level	HIGH ALARM
40	8	MFPETLVLLL	Montford Point Elv. Tank Level	LOW ALARM
41	10	PPETLVLH	Paradise Point Elv. Tank Level	HIGH ALARM
42	8	PPETLVLLL	Paradise Point Elv. Tank Level	LOW ALARM
43	10	BMETLVLH	Berkley Manor Elv. Tank Level	HIGH ALARM
44	8	BMETLVLLL	Berkley Manor Elv. Tank Level	LOW ALARM
45	10	MPETLVLH	Midway Park Elv. Tank Level	HIGH ALARM
46	8	MPETLVLLL	Midway Park Elv. Tank Level	LOW ALARM
47	10	TTRSVLH	Tarawa Terrace Reservoir Level	HIGH ALARM
48	8	TTRSVLL	Tarawa Terrace Reservoir Level	LOW ALARM
49	10	PINFLWH	Plant Influent Flow	HIGH ALARM
50	8	PINFLWL	Plant Influent Flow	LOW ALARM
51	10	HBDFLWH	Holcomb Blvd Flow	HIGH ALARM
52	8	HBDFLWL	Holcomb Blvd Flow	LOW ALARM
53	10	RWTRFLWH	Raw Water Flow	HIGH ALARM
54	8	RWTRFLWL	Raw Water Flow	LOW ALARM
55	10	TRMFLWH	Transmission Main Flow	HIGH ALARM
56	8	TRMFLWL	Transmission Main Flow	LOW ALARM
57	10	RWTRESVH	Raw Water Reservoir Level	HIGH ALARM
58	8	RWTRESVL	Raw Water Reservoir Level	LOW ALARM
59	10	TRMWRSVH	Trans. Main Fnsh Wtr Rsvr Lvl	HIGH ALARM
60	8	TRMWRSVL	Trans. Main Fnsh Wtr Rsvr Lvl	LOW ALARM
61	10	HBFWRLH	Holcomb Blvd Fnsh Wtr Rsvr Lvl	HIGH ALARM
62	8	HBFWRLL	Holcomb Blvd Fnsh Wtr Rsvr Lvl	LOW ALARM



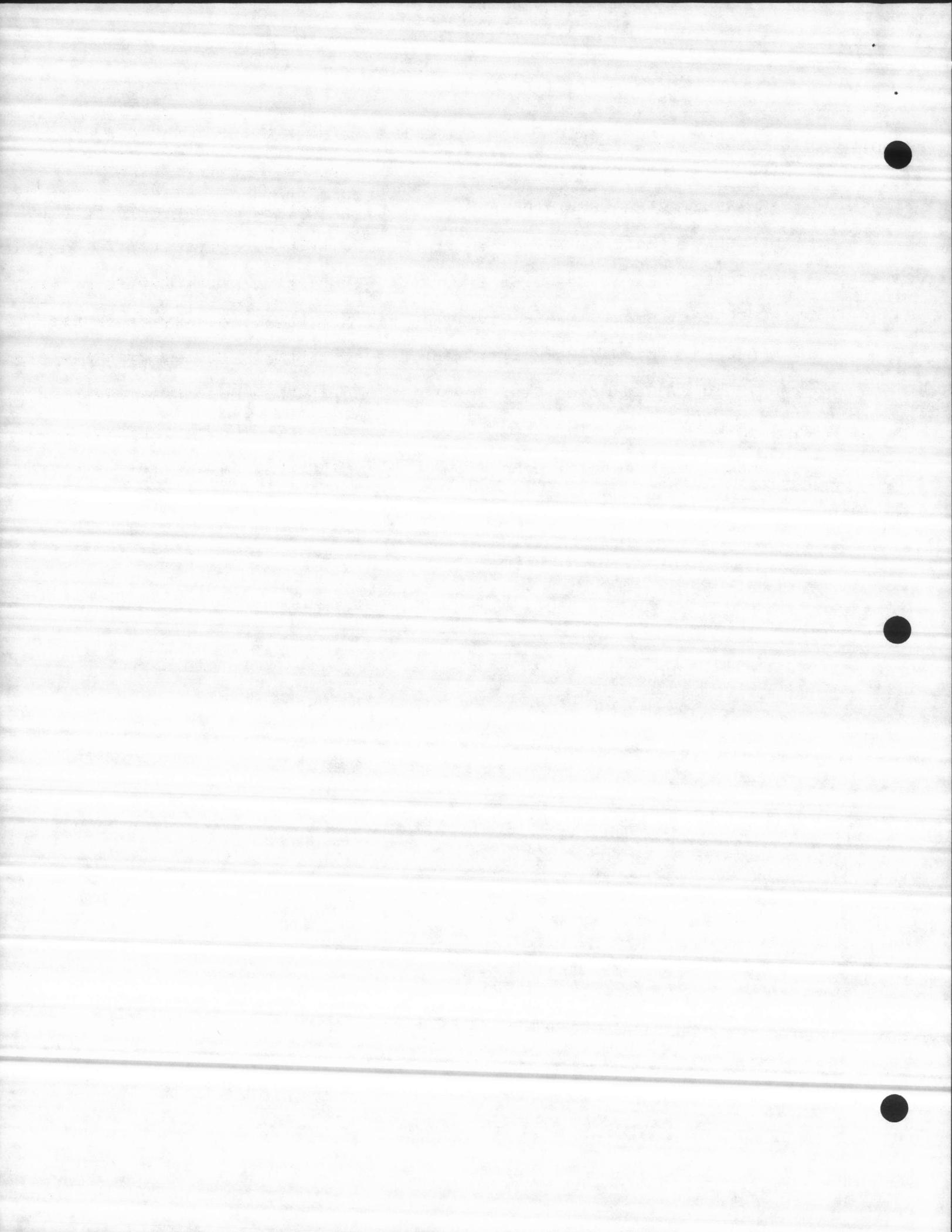
## MANUAL FAIL SIGNALS

Ind.	Tag No.	Description
01	MLMFP1F	Manual Lime Feed Pump 1
02	MLMFP2F	Manual Lime Feed Pump 2
03	MLMFP3F	Manual Lime Feed Pump 3
04	MLMFP4F	Manual Lime Feed Pump 4
05	MLMFP5F	Manual Lime Feed Pump 5
06	MLMFP6F	Manual Lime Feed Pump 6
07	MFFP1F	Fluoride Feed Pump 1 Manu
08	MFFP2F	Fluoride Feed Pump 2 Manu
09	MAFP1F	Acid Feed Pump 1 Manual
10	MAFP1F	Acid Feed Pump 2 Manual
11	MPEFP1F	Chlorine Booster P2 Manu
12	MRWP1F	Raw Water Pump 1 Manual
13	MRWP2F	Raw Water Pump 2 Manual
14	MRWP3F	Raw Water Pump 3 Manual
15	MRWP4F	Raw Water Pump 4 Manual
16	MTMP1F	Trans. Main Pump 1 Manual
17	MTMP2F	Trans. Main Pump 2 Manual
18	MTMP3F	Trans. Main Pump 3 Manual
19	MTMBPF	Trans. Main BkWsh P Manual
20	MHBP1F	Holcomb Blvd Pump 1 Manual
21	MHBP2F	Holcomb Blvd Pump 2 Manual
22	MHBP3F	Holcomb Blvd Pump 3 Manual
23	MHBP4F	Holcomb Blvd Pump 4 Manual
24	MHBP5F	Holcomb Blvd Pump 5 Manual
25	MHBPF	Holcomb Blvd BKW P Manual



## M O R E A N A L O G A L A R M S (low low)

index	bit #	Tag No.	Description	
01		TRMWRLLL	Trans Main Fnsh Wtr Res Lvl	[Low Low Alarm]
02		HBFWRLLL	Holcomb Blvd Fnsh Wtr Res Lvl	[Low Low Alarm]
03		PPETLLL	Paradise Point Elv. Tank Lvl	[Low Low Alarm]
04		BMETLLL	Berkly Manor Elv. Tank Lvl	[Low Low Alarm]
05		MPETLLL	Midway Park Elv. Tank Lvl	[Low Low Alarm]
06		RWTRLLE	Raw Water Res. Level	[Low Low Alarm]
07		TTRSVLLL	Taraw Terrace Level	[Low Low Alarm]
08		PINFLLL	Plant Influent Flow	[Low Low Alarm]

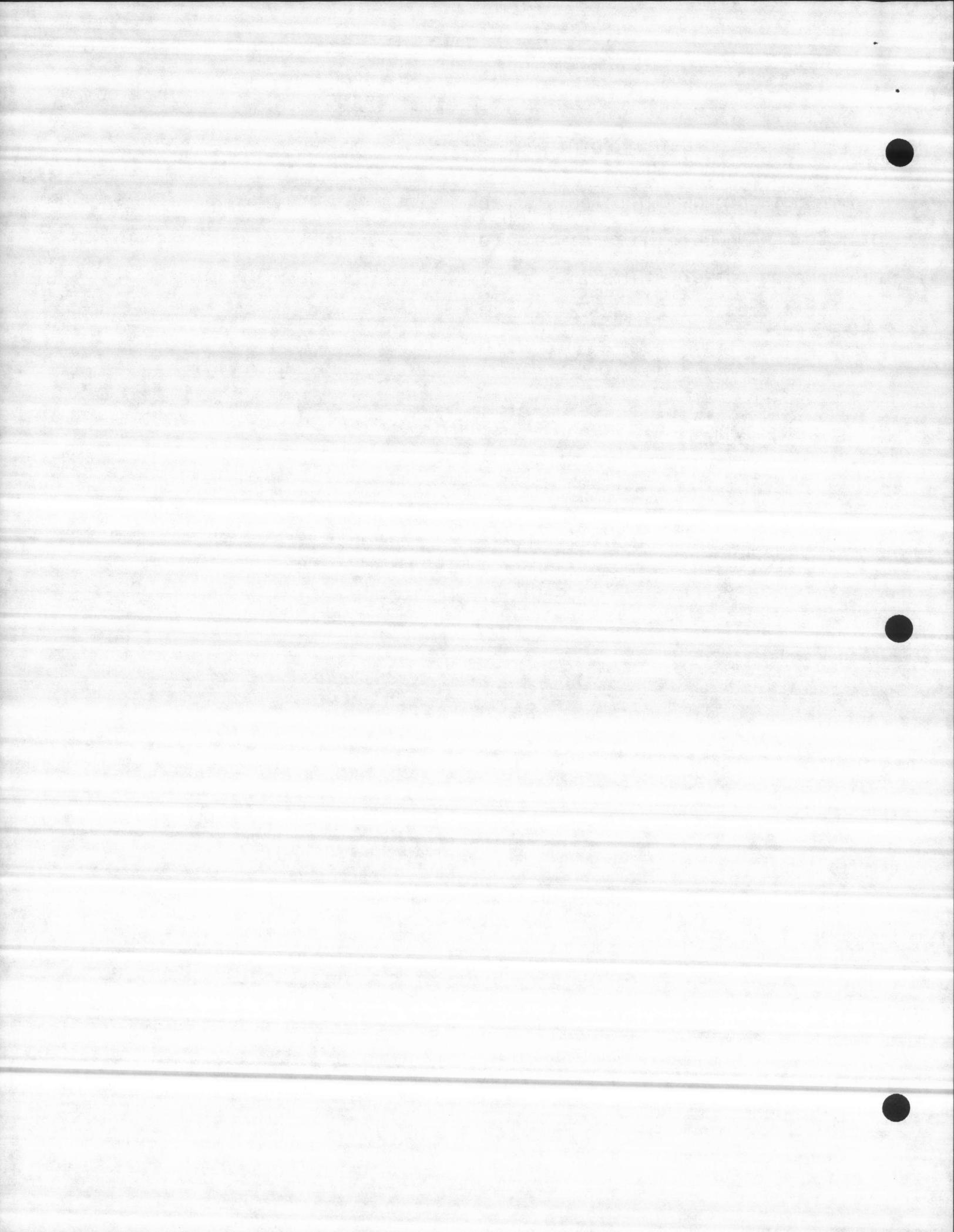


APPENDIX - C : ANALOG OUTPUT POINTS :

\*\* TOTAL REMOTE ANALOG OUTPUT POINTS = 0

\*\* TOTAL LOCAL ANALOG OUTPUT POINTS = 0

\*\* TOTAL ANALOG OUTPUT POINTS = 1



APPENDIX - D : DISCRETE OUTPUT POINTS :

\*\* TOTAL REMOTE DISCRETE OUTPUT POINTS = 30

\*\* TOTAL LOCAL DISCRETE OUTPUT POINTS = 90

\*\* TOTAL DISCRETE OUTPUT POINTS = 120

REMOTE DISCRETE OUTPUT POINTS

Remote 01 - Well Site No. 643

Transmit Word 1

Rmt#	bit	Tag No.	description
------	-----	---------	-------------

01	1	-----	Not Used
----	---	-------	----------

.	.	800	STWP43	Start Well Pump 643
---	---	-----	--------	---------------------

Remote 02 - Well Site No. 644

Transmit Word 1

Rmt#	bit	Tag No.	description
------	-----	---------	-------------

02	1	-----	Not Used
----	---	-------	----------

.	.	800	STWP44	Start Well Pump 644
---	---	-----	--------	---------------------

Remote 03 - Well Site No. 645

Transmit Word 1

Rmt#	bit	Tag No.	description
------	-----	---------	-------------

03	1	-----	Not Used
----	---	-------	----------

.	.	800	STWP45	Start Well Pump 645
---	---	-----	--------	---------------------

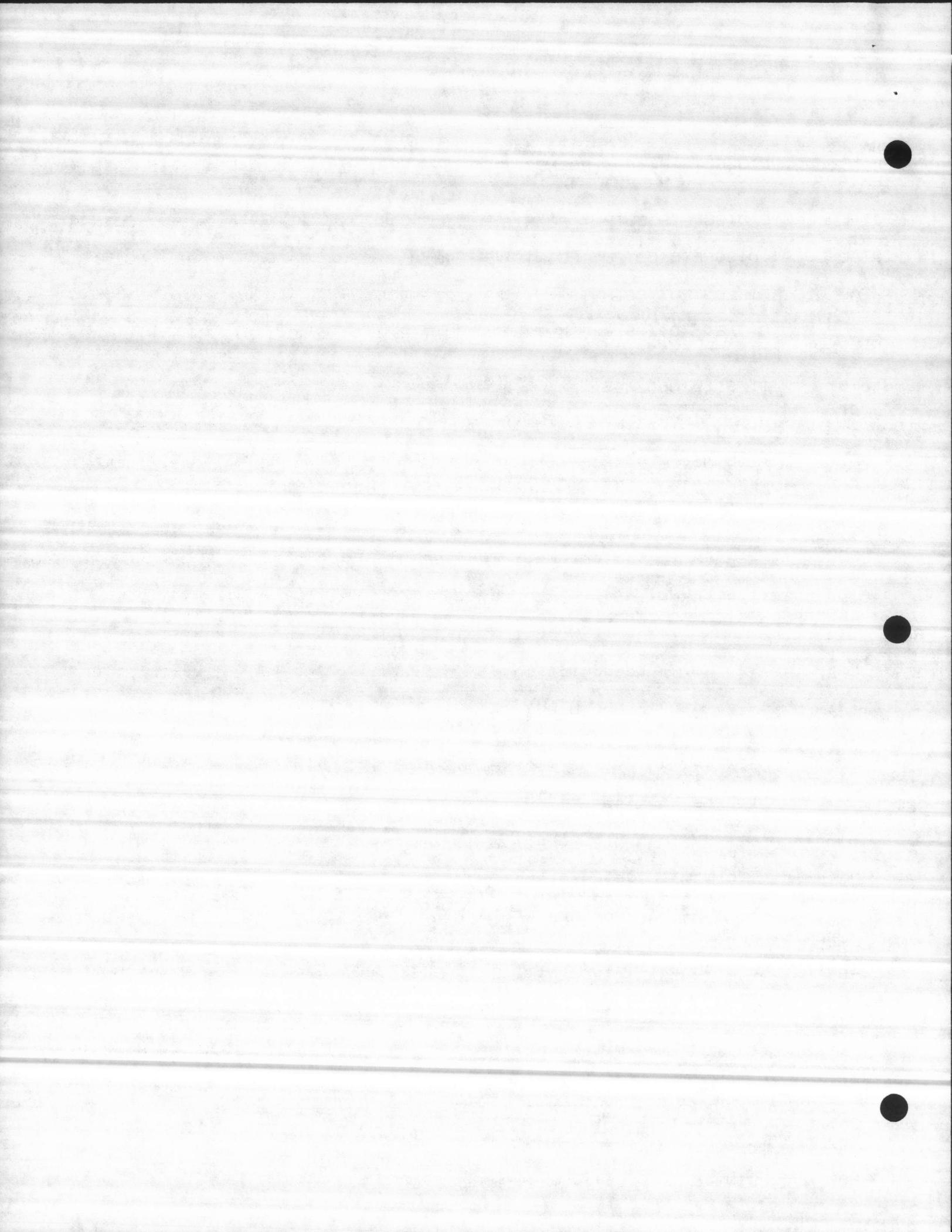
Remote 04 - Well Site No. 646

Transmit Word 1

Rmt#	bit	Tag No.	description
------	-----	---------	-------------

04	1	-----	Not Used
----	---	-------	----------

.	.	800	STWP46	Start Well Pump 646
---	---	-----	--------	---------------------



Remote 05 - Well Site No. 647

Transmit Word 1

Rmt#	bit	Tag No.	description
04	1	-----	Not Used
.	.		
800		STWP47	Start Well Pump 647

Remote 06 - Well Site No. 648

Transmit Word 1

Rmt#	bit	Tag No.	description
06	1	-----	Not Used
.	.		
800		STWP48	Start Well Pump 648

Remote 07 - Well Site No. 649

Transmit Word 1

Rmt#	bit	Tag No.	description
07	1	-----	Not Used
.	.		
800		STWP49	Start Well Pump 649

Remote 08 - Well Site No. 650

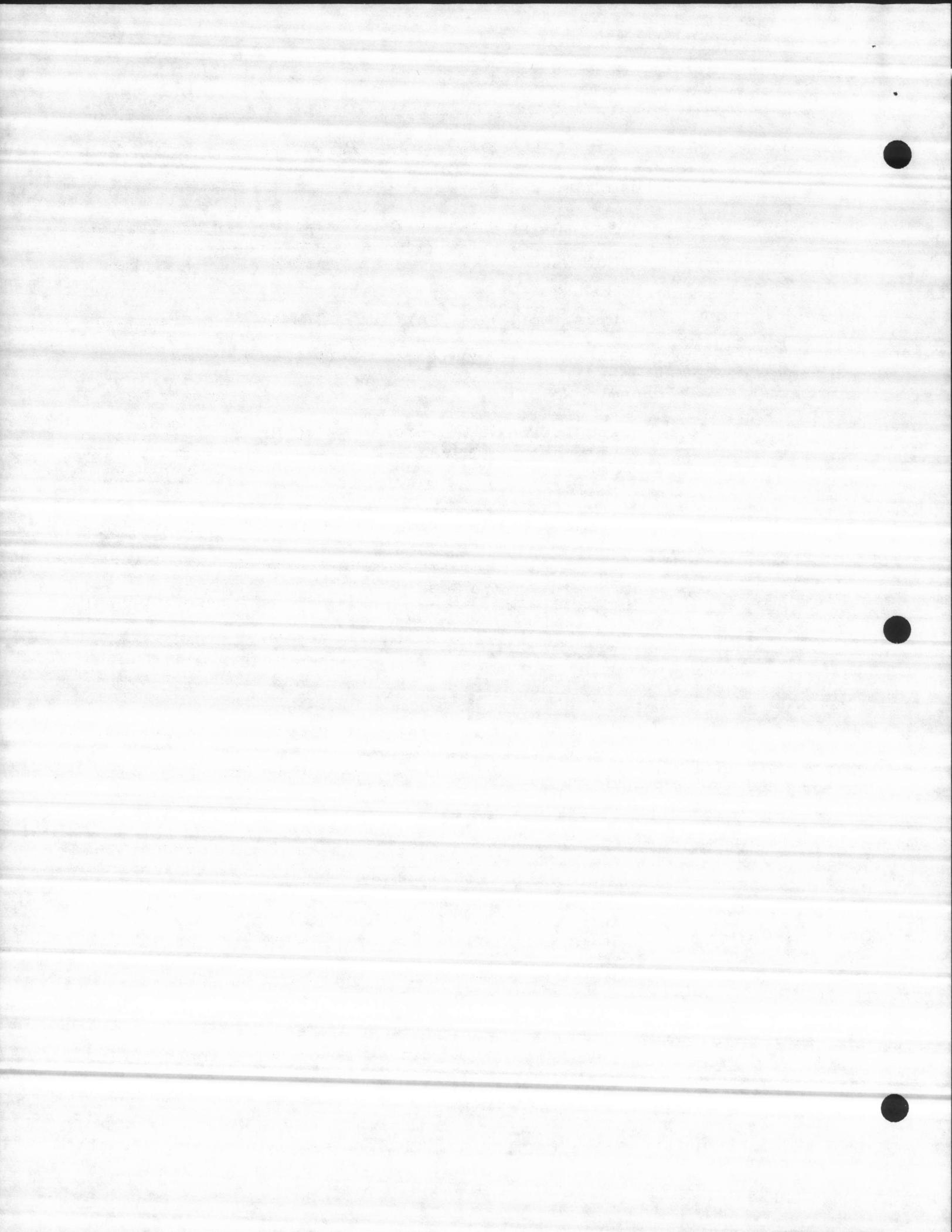
Transmit Word 1

Rmt#	bit	Tag No.	description
08	1	-----	Not Used
.	.		
800		STWP50	Start Well Pump 650

Remote 09 - Well Site No. 001

Transmit Word 1

Rmt#	bit	Tag No.	description
09	1	-----	Not Used
.	.		
800		STWP01	Start Well Pump 001



Remote 10 - Well Site No. 002

Transmit Word 1

Rmt#	bit	Tag No.	description
10	1		Not Used
.	.		
800		STWP02	Start Well Pump 002

Remote 11 - Well Site No. 003

Transmit Word 1

Rmt#	bit	Tag No.	description
11	1		Not Used
.	.		
800		STWP03	Start Well Pump 003

Remote 12 - Well Site No. 004

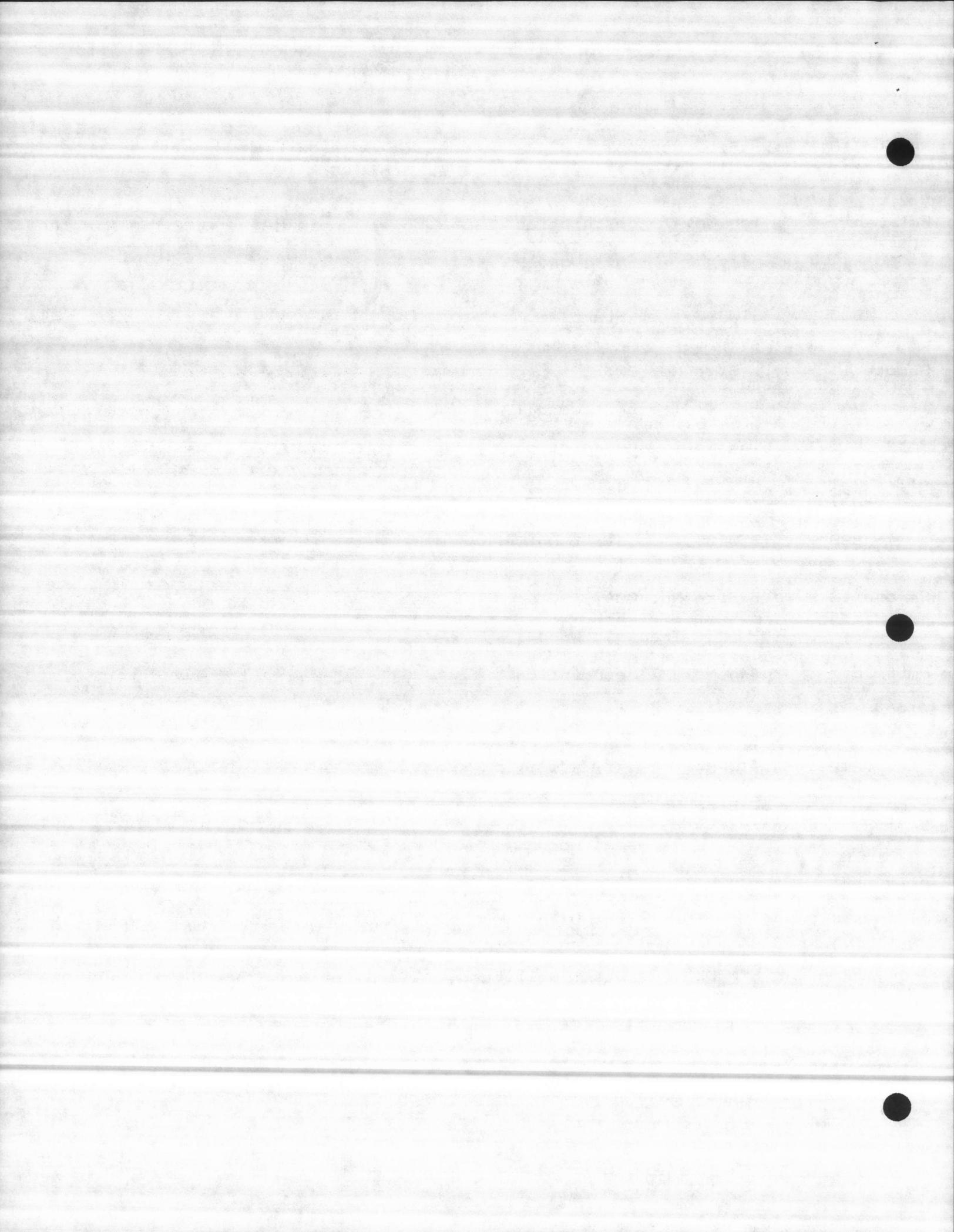
Transmit Word 1

Rmt#	bit	Tag No.	description
12	1		Not Used
.	.		
800		STWP04	Start Well Pump 004

Remote 13 - Well Site No. 005

Transmit Word 1

Rmt#	bit	Tag No.	description
13	1		Not Used
.	.		
800		STWP05	Start Well Pump 005



Remote 14 - Well Site No. 006

Transmit Word 1

Rmt#	bit	Tag No.	description
14	1	-----	Not Used
.	.		
800		STWP06	Start Well Pump 006

Remote 15 - Well Site No. 007

Transmit Word 1

Rmt#	bit	Tag No.	description
15	1	-----	Not Used
.	.		
800		STWP07	Start Well Pump 007

Remote 16 - Well Site No. 008

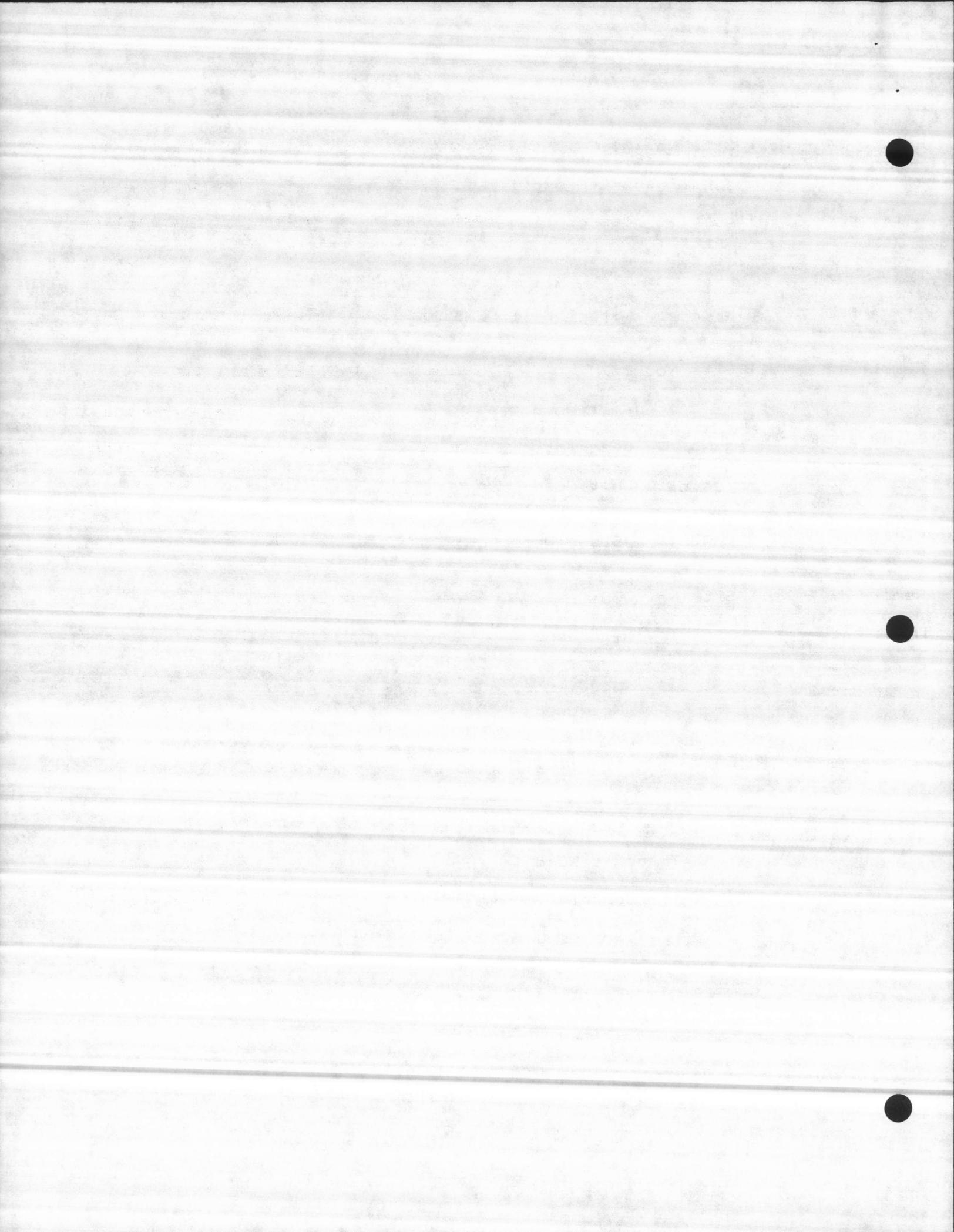
Transmit Word 1

Rmt#	bit	Tag No.	description
16	1	-----	Not Used
.	.		
800		STWP08	Start Well Pump 008

Remote 17 - Well Site No. 009

Transmit Word 1

Rmt#	bit	Tag No.	description
17	1	-----	Not Used
.	.		
800		STWP09	Start Well Pump 009



Remote 18 - Well Site No. 010

Transmit Word 1

Rmt#	bit	Tag No.	description
18	1		Not Used
800		STWP10	Start Well Pump 010

Remote 24 - Tarawa Terrace Reservoir & Pump Station

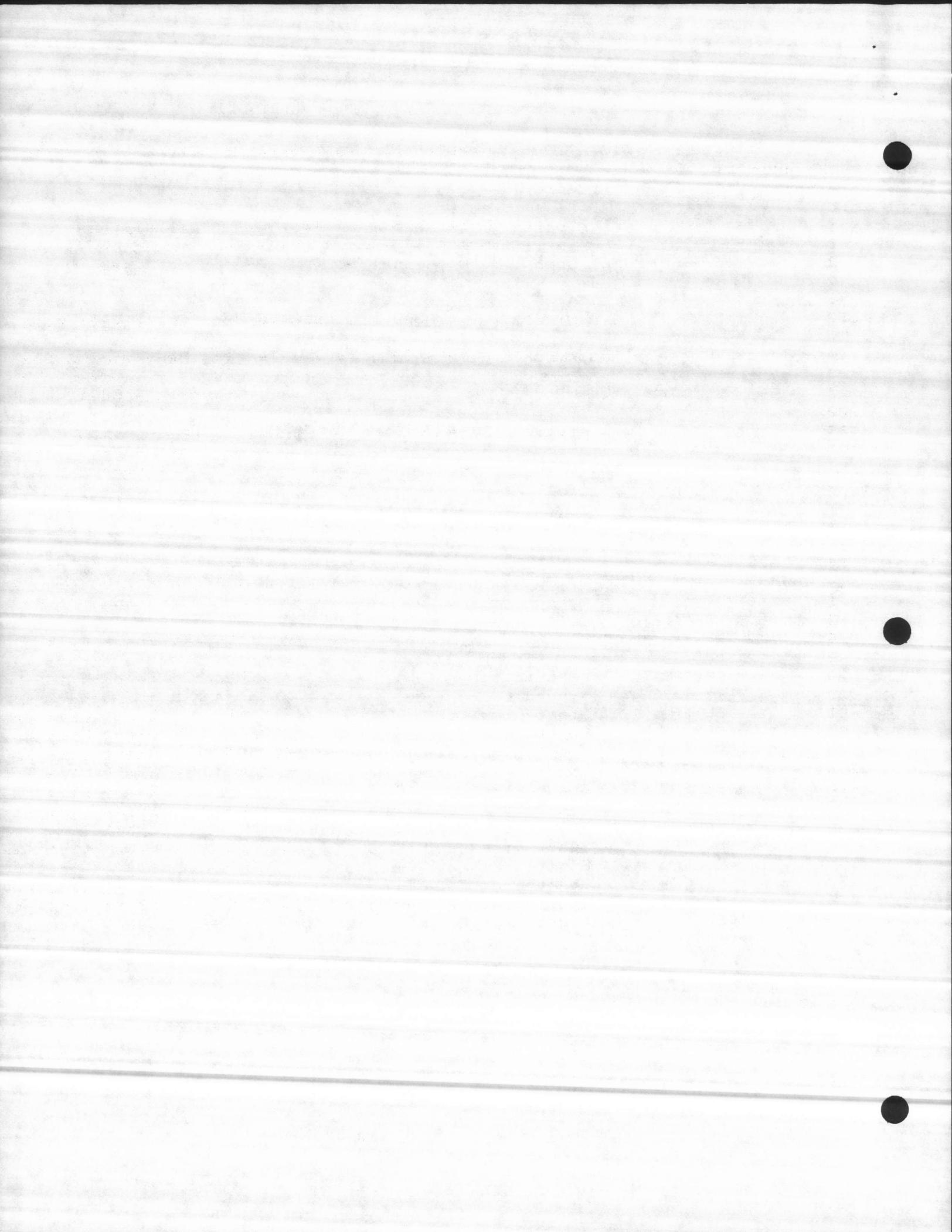
Transmit Word 1

Rmt#	bit	Tag No.	description
24	1		Not Used
800		BPLLCUT	Booster Pump Low Level Cutout

L O C A L   D I S C R E T E   O U T P U T   P O I N T S

in[ ]t rack addr 15 lcl\_discrete word #

port	bit	Tag No.	description
C6-P1	1	SLMFR1	Start Lime Feeder 01
	2	SLMFR2	Start Lime Feeder 02
	4	SLMFR3	Start Lime Feeder 03
	8	SLMFR4	Start Lime Feeder 04
	10	SLMFR5	Start Lime Feeder 05
	20	SLMFR6	Start Lime Feeder 06
	40	SLMFP1	Start lime Feed Pump 01
	80	SLMFP2	Start Lime Feed Pump 02
	100	SLMFP3	Start Lime Feed Pump 03
	200	SLMFP4	Start Lime Feed Pump 04
	400	SLMFP5	Start Lime Feed Pump 05
	800	SLMFP6	Start Lime Feed Pump 06



---

```
input rack addr 16 lcl_discrete word #
```

---

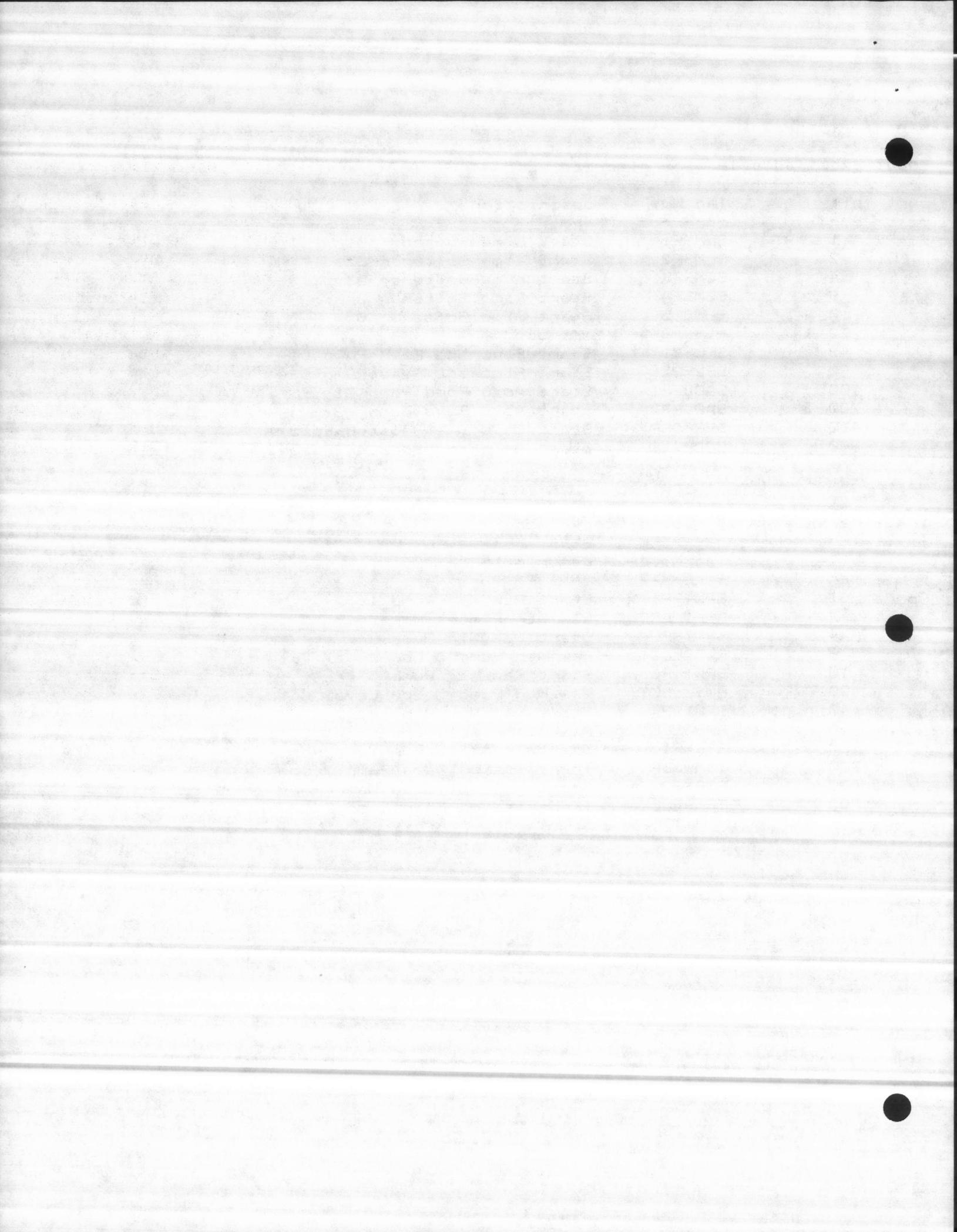
port	bit	Tag No.	description
C6-P2	1	SLMSF1	Start Lime Softener 01
	2	SLMSF2	Start Lime Softener 02
	4	SLMSF3	Start Lime Softener 03
	8	SLMSF4	Start Lime Softener 04
	10	SLMSF5	Start Lime Softener 05
	20	-----	Not Used
	40	SFLFP1	Start Fluoride Feed Pump 01
	80	SFLFP2	Start Fluoride Feed Pump 02
	100	SACDFP1	Start Acid Feed Pump 01
	200	SACDFP2	Start Acid Feed Pump 02
	400	SPEFCBP	Start Chlorine&Booster Pump 1
	800	SCBP2	Start Chlorine&Booster Pump 2

---

```
input rack addr 17 lcl_discrete word #
```

---

port	bit	Tag No.	description
C6-P3	1	SRWTRP1	Start Raw Wtr Pump 1
	2	RWP1LLCT	Raw Wtr Pump 1 Low Level Cutout
	4	SRWTRP2	Start Raw Wtr Pump 2
	8	RWP2LLCT	Raw Wtr Pump 2 Low Level Cutout
	10	SRWTRP3	Start Raw Wtr Pump 3
	20	RWP3LLCT	Raw Wtr Pump 3 Low Level Cutout
	40	SRWTRP4	Start Raw Wtr Pump 4
	80	RWP4LLCT	Raw Wtr Pump 4 Low Level Cutout
	100	STRMP1	Start Transmission Main Pump 1
	200	TRMP1LCT	Transmission Main Pump 1 Low Level Cutout
	400	STRMP2	Start Transmission Main Pump 2
	800	TRMP2LCT	Transmission Main Pump 2 Low Level Cutout



---

```
input rack addr 18 lcl_discrete word #
```

---

port	bit	Tag No.	description
C7-P1	1	STRMP3	Start Transmission Main Pump 3
	2	TRMP3LLCT	Transmission Main Pump 3 Low Level Cutout
	4	ETMBP	Enable Transmission Blvd. Backwash Pump
	8	TRMBWLCT	Transmission Main Backwash Pump Low Lvl Cut
	10	SHBP1	Start Holcomb Blvd Pump 1
	20	HBP1LLCT	Holcomb Blvd Pump 1 Low Level Cutout
	40	SHBP2	Start Holcomb Blvd Pump 2
	80	HBP2LLCT	Holcomb Blvd Pump 2 Low Level Cutout
	100	SHBP3	Start Holcomb Blvd Pump 3
	200	HBP3LLCT	Holcomb Blvd Pump 3 Low Level Cutout
	400	SHBP4	Start Holcomb Blvd Pump 4
	800	HBP4LLCT	Holcomb Blvd Pump 4 Low Level Cutout

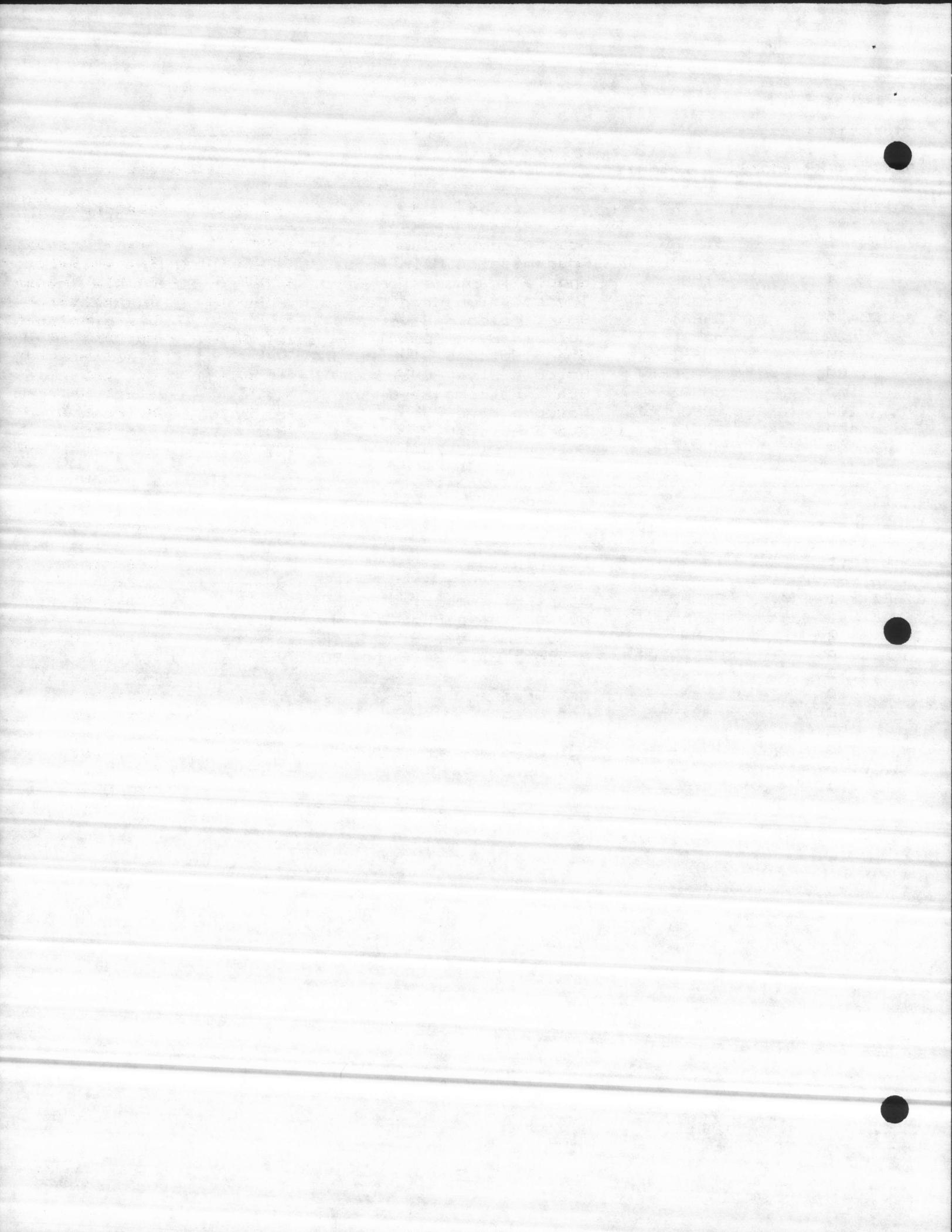
---

```
input rack addr 19 lcl_discrete word #
```

---

port	bit	Tag No.	description
C7-P2	1	SHBP5	Start Holcomb Blvd Pump 5
	2	HBP5LLCT	Holcomb Blvd Pump 5 Low Level Cutout
	4	EHMBP	Enable Holcomb Blvd Backwash Pump
	8	HBBWPLCT	Holcomb Blvd Backwash Pump Low Level Cutout
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used

---

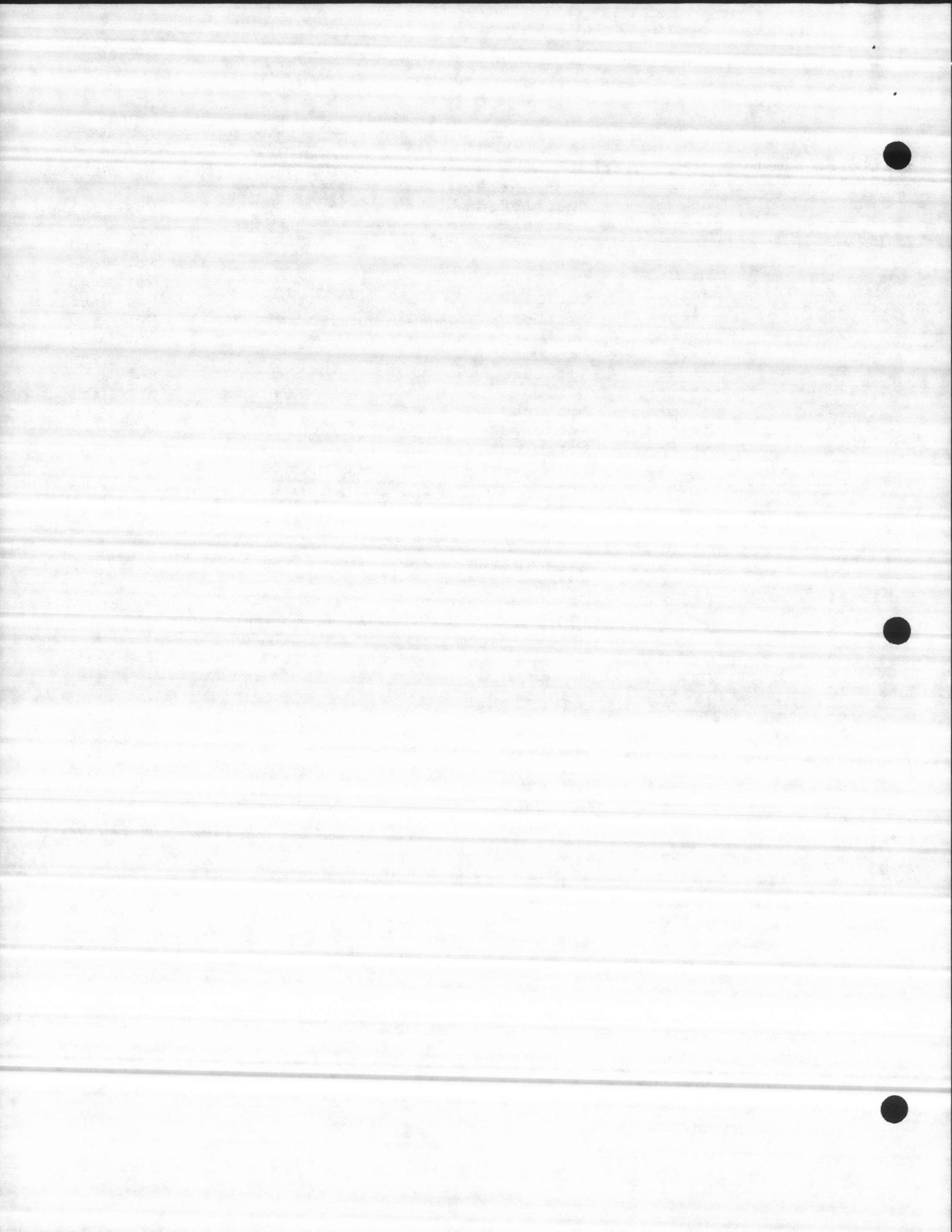


input rack addr 20 lcl\_discrete word #

port	bit	Tag No.	description
C7-P3	1	-----	Not Used
	2	-----	Not Used
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used

input rack addr 21 lcl\_discrete word #

port	bit	Tag No.	description
C8-P1	1	COPERAT	Computer Operating (2 sec On - 2 sec Off continuous Pulsing)
	2	COMALRM	Common Alarm
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used



---

```
input rack addr 22 lcl_discrete word #
```

---

port	bit	Tag No.	description
C8-P2	1	-----	Not Used
	2	-----	Not Used
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used

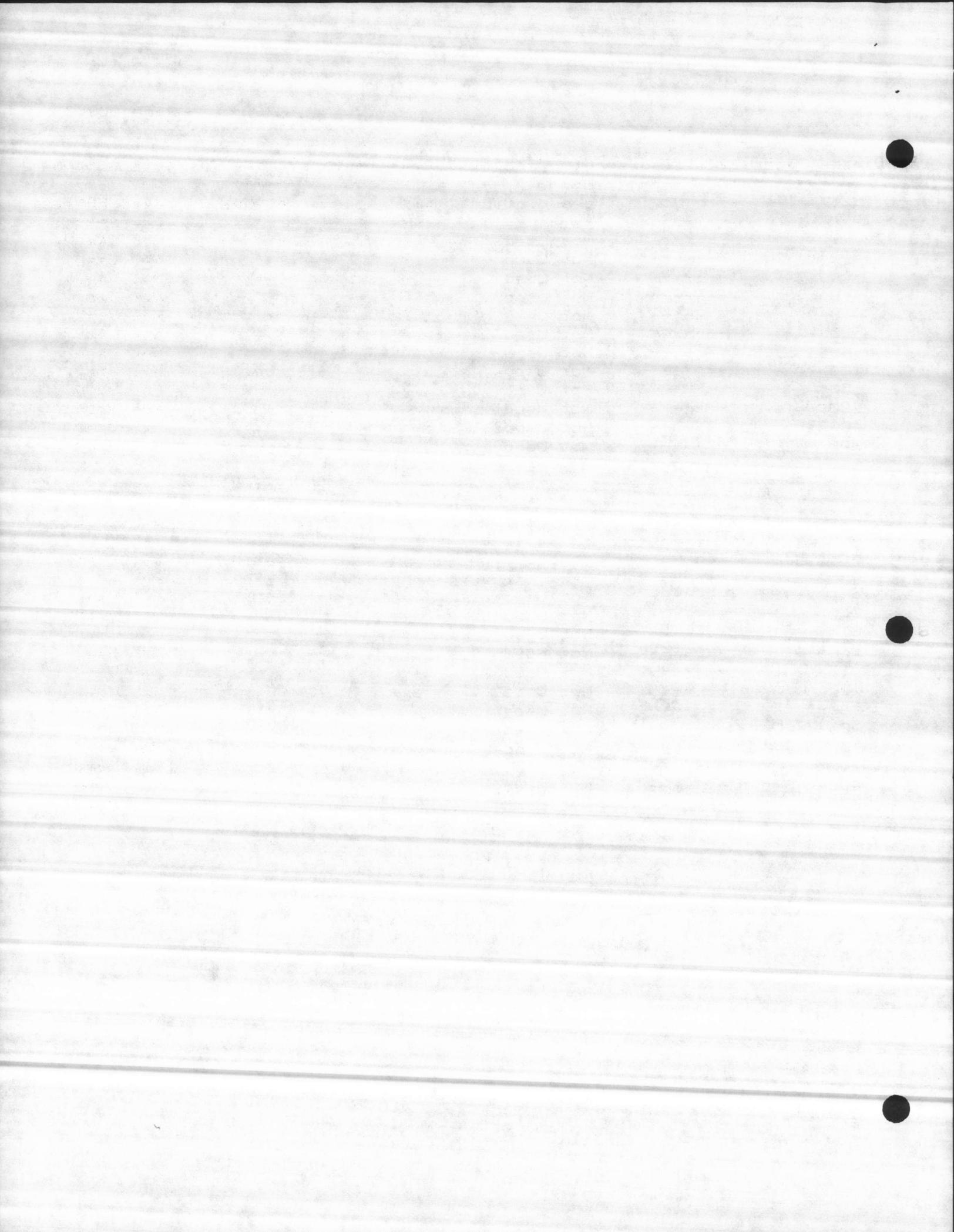
---

```
input rack addr 23 lcl_discrete word #
```

---

port	bit	Tag No.	description
C8-P3	1	-----	Not Used
	2	-----	Not Used
	4	-----	Not Used
	8	-----	Not Used
	10	-----	Not Used
	20	-----	Not Used
	40	-----	Not Used
	80	-----	Not Used
	100	-----	Not Used
	200	-----	Not Used
	400	-----	Not Used
	800	-----	Not Used

---



## APPENDIX - E : POINT CLASSIFICATIONS:

\*\* Number Of Flows = 22

- 18 (R) Well Pump Flows
- 1 (L) Plant Effluent Flow
- 1 (L) Holcomb Blvd. Flow
- 1 (L) Raw Water Flow
- 1 (L) Transmission Main Flow

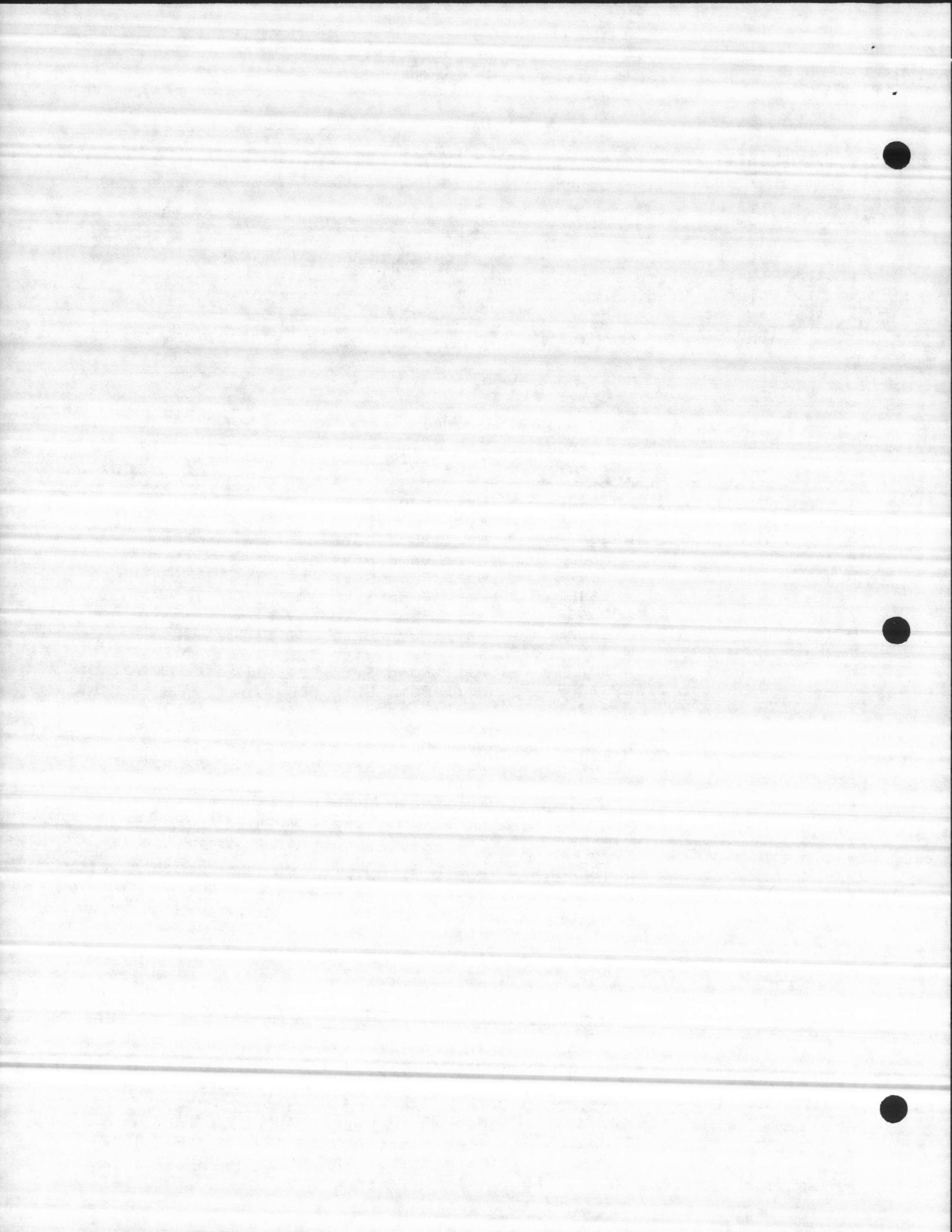
## F L O W S

anlg indx	flow indx	R/L *** or adr	RMT #	Wd	Tag No.	Description
1	1	R	1	2	WPS643F	Well Pump Site 643 Flow
2	2	R	2	2	WPS644F	Well Pump Site 644 Flow
3	3	R	3	2	WPS645F	Well Pump Site 645 Flow
4	4	R	4	2	WPS646F	Well Pump Site 646 Flow
5	5	R	5	2	WPS647F	Well Pump Site 647 Flow
6	6	R	6	2	WPS648F	Well Pump Site 648 Flow
7	7	R	7	2	WPS649F	Well Pump Site 649 Flow
8	8	R	8	2	WPS650F	Well Pump Site 650 Flow
9	9	R	9	2	WPS001F	Well Pump Site 001 Flow
10	10	R	10	2	WPS002F	Well Pump Site 002 Flow
11	11	R	11	2	WPS003F	Well Pump Site 003 Flow
12	12	R	12	2	WPS004F	Well Pump Site 004 Flow
13	13	R	13	2	WPS005F	Well Pump Site 005 Flow
14	14	R	14	2	WPS006F	Well Pump Site 006 Flow
15	15	R	15	2	WPS007F	Well Pump Site 007 Flow
16	16	R	16	2	WPS008F	Well Pump Site 008 Flow
17	17	R	17	2	WPS009F	Well Pump Site 009 Flow
18	18	R	18	2	WPS010F	Well Pump Site 010 Flow
19	19	L	00		PINFLW	Plant Influent Flow
20	20	L	01		HBDFLW	Holcomb Blvd Flow
21	21	L	02		RWTRFLW	Raw Water Flow
22	22	L	03		TRMFLW	Transmission Main Flow

\*\* Number Of Reservoirs = 4

## R E S E R V O I R S

anlg indx	resv indx	R/L *** or adr	RMT #	Wd	Tag No.	Description
23	1	R	24	2	TTRSVL	Trawa Terrace Reservoir Level
24	2	L	04		RWTRESV	Raw Water Reservoir Level
25	3	L	05		TRMWRSV	Transmission Main Fnsh Wtr Reserv Lvl
26	4	L	06		HBFWRL	Holcomb Blvd Fnsh Wtr Reservoir Level



\*\* Number Of Tanks = 5

### T A N K S

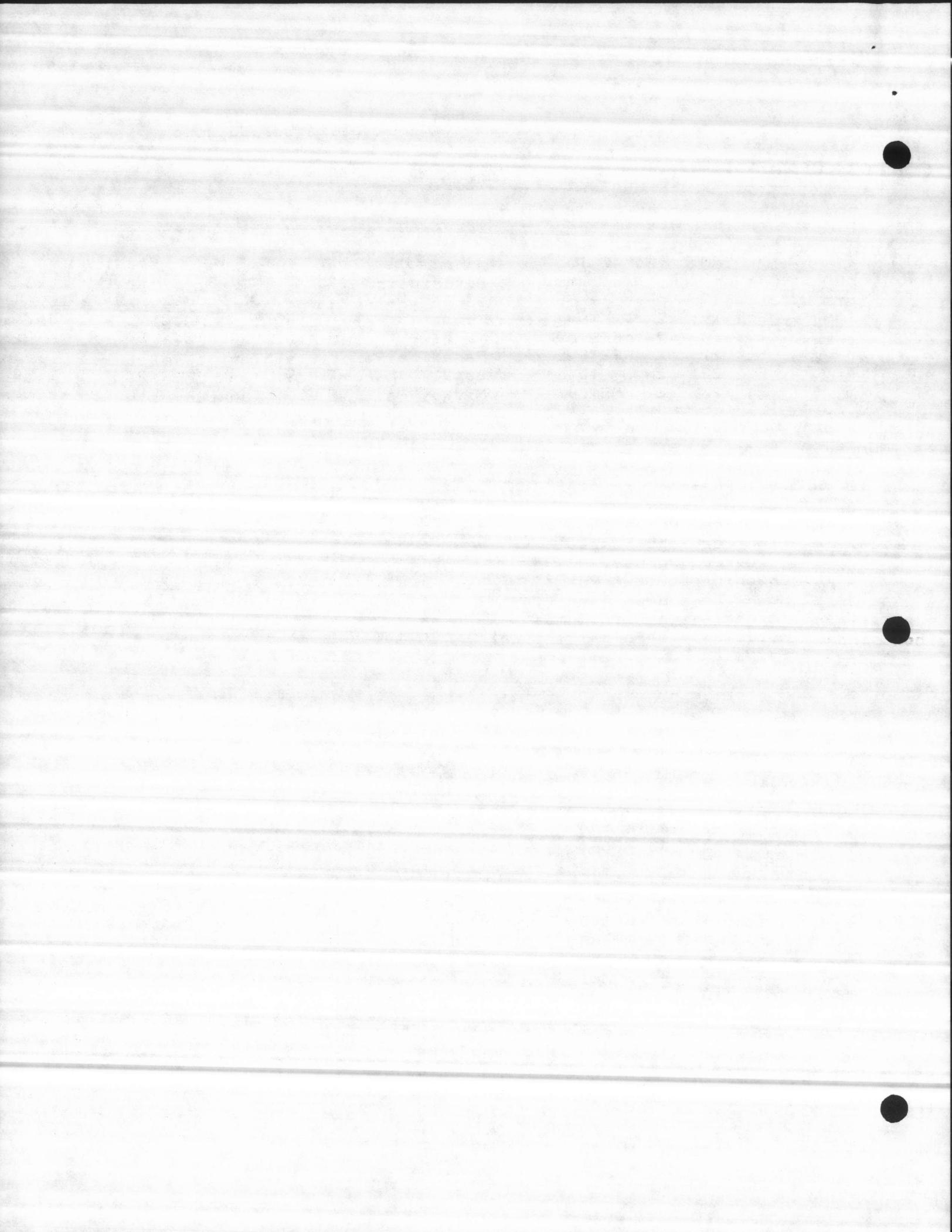
anlg indx	tank indx	R/L ***	RMT or adr	Wd #	Tag No.	Description
27	1	R	19	2	TTETLVL	Trawa Terrace Elv. Tank Level
28	2	R	20	2	MFPETLVL	Montford Point Elv. Tank Level
29	3	R	21	2	PPETLVL	Paradise Point Elv. Tank Level
30	4	R	22	2	BMETLVL	Berkley Manor Elv. Tank Level
31	5	R	23	2	MPETLVL	Midway Park Elv. Tank Level

\*\* Number Of Pumps = 48

- 18 (R) Well Pumps
- 4 (R) Booster Pumps
- 6 (L) Lime Feed Pumps
- 2 (L) Fluoride Feed Pumps
- 2 (L) Acid Feed Pumps
- 2 (L) Chlorine Booster Pumps
- 4 (L) Raw Water Pumps
- 3 (L) Transmission Main Pumps
- 5 (L) Holcomb Blvd Pumps
- 2 (L) Backwash Pumps.

### P U M P S

pump indx	well indx	R/L ***	RMT #	Wd #	Tag No.	Description
1	1	R	1	1	WP643	Well Pump 643
2	2	R	2	1	WP644	Well Pump 644
3	3	R	3	1	WP645	Well Pump 645
4	4	R	4	1	WP646	Well Pump 646
5	5	R	5	1	WP647	Well Pump 647
6	6	R	6	1	WP648	Well Pump 648
7	7	R	7	1	WP649	Well Pump 649
8	8	R	8	1	WP650	Well Pump 650
9	9	R	9	1	WP001	Well Pump 001
10	10	R	10	1	WP002	Well Pump 002
11	11	R	11	1	WP003	Well Pump 003
12	12	R	12	1	WP004	Well Pump 004
13	13	R	13	1	WP005	Well Pump 005
14	14	R	14	1	WP006	Well Pump 006
15	15	R	15	1	WP007	Well Pump 007
16	16	R	16	1	WP008	Well Pump 008
17	17	R	17	1	WP009	Well Pump 009
18	18	R	18	1	WP010	Well Pump 010



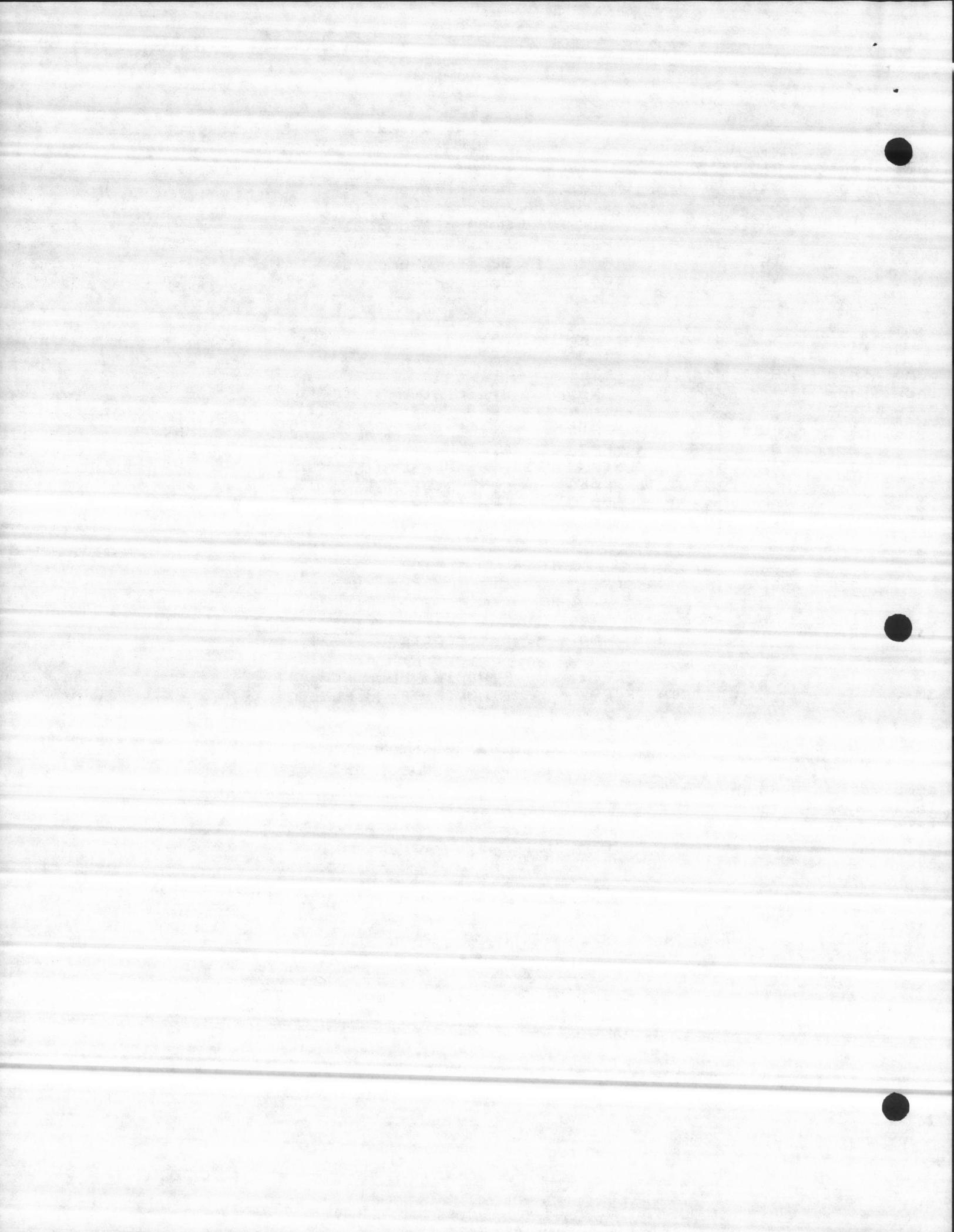
pump indx	boost indx	R/L ***	RMT #	Wd #	Tag No.	Description
19	1	R	24	1	TTP1	Booster Pump 01
20	2	R	24	1	TTP2	Booster Pump 02
21	3	R	24	1	TTP3	Booster Pump 03
22	4	R	24	1	TTP4	Booster Pump 04

pump indx	lmpfd indx	R/L ***	addr #	Wd #	Tag No.	Description
23	1	L	12		LMFP1	Lime Feed Pump 1
24	2	L	12		LMFP2	Lime Feed Pump 2
25	3	L	12		LMFP3	Lime Feed Pump 3
26	4	L	12		LMFP4	Lime Feed Pump 4
27	5	L	12		LMFP5	Lime Feed Pump 5
28	6	L	12		LMFP6	Lime Feed Pump 6

pum indx	flrid indx	R/L ***	addr #	Wd #	Tag No.	Description
29	1	L	11		FLFP1	Fluoride Feed Pump 1
30	2	L	11		FLFP2	Fluoride Feed Pump 2

pump indx	acid indx	R/L ***	addr #	Wd #	Tag No.	Description
31	1	L	11		ACDFP1	Acid Feed Pump 1
32	2	L	11		ACDFP2	Acid Feed Pump 2

pump indx	chl r indx	R/L ***	addr #	Wd #	Tag No.	Description
33	1	L	11		CBSP1	Chlorine Booster Pump 1
34	2	L	11		CBSP2	Chlorine Booster Pump 2



pump indx	Rawtr indx	R/L ***	addr #	Wd #	Tag No.	Description
35	1	L	12		RWP1	Raw Water Pump 1
36	2	L	12		RWP2	Raw Water Pump 2
37	3	L	12		RWP3	Raw Water Pump 3
38	4	L	12		RWP4	Raw Water Pump 4

pump indx	trnsm indx	R/L ***	addr #	Wd #	Tag No.	Description
39	1	L	12		TRMP1	Transmission Main Pump 1
40	2	L	12		TRMP2	Transmission Main Pump 2
41	3	L	12		TRMP3	Transmission Main Pump 3

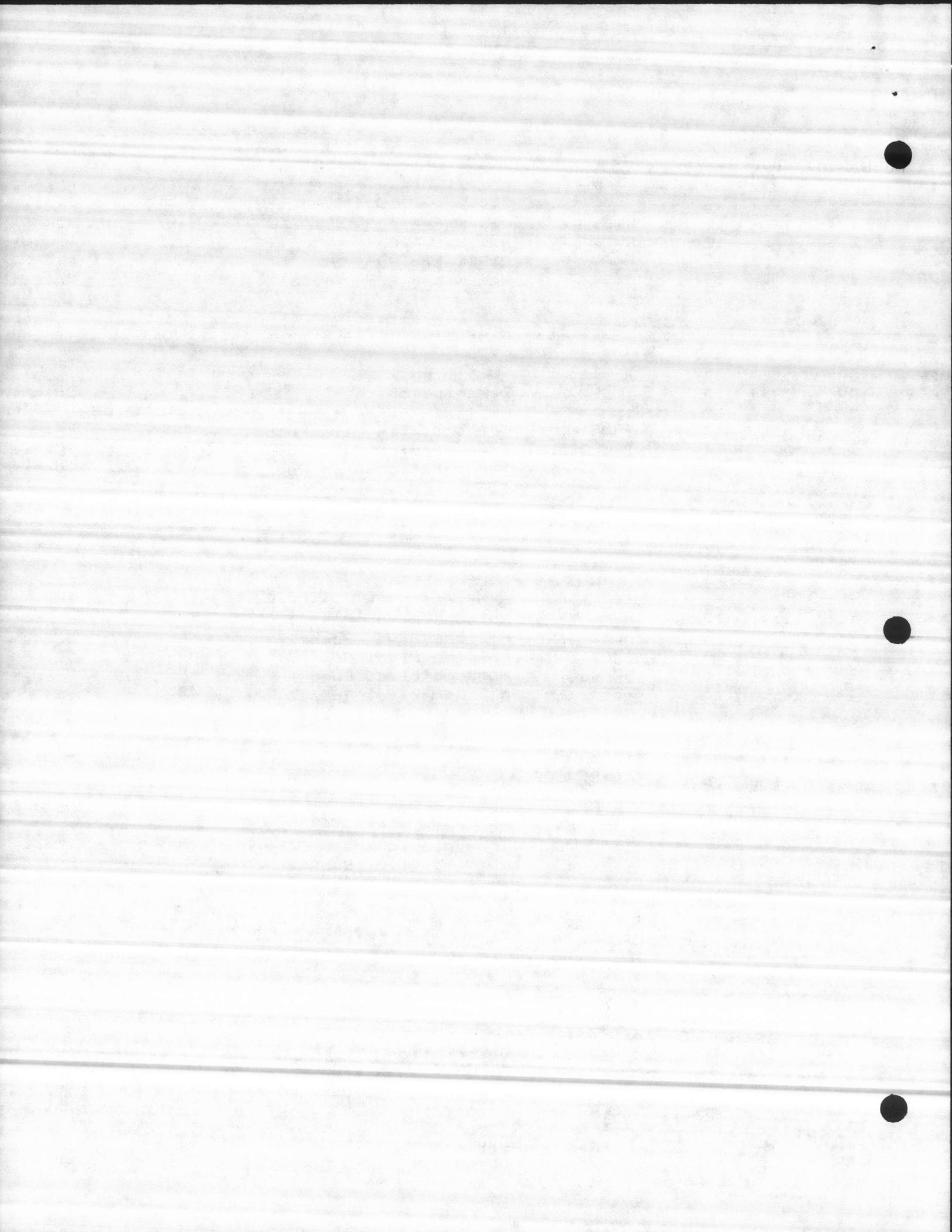
pump indx	hlcmb indx	R/L ***	addr #	Wd #	Tag No.	Description
42	1	L	13		HBP1	Holcomb Blvd. Pump 1
43	2	L	13		HBP2	Holcomb Blvd. Pump 2
44	3	L	13		HBP3	Holcomb Blvd. Pump 3
45	4	L	13		HBP4	Holcomb Blvd. Pump 4
46	5	L	13		HBP5	Holcomb Blvd. Pump 5

pump indx	bkwsh indx	R/L ***	addr #	Wd #	Tag No.	Description
47	1	L	12		TRMBWP	Transmission Main Backwash Pump
48	2	L	13		HBBWP	Holocmb Blvd. Backwash Pump

\*\* Number Of Filters = 5

#### F I L T E R S

fltr indx	R/L ***	addr #	Wd #	Tag No.	Description
1	L	09		FLTR1	Filter 1
2	L	09		FLTR2	Filter 2
3	L	09		FLTR3	Filter 3
4	L	09		FLTR4	Filter 4
5	L	09		FLTR5	Filter 5



\*\* Number Of Lime Feeders = 6

---

L I M E   F E E D E R S

---

lmfdr indx	R/L ***	addr #	Wd #	Tag No.	Description
1	L	10		LMFDR1	Lime Feeder 1
2	L	10		LMFDR2	Lime Feeder 2
3	L	10		LMFDR3	Lime Feeder 3
4	L	10		LMFDR4	Lime Feeder 4
5	L	10		LMFDR5	Lime Feeder 5
6	L	10		LMFDR6	Lime Feeder 6

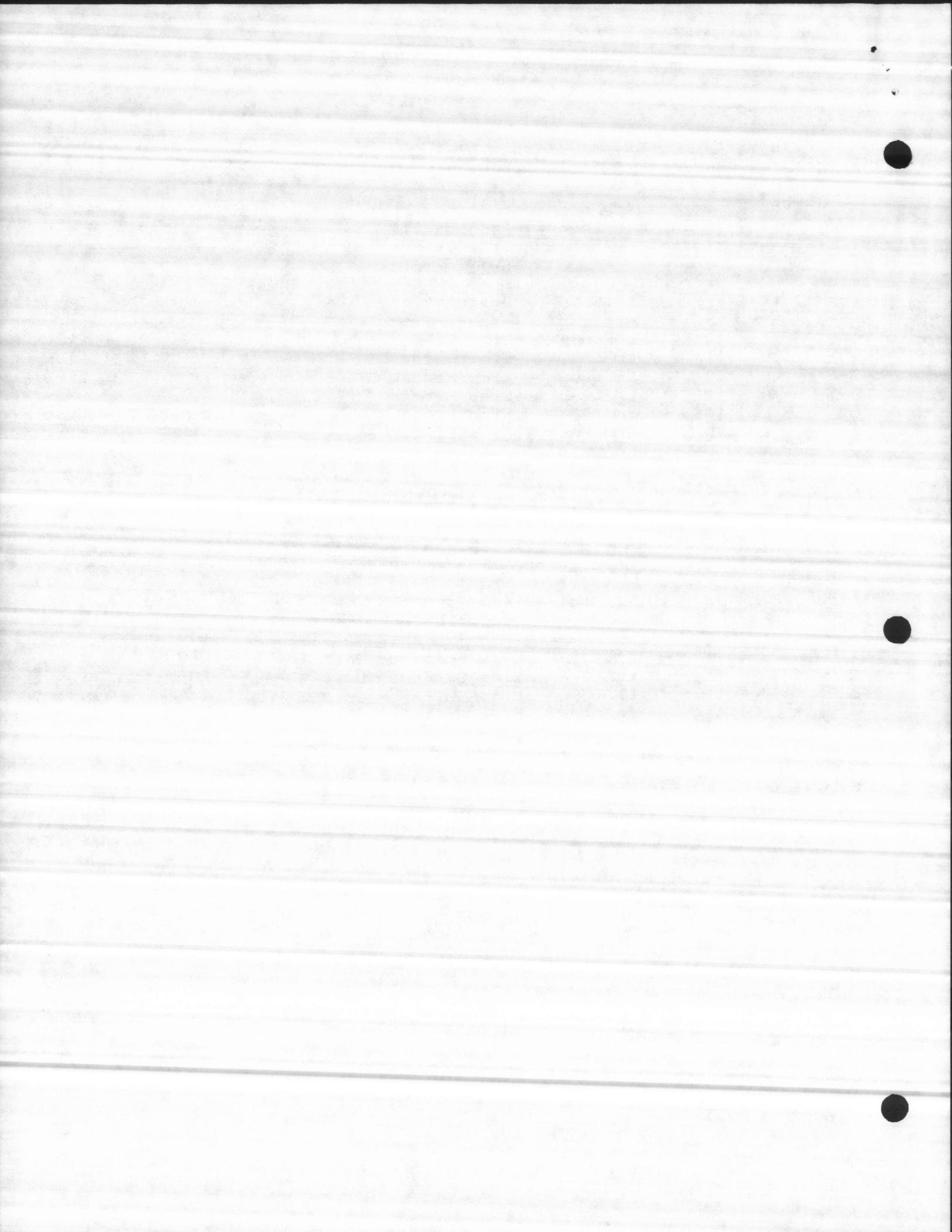
\*\* Number Of Lime Softener Units = 5

---

L I M E   S O F T E N E R   U N I T S

---

lmfdr indx	R/L ***	addr #	Wd #	Tag No.	Description
1	L	11		LMSU1	Lime Softener Unit 1
2	L	11		LMSU2	Lime Softener Unit 2
3	L	11		LMSU3	Lime Softener Unit 3
4	L	11		LMSU4	Lime Softener Unit 4
5	L	11		LMSU5	Lime Softener Unit 5



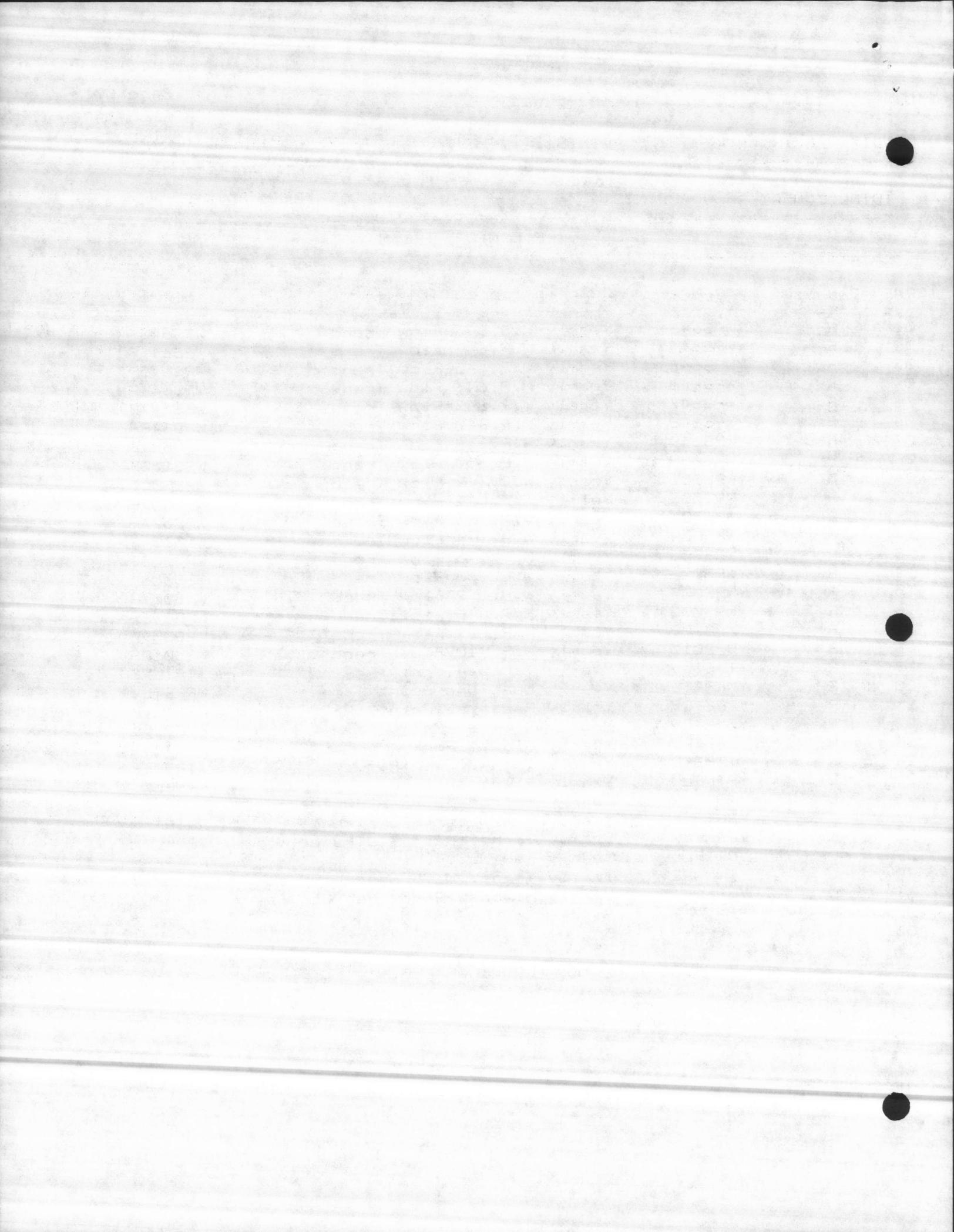
## APPENDIX - F : YEARLY TREND POINTS :

\*\* TOTAL YEARLY TREND POINTS = 33

## YEARLY TREND POINTS

Index	anlg type	Tag No.	description	source id
01	R	YWPS643	Yearly W.P.643 Flow Trend	WPS643F
02	R	YWPS644	Yearly W.P.644 Flow Trend	WPS644F
03	R	YWPS645	Yearly W.P.645 Flow Trend	WPS645F
04	R	YWPS646	Yearly W.P.646 Flow Trend	WPS646F
05	R	YWPS647	Yearly W.P.647 Flow Trend	WPS647F
06	R	YWPS648	Yearly W.P.648 Flow Trend	WPS648F
07	R	YWPS649	Yearly W.P.649 Flow Trend	WPS649F
08	R	YWPS650	Yearly W.P.650 Flow Trend	WPS650F
09	R	YWPS001	Yearly W.P. 1 Flow Trend	WPS001F
10	R	YWPS002	Yearly W.P. 2 Flow Trend	WPS002F
11	R	YWPS003	Yearly W.P. 3 Flow Trend	WPS003F
12	R	YWPS004	Yearly W.P. 4 Flow Trend	WPS004F
13	R	YWPS005	Yearly W.P. 5 Flow Trend	WPS005F
14	R	YWPS006	Yearly W.P. 6 Flow Trend	WPS006F
15	R	YWPS007	Yearly W.P. 7 Flow Trend	WPS007F
16	R	YWPS008	Yearly W.P. 8 Flow Trend	WPS008F
17	R	YWPS009	Yearly W.P. 9 Flow Trend	WPS009F
18	R	YWPS010	Yearly W.P. 10 Flow Trend	WPS010F
19	R	YTETLVL	Yearly Tarawa Terr.	TTETLVL
20	R	YMFETLVL	Yearly Montford Point	MFPETLVL
21	R	YPETLVL	Yearly Paradise Point	PPETLVL
22	R	YBMETLVL	Yearly Berkley Park	BMETLVL
23	R	YMPETLVL	Yearly Midway Park	MPETLVL
24	R	YTTRSVL	Yearly Tarawa Terr Resv	TTRSVL
25	L	YPINFLW	Yearly Plant Infl. Flow Trend	PINFLW
26	L	YHBDFLW	Yearly Holcomb Blvd Flow Trend	HBDFLW
27	L	YRWTFLW	Yearly Raw Water Flow Trend	RWTRFLW
28	L	YTRMFLW	Yearly Trans. Main F. Trend	TRMFLW
29	L	YRWTRESV	Yearly Raw Water Resv.	RWTRESV
30	L	YTRMWRSV	Yearly Trans. Main Fnsh Wtr Resv	TRMWRSV
31	L	YHBFWRL	Yearly Holcomb Blvd Fnsh Wtr Resv	HBFWRL
32	L	-----	Spare	
33	L	-----	Spare	

WHERE R = REMOTE ANALOG &amp; L = LOCAL ANALOG.

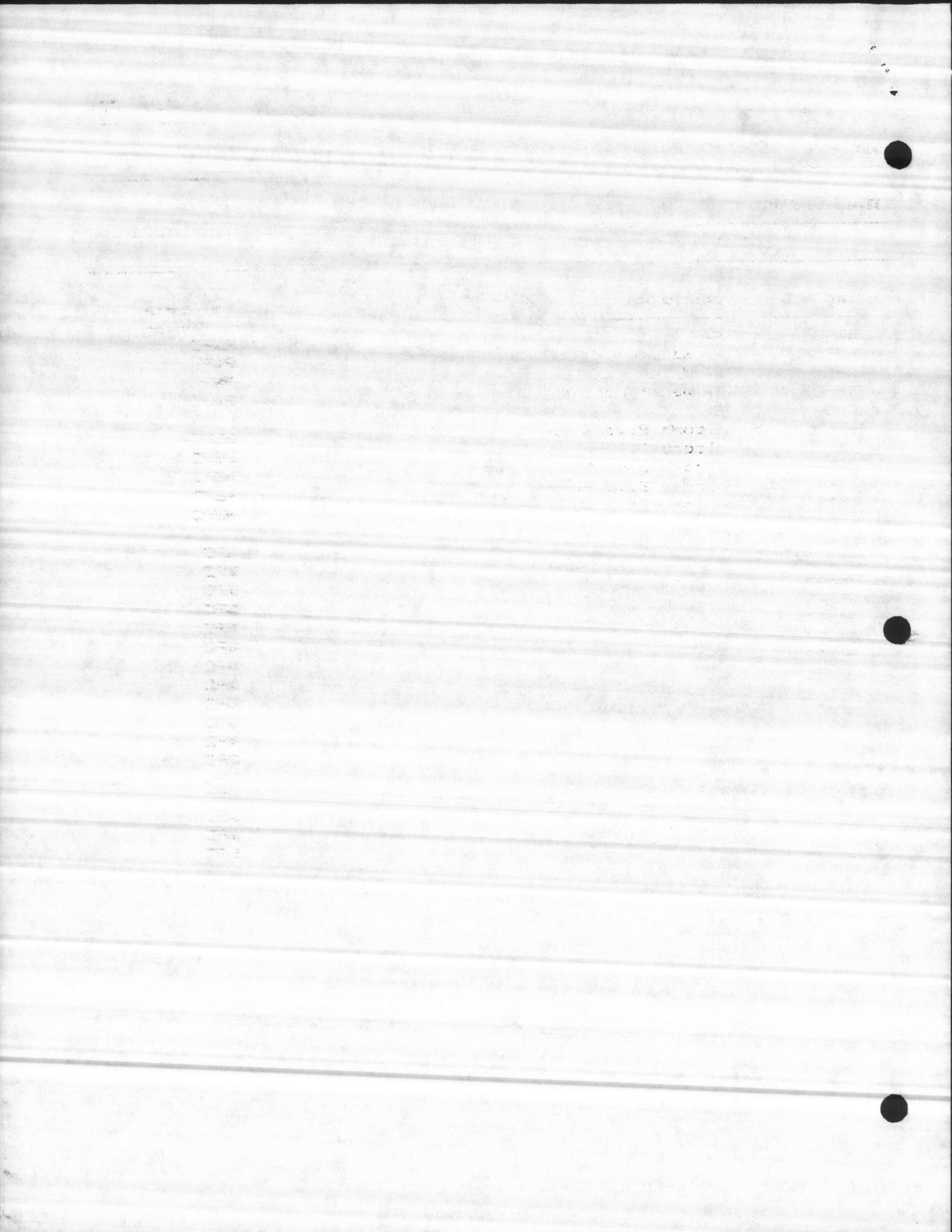


APPENDIX - G : CONTROLS :

\*\* TOTAL CONTROLS = 6

C O N T R O L S

Index	Tag No.	description
1	RWPC	RAW WATER PUMP CONTROL
2	HBPC	HOLCMB BLVD PUMP CONTROL
3	WPC	WELL PUMP CONTROL
4	TMPC	TRANS. MAIN PUMP CONTROL
5	CFC	CHEMICAL FEED CONTROL
6	-----	SPARE



## APPENDIX - H : CONTROL LEVEL POINTS :

\*\* TOTAL CONTROL LEVEL POINTS = 60

## C O N T R O L   L E V E L   P O I N T S

Index	Tag No.	description	Control type
01	RWP1	Raw Water Pump 1	RWPC
02	RWP2	Raw Water Pump 2	RWPC
03	RWP3	Raw Water Pump 3	RWPC
04	RWP4	Raw Water Pump 4	RWPC
05	HBP1	Holcomb Blvd Pump 1	HBPC
06	HBP2	Holcomb Blvd Pump 2	HBPC
07	HBP3	Holcomb Blvd Pump 3	HBPC
08	HBP4	Holcomb Blvd Pump 4	HBPC
09	HBP5	Holcomb Blvd Pump 5	HBPC
10	WP643	Well Pump 643	WPC
11	WP644	Well Pump 644	WPC
12	WP645	Well Pump 645	WPC
13	WP646	Well Pump 646	WPC
14	WP647	Well Pump 647	WPC
15	WP648	Well Pump 648	WPC
16	WP649	Well Pump 649	WPC
17	WP650	Well Pump 650	WPC
18	WP1	Well Pump 1	WPC
19	WP2	Well Pump 2	WPC
20	WP3	Well Pump 3	WPC
21	WP4	Well Pump 4	WPC
22	WP5	Well Pump 5	WPC
23	WP6	Well Pump 6	WPC
24	WP7	Well Pump 7	WPC
25	WP8	Well Pump 8	WPC
26	WP9	Well Pump 9	WPC
27	WP10	Well Pump 10	WPC

092  
093  
094

095  
096  
097  
098  
099  
100  
101  
102  
103  
104  
105  
106

107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200

29	TMP1	Transmission Main Pump 1	TMPC
29	TMP2	Transmission Main Pump 2	TMPC
30	TMP3	Transmission Main Pump 3	TMPC
31	FL1	Lime Feeder 1	CFC
32	FL2	Lime Feeder 2	CFC
33	FL3	Lime Feeder 3	CFC
34	FL4	Lime Feeder 4	CFC
35	FL5	Lime Feeder 5	CFC
36	FL6	Lime Feed Pump 1	CFC
37	LFP1	Lime Feed Pump 2	CFC
38	LFP2	Lime Feed Pump 3	CFC
39	LFP3	Lime Feed Pump 4	CFC
40	LFP4	Lime Feed Pump 5	CFC
41	LFP5	Lime Feed Pump 6	CFC
42	LFP6	Lime Softner Unit 1	CFC
43	LSU1	Lime Softner Unit 2	CFC
44	LSU2	Lime Softner Unit 3	CFC
45	LSU3	Lime Softner Unit 4	CFC
46	LSU4	Lime Softner Unit 5	CFC
47	LSU5	Flouride Feed Pump 1	CFC
48	FFP1	Flouride Feed Pump 2	CFC
49	FFP2	Acid Feed Pump 1	CFC
50	AFP1	Acid Feed Pump 2	CFC
51	AFP2	Plant Effl. Ch1 Feed Pump	CFC
52	PECFP	Lime Feeder 1	CFC
53	HOLBW1	Holcomb Backwash Control	RWPC
54	TRNSBW1	Trans. Main Backwash Ctrl	RWPC
55	-----	Spare	
56	-----	Spare	
57	-----	Spare	
58	-----	Spare	
59	-----	Spare	
60	-----	Spare	

1960 9069

RECEIVED IN LIBRARY

LIBRARY

SEARCHED AND INDEXED  
SERIALIZED AND FILED  
JULY 1960  
SEARCHED AND INDEXED  
SERIALIZED AND FILED  
JULY 1960  
SEARCHED AND INDEXED  
SERIALIZED AND FILED  
JULY 1960

SEARCHED

INDEXED

FILED

## APPENDIX - I : REPORTS :

\*\* TOTAL REPORTS = 25

## R E P O R T S (L O G S)

Index	Tag No.	description
01	HBDR	Holcomb Blvd Daily Rpt
02	DLVLSUM	Daily Level Summary
03	DRTS	Daily Run Time Summary
04	DF&SUIS	Daily Fltr & Softners.
05	HBMR	Holcomb Blvd Monthly Rpt
06	MLVLSUM	Monthly Level Summary
07	MRTS	Monthly Run Time Summary
08	MF&SUIS	Monthly Filter & Sioftners Rpt
09	-----	Spare
10	-----	Spare
11	-----	Spare
12	-----	Spare
13	-----	Spare
14	-----	Spare
15	-----	Spare
16	-----	Spare
17	-----	Spare
18	-----	Spare
19	-----	Spare
20	-----	Spare
21	-----	Spare
22	-----	Spare
23	-----	Spare
24	-----	Spare
25	-----	Spare

101 = 101

## ROUTE DIRECTORY

## cription

Site 1 Displa  
 Site 2 Displa  
 Site 3 Displa  
 Site 4 Displa  
 Site 5 Displa  
 Site 6 Displa  
 Site 7 Displa  
 Site 8 Displa  
 Site 9 Displa  
 Site 10 Displa  
 Site 943 Displa  
 Site 944 Displa  
 Site 945 Displa  
 Site 946 Displa  
 Site 947 Displa  
 Site 948 Displa  
 Site 949 Displa  
 Site 950 Displa  
 Site 951 Displa  
 Site 952 Displa  
 Site 953 Displa  
 Site 954 Displa  
 Site 955 Displa  
 Site 956 Displa  
 Site 957 Displa  
 Site 958 Displa  
 Site 959 Displa  
 Site 960 Displa  
 Site 961 Displa  
 Site 962 Displa  
 Site 963 Displa  
 Site 964 Displa  
 Site 965 Displa  
 Site 966 Displa  
 Site 967 Displa  
 Site 968 Displa  
 Site 969 Displa  
 Site 970 Displa  
 Site 971 Displa  
 Site 972 Displa  
 Site 973 Displa  
 Site 974 Displa  
 Site 975 Displa  
 Site 976 Displa  
 Site 977 Displa  
 Site 978 Displa  
 Site 979 Displa  
 Site 980 Displa  
 Site 981 Displa  
 Site 982 Displa  
 Site 983 Displa  
 Site 984 Displa  
 Site 985 Displa  
 Site 986 Displa  
 Site 987 Displa  
 Site 988 Displa  
 Site 989 Displa  
 Site 990 Displa  
 Site 991 Displa  
 Site 992 Displa  
 Site 993 Displa  
 Site 994 Displa  
 Site 995 Displa  
 Site 996 Displa  
 Site 997 Displa  
 Site 998 Displa  
 Site 999 Displa  
 Site 000 Displa

## APPENDIX - J : GROUP DISPLAYS :

\*\* TOTAL GROUP DISPLAYS = 101

## GROUP DISPLAYS

Index	Tag No.	description
01	WS1	Well Site 1 Display
02	WS2	Well Site 2 Display
03	WS3	Well Site 3 Display
04	WS4	Well Site 4 Display
05	WS5	Well Site 5 Display
06	WS6	Well Site 6 Display
07	WS7	Well Site 7 Display
08	WS8	Well Site 8 Display
09	WS9	Well Site 9 Display
10	WS10	Well Site 10 Display
11	WS643	Well Site 643 Display
12	WS644	Well Site 644 Display
13	WS645	Well Site 645 Display
14	WS646	Well Site 646 Display
15	WS647	Well Site 647 Display
16	WS648	Well Site 648 Display
17	WS649	Well Site 649 Display
18	WS650	Well Site 650 Display
19	TTET	Tarawa Terr Elev Tank
20	MPET	Montford Pt Elev Tank
21	PPET	Paradise Pt Elev Tank
22	BMET	Berkely Man Elev Tank
23	MWPET	Midway Park Elev Tank
24	TTPS	Tarawa Terr Pump Station
25	RWPS	Raw Water Pump Station
26	TMPS	Trans. Main Pump Station
27	HBPS	Holcomb Blvd Pump Station
28	CFSD	Chemical Feed Pump Display
29	FSDP	Filter Status Display
30	SYSTEM	Graphic System Display
31	SYMBOLS1	Symbol 1 Listing
32	SYMBOLS2	Symbol 2 Listing
33	RWPED	Raw Water Pump Control
34	HBPCD	Hlcmbl Blvd Pump Control
35	WPCD	Well Pump Control
36	TMPCD	Trans Main Pump Control
37	CFCD	Chemical Feed Control
38	ETL	Elev Tank Levels
39	-----	Spare
40	-----	Spare
.		
101	-----	Spare

ATTENTION

L-130

#12020

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%

100%