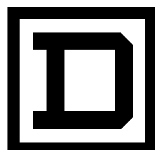
CRISP®/32

Application Notebook



SQUARE D
GROUPE SCHNEIDER

CRISP®/32
Application Notebook
Document number: 500 064 - 001, Rev. 2

Document History

Revision	Date	Pages affected/Description of change
1	11/30/92	Initial Release. ECN xxxx
2	1/18/93	Update Total Document. ECN 4303

Software Version CRISP/32 Rev. 2.8 and Later

This information furnished by Square D Company is believed to be accurate and reliable. However, Square D Company neither assumes responsibility for its use nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Square D Company. This information is subject to change without notice.

Copyright 1992 by
Square D Company
5160 Paul G. Blazer Memorial Parkway
Dublin, Ohio 43017
USA

WARNING: Any unauthorized sale, modification or duplication of this material may be an infringement of copyright.

CRISP® is a registered trademark of Square D Company.

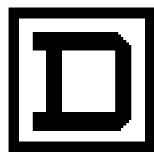
I/ONYX® is a registered trademark of Square D Company.

The following are trademarks of Digital Equipment Corporation: VMS, DEC, RSX-IIM Plus, VAX, MicroVAX, and PDP-II.

CRISP/32

Application

Notebook



SQUARE D COMPANY

CRISP AUTOMATION SYSTEMS

CRISP/32
Application Notebook
Document number: 500 064 - 001, Rev. 2

Document History

Revision	Date	Pages affected/Description of change
1	11/30/92	Initial Release. ECN xxxx
2	1/18/93	Update Total Document. ECN xxxx

Software Version CRISP/32 Rev. 2.8 and Later

This information furnished by Square D Company is believed to be accurate and reliable. However, Square D Company neither assumes responsibility for its use nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Square D Company. This information is subject to change without notice.

Copyright 1992 by
Square D Company
5160 Paul G. Blazer Memorial Parkway
Dublin, Ohio 43017
USA

WARNING: Any unauthorized sale, modification or duplication of this material may be an infringement of copyright.

CRISP® is a registered trademark of Square D Company.

I/ONYX® is a registered trademark of Square D Company.

The following are trademarks of Digital Equipment Corporation: VMS, DEC, RSX-IIM Plus, VAX, MicroVAX, and PDP-II.

Table of Contents

Introduction

Introduction....1-1

Security

General....2-1
PCWS Remote Security....2-3
Windowed Workstation Security....2-5
Screen Specific Security....2-9
  Solution A....2-9
  Explanation A....2-9
  Example 1.2-A....2-9
  Solution B....2-15
  Explanation B....2-15
  Example 1.2-B....2-15
  Solution C....2-19
  Explanation C....2-19
  Example 1.2-C....2-19
CHART/II Security....2-25

Alarm Handling

General....3-1
Alarm Messages to Disk or File....3-3
  Solution....3-3
  Explanation....3-3
  Example 2.1....3-4
File Maintenance....3-11
  Explanation....3-11
  Example 2.1 B....3-11
Alarm Scrolling....3-15
  Explanation....3-15
  Example 2.2....3-15
Alarm Scrolling (2 Lines)....3-21
  Explanation....3-21
  Example 2.3....3-21
Group Alarm Acknowledgment....3-29
  Explanation....3-29
  Example 2.4....3-29
Individual Alarm Acknowledgment....3-35
  Explanation....3-35
  Example 2.5....3-35
Enabling/Disabling Alarms....3-43
  Solution A....3-43
  Explanation A....3-43
  Example 2.6 A....3-43
  Solution B....3-49
  Explanation B....3-49
  Example 2.6 B....3-49

Historical Data Recall

General....4-1
  Solution....4-1
  Explanation....4-1
  SCAN_HIST Edit Function....4-1
  Example 3.1....4-2

© 1991 Square D All Rights Reserved

Device Communication

General....5-1
Redundant PLC Interface....5-3
  Example 4.1....5-3
Multi-drop Interface....5-11
  Example 4.2....5-11
  Sample UCF Code....5-16

File Handling

General....6-1
  Solution A....6-1
  Explanation A....6-1
  Example 5.1 A....6-1
  Solution B....6-9
  Explanation B....6-9
  Example 5.1 B....6-9
  Solution C....6-19
  Explanation C....6-19
  Example 5.1 C....6-19

© 1991 Square D All Rights Reserved

User's Critique

Introduction

To make this Application Notebook more useful, please submit any comments and recommendations that you feel are required to improve this document. After using this document, please take the first opportunity to complete this questionnaire and return it, via facsimile (614-764-4279) to Square D/CRISP Automation where your comments will be given every consideration.

Organization

Was the Table of Contents detailed enough and useful?

Yes p No p Comments ______________________________

Were the Notebook Chapters well organized?

Yes p No p Comments ______________________________

General

Please answer the following questions using one of the following rating categories.

1 - Excellent 2 - Above Average 3 - Average 4 - Below Average 5 - Poor 6 - Unknown (not used)

For each chapter, how would you rate the quality of the Application descriptions?

___Security ___Alarms ___HISTORIAN ___Communications ___Files

For each chapter, how would you rate the usefulness of the Application examples and solutions?

___Security ___Alarms ___HISTORIAN ___Communications ___Files

For each chapter, how would you rate the quality of the displays, diagrams, etc.?

___Security ___Alarms ___HISTORIAN ___Communications ___Files

Do you intend to use any of the Application code solutions presented?

Yes p No p If yes, list Chapter(s) ______________________________

If there are particular Chapters, problems, solutions, etc., you feel require clarification or rewriting, please identify them and add your comments.

____
____
____
____

Comments

(For example, additional chapters you would like to see presented)

____
____
____
____

Have you attended any formal CRISP Automation training programs

Yes p No p If yes, list what/when ______________________________

Do you have a unique solution to a specific problem that could be included in this notebook?

Yes p No p If yes, please fill in the following contact information.

Contact

Respondent ______________________ Title ______________

Company ______________________________

Address ______________________________

City/State/Zip ______________________________

© 1993 Square D All Rights Reserved

Introduction

This document provides the user with sample solutions to common application problems.

This manual is broken down into the following sections.

Section	**Description**
Security	Defines how to configure the various types of security systems available on the CRISP/32 workstations.
Alarm Handling	Defines how to configure the various types of alarm handling functions available on CRISP/32.
Historical Data Recall	Defines how to use the SCAN_HIST function call, including data point edit, and how to write the HISTORIAN data to an ASCII file.
Device Communication	Defines how to communicate with the various types of devices available on CRISP/32.
File Handling	Defines how to open, read, write, and close small and large files.

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

General

This section defines how to extend the functionality of various types of security systems available on the CRISP/32 workstations.

This section is broken down into the following subsections.

Subsection	Description
PCWS Remote Security *(page 2-3)*	Defines how to modify PC Workstation security from a CRISP logic.
Windowed Workstation Security *(page 2-5)*	Defines how to configure CRT-specific security on a Windowed Workstation.
Screen-Specific Security *(page 2-9)*	Defines three methods of implementing screen-specific security.
CHART/II Security *(page 2-25)*	Defines how to incorporate security within a CHART/II application.

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

PCWS Remote Security

This section describes the procedure for modifying PCWS/CWS security from a CRISP logic. This can be useful to the application developer who needs to customize workstation security for a particular end user.

To configure PCWS Remote Security you must go to the **UTILITY MENU** and select **CONFIGURE SECURITY**, which displays the Remote Security Table. Each user (refered to as Account Name) should have a Password and an associated Screen Security Link Level (referred to as HIGH on the Remote Security Table). When a user (Account Name) is logged in, the user **only** has access to those display links with a Link Level (denoted by `/L:#`) between the `HIGH` and `LOW` or that `Account Name` on the Security Table.

There are certain things to keep in mind if you want to force the PCWS to a specified security level (i.e., the DEFAULT security). The logic must have in its declarations a Numeric array `SET_LOG_NUM_ARRAY(MAX_CRT_NO)` where `MAX_CRT_NO` is maximum number of CRTs on the system. This variable (which is used to force the PCWS to a specified security level) must be linked on the Local system control display, screen #21. Even if the security is for Remote displays, `%SET_LOG_NUM` should be on both Local and Remote screen #21, and linked to `DATABASE:SET_LOG_NUM_ARRAY(#)` (PCWS replaces the `#` with the CRT number that you are on). The user logic using the `SET_LOG_NUM_ARRAY` should be in the Local and Remote Node Database Table.

If the security is for the remote displays, `WSDIR` on the Local Node Database Table, should be pointing to the remote node number. Also, the Local Node Database Display (page 2) must have `D_HOST` pointing to the remote node number.

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Windowed Workstation Security

This section describes how to configure a workstation-specific security for a requirement calling for limiting process changes at certain graphic screens if the operator is not at the proper workstation. The screens are viewable on all workstations, but no process changes are possible unless the operator is logged in at the correct workstation.

The solution is implemented by configuring separate copies of the CRISPWIN security file (CRISPWIN_SECURITY.DAT) for each of the workstations. To ensure that each CRISPwindows process gets the right security file, the security files must be placed in different subdirectories along with the CRISPWIN startup process.

The following user setup file must be added to the [CRISP] directory. This definition tells a process using the CRISPWIN_SECURITY logical to look for the security file in the same directory where the process is being started.

```
$ ! NODE::CRISP$:USER_SETUP_CRISPWIN.COM
$ !
$ ! Define a logical to re-direct the CRISPWIN_SECURITY file access
$ ! from the default directory to an individual CRT subdirectory
$ !
$ Define/system crispwin_security sys$disk:[]crispwin_security.dat
$ !
$ Exit
```

The startup procedure for CRISPWIN processes is defined in the CRISPWIN_PROCDEF.DAT file. A typical portion of the startup definition is as follows.

```
!
! NODE::CRISP$CFG:CRISPWIN_PROCDEF.DAT
!
!       Define startup parameters for
!       CWIN and CDRW processes
!

!--------------------------------------------------
!CRT1
!--------------------------------------------------
!
1,                                    ! session_number,
NODE,                                 ! run_node_name,
LAT_xxxxxxxxxx,                       ! display_node_name,
CWIN,                                 ! process_type,
crisp$device:[crisp.wws.CRT1]         ! default_directory
!
2,                                    ! session_number,
NODE,                                 ! run_node_name,
LAT_xxxxxxxxxx,                       ! display_node_name,
CDRW,                                 ! process_type,
crisp$device:[crisp.wws.CRT1],        ! default_directory,
crisp$wws_etc:crispdraw_config.dat,   ! configuration file specification,
noautostart                           ! autostart flag
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Windowed Workstation Security (cont)

```
!
!--------------------------------------------------
!CRT2
!--------------------------------------------------
!
3,                                   ! session_number,
NODE,                                ! run_node_name,
LAT_xxxxxxxxxx,                      ! display_node_name,
CWIN,                                ! process_type,
crisp$device:[crisp.wws.CRT2]        ! default_directory
!
4,                                   ! session_number,
NODE,                                ! run_node_name,
LAT_xxxxxxxxxx,                      ! display_node_name,
CDRW,                                ! process_type,
crisp$device:[crisp.wws.CRT2],       ! default_directory,
crisp$wws_etc:crispdraw_config.dat,  ! configuration file specification,
noautostart                          ! autostart flag
```

After completing configuration for the workstations, the next task is to modify the security definitions in each of the security files. This is best accomplished by first creating a single master security file and copying it to the individual subdirectories. The copies are then edited to provide the specific limitations.

The following examples show how these edited files might look for two workstations and two displays. In the first security file, display A and B have the same security masks meaning that tag variables linked to these displays may be modified if the operator has logged in and his mask matches the mask of the individual tags.

```
!
! NODE::DISK$USER:[CRISP.WWS.CRT1]CRISPWIN_SECURITY.DAT
!
!        Screen Security Configuration (CRT1)
!
!
! Default mask for all TAGs
!
FFFFFFFF

!
!-------------------
! Display A Security
!-------------------
!
! Mask   TAG Name
!------- ---------------------------
00000044 NODE::DBASE:DSP01_NEXT_SP(1)
00000044 NODE::DBASE:DSP01_NEXT_SP(2)
00000044 NODE::DBASE:DSP01_NEXT_SP(3)
00000044 NODE::DBASE:DSP01_NEXT_SP(4)
00000044 NODE::DBASE:DSP01_CANCEL_PB
00000044 NODE::DBASE:DSP01_EXECUTE_PB
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Windowed Workstation Security (cont)

```
!
!-------------------
! Display B Security
!-------------------
!
! Mask   TAG Name
!------- ----------------------------
00000044 NODE::DBASE:DSP02_NEXT_SP(1)
00000044 NODE::DBASE:DSP02_NEXT_SP(2)
00000044 NODE::DBASE:DSP02_NEXT_SP(3)
00000044 NODE::DBASE:DSP02_NEXT_SP(4)
00000044 NODE::DBASE:DSP02_CANCEL_PB
00000044 NODE::DBASE:DSP02_EXECUTE_PB
```

In the second security file (refer to the following), the mask for display A has been set to all zeros. Therefore, all operators, regardless of what security level they log in with, will not be able to modify the tags from CRT2. Display B has the same security masks as display B did on CRT1 meaning that operation of this display is not affected by which CRISPwindows process is accessing the data.

```
! NODE::DISK$USER:[CRISP.WWS.CRT2]CRISPWIN_SECURITY.DAT
!
!        SCREEN SECURITY CONFIGURATION (CRT2)
!
!
! Default mask for all TAGs
!
FFFFFFFF

!
!-------------------
! Display A Security
!-------------------
!
! Mask   TAG Name
!------- ----------------------------
00000000 NODE::DBASE:DSP01_NEXT_SP(1)
00000000 NODE::DBASE:DSP01_NEXT_SP(2)
00000000 NODE::DBASE:DSP01_NEXT_SP(3)
00000000 NODE::DBASE:DSP01_NEXT_SP(4)
00000000 NODE::DBASE:DSP01_CANCEL_PB
00000000 NODE::DBASE:DSP01_EXECUTE_PB

!
!-------------------
! Display B Security
!-------------------
!
! Mask   TAG Name
!------- ----------------------------
00000044 NODE::DBASE:DSP02_NEXT_SP(1)
00000044 NODE::DBASE:DSP02_NEXT_SP(2)
00000044 NODE::DBASE:DSP02_NEXT_SP(3)
00000044 NODE::DBASE:DSP02_NEXT_SP(4)
00000044 NODE::DBASE:DSP02_CANCEL_PB
00000044 NODE::DBASE:DSP02_EXECUTE_PB
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Screen Specific Security

This subsection contains three methods of implementing screen-specific security. This means using CRISP logic to create a way for an operator to log in/out directly from his current screen rather than exiting to the main menu and selecting the security option.

Solution A

The application security requirement addressed in this example involved a customer who wanted operators to log in on the same screen as they were performing their functions and for the security to reset automatically after a fixed time period. The following logic enables a password to be entered on a display, forcing a specific security level (determined by the Account Number) to the screen.

Explanation A

The Account Number used in this logic is first defined in the PCWS security table with the appropriate User Lock Level and User Lock Range. The logic-authorized account number is passed to the `SET_LOG_NUM_ARRAY` variable to activate the requested security level. This variable must be defined/linked on the PCWS Screen 21 to work properly.

Example 1.2-A

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to implement screen-specific security. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!**************************************************************************
!*      Sample code for Example 1.2 A                                    *
!*                                                                       *
!*      Task 1.2 - How to implement screen-specific security            *
!**************************************************************************

!--------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!--------------------------------------------------------------------

!------------------------------------------------------------------------
!              The following 16 Logicals are used by the CRISP
!              system.  Do not place any other Declarations
!              before these.
!------------------------------------------------------------------------
LOGICAL;      NEW_DB:true, NEW_CLE, ACTIVE, CACDSA, RESV5, RESV6,\
              RESV7, RESV8, RESV9, RESV10, RESV11, ICCDSA, \
              PAUSE, RESV14, RESV15, RESV16

!------------------------------------------------------------------------
!              The follow 32 logical make up the "Message Mask".
!              These are variables that represent each potential print
!              device or file were messages can be routed.
!------------------------------------------------------------------------
     LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                     CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
```

(Continued on next page.)

Example 1.2-A (cont)

```
                    CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                    CRISP_DEV04  , REPORT_FILE , CRISP_DEV06 , CRISP_DEV07 , \
                    CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                    CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                    CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                    CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


CONSTANT;     ZERO:0, ONE:1
STRING;QUESTION_PSWD[3]:"???"
STRING;BLANK_PSWD[3]   :"   "

!-----------------------------------------------------------------------------
!          Initialize the number of PCWS screens, the number of PASSWORDS
!          and the screen numbers
!-----------------------------------------------------------------------------
CONSTANT;     CRT1:1, CRT1_PTR:0
CONSTANT;     MAX_CRT_NO:1
CONSTANT;     PSWD_LMT:5
CONSTANT;     DEFAULT_PSWD_LEVEL:0

!-------------------------------------------------------------------------
!          Password variable declaration for CRT number 1
!-------------------------------------------------------------------------
LOGICAL;      GET_PSWD_CRT1:TRUE,          \
              TEST_PSWD_CRT1,              \
              VALID_PSWD_CRT1,             \
              INVALID_PSWD_CRT1

LOGICAL;      TIMEOUT_PSWD_CRT1,           \
              NEW_PSWD_CRT1,        \
              GOOD_PSWD_CRT1,              \
              BAD_PSWD_CRT1

NUMERIC;      PSWD_INDEX_CRT1, PSWD_PTR_CRT1

STRING;PCWS_CRT1_MSG[50]

!------------------------------------------------------------------------
!          Declarations to force the PCWS to a specified security level
!          this variable must be linked on the PCWS screen 21
!------------------------------------------------------------------------
NUMERIC;      SET_LOG_NUM_ARRAY(MAX_CRT_NO)

!------------------------------------------------------------------------
!          Declarations for the password timeout (1 Hour)
!------------------------------------------------------------------------
TIMER;        PSWD_TIMER_CRT1:3600    !seconds


!------------------------------------------------------------------------
!       Declarations for the password string variables
!------------------------------------------------------------------------
STRING;ENTER_PSWD_CRT(MAX_CRT_NO)[3]
STRING;CURRENT_PSWD_CRT(MAX_CRT_NO)[3]
NUMERIC;      CURRENT_DSP(MAX_CRT_NO)
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 1.2-A (cont)

```
!------------------------------------------------------------------------
!       Declarations for the login status messages
!------------------------------------------------------------------------
STRING;PSWD_ERROR_MSG[]:"Invalid Password Entered - Try Again          "
STRING;VALID_PSWD_MSG[]:"Password Accepted, Modify Data and Logout!    "
STRING;CLEAR_MSG[]     :"Password Expired - Enter Password to continue"

!------------------------------------------------------------------------
!          This is the defined list of passwords for the system
!          three Digit passwords corresponding to operator initials.
!------------------------------------------------------------------------
STRING;         PSWD_LIST(PSWD_LMT)[3]     \
                            :"AAA"\     ! Operator 1
                            :"BBB"\     ! Maitenance
                            :"CCC"\     ! Technician 1
                            :"DDD"\     ! Engineer
                            :"EEE"      ! Manager

LONG;         PSWD_LEVEL(PSWD_LMT)  \
                            :1            \      ! Operator 1
                            :5            \      ! Maitenance
                            :5            \      ! Technician 1
                            :10           \      ! Engineer
                            :20                  ! Manager

!--------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!--------------------------------------------------------------------
TABLES;


!--------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!--------------------------------------------------------------------
RESTART;


!-----------------------------------------------------------------
!      Monitor each CRT for a display screen other than the main menu
!-----------------------------------------------------------------
RECALL; RDDSP, , CRT1, CURRENT_DSP(CRT1_PTR)


!------------------------------------------------------------------------------
!      Have a Password to test whenever the Enter Password field on
!      the display screen is not the Default Password (???)
!------------------------------------------------------------------------------
RECALL; CMPRS, GET_PSWD_CRT1, \
       ENTER_PSWD_CRT(CRT1_PTR), QUESTION_PSWD, ONE, PSWD_PTR_CRT1

NEW_PSWD_CRT1 = ((PSWD_PTR_CRT1 < ZERO) & (CURRENT_DSP(CRT1_PTR) > 0) & \
                 GET_PSWD_CRT1)
```

(Continued on next page.)

Example 1.2-A (cont)

```
!--------------------------------------------------------------------------------
!      Determine if the password entered was found in the list of
!      defined valid passwords.
!--------------------------------------------------------------------------------
GOOD_PSWD_CRT1 = ((PSWD_INDEX_CRT1 >= ZERO) & TEST_PSWD_CRT1)

BAD_PSWD_CRT1 = ((PSWD_INDEX_CRT1 < ZERO) & TEST_PSWD_CRT1)


!--------------------------------------------------------------------------------
!          Clear the password on a timeout
!--------------------------------------------------------------------------------
TIMEOUT_PSWD_CRT1 = TIMER (VALID_PSWD_CRT1, VALID_PSWD_CRT1, PSWD_TIMER_CRT1)


!--------------------------------------------------------------------------------
!          The State Ring contains Two paths, one for a valid
!          password and the other for an invalid password.
!--------------------------------------------------------------------------------

VALID_PSWD_CRT1   = (TEST_PSWD_CRT1 & GOOD_PSWD_CRT1) | \
                    VALID_PSWD_CRT1 & ~GET_PSWD_CRT1

INVALID_PSWD_CRT1 = (TEST_PSWD_CRT1 & BAD_PSWD_CRT1) | \
                    INVALID_PSWD_CRT1 & ~GET_PSWD_CRT1

TEST_PSWD_CRT1    = (GET_PSWD_CRT1 & NEW_PSWD_CRT1) | \
                    TEST_PSWD_CRT1 & (~VALID_PSWD_CRT1 & ~INVALID_PSWD_CRT1)

GET_PSWD_CRT1     = (VALID_PSWD_CRT1 & TIMEOUT_PSWD_CRT1) | \
                    INVALID_PSWD_CRT1 | \
                    GET_PSWD_CRT1 & ~TEST_PSWD_CRT1


!--------------------------------------------------------------------------------
!          Compare Current password to all Valid Passwords.
!--------------------------------------------------------------------------------
CALL; CMPRS, TEST_PSWD_CRT1, \
          ENTER_PSWD_CRT(CRT1_PTR), PSWD_LIST(0), PSWD_LMT, PSWD_INDEX_CRT1


!-------------------------------------------------------------------------------
! On a valid password send the defined account number to the array variable
! which will correspond to the crt number minus 1.  For example CRT1 = array(0)
! CRT2 = array(1). The account number will change the login security level
! for the PCWS. The account number used for this application is "1", this
! number can be any valid account number from 1 to 31.
!-------------------------------------------------------------------------------
SET; VALID_PSWD_CRT1, SET_LOG_NUM_ARRAY(CRT1_PTR) = PSWD_LEVEL(PSWD_INDEX_CRT1)

SET; TIMEOUT_PSWD_CRT1, SET_LOG_NUM_ARRAY(CRT1_PTR) = DEFAULT_PSWD_LEVEL


!-------------------------------------------------------------------------------
!          Clear Password / Send error messages
!-------------------------------------------------------------------------------
CALL; SCOPY, (VALID_PSWD_CRT1 | INVALID_PSWD_CRT1),  \
    QUESTION_PSWD, ZERO, ENTER_PSWD_CRT(CRT1_PTR), ZERO, ONE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 1.2-A (cont)

```
CALL; SCOPY, INVALID_PSWD_CRT1, PSWD_ERROR_MSG, ZERO, PCWS_CRT1_MSG, ZERO, ONE

CALL; SCOPY, VALID_PSWD_CRT1, VALID_PSWD_MSG, ZERO, PCWS_CRT1_MSG, ZERO, ONE

CALL; SCOPY, GET_PSWD_CRT1, CLEAR_MSG, ZERO, PCWS_CRT1_MSG, ZERO, ONE

END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Solution B

Certain applications require security to automatically reset if the user does not log out after performing a privileged operation. In this example, CRISP logic is used to restore the default security (lowest level) to the PCWS if a screen other than the Main Menu is displayed longer than a fixed time period, the operator returns to the main menu or he selects a Clear pushbutton on the screen.

Explanation B

1. Security will be reset every time the Main Menu is displayed.
2. Security will reset after 10 minutes has elapsed.
3. Security will reset when a CLEAR variable is set.

Example 1.2-B

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to implement screen-specific security. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 1.2 B                                      *
!*                                                                         *
!*      Task 1.2 - How to implement screen-specific security               *
!***************************************************************************

!---------------------------------------------------------------------
!       Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------

!-------------------------------------------------------------------------
!              The following 16 Logicals are used by the CRISP
!              system.  Do not place any other Declarations
!              before these.
!-------------------------------------------------------------------------
LOGICAL;       NEW_DB:true, NEW_CLE, ACTIVE, CACDSA, RESV5, RESV6,\
               RESV7, RESV8, RESV9, RESV10, RESV11, ICCDSA, \
               PAUSE, RESV14, RESV15, RESV16
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 1.2-B (cont)

```
!---------------------------------------------------------------------
!              The follow 32 logical make up the "Message Mask".
!              These are variables that represent each potential print
!              device or file were messages can be routed.
!---------------------------------------------------------------------
     LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                     CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                     CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                     CRISP_DEV04  , REPORT_FILE , CRISP_DEV06 , CRISP_DEV07 , \
                     CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                     CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                     CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                     CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


CONSTANT;      DEFAULT_PSWD_LEVEL:0

!-------------------------------------------------------------------------------
!       Initialize the PCWS CRT number(s), array pointers and
!       the highest CRT number
!-------------------------------------------------------------------------------
CONSTANT;      CRT1:1, CRT1_PTR:0   ! Pointer is one less than CRT number
CONSTANT;      MAX_CRT_NO:1

NUMERIC;       CURRENT_DSP(MAX_CRT_NO)


!---------------------------------------------------------------------
!       This variable would be linked to a button on each display,
!       which would clear the security level by forcing PCWS to the
!       default security level whenever the logical is set.
!---------------------------------------------------------------------
LOGICAL;       CLEAR_PSWD_PB(MAX_CRT_NO)

!---------------------------------------------------------------------
!          Password variable declarations for CRT1
!---------------------------------------------------------------------
LOGICAL;       START_TMR_CRT1,                     \
               TIMEOUT_CRT1

TIMER;         PSWD_TIMER_CRT1:600


!---------------------------------------------------------------------
!       Declarations to force the PCWS to a specified security level
!       this variable must be linked on the PCWS screen 21
!---------------------------------------------------------------------
NUMERIC;       SET_LOG_NUM_ARRAY(MAX_CRT_NO)



!-----------------------------------------------------------------
!       All logic placed between the Key Words TABLES; and
!       RESTART;  will be executed only on the first pass of
!       logic.  All code in this area is Initialization Code.
!-----------------------------------------------------------------
TABLES;
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 1.2-B (cont)

```
!--------------------------------------------------------------------
!       The Logic between the Key Word RESTART; and the Key Word
!       END; will be executed during each pass of Logic.
!--------------------------------------------------------------------
RESTART;

!--------------------------------------------------------------------
!       Monitor each CRT for a display screen other than the main menu
!--------------------------------------------------------------------
RECALL; RDDSP, , CRT1, CURRENT_DSP(CRT1_PTR)



!--------------------------------------------------------------------
!       Initialize each CRT timer if a display screen is on the CRT
!--------------------------------------------------------------------
START_TMR_CRT1 = (CURRENT_DSP(CRT1_PTR) > 0)


!------------------------------------------------------------------------------
!       Clear the password on a timeout or on return to
!       main display (CURRENT_DSP = 0)
!------------------------------------------------------------------------------
TIMEOUT_CRT1 = TIMER (START_TMR_CRT1, START_TMR_CRT1, PSWD_TIMER_CRT1)

!-----------------------------------------------------------------------------
!       When the timer times out, the CLEAR button is pressed, or the Main
!       Menu is displayed, send the default account number to the set
!       number array variable which will correspond to the crt number minus 1.
!       For example CRT1 = array(0), CRT2 = array(1).  The account number will
!       change the login security level for the PCWS.
!-----------------------------------------------------------------------------
SET; TIMEOUT_CRT1 | CLEAR_PSWD_PB(CRT1_PTR) | (CURRENT_DSP(CRT1_PTR) == 0), \
       SET_LOG_NUM_ARRAY(CRT1_PTR) = DEFAULT_PSWD_LEVEL

!----------------------------------------------------------------------------
!       Reset CLEAR pushbuttons at end of logic scan
!----------------------------------------------------------------------------
RECALL; BZERO, , CLEAR_PSWD_PB(0), MAX_CRT_NO

END;
```

© 1991 Square D All Rights Reserved

Notes:

 © 1991 Square D All Rights Reserved

Solution C

The application security requirement addressed in this example involved a customer who wanted operators to log in on the same screen as they were performing their functions and for the security level to reset when a screen change occurs. The following logic allows a password to be entered on a display, forcing a specific security level (determined by the Account Number) to the screen.

Explanation C

The Account Number used in this logic is first defined in the PCWS security table with the appropriate User Lock Level and User Lock Range. The logic-authorized account number is passed to the SET_LOG_NUM_ARRAY variable to activate the requested Security Level. This variable must be defined/linked on the PCWS Screen 21 to work properly.

Example 1.2-C

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to implement screen-specific security. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 1.2 C                                      *
!*                                                                         *
!*      Task 1.2 - How to implement screen-specific security               *
!***************************************************************************

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------


!-------------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!-------------------------------------------------------------------------
LOGICAL;      NEW_DB:true, NEW_CLE, ACTIVE, CACDSA, RESV5, RESV6,\
              RESV7, RESV8, RESV9, RESV10, RESV11, ICCDSA, \
              PAUSE, RESV14, RESV15, RESV16

!-------------------------------------------------------------------------
!              The follow 32 logical make up the "Message Mask".
!              These are variables that represent each potential print
!              device or file were messages can be routed.
!-------------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 1.2-C (cont)

```
     LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                     CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                     CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                     CRISP_DEV04  , REPORT_FILE , CRISP_DEV06 , CRISP_DEV07 , \
                     CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                     CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                     CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                     CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


CONSTANT;             ZERO:0, ONE:1
STRING;        QUESTION_PSWD[3]:"???"
STRING;        BLANK_PSWD[3]   :"   "
!-------------------------------------------------------------------------------
!          Initialize the number of PCWS screens, the number of PASSWORDS
!          and the screen numbers
!-------------------------------------------------------------------------------

CONSTANT;      CRT1:1, CRT1_PTR:0

CONSTANT;      MAX_CRT_NO:1
CONSTANT;      PSWD_LMT:5
CONSTANT;      DEFAULT_PSWD_LEVEL:0

!---------------------------------------------------------------------------
!          Password variable declaration for CRT number 1
!---------------------------------------------------------------------------
LOGICAL;       GET_PSWD_CRT1:TRUE,          \
               TEST_PSWD_CRT1,              \
               VALID_PSWD_CRT1,             \
               INVALID_PSWD_CRT1

LOGICAL;       RESET_PSWD_CRT1,             \
               NEW_PSWD_CRT1,        \
               GOOD_PSWD_CRT1,              \
               BAD_PSWD_CRT1

NUMERIC;       PSWD_INDEX_CRT1, PSWD_PTR_CRT1

STRING;        PCWS_CRT1_MSG[50]

!--------------------------------------------------------------------------
!          Declarations to force the PCWS to a specified security level
!          this variable must be linked on the PCWS screen 21
!--------------------------------------------------------------------------
NUMERIC;       SET_LOG_NUM_ARRAY(MAX_CRT_NO)
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 1.2-C (cont)

```
!-----------------------------------------------------------------------
!       Declarations for the password variables
!-----------------------------------------------------------------------
STRING;ENTER_PSWD_CRT(MAX_CRT_NO)[3]
NUMERIC;       CURRENT_DSP(MAX_CRT_NO)
NUMERIC;       OLD_DSP(MAX_CRT_NO)


!-----------------------------------------------------------------------
!       Declarations for the login status messages
!-----------------------------------------------------------------------
STRING;PSWD_ERROR_MSG[]:"Invalid Password Entered - Try Again         "
STRING;VALID_PSWD_MSG[]:"Password Accepted, Modify Data and Logout!   "
STRING;CLEAR_MSG[]     :"Cleared Password - Enter Password to continue"

!-----------------------------------------------------------------------
!          This is the defined list of passwords for the system
!          three Digit passwords corresponding to operator initials.
!-----------------------------------------------------------------------


STRING;         PSWD_LIST(PSWD_LMT)[3]     \
                            :"AAA"\     ! Operator 1
                            :"BBB"\     ! Maitenance
                            :"CCC"\     ! Technician 1
                            :"DDD"\     ! Engineer
                            :"EEE"      ! Manager

LONG;         PSWD_LEVEL(PSWD_LMT)  \
                            :1            \     ! Operator 1
                            :5            \     ! Maitenance
                            :5            \     ! Technician 1
                            :10           \     ! Engineer
                            :20                 ! Manager

!-------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!-------------------------------------------------------------------
TABLES;


!-------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!-------------------------------------------------------------------
RESTART;
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 1.2-C (cont)

```
!-------------------------------------------------------------------
!      Monitor each CRT for a display screen other than the main menu
!-------------------------------------------------------------------
RECALL; RDDSP, , CRT1, CURRENT_DSP(CRT1_PTR)



!--------------------------------------------------------------------------------
!      Have a Password to test whenever the Enter Password field on
!      the display screen is not the Default Password (???)
!--------------------------------------------------------------------------------
RECALL; CMPRS, GET_PSWD_CRT1, \
        ENTER_PSWD_CRT(CRT1_PTR), QUESTION_PSWD, ONE, PSWD_PTR_CRT1

NEW_PSWD_CRT1 = ((PSWD_PTR_CRT1 < ZERO) & (CURRENT_DSP(CRT1_PTR) > 0) & \
                 GET_PSWD_CRT1)

!--------------------------------------------------------------------------------
!      Determine if the password entered was found in the list of
!      defined valid passwords.
!--------------------------------------------------------------------------------


GOOD_PSWD_CRT1 = ((PSWD_INDEX_CRT1 >= ZERO) & TEST_PSWD_CRT1)

BAD_PSWD_CRT1 = ((PSWD_INDEX_CRT1 < ZERO) & TEST_PSWD_CRT1)


!--------------------------------------------------------------------------------
!          Clear the password on a screen change
!--------------------------------------------------------------------------------
RESET_PSWD_CRT1 = (OLD_DSP(CRT1_PTR) <> CURRENT_DSP(CRT1_PTR))
OLD_DSP(CRT1_PTR) = CURRENT_DSP(CRT1_PTR)

!--------------------------------------------------------------------------------
!          The State Ring contains Two paths, one for a valid
!          password and the other for an invalid password.
!--------------------------------------------------------------------------------

VALID_PSWD_CRT1   = (TEST_PSWD_CRT1 & GOOD_PSWD_CRT1) | \
                    VALID_PSWD_CRT1 & ~GET_PSWD_CRT1

INVALID_PSWD_CRT1 = (TEST_PSWD_CRT1 & BAD_PSWD_CRT1) | \
                    INVALID_PSWD_CRT1 & ~GET_PSWD_CRT1

TEST_PSWD_CRT1    = (GET_PSWD_CRT1 & NEW_PSWD_CRT1) | \
                    TEST_PSWD_CRT1 & (~VALID_PSWD_CRT1 & ~INVALID_PSWD_CRT1)

GET_PSWD_CRT1     = (VALID_PSWD_CRT1 & RESET_PSWD_CRT1) | \
                    INVALID_PSWD_CRT1 | \
                    GET_PSWD_CRT1 & ~TEST_PSWD_CRT1
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 1.2-C (cont)

```
!--------------------------------------------------------------------------------
!          Compare Current password to all Valid Passwords.
!--------------------------------------------------------------------------------
CALL; CMPRS, TEST_PSWD_CRT1, \
           ENTER_PSWD_CRT(CRT1_PTR), PSWD_LIST(0), PSWD_LMT, PSWD_INDEX_CRT1


!-------------------------------------------------------------------------------
! On a valid password send the defined account number to the array variable
! which will correspond to the crt number minus 1.  For example CRT1 = array(0)
! CRT2 = array(1). The account number will change the login security level
! for the PCWS. The account number used for this application is "1", this
! number can be any valid account number from 1 to 31.
!-------------------------------------------------------------------------------
SET; VALID_PSWD_CRT1, SET_LOG_NUM_ARRAY(CRT1_PTR) = PSWD_LEVEL(PSWD_INDEX_CRT1)

SET; RESET_PSWD_CRT1, SET_LOG_NUM_ARRAY(CRT1_PTR) = DEFAULT_PSWD_LEVEL


!-------------------------------------------------------------------------------
!          Clear Password / Send error messages
!-------------------------------------------------------------------------------
CALL; SCOPY, (VALID_PSWD_CRT1 | INVALID_PSWD_CRT1),  \
    QUESTION_PSWD, ZERO, ENTER_PSWD_CRT(CRT1_PTR), ZERO, ONE

CALL; SCOPY, INVALID_PSWD_CRT1, PSWD_ERROR_MSG, ZERO, PCWS_CRT1_MSG, ZERO, ONE

CALL; SCOPY, VALID_PSWD_CRT1, VALID_PSWD_MSG, ZERO, PCWS_CRT1_MSG, ZERO, ONE

CALL; SCOPY, GET_PSWD_CRT1, CLEAR_MSG, ZERO, PCWS_CRT1_MSG, ZERO, ONE


END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

CHART/II Security

If access to some or all user screens must be controlled with security, this can be accomplished using the security features of the VMS operating system. First, set up VMS login accounts using the UAF utilities (refer to VMS manuals for specific instructions). Give each user a separate directory and place a LOGIN.COM file in each home directory. The following examples define how an OPERATOR and ADMIN account might be set up.

```
$!
$!------------------------------------------
$! File: DISK$USER:[OPERATOR]LOGIN.COM
$!------------------------------------------
$!
$!
$ @[crisp]crisp_login
$ @[crisp]user_login_ch2
$ set def [crisp.ch2.operator_dsp]
$ cht operator_menu
$ lo
```

```
$!
$!------------------------------------------
$! File: DISK$USER:[ADMIN]LOGIN.COM
$!------------------------------------------
$!
$!
$ @[crisp]crisp_login.com
$ @[crisp]user_login_ch2
$ set def [crisp.ch2.admin_dsp]
$ cht admin_menu
$ lo
```

In both examples, CHART/II starts up with a custom menu display. The user may move to successive displays and perform his duties. To log out, press the PF4 key on the keyboard and CHART/II exits and returns to the command program, which will perform a VMS logout.

Note that the CHART/II screens for the two accounts are also located in separate directories. This means that both directories need a copy of the CTPINI.CTP file in order for CHART/II to run. Also note that any screens that must be accessable to both users should be duplicated in both directories.

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

General

This section defines how to configure the various types of alarm handling functions available on CRISP/32.

This section is broken down into the following subsections.

Subsection	**Description**
Alarm Messages to Disk or Files *(page 3-3)*	Defines how to send alarm messages to disk files and to printers.
File Maintenance *(page 3-11)*	Defines how to purge/delete alarm files.
Alarm Scrolling *(page 3-15)*	Defines how to send a CRISP alarm to a CRISP display.
Alarm Scrolling, 2 Lines *(page 3-21)*	Defines how to display two line CRISP alarm messages.
Group Alarm Acknowledgment *(page 3-29)*	Defines how to perform a group alarm acknowledgment.
Inidividual Alarm Acknowledgment *(page 3-35)*	Defines how to perform an individual alarm acknowledgment.
Alarm Enabling/Disabling *(page 3-43)*	Defines two methods of how to enable and disable alarm handling.

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Alarm Messages to Disk or File

The CRISP/32 User Logic can be used to send Alarm messages to disk files and to printers. Although messages can be sent to a printer, it is important to remember that the printer must not be a spooled device.

NOTE

The method described here can be used to send any type of message to a printer or to a file. It is not limited to only alarm messages.

When sending messages to a file, it is often necessary to open a new file at a specific time or when a specific event occurs. In the example to follow, a new file is opened at the beginning of each month. Once the user understands the conditions that will open a new file every month, the user will understand how to change the conditon to open a new file every day or at the beginning of a new batch.

Solution

The user can accomplish the task using the following CRISP Function Calls:

- FIXDAT
- NEW_DESTINATION
- SETMSG_C16
- STR_MERGE
- TIME_AND_DATE
- VALUE_C16

Explanation

Designate one of the 32 CRISP Logical variables in the Message Mask to represent the CRISP Alarm file (i.e., `CRISP_ALMFIL`). Designate another Logical variable in the Message Mask to represent a printer device (i.e., `PRINTER1`).

NOTE

The device PRINTER1 does not appear in the Message Mask defined with the SETMSG_C16 Function Call in the CRISP/32 Function Calls Reference Manual. Any of the devices that appear in the Message Mask may be renamed to be more representative of their destination.

If the `DISK$USER:[CRISP.ALM]` directory does not exist, it can be created using the following VMS command.

(Continued on next page.)

© 1991 Square D All Rights Reserved

Explanation (cont)

```
$ Create /Dir [crisp.alm]
```

Lines of code placed after `RESTART;` test the value of the current month to determine when it changes. At the beginning of each month, `ALARM_FILE_SPEC` is given a new value. `ALARM_FILE_SPEC` is a string with the format `ALMFIL.MMM-YYYY`. `ALARM_FILE_SPEC` is the second argument to one of two NEW_DESTINATION Function Calls used in the example to follow. The NEW_DESTINATION Function Call will execute on the first pass of logic and again with each new month.

The Message Mask variables `CRISP_ALMFIL` and `PRINTER1` must be set TRUE before the VALUE_C16 Function Call. The same Message Mask variables must also be cleared to FALSE after the VALUE_C16 Function Call. The message created by the VALUE_C16 Function Call will be sent to each device or file whose Logical variable in the Message Mask is TRUE when the VALUE_C16 Function Call executes.

Example 2.1

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to send messages to printers and files. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follow.

```
!***********************************************************************
!*   Sample code for Example 2.1                                       *
!*                                                                     *
!*   Task 2.1 - Log CRISP alarms to a File and/or Printer              *
!***********************************************************************

!-----------------------------------------------------------------------
!   Code for the Declaration Section (before "TABLES")
!-----------------------------------------------------------------------

!-----------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!-----------------------------------------------------------------------
LOGICAL;      NEW_DB:true, NEW_CLE, ACTIVE, CACDSA, RESV5, RESV6,\
              RESV7, RESV8, RESV9, RESV10, RESV11, ICCDSA, \
              PAUSE, RESV14, RESV15, RESV16
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.1 (cont)

```
!--------------------------------------------------------------------------
!               The follow 32 logical make up the "Message Mask".
!               These are variables that represent each potential print
!               device or file were messages can be routed.
!--------------------------------------------------------------------------
     LOGICAL;        CRISP_TT:TRUE, CRISP_LP     , CRISP_CL     , CRISP_SS     , \
                     CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                     CRISP_DEV00  , PRINTER1     , PRINTER2     , DECTALK_DEV , \
                     CRISP_DEV04  , REPORT_FILE , CRISP_DEV06 , CRISP_DEV07 , \
                     CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                     CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                     CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                     CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


!--------------------------------------------------------------------
!      The following variables are used for the TIME_AND_DATE
!      Function Call and to detect a new month.
!--------------------------------------------------------------------
NUMERIC;      YEAR_N,       MONTH_N,      DAY_N,        HOUR_N,\
              MIN_N,               SEC_N,        HNDRD_N, , , , , , , , , ,


!---------------------------------------
!        constants
!---------------------------------------
CONSTANT;       ZERO:0,             ONE:1,\
                NUMBER_OF_ANALOG_ALARMS:100

!---------------------------------------------------------------
!      The folling variables will be used as offsets in the
!      STR_MERGE Function Call.
!---------------------------------------------------------------
NUMERIC;      FOUR:4,       OFFSET:29

!--------------------------------------------------------
!      Variable used to determine change of months.
!--------------------------------------------------------
NUMERIC;      OLD_MONTH


!----------------------------------------
!      array index for checking alarms
!----------------------------------------
NUMERIC;        ALARM_INDEX,\
                INDEX
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.1 (cont)

```
!------------------------------------------
!    Event flag for the VALUE_C16 call.
!------------------------------------------
LOGICAL;       NEW_ALARM


!--------------------------------------------
!    Variables for points in the system.
!--------------------------------------------
STRING;         LABEL_STR(100)[8],\
                ID_NAME_STR(100)[16],\
                ALM_MSG(100)[40]


!-----------------------------------------------------------
!              Variables used in alarm checking
!-----------------------------------------------------------
LOGICAL;        ALARM_DISABLED(100),\
                NORMAL(100),\
                ALARM(100),\
                HIGH_ALARM(100),\
                LOW_ALARM(100),\
                ALARM_ON(100)

FLOAT;         DEADBAND_LIMITS(100),\
               HIGH_ALARM_LIMITS(100),\
               LOW_ALARM_LIMITS(100),\
               ANALOG_TEMP(100)


!----------------------------------------------
!       Strings used for alarm messages.
!----------------------------------------------
STRING; \
        SPACE[]:      " ",           \
        LOW_STR[]:"Lo Alarm  ",\
        HIGH_STR[]:"Hi Alarm  ",\
        ALARM_TYPE(100)[10]


!---------------------------------------------------------
!      The following Strings will receive the current
!      date and time.
!---------------------------------------------------------
STRING; \
       DATE_STR[11],                \
       TIME_STR[8]
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.1 (cont)

```
!----------------------------------------------------------------------
!       The name of the file that all alarms will be sent to
!       id declared in the String ALARM_FILE_SPEC.  The current
!       month and year will be merged into the string.  If the
!       current month and year is JULY 1992, then the value of
!       ALARM_FILE_SPEC will be DISK$USER:[CRISP.ALM]ALMFIL.JUL-1992.
!       This will be an ASCII file that can be viewed with the VMS
!       TYPE command or with a DEC Editor such as TPU.
!----------------------------------------------------------------------
STRING;ALARM_FILE_SPEC[36]:"DISK$USER:[CRISP.ALM]ALMFIL. "

!-------------------------------------------------------------
!       The following string has a value which is a print
!       device.
!-------------------------------------------------------------
STRING;         PRINTER1_STR[7]:"LTA154:"

!---------------------------------------------------------------------
!      VARVAL is a variable value.
!      The value of the variable TERMINATOR will tell the
!      printer to do the following:
!
!      •  @G = Ring bell
!      •  @M = Carriage return
!      •  @J = Line feed
!----------------------------------------------------------------------
STRING;VARVAL[]:"\\\\.\\",    \
              TERMINATOR[]:"@G@M@J"


!-----------------------------------------------------------------
!       This varible is used to control the RMS file options for
!       the NEW_DESTINATION Function Call.  If the value is zero,
!       the file will be opened for append.  If the value is one,
!       a new version of the file will be opened.
!-----------------------------------------------------------------
LONG;  RMS_OPTIONS:0




!---------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!---------------------------------------------------------------------
TABLES;
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.1 (cont)

```
!--------------------------------------------------------------
!       This Function Call establishes the Message Mask using
!       32 CRISP Logical variables beginning with CRISP_TT.
!--------------------------------------------------------------
RECALL; SETMSG_C16, , CRISP_TT

!------------------------------------------------------------
!       This Function Call will use YEAR_N and the next six
!       Numeric variables following YEAR_N to receive the
!       current values for Time and Date.
!------------------------------------------------------------
RECALL; TIME_AND_DATE, , YEAR_N

!---------------------------------------------------------------
!      This Function Call will perform a VMS Define by assigning
!      the value of PRINTER1_STR to the Logical Variable
!      PRINTER1.
!---------------------------------------------------------------
CALL; NEW_DESTINATION, ,PRINTER1, PRINTER1_STR, RMS_OPTIONS




!--------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!--------------------------------------------------------------------
RESTART;


!------------------------------------------------------------------
!       Update the Date & Time Strings to current values.
!------------------------------------------------------------------
RECALL;FIXDAT, , DATE_STR
RECALL;FIXTIM_C16, , TIME_STR

!------------------------------------------------------------------
!       During each pass of logic, the following three lines
!       of code will test the value of the current month to
!       determine if it has changed.  At the beginning of a
!       new month, a new alarm file is created with a name in
!       the format ALMFIL.MMM-YYYY.  All alarms from this Logic
!       will go to the most recently created alarm file.
!------------------------------------------------------------------
CALL;   SMERGE_C16, (MONTH_N <> OLD_MONTH), \
        DATE_STR, FOUR, ALARM_FILE_SPEC, OFFSET

CALL;   NEW_DESTINATION,(MONTH_N <> OLD_MONTH),\
        CRISP_ALMFIL, ALARM_FILE_SPEC, RMS_OPTIONS
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.1 (cont)

```
SET;    (MONTH_N <> OLD_MONTH), OLD_MONTH = MONTH_N


ALARM_INDEX = 0
LABEL;NEXT_ANALOG

!---------------------------------------------------------------------
!       This call will check the current temperature "ANALOG_TEMP"
!       against the hi/lo limits and set a hi/lo logical and an
!       alarm bit when in alarm.
!---------------------------------------------------------------------
    RECALL; ABSALM_C16, , \
           ANALOG_TEMP(ALARM_INDEX), \
           LOW_ALARM_LIMITS(ALARM_INDEX), \
           HIGH_ALARM_LIMITS(ALARM_INDEX), \
           DEADBAND_LIMITS(ALARM_INDEX), \
           LOW_ALARM(ALARM_INDEX), \
           HIGH_ALARM(ALARM_INDEX), \
           NORMAL(ALARM_INDEX),\
           ALARM(ALARM_INDEX)

!-----------------------------------------------------------
!   A point is in alarm whenever there is a
!   low alarm, or a high alarm for that point.
!-----------------------------------------------------------
    ALARM(ALARM_INDEX) = LOW_ALARM(ALARM_INDEX) | HIGH_ALARM(ALARM_INDEX)

    NEW_ALARM          = ALARM(ALARM_INDEX) & ~ALARM_ON(ALARM_INDEX)

    ALARM_ON(ALARM_INDEX) = ALARM(ALARM_INDEX)


!------------------------------------------------------------
!  Determine what type of alarm (high/low), and
!  insert the hi/low string into the ALARM_TYPE
!  array, this will become part of the alarm message.
!------------------------------------------------------------
    RECALL; SCOPY, HIGH_ALARM(ALARM_INDEX),\
            HIGH_STR, ZERO, ALARM_TYPE(ALARM_INDEX),ZERO, ONE

    RECALL; SCOPY, LOW_ALARM(ALARM_INDEX),\
            LOW_STR, ZERO, ALARM_TYPE(ALARM_INDEX),ZERO, ONE

!--------------------------------------------------------------
!      Clearing CRISP_TT to FALSE means the alarm messages
!      will NOT be sent to the System Console.
!--------------------------------------------------------------
         CRISP_TT     = FALSE
         CRISP_ALMFIL = TRUE
         PRINTER1     = TRUE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.1 (cont)

```
!-------------------------------------------------------------------
!       This VALUE_C16 Function Call will send the following
!       messages to PRINTER1 and CRISP_ALMFIL:
!
!       •  Current Date and Time
!       •  A Label
!       •  An ID Name
!       •  An Alarm Message
!       •  An Alarm Type
!       •  The Current Temperature
!       •  The High Limit
!       •  The Low Limit
!-------------------------------------------------------------------
         RECALL; VALUE_C16,NEW_ALARM ,\
                 DATE_STR, SPACE, \
                 TIME_STR, SPACE, \
                 LABEL_STR(ALARM_INDEX), SPACE, \
                 ID_NAME_STR(ALARM_INDEX), SPACE,\
                 ALM_MSG(ALARM_INDEX), SPACE,\
                 ALARM_TYPE(ALARM_INDEX), SPACE, \
                 VARVAL, ANALOG_TEMP(ALARM_INDEX), SPACE, \
                 VARVAL, HIGH_ALARM_LIMITS(ALARM_INDEX), SPACE, \
                 VARVAL, LOW_ALARM_LIMITS(ALARM_INDEX), SPACE, \
                 TERMINATOR



!--------------------------------------------------------------
!      Reset the Message Mask variables to enable the
!      System Console to receive system messages.
!--------------------------------------------------------------
          CRISP_TT     = TRUE
          CRISP_ALMFIL = FALSE
          PRINTER1     = FALSE



ALARM_INDEX = ALARM_INDEX + 1

!---------------------------------------------------------------
!      Reset the event flag for the VALUE_C16 Function Call.
!---------------------------------------------------------------
RECALL; MAKZERO, , NEW_ALARM


JUMP;  NEXT_ANALOG, ALARM_INDEX < NUMBER_OF_ANALOG_ALARMS


END;
```


© 1991 Square D All Rights Reserved

File Maintenance

The user can accomplish file maintenance for logging alarms to disk using the Function Call SUBMIT_BATCH.

Explanation

The SUBMIT_BATCH call has an argument FILE_PURGE_BATCH_FILESPEC, which will be the name of a DCL command procedure file. The COM file will delete any old log files using the /BEFORE qualifier.

Example 2.1 B

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to delete/purge alarm files from disk. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!**************************************************************************
!*   Sample code for Example 2.1 B                                       *
!*                                                                       *
!*   Task 2.1 B  - File Maintenance for logging alarms to disk.          *
!**************************************************************************

!-------------------------------------------------------------------------
!   Code for the Declaration Section (before "TABLES")
!-------------------------------------------------------------------------

!--------------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!--------------------------------------------------------------------------
LOGICAL;      NEW_DB:true, NEW_CLE, ACTIVE, CACDSA, RESV5, RESV6,\
              RESV7, RESV8, RESV9, RESV10, RESV11, ICCDSA, \
              PAUSE, RESV14, RESV15, RESV16

!--------------------------------------------------------------------------
!              The follow 32 logical make up the "Message Mask".
!              These are variables that represent each potential print
!              device or file were messages can be routed.
!--------------------------------------------------------------------------
     LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                     CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                     CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                     CRISP_DEV04  , REPORT_FILE , CRISP_DEV06 , CRISP_DEV07 , \
                     CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                     CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                     CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                     CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.1 B (cont)

```
!------------------------------------------------------------------
!      The following variables are used for the TIME_AND_DATE
!      Function Call and to detect a new month.
!------------------------------------------------------------------
NUMERIC;      YEAR_N,MONTH_N,      DAY_N,        HOUR_N,\
              MIN_N,        SEC_N,        HNDRD_N, , , , , , , , , ,

!-----------------------------------------------------------------------------
!             Declarations for "File Purging" function (Alarm Logs)
!-----------------------------------------------------------------------------
LOGICAL;      FILE_PURGE_SUBMIT_FLAG,                             \
              FILE_PURGE_DONE_FLAG,                        \
              FILE_PURGE_PRINT_FLAG:FALSE,                 \
              FILE_PURGE_NOTIFY_FLAG:TRUE,                 \
              FILE_PURGE_FIRST_PASS:TRUE

NUMERIC;      FILE_PURGE_OLD_DAY_NUMBER:0,                 \
              NEW_DAY_NUMBER

LONG;         FILE_PURGE_STATUS

STRING;\
       FILE_PURGE_BATCH_FILESPEC[]:"DISK$USER:[CRISP.ALM]ALMFIL_PURGE.COM", \
       FILE_PURGE_AFTER_TIME[]:"00:01:00"




!--------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!--------------------------------------------------------------------
TABLES;




!--------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!--------------------------------------------------------------------
RESTART;


RECALL; YRDAY_C16, ~FILE_PURGE_FIRST_PASS, YEAR_N, NEW_DAY_NUMBER

!-----------------------------------------------------------------------------
!      Code for "File Purging" function (Alarm Logs)
!
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.1 B (cont)

```
!       The system will automatically purge Alarm log files at a
!       given time each day, retaining only those created during
!       the last ninety days.  This code uses the CRISP/32 function
!       call "SUBMIT_BATCH" to submit a DCL command procedure file
!       for execution in a batch job queue.  The DCL command procedure
!       filespec is "DISK$USER:[CRISP.ALM]ALMFIL_PURGE.COM", and
!       includes the following command lines:
!
!             $ SET DEF DISK$USER:[CRISP.ALM]
!             $ DELETE ALMFIL.*;* /LOG/BEFORE="-90-::"
!-------------------------------------------------------------------------------
LET;    , FILE_PURGE_SUBMIT_FLAG = ~FILE_PURGE_FIRST_PASS & \
       (NEW_DAY_NUMBER <> FILE_PURGE_OLD_DAY_NUMBER)

CALL;   SUBMIT_BATCH, FILE_PURGE_SUBMIT_FLAG, FILE_PURGE_DONE_FLAG, \
       FILE_PURGE_STATUS, FILE_PURGE_BATCH_FILESPEC, , \
       FILE_PURGE_AFTER_TIME, FILE_PURGE_PRINT_FLAG, FILE_PURGE_NOTIFY_FLAG

LET;    ~FILE_PURGE_FIRST_PASS, FILE_PURGE_OLD_DAY_NUMBER = NEW_DAY_NUMBER


!---------------------------------------------------------
! "TIME_AND_DATE" has had enough time to load "YEAR_N"
!---------------------------------------------------------
FILE_PURGE_FIRST_PASS = FALSE



END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Alarm Scrolling

The user can send a CRISP Alarm to a CRISP Display using the Function Call DISPLAY_MESSAGES.

Explanation

The Alarm Handler should have some way of identifying alarms. In the following sample code, a system is considered to have 100 analog points that can alarm based on a value going outside of the points high or low limits. The CRISP call used to identify the alarms is the Function Call ABSALM_C16.

ALARM_ON is used to mark a point which is either a high alarm, or a low alarm. ALARM_ON is used instead of ALARM because the variable ALARM within the Function Call ABSALM_C16 gets set when a point goes into alarm, but then the next pass of logic it is reset.

The DISPLAY_MESSAGES Function Call displays alarm messages in a scrolling region based on a logical variable DISPLAY_ALARM which is updated by ALARM_ON. As long as the logical DISPLAY_ALARM is true, the associated array element in LABEL_STR, ALM_MSG, ALARM_TYPE, and CR all appear as one long string in the variable ALM_DISPLAY_STR, which would be linked to a CRISP display, forming an alarm scrolling region.

Each DISPLAY_MESSAGES call has its own `DISPLAY_RETURN_STATUS` variable, and `DSP_MSG_MASK` variable. The `DSP_MSG_MASK` is a Display Message Mask, which is an array of 32 bits.

- When the first bit (array location zero) is set to TRUE, a time and date stamp is attached on the front end of each message sent to the scrolling region.
- The second bit (array location one) enables a string substitution. In the DISPLAY_MESSAGES call, the two variables following the DSP_MSG_MASK argument are the true_string and false_string. Those strings are substituted into the alarm message in place of a logical `1` or `0` when the second bit in the display message mask is TRUE and when an array of logicals is given as part of the message created in the DISPLAY_MESSAGES Function Call.
- The third bit (array location two) enables a blink option using the ALM_BLINK argument of the Call. ALM_BLINK is an array of logicals, which can be associated with each line in the scrolling region.

Example 2.2

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to send alarm messages to a display linked as a scrolling region. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.2 (cont)

```
!*********************************************************************
!*      Sample code for Example 2.2                                  *
!*                                                                   *
!*      Task 2.2 - How to send a CRISP Alarm to a CRISP Display.     *
!*********************************************************************

!-------------------------------------------------------------------
!   This is an example of a system containing (100) analog points.
!-------------------------------------------------------------------

!-------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!-------------------------------------------------------------------

!-----------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!-----------------------------------------------------------------------
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, DEBUG , DEBUG_BREAK ,


!-----------------------------------------------------------------------
!               The follow 32 logical make up the "Message Mask".
!               These are variables that represent each potential print
!               device or file were messages can be routed.
!-----------------------------------------------------------------------
LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


!--------------------------------------
!         constants
!--------------------------------------
CONSTANT;      ZERO:0,       ONE:1,\
               NUMBER_OF_ANALOG_ALARMS:100

!---------------------------------------
!      array index for checking alarms
!---------------------------------------
NUMERIC;       ALARM_INDEX,\
               INDEX
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.2 (cont)

```
!--------------------------------------------------------------------------
!      This logical would be defined for each point indicating
!      a critical/noncritical point, depending on its value.
!      Then when this point goes into alarm, the DISPLAY_MESSAGES
!      call can substitute a string into the scrolling region based
!      on the logical to indicate when a critical point goes into alarm.
!---------------------------------------------------------------------------
LOGICAL;      CR(100)


!------------------------------------------------------------
!             Variables used in alarm checking
!------------------------------------------------------------
LOGICAL;      ALARM_DISABLED(100),\
              NORMAL(100),\
              ALARM(100),\
              HIGH_ALARM(100),\
              LOW_ALARM(100),\
              ALARM_ON(100)

FLOAT;        DEADBAND_LIMITS(100),\
              HIGH_ALARM_LIMITS(100),\
              LOW_ALARM_LIMITS(100),\
              ANALOG_TEMP(100)


!---------------------------------------
!      Alarms to be displayed
!---------------------------------------
LOGICAL;      DISPLAY_ALARM(100)

!-------------------------------------------------------------------------
!       Size of scrolling region for display of alarms, the scrolling
!       region is set to the maximun number of possible alarms so that
!       alarm messages will never be lost, out of the boundaries of the
!       scrolling regions, however the alarm display may only have the
!       first 20-25 slots within the scrolling regions actually linked
!       for view.
!-------------------------------------------------------------------------
NUMERIC;      NUM_LINES:100

!-------------------------------------------------------------------------
!       Option mask to be used in the DISPLAY_MESSAGES Function Call.
!-------------------------------------------------------------------------
LOGICAL;      DSP_MSG_MASK(32)

!---------------------------------------------------------------
!        Variables used in the DISPLAY_MESSAGES call
!---------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.2 (cont)

```
LOGICAL;         ALM_BLINK(100),\
                 GEN_BLINK,\
                 GEN_ALARM,\
                 ALARM_ACK

STRING;          LABEL_STR(100)[8],\
                 ALM_MSG(100)[30]

!-------------------------------------------------------------------
!      String declarations for Display_Message scrolling region.
!-------------------------------------------------------------------
STRING;  ALM_DISPLAY_STR(100)[75]


!--------------------------------------------------------------------
!         Return Status for the DISPLAY_MESSAGES Function Call.
!         The statuses and their meanings are listed in the
!         documentation for this call.
!--------------------------------------------------------------------
LONG;          DISPLAY_RETURN_STATUS


!---------------------------------------------
!       Strings used for displaying messages.
!---------------------------------------------
STRING; \
        LOW_STR[]:"Lo Alarm  ",\
        HIGH_STR[]:"Hi Alarm  ",\
        CR_STR[2]:"CR"  ,\
       BLANK_MESSAGE[]:" ",\
        ALARM_TYPE(100)[10]



!---------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!---------------------------------------------------------------------
TABLES;


RECALL; SETMSG_C16, , CRISP_TT

!--------------------------------------------------------------------------
!     Put date and time in front of each message in scroll message region.
!--------------------------------------------------------------------------
DSP_MSG_MASK(0) = TRUE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.2 (cont)

```
!-----------------------------------
!     Enable string substitution.
!-----------------------------------
DSP_MSG_MASK(1) = TRUE

!--------------------------
!     Enable blink option.
!--------------------------
DSP_MSG_MASK(2)   = TRUE


!----------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;

!-----------------------------------
!      Alarm Checking for ANALOGS
!-----------------------------------

ALARM_INDEX = ZERO
LABEL; NEXT_ANALOG

!-------------------------------------------------------------
! If an alarm is disabled then jump over the code
! for checking alarms, to the label JUMP_TO_NEXT_ANALOG.
!-------------------------------------------------------------
JUMP; JUMP_TO_NEXT_ANALOG,  ALARM_DISABLED(ALARM_INDEX)

!---------------------------------------------------------------------
!       This call will check the current temperature "ANALOG_TEMP"
!       against the hi/lo limits and set a hi/lo logical and an
!       alarm bit when in alarm.
!---------------------------------------------------------------------
    RECALL; ABSALM_C16, , \
           ANALOG_TEMP(ALARM_INDEX), \
           LOW_ALARM_LIMITS(ALARM_INDEX), \
           HIGH_ALARM_LIMITS(ALARM_INDEX), \
           DEADBAND_LIMITS(ALARM_INDEX), \
           LOW_ALARM(ALARM_INDEX), \
           HIGH_ALARM(ALARM_INDEX), \
           NORMAL(ALARM_INDEX),\
           ALARM(ALARM_INDEX)

!------------------------------------------------
!   A point is in alarm whenever there is a
!   low alarm, or a high alarm for that point.
!------------------------------------------------
    ALARM_ON(ALARM_INDEX) = LOW_ALARM(ALARM_INDEX) | HIGH_ALARM(ALARM_INDEX)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.2 (cont)

```
!----------------------------------------------------------
!  Determine what type of alarm (high/low), and
!  insert the hi/low string into the ALARM_TYPE
!  array, to be displayed in the scrolling region.
!----------------------------------------------------------
    RECALL; SCOPY, HIGH_ALARM(ALARM_INDEX),\
            HIGH_STR, ZERO, ALARM_TYPE(ALARM_INDEX),ZERO, ONE

    RECALL; SCOPY, LOW_ALARM(ALARM_INDEX),\
            LOW_STR, ZERO, ALARM_TYPE(ALARM_INDEX),ZERO, ONE

!----------------------------------------------
!  Mark all the analog alarms to be displayed.
!----------------------------------------------
     DISPLAY_ALARM(ALARM_INDEX) = ALARM_ON(ALARM_INDEX)

LABEL; JUMP_TO_NEXT_ANALOG

ALARM_INDEX = ALARM_INDEX + 1

JUMP;  NEXT_ANALOG, ALARM_INDEX < NUMBER_OF_ANALOG_ALARMS


!---------------------------------------------------------------------
!       This Display call will place all the strings which make
!       up the message into ALM_DISPLAY_STR (which will be linked
!       to a CRISP display) for each element of the array whose
!       DISPLAY_ALARM bit is on.  The CRISP display will consist of
!       strings the same length as ALM_DISPLAY_STR and as many
!       as you want lines in your scrolling region.  Linked to
!       ALM_DISPLAY_STR(0), ALM_DISPLAY_STR(1), ALM_DISPLAY_STR(2),...
!---------------------------------------------------------------------
RECALL;DISPLAY_MESSAGES, ,\
              DISPLAY_RETURN_STATUS, \
              DSP_MSG_MASK(0), \
              CR_STR, \
              BLANK_MESSAGE, \
              NUM_LINES, \
              ALM_DISPLAY_STR(0), \
              ALM_BLINK(0), \
              GEN_BLINK, \
              ALARM_ACK, \
              GEN_ALARM, \
              NUMBER_OF_ANALOG_ALARMS, \
              DISPLAY_ALARM(0), \
              LABEL_STR(0), \
              ALM_MSG(0),\
              ALARM_TYPE(0),\
              CR(0)


END;
```


© 1991 Square D All Rights Reserved

Alarm Scrolling (2 Lines)

Displaying two-line CRISP Alarm messages is similar to the previous example. However, the message going to the alarm scrolling region is too long to fit on one line, so we have to make a 2-line message. This is accomplished by using two DISPLAY_MESSAGES Function Calls.

Explanation

The Display_Message calls will each have a Dispay Return Status, and a Display Message Mask. The alarm trigger will be the same in each call, DISPLAY_ALARM. After the trigger, the variables in the DISPLAY_MESSAGES call are the pieces of each message for line 1 and line 2 of the scrolling region.

NOTE

The time and date stamp only needs to be on the first line of the messages, therefore only DSP_MSG_MASK(0) would be initialized to TRUE, while DSP_MSG_MASK_2(0) would remain FALSE.

Example 2.3

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to send 2-line alarm messages to a display linked as a scrolling region. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The Example Declarations and Logic follows.

```
!***************************************************************************
!*      Sample code for Example 2.3                                        *
!*                                                                         *
!*      Task 2.3 - How to display two-line CRISP Alarm to a CRISP Display. *
!***************************************************************************

!---------------------------------------------------------------------
!   This is an example of a system containing (100) analog points.
!---------------------------------------------------------------------

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------

!-------------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!-------------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.3 (cont)

```
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, DEBUG , DEBUG_BREAK ,

!-------------------------------------------------------------------------
!               The following 32 logicals make up the "Message Mask".
!               These are variables that represent each potential print
!               device or file were messages can be routed.
!-------------------------------------------------------------------------
LOGICAL;        CRISP_TT:TRUE, CRISP_LP     , CRISP_CL     , CRISP_SS     , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , PRINTER1     , PRINTER2     , DECTALK_DEV , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


!-------------------------------------
!         constants
!-------------------------------------
CONSTANT;       ZERO:0,             ONE:1,\
                NUMBER_OF_ANALOG_ALARMS:100

!--------------------------------------
!      array index for checking alarms
!--------------------------------------
NUMERIC;        ALARM_INDEX,\
                INDEX


!-------------------------------------------------------------------------
!      This logical would be defined for each point indicating
!      a critical/noncritical point, depending on its value.
!      Then, when this point goes into alarm, the DISPLAY_MESSAGES
!      call can substitute a string into the scrolling region based
!      on the logical to indicate when a critical point goes into alarm.
!--------------------------------------------------------------------------
LOGICAL;       CR(100)


!-------------------------------------------------------
!              Variables used in alarm checking
!-------------------------------------------------------
LOGICAL;        ALARM_DISABLED(100),\
                NORMAL(100),\
                ALARM(100),\
                HIGH_ALARM(100),\
                LOW_ALARM(100),\
                ALARM_ON(100)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.3 (cont)

```
FLOAT;          DEADBAND_LIMITS(100),\
                HIGH_ALARM_LIMITS(100),\
                LOW_ALARM_LIMITS(100),\
                ANALOG_TEMP(100)

!-------------------------------------
!      Alarms to be displayed
!-------------------------------------
LOGICAL;      DISPLAY_ALARM(100)

!----------------------------------------------------------------------
!       Size of scrolling region for display of alarms, the scrolling
!       region is set to the maximun number of possible alarms so that
!       alarm messages will never be lost, out of the boundaries of the
!       scrolling regions, however the alarm display may only have the
!       first 20-25 slots within the scrolling regions actually linked
!       for view.
!----------------------------------------------------------------------
NUMERIC;        NUM_LINES:100

!----------------------------------------------------------------------
!      Option masks used in the DISPLAY_MESSAGES Function Calls.
!----------------------------------------------------------------------
LOGICAL;        DSP_MSG_MASK(32),\
                DSP_MSG_MASK_2(32)

!-----------------------------------------------------------------
!        Variables used in the DISPLAY_MESSAGES Function Calls.
!-----------------------------------------------------------------
LOGICAL;        ALM_BLINK(100),\
                GEN_BLINK,\
                GEN_ALARM,\
                ALARM_ACK

STRING;         LABEL_STR(100)[8],\
                ID_NAME_STR(100)[16],\
                ALM_MSG(100)[40]

!------------------------------------------------------------------
!      String declarations for Display_Messages scrolling region.
!------------------------------------------------------------------
STRING;   ALM_DISPLAY_STR(100)[75],\
          ALM_DISPLAY_STR_2(100)[75]


!-------------------------------------------------------------------
!         Return Statuses for the DISPLAY_MESSAGES Function Call.
!         The statuses and their meanings are listed in the
!         documentation for this call.
!-------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.3 (cont)

```
LONG;           DISPLAY_RETURN_STATUS,\
                DISPLAY_RETURN_STATUS_2


!----------------------------------------------
!       Strings used for displaying messages.
!----------------------------------------------
STRING; \
        LOW_STR[]:"Lo Alarm  ",\
        HIGH_STR[]:"Hi Alarm  ",\
        CR_STR[2]:"CR"  ,\
        BLANK_MESSAGE[]:" ",\
        ALARM_TYPE(100)[10]



!----------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!----------------------------------------------------------------------
TABLES;


RECALL; SETMSG_C16, , CRISP_TT

!---------------------------------------------------------------------------
!     Put date and time in front of each message in scroll message region.
!---------------------------------------------------------------------------
DSP_MSG_MASK(0) = TRUE

!-----------------------------------
!     Enable string substitution.
!-----------------------------------
DSP_MSG_MASK(1) = TRUE


!---------------------------------------------------------
!     To Enable blink option for both lines
!     set DSP_MSG_MASK(2) = TRUE and set
!     DSP_MSG_MASK_2(2) = TRUE.
!---------------------------------------------------------


!----------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.3 (cont)

```
!----------------------------------
!      Alarm Checking for ANALOGS
!----------------------------------

ALARM_INDEX = ZERO
LABEL; NEXT_ANALOG

!---------------------------------------------------------------
! If an alarm is disabled then jump over the code
! for checking alarms, to the label JUMP_TO_NEXT_ANALOG.
!---------------------------------------------------------------
JUMP; JUMP_TO_NEXT_ANALOG,  ALARM_DISABLED(ALARM_INDEX)

!---------------------------------------------------------------------
!       This call will check the current temperature "ANALOG_TEMP"
!       against the hi/lo limits and set a hi/lo logical and an
!       alarm bit when in alarm.
!---------------------------------------------------------------------
    RECALL; ABSALM_C16, , \
            ANALOG_TEMP(ALARM_INDEX), \
            LOW_ALARM_LIMITS(ALARM_INDEX), \
            HIGH_ALARM_LIMITS(ALARM_INDEX), \
            DEADBAND_LIMITS(ALARM_INDEX), \
            LOW_ALARM(ALARM_INDEX), \
            HIGH_ALARM(ALARM_INDEX), \
            NORMAL(ALARM_INDEX),\
            ALARM(ALARM_INDEX)


!----------------------------------------------
!   A point is in alarm whenever there
!   is a low alarm, or a high alarm.
!----------------------------------------------
    ALARM_ON(ALARM_INDEX) = LOW_ALARM(ALARM_INDEX) | HIGH_ALARM(ALARM_INDEX)


!------------------------------------------------------------
!  Determine what type of alarm (high/low), and
!  insert the hi/low string into the ALARM_TYPE
!  array, to be displayed in the scrolling region.
!------------------------------------------------------------
    RECALL; SCOPY, HIGH_ALARM(ALARM_INDEX),\
            HIGH_STR, ZERO, ALARM_TYPE(ALARM_INDEX),ZERO, ONE

    RECALL; SCOPY, LOW_ALARM(ALARM_INDEX),\
            LOW_STR, ZERO, ALARM_TYPE(ALARM_INDEX),ZERO, ONE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.3 (cont)

```
!-----------------------------------------------
!  Mark all the analog alarms to be displayed.
!-----------------------------------------------
     DISPLAY_ALARM(ALARM_INDEX) = ALARM_ON(ALARM_INDEX)


LABEL; JUMP_TO_NEXT_ANALOG

ALARM_INDEX = ALARM_INDEX + 1

JUMP;  NEXT_ANALOG, ALARM_INDEX < NUMBER_OF_ANALOG_ALARMS



!----------------------------------------------------------------------
!       To display a two-line alarm message the following two
!       DISPLAY_MESSAGES calls would be used.  Notice the trigger
!       to display the alarm DISPLAY_ALARM would be the same in
!       both calls.  The scrolling region on the CRISP display
!       starting with the first line, would be linked to
!       ALM_DISPLAY_STR(0) and every other line to follow
!       would be linked to ALM_DISPLAY_STR(1), (2), (3), ....
!       On the same display starting with the 2nd line, link
!       it to ALM_DISPLAY_STR_2(0) and every other line to
!       follow link to ALM_DISPLAY_STR_2(1), (2), (3), ....
!----------------------------------------------------------------------
!----------------------------------------------------------------------
!      Display the 1st line of the Alarm Message on a screen display.
!----------------------------------------------------------------------
RECALL; DISPLAY_MESSAGES, ,\
        DISPLAY_RETURN_STATUS, \
        DSP_MSG_MASK(0), \
        CR_STR, \
        BLANK_MESSAGE, \
        NUM_LINES, \
        ALM_DISPLAY_STR(0), \
        ALM_BLINK(0), \
        GEN_BLINK, \
        ALARM_ACK, \
        GEN_ALARM, \
        NUMBER_OF_ANALOG_ALARMS, \
        DISPLAY_ALARM(0), \
        LABEL_STR(0), \
        ALM_MSG(0),\
        CR(0)

!----------------------------------------------------------------------
!      Display the 2nd line of the Alarm Message on a screen display.
!----------------------------------------------------------------------
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.3 (cont)

```
RECALL; DISPLAY_MESSAGES, ,\
        DISPLAY_RETURN_STATUS_2, \
        DSP_MSG_MASK_2(0), \
        , \
        , \
        NUM_LINES, \
        ALM_DISPLAY_STR_2(0), \
        , \
        , \
        , \
        , \
        NUMBER_OF_ANALOG_ALARMS, \
        DISPLAY_ALARM(0), \
        ID_NAME_STR(0),\
        ALARM_TYPE(0),\
        ANALOG_TEMP(0), \
        HIGH_ALARM_LIMITS(0), \
        LOW_ALARM_LIMITS(0)


END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Group Alarm Acknowledgment

Group Alarm Acknowledgment is accomplished by creating an array of logicals to represent unacknowledged alarms. When a group acknowledgment occurs, all elements in the array UNACK_ALARM are set to FALSE to indicate acknowledgement.

Explanation

When a point goes into alarm, the UNACK_ALARM logical for that point will be set to TRUE. A group acknowledgement button will be on the Alarm Display, and, when the button is set, that will be the condition to trigger a BZER0 Function Call, which sets all elements in the UNACK_ALARM array to FALSE.

The trigger DISPLAY_ALARM that forces the messages into the scrolling regions is TRUE when ALARM_ON or UNACK_ALARM is TRUE. This means that alarms will appear in the scrolling region as long as one of the two conditions are TRUE.

Example 2.4

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to perform group acknowledgment of alarm messages. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 2.4                                        *
!*                                                                         *
!*      Task 2.4 - An example of Group Alarm Acknowledgement.              *
!***************************************************************************

!---------------------------------------------------------------------
!   This is an example of a system containing (100) analog points.
!---------------------------------------------------------------------

!-------------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!-------------------------------------------------------------------------
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, DEBUG , DEBUG_BREAK ,

!-------------------------------------------------------------------------
!               The follow 32 logical make up the "Message Mask".
!               These are variables that represent each potential print
!               device or file were messages can be routed.
!-------------------------------------------------------------------------
```

(Continued on next page.)

Example 2.4 (cont)

```
LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


!---------------------------------------
!         constants
!---------------------------------------
CONSTANT;       ZERO:0,             ONEHUNDRED:100, \
                NUMBER_OF_ANALOG_ALARMS:100

!----------------------------------------
!     Array indexes for checking alarm
!----------------------------------------
NUMERIC;        ALARM_INDEX,\
                INDEX

!------------------------------------------------
!     Variables for each point in the system.
!------------------------------------------------
STRING;         LABEL_STR(100)[8],\
                ID_NAME_STR(100)[16],\
                ALM_MSG(100)[40]

FLOAT;          DEADBAND_LIMITS(100),\
                HIGH_ALARM_LIMITS(100),\
                LOW_ALARM_LIMITS(100),\
                ANALOG_TEMP(100)

!------------------------------------------------------------
!             Variables used in alarm checking
!------------------------------------------------------------
LOGICAL;        ALARM_DISABLED(100),\
                NORMAL(100),\
                ALARM(100),\
                HIGH_ALARM(100),\
                LOW_ALARM(100)

!---------------------------------------
!       Unacknowledged alarms
!---------------------------------------
LOGICAL;        UNACK_ALARM(100)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.4 (cont)

```
!--------------------------------------
!      Alarms to be displayed
!--------------------------------------
LOGICAL;        DISPLAY_ALARM(100)


!------------------------------------------------------------------------------
!      NEW_ALARM logicals--one for each variable in the system.  This array
!      is generated by comparing the current array of ALARM logicals with
!      those from the previous pass through the logic (the ALARM_ON array).
!      NEW_ALARM logicals are set when an ALARM logical is set and the
!      corresponding ALARM_ON logical is FALSE.
!------------------------------------------------------------------------------
LOGICAL;        NEW_ALARM_I(100)


!------------------------------------------------------------------------------
!      ALARM_ON logicals--one for each variable in the system.  The ALARM
!      array is copied into this array at the end of each pass of logic.
!      It is used for determining changes of state in the ALARM array, i.e.,
!      new alarms or return to normal conditions.
!
!      UNACK_ALARM_ON logicals work same as ALARM_ON logicals, but are
!       to keep track new alarm acknowledgments.
!------------------------------------------------------------------------------
LOGICAL;        ALARM_ON(100),\
                UNACK_ALARM_ON(100)


!------------------------------------------------------------------------
!      Size of scrolling region for display of alarms.
!------------------------------------------------------------------------
NUMERIC;        NUM_LINES:100

!------------------------------------------------------------------------
!      Option masks for use with the DISPLAY_MESSAGES calls.
!------------------------------------------------------------------------
LOGICAL;        DSP_MSG_MASK(32),\
                DSP_MSG_MASK_2(32)

!---------------------------------------------------------------
!        Variables used in the DISPLAY_MESSAGE calls
!---------------------------------------------------------------
LOGICAL;        ALM_BLINK(100),\
                GEN_BLINK,\
                GEN_ALARM,\
                ALARM_ACK,\
                ALARM_ACK_ALL
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.4 (cont)

```
!-------------------------------------------------------------------
!      String declarations for Display_Message scrolling region.
!-------------------------------------------------------------------
STRING;        ALM_DISPLAY_STR(100)[75]


!-----------------------------------------------------------------------
!       Return Status for the display_message call.  The statuses and
!       their meanings are listed in the documentation for this call.
!-----------------------------------------------------------------------
LONG;          DISPLAY_RETURN_STATUS



!---------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!---------------------------------------------------------------------
TABLES;


RECALL; SETMSG_C16, , CRISP_TT

!--------------------------------------------------------------------------
!     Put date and time in front of each message in scroll message region.
!--------------------------------------------------------------------------
DSP_MSG_MASK(0) = TRUE



!---------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!---------------------------------------------------------------------
RESTART;

!-----------------------------------
!      Alarm Checking for ANALOGS
!-----------------------------------

ALARM_INDEX = ZERO
LABEL; NEXT_ANALOG

!-------------------------------------------------------
! If an alarm is disabled then jump over the code
! to the label JUMP_TO_NEXT_ANALOG.
!-------------------------------------------------------
JUMP; JUMP_TO_NEXT_ANALOG,  ALARM_DISABLED(ALARM_INDEX)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.4 (cont)

```
!---------------------------------------------------------------------
!      For each analog variable, do alarm checking if enabled
!---------------------------------------------------------------------
    RECALL; ABSALM_C16, ,\
            ANALOG_TEMP(ALARM_INDEX), \
            LOW_ALARM_LIMITS(ALARM_INDEX), \
            HIGH_ALARM_LIMITS(ALARM_INDEX), \
            DEADBAND_LIMITS(ALARM_INDEX), \
            LOW_ALARM(ALARM_INDEX), \
            HIGH_ALARM(ALARM_INDEX), \
            NORMAL(ALARM_INDEX),\
            ALARM(ALARM_INDEX)

!-------------------------------------------------------------------
!   A point is in alarm whenever a low alarm, or a high alarm.
!-------------------------------------------------------------------
    ALARM(ALARM_INDEX) = LOW_ALARM(ALARM_INDEX) | HIGH_ALARM(ALARM_INDEX)


!---------------------------------
!       Check if new alarm
!---------------------------------
    NEW_ALARM_I(ALARM_INDEX) = ALARM(ALARM_INDEX) & ~ALARM_ON(ALARM_INDEX)


!------------------------------------------------------
!   For every new alarm, set the unacknowledge bit.
!------------------------------------------------------
    LET; NEW_ALARM_I(ALARM_INDEX),\
         UNACK_ALARM(ALARM_INDEX) = TRUE


!-------------------------------------------------------------------
!  Update arrays of history bits for alarms & acknowleged alarms.
!-------------------------------------------------------------------
       ALARM_ON(ALARM_INDEX) = ALARM(ALARM_INDEX)
       UNACK_ALARM_ON(ALARM_INDEX) = UNACK_ALARM(ALARM_INDEX)

!----------------------------------------------
!  Mark all the analog alarms to be displayed.
!----------------------------------------------
     DISPLAY_ALARM(ALARM_INDEX) = ALARM_ON(ALARM_INDEX) | \
                                  UNACK_ALARM(ALARM_INDEX)


LABEL; JUMP_TO_NEXT_ANALOG

ALARM_INDEX = ALARM_INDEX + 1

JUMP;  NEXT_ANALOG, ALARM_INDEX < NUMBER_OF_ANALOG_ALARMS
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.4 (cont)

```
!------------------------------------------------------------------
!      Display Alarms on the Alarm Screen
!------------------------------------------------------------------
RECALL; DISPLAY_MESSAGES, ,\
        DISPLAY_RETURN_STATUS, \
        DSP_MSG_MASK(0), \
        , \
        , \
        NUM_LINES, \
        ALM_DISPLAY_STR(0), \
        ALM_BLINK(0), \
        GEN_BLINK, \
        ALARM_ACK, \
        GEN_ALARM, \
        NUMBER_OF_ANALOG_ALARMS, \
        DISPLAY_ALARM(0), \
        LABEL_STR(0), \
        ID_NAME_STR(0),\
        ALM_MSG(0)




!------------------------------------------------------------------------------
!      (GROUP ACKNOWLEDGEMENT)
!      The following code is used to acknowledge a group of alarms.
!      ALARM_ACK_ALL would be linked to a button on an alarm screen,
!      when ALARM_ACK_ALL becomes true, all the current alarms
!      in the scrolling region would be acknowledged.
!
!      NOTE:  Alarms will remain in the scrolling region until
!      The alarm condition returns to normal "AND" the alarm has
!      been acknowledged.
!------------------------------------------------------------------------------
RECALL; BZERO, ALARM_ACK_ALL, UNACK_ALARM(0), ONEHUNDRED

!-------------------------
!  reset the ack button
!-------------------------
RECALL; MAKZERO, , ALARM_ACK_ALL


END;
```


© 1991 Square D All Rights Reserved

Individual Alarm Acknowledgment

Individual Alarm Acknowledgment is accomplished by creating an array of logicals to represent unacknowledged alarms, as done for the group acknowledgment. On the CRISP display there will be an **ack** button next to each line in the alarm scrolling region. Each alarmable point in the system will have a unique Label, and the SMERGE_C16 and CMPRS Function Calls are used in determining which alarm was acknowledged.

Explanation

When a point goes into alarm, the associated UNACK_ALARM bit is set to TRUE. On the alarm scrolling region, the buttons associated with each line in the scrolling region are linked to ACK_ALM_DSP_MSG. When an **ack** button is set, the ACK_ALM_DSP_MSG logical becomes TRUE. As the logic loops through all the points, when an ACK_ALM_DSP_MSG bit is TRUE, a SMERGE_C16 Function Call will execute, which skips over the time and date stamp and copies the next eight characters (which, in this case, would be the length of the unique LABEL_STR for each point in the system) into an ACK_LABEL variable. Then a CMPRS Function Call would compare the ACK_LABEL with each element in the LABEL_STR array until it located a match. When a match is located the CMPRS Function Call then returns the index of that match into ACK_ALARM_INDEX. Finally, using the ACK_ALARM_INDEX the alarm can now be acknowleged by setting UNACK_ALARM(ACK_ALARM_INDEX) = FALSE.

Example 2.5

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to perform individual alarm acknowledgment. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 2.5                                        *
!*                                                                         *
!*      Task 2.5 - An example of Individual Alarm Acknowledgement.         *
!***************************************************************************

!---------------------------------------------------------------------
!   This is an example of a system containing (100) analog points.
!---------------------------------------------------------------------

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.5 (cont)

```
!------------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!------------------------------------------------------------------------
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, DEBUG , DEBUG_BREAK ,

!------------------------------------------------------------------------
!               The follow 32 logical make up the "Message Mask".
!               These are variables that represent each potential print
!               device or file were messages can be routed.
!------------------------------------------------------------------------
LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


!--------------------------------------
!         constants
!--------------------------------------
CONSTANT;       ZERO:0,         ONE:1,\
                ONEHUNDRED:100, TWENTY:20,\
                TWENTYFIVE:25,  NUMBER_OF_ANALOG_ALARMS:100

!-----------------------------------------
!     Array indexes for checking alarm
!-----------------------------------------
NUMERIC;        ACK_ALARM_INDEX,    ALARM_INDEX,\
                ACK_INDEX,          INDEX

!----------------------------------------------------------
!      LABEL_STR is unique for each point in the system.
!----------------------------------------------------------
STRING;         LABEL_STR(100)[8],\
                ID_NAME_STR(100)[16],\
                ALM_MSG(100)[40]

FLOAT;          DEADBAND_LIMITS(100),\
                HIGH_ALARM_LIMITS(100),\
                LOW_ALARM_LIMITS(100),\
                ANALOG_TEMP(100)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.5 (cont)

```
!------------------------------------------------------------
!              Variables used in alarm checking
!------------------------------------------------------------
LOGICAL;        ALARM_DISABLED(100),\
                NORMAL(100),\
                ALARM(100),\
                HIGH_ALARM(100),\
                LOW_ALARM(100)

!---------------------------------------
!       Unacknowledged alarms
!---------------------------------------
LOGICAL;        UNACK_ALARM(100)

!---------------------------------------
!      Alarms to be displayed
!---------------------------------------
LOGICAL;        DISPLAY_ALARM(100)


!------------------------------------------------------------------------------
!      NEW_ALARM logicals--one for each variable in the system.  This array
!      is generated by comparing the current array of ALARM logicals with
!      those from the previous pass through the logic (the ALARM_ON array).
!      NEW_ALARM logicals are set when an ALARM logical is set and the
!      corresponding ALARM_ON logical is FALSE.
!------------------------------------------------------------------------------
LOGICAL;        NEW_ALARM_I(100)


!------------------------------------------------------------------------------
!       ALARM_ON logicals--one for each variable in the system.  The ALARM
!       array is copied into this array at the end of each pass of logic.
!       It is used for determining changes of state in the ALARM array, i.e.,
!       new alarms or return to normal conditions.
!
!       UNACK_ALARM_ON logicals work same as ALARM_ON logicals, but are
!       to keep track new alarm acknowledgments.
!------------------------------------------------------------------------------
LOGICAL;        ALARM_ON(100),\
                UNACK_ALARM_ON(100)


!------------------------------------------------------------------------
!      Size of scrolling region for display of alarms.
!------------------------------------------------------------------------
NUMERIC;        NUM_LINES:100
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.5 (cont)

```
!----------------------------------------------------------------------
!      Option mask use in the DISPLAY_MESSAGES call.
!----------------------------------------------------------------------
LOGICAL;        DSP_MSG_MASK(32)


!---------------------------------------------------------------
!        Variables used in the DISPLAY_MESSAGES Function Call.
!---------------------------------------------------------------
LOGICAL;        ALM_BLINK(100),\
                GEN_BLINK,\
                GEN_ALARM,\
                ALARM_ACK


!------------------------------------------------------------------
!      String declarations for Display_Message scrolling region.
!------------------------------------------------------------------
STRING;         ALM_DISPLAY_STR(100)[75]


!-------------------------------------------------------------------------
!       Return Status for the display_message call.  The statuses
!       and their meanings are listed in the documentation for this call.
!-------------------------------------------------------------------------
LONG;           DISPLAY_RETURN_STATUS

!-----------------------------------------------------------------------
!       This logical is used to acknowledge alarms within the
!       alarm scrolling region.
!-----------------------------------------------------------------------
LOGICAL;        ACK_ALM_DSP_MSG(100)

!--------------------------------------------------------------
!               This variable is used to hold the label
!               of the alarm being acknowledged, in which
!               is then compared to the array of labels
!               inorder to find the index for that point
!               in alarm.
!--------------------------------------------------------------
STRING;         ACK_LABEL[8]



!--------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!--------------------------------------------------------------------
TABLES;
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.5 (cont)

```
RECALL; SETMSG_C16, , CRISP_TT

!---------------------------------------------------------------------------
!     Put date and time in front of each message in scroll message region.
!---------------------------------------------------------------------------
DSP_MSG_MASK(0) = TRUE




!----------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;

!------------------------------------
!      Alarm Checking for ANALOGS
!------------------------------------

ALARM_INDEX = ZERO
LABEL; NEXT_ANALOG

!--------------------------------------------------------
! If an alarm is disabled then jump over the code
! to the label JUMP_TO_NEXT_ANALOG.
!--------------------------------------------------------
JUMP; JUMP_TO_NEXT_ANALOG,  ALARM_DISABLED(ALARM_INDEX)

!---------------------------------------------------------------------
!      Perform alarm handling, identify hi/lo alarms.
!---------------------------------------------------------------------
    RECALL; ABSALM_C16, ,\
            ANALOG_TEMP(ALARM_INDEX), \
            LOW_ALARM_LIMITS(ALARM_INDEX), \
            HIGH_ALARM_LIMITS(ALARM_INDEX), \
            DEADBAND_LIMITS(ALARM_INDEX), \
            LOW_ALARM(ALARM_INDEX), \
            HIGH_ALARM(ALARM_INDEX), \
            NORMAL(ALARM_INDEX),\
            ALARM(ALARM_INDEX)

!------------------------------------------------
!   A point is in alarm whenever there is a
!   low alarm, or a high alarm for that point.
!------------------------------------------------
    ALARM(ALARM_INDEX) = LOW_ALARM(ALARM_INDEX) | HIGH_ALARM(ALARM_INDEX)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.5 (cont)

```
    !-----------------------------------
    !       Check if new alarm
    !-----------------------------------
    NEW_ALARM_I(ALARM_INDEX) = ALARM(ALARM_INDEX) & ~ALARM_ON(ALARM_INDEX)


!------------------------------------------------------
!   For every new alarm, set the unacknowledge bit.
!------------------------------------------------------
    LET; NEW_ALARM_I(ALARM_INDEX),\
         UNACK_ALARM(ALARM_INDEX) = TRUE


!-------------------------------------------------------------------
!  Update arrays of history bits for alarms & acknowleged alarms.
!-------------------------------------------------------------------
       ALARM_ON(ALARM_INDEX) = ALARM(ALARM_INDEX)
       UNACK_ALARM_ON(ALARM_INDEX) = UNACK_ALARM(ALARM_INDEX)

!------------------------------------------------
!  Mark all the analog alarms to be displayed.
!------------------------------------------------
     DISPLAY_ALARM(ALARM_INDEX) = ALARM_ON(ALARM_INDEX) | \
                                  UNACK_ALARM(ALARM_INDEX)


LABEL; JUMP_TO_NEXT_ANALOG

ALARM_INDEX = ALARM_INDEX + 1

JUMP;  NEXT_ANALOG, ALARM_INDEX < NUMBER_OF_ANALOG_ALARMS


!------------------------------------------------------------------
!      Display Alarms on the Alarm Screen
!------------------------------------------------------------------
RECALL; DISPLAY_MESSAGES, ,\
        DISPLAY_RETURN_STATUS, \
        DSP_MSG_MASK(0), \
        , \
        , \
        NUM_LINES, \
        ALM_DISPLAY_STR(0), \
        ALM_BLINK(0), \
        GEN_BLINK, \
        ALARM_ACK, \
        GEN_ALARM, \
        NUMBER_OF_ANALOG_ALARMS, \
        DISPLAY_ALARM(0), \
        LABEL_STR(0), \
        ID_NAME_STR(0),\
        ALM_MSG(0)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.5 (cont)

```
!-----------------------------------------------------------------------------
!       NOTE: Individual Alarm Acknowledgements will be done
!       on the CRISP display, by setting the button (the logical
!       ACK_ALM_DSP_MSG) next to the alarm message you wish to
!       acknowledge.
!-----------------------------------------------------------------------------


ACK_INDEX = 0
LABEL; ACK_ALARMS
!---------------------------------------------------------------------
!         Get description from alarm which was acknoledged:
!         The logical ACK_ALM_DSP_MSG will be linked to a button
!         on the CRISP display, an array corresponding to each
!         line in the scrolling region.  When the logical is
!         set (acknowledged by an operator) the corresponding
!         alarm message is used to find the LABEL_STR which
!         identifies the point.   The offset of TWENTFIVE in
!         this call is to skip over the time & date.
!---------------------------------------------------------------------
            RECALL; SMERGE_C16, ACK_ALM_DSP_MSG(ACK_INDEX),\
                    ALM_DISPLAY_STR(ACK_INDEX),\
                    TWENTYFIVE, ACK_LABEL, ONE

!------------------------------------------------------------
!         Find index for alarm which was acknowledged,
!         by comparing it with the array of LABEL_STR.
!         The labels index location within the array
!         LABEL_STR is returned into ACK_ALARM_INDEX.
!------------------------------------------------------------
            RECALL; CMPRS, ACK_ALM_DSP_MSG(ACK_INDEX), ACK_LABEL, \
                    LABEL_STR(0), NUMBER_OF_ANALOG_ALARMS, ACK_ALARM_INDEX

!------------------------------------------------------------------
!         If alarm was acknowledged then turn
!         off its UNACK_ALARM bit.
!------------------------------------------------------------------
           LET; ACK_ALM_DSP_MSG(ACK_INDEX) & (ACK_ALARM_INDEX <> -1), \
                UNACK_ALARM(ACK_ALARM_INDEX) = FALSE


ACK_INDEX = ACK_INDEX + 1
JUMP; ACK_ALARMS, ACK_INDEX < ONEHUNDRED

!-------------------------------------------------------------
!  Reset all individual alarm ack buttons each pass of loic.
!-------------------------------------------------------------
RECALL; BZERO, , ACK_ALM_DSP_MSG(0),ONEHUNDRED


END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Enabling/Disabling Alarms

This subsection defines two methods used to enable and disable alarm handling.

Solution A

Alarm Enabling/Disabling may be accomplished by creating an array of logicals, one for each alarmable point in the system, and through the logic or through a configuration screen set the points which need to be alarm disabled.

Explanation A

All of the Alarm Handling functions will be together in one block of code. An index would be incremented after the Alarm Handler section. Just before the Alarm Handling block of code, a JUMP statement would jump over all of the Alarm Handling if Logical ALARM_DISABLED(index) is TRUE for any point in the system.

An Alarms-Disabled scrolling region is implemented to indicate those alarms that are disabled. This logic supports an ENABLED ALL button and a DISABLED ALL button for the Alarms-Disabled display.

Example 2.6 A

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to perform alarm enabling/disabling. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 2.6                                        *
!*                                                                         *
!*      Task 2.6 - An example of Alarm Enable/Disable.                     *
!***************************************************************************

!---------------------------------------------------------------------
!   This is an example of a system containing (100) analog points.
!---------------------------------------------------------------------

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------

!-------------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!-------------------------------------------------------------------------
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, DEBUG , DEBUG_BREAK ,
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.6 A (cont)

```
!-----------------------------------------------------------------------
!              The follow 32 logical make up the "Message Mask".
!              These are variables that represent each potential print
!              device or file were messages can be routed.
!-----------------------------------------------------------------------
LOGICAL;       CRISP_TT:TRUE, CRISP_LP     , CRISP_CL     , CRISP_SS     , \
               CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
               CRISP_DEV00  , PRINTER1     , PRINTER2     , DECTALK_DEV , \
               CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
               CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
               CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
               CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
               CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23



!---------------------------------------------------------
!             Variables used to enable/disable alarms
!---------------------------------------------------------
LOGICAL;       ALARM_DISABLED(100),\
               ENABLE_ALL_ALARMS,\
               DISABLE_ALL_ALARMS

NUMERIC;      ZERO:0,       NUMBER_OF_ANALOG_ALARMS:100,\
              ALARM_INDEX,          INDEX


!---------------------------------------------------------------------
!      Size of scrolling region for displaying points that are
!      alarm disabled.
!---------------------------------------------------------------------
NUMERIC;      NUM_LINES:100

!---------------------------------------------------------------------
!      Option mask to be used in the DISPLAY_MESSAGES Function Call.
!---------------------------------------------------------------------
LOGICAL;      DSP_MSG_MASK(32)

!------------------------------------------------------------
!        Variables used in the DISPLAY_MESSAGES call
!------------------------------------------------------------
STRING;        LABEL_STR(100)[8],\
               ID_NAME_STR(100)[10]

!-----------------------------------------------------------------
!      String declarations for Display_Message scrolling region.
!-----------------------------------------------------------------
STRING;       ALM_DISPLAY_STR(100)[75]
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.6 A (cont)

```
!--------------------------------------------------------------------
!         Return Status for the DISPLAY_MESSAGES Function Call.
!         The statuses and their meanings are listed in the
!        documentation for this call.
!--------------------------------------------------------------------
LONG;          DISPLAY_RETURN_STATUS


!---------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!---------------------------------------------------------------------
TABLES;

!---------------------------------------------------------------------------
!     Put date and time in front of each message in scroll message region.
!---------------------------------------------------------------------------
DSP_MSG_MASK(0) = TRUE



!---------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!---------------------------------------------------------------------
RESTART;



ALARM_INDEX = ZERO
LABEL; NEXT_ANALOG


JUMP; SKIP_ALARM_HANDLING, ALARM_DISABLED(ALARM_INDEX)

!**************************
!*                       *
!*  Alarm Handling Code   *
!*                       *
!**************************


LABEL; SKIP_ALARM_HANDLING

ALARM_INDEX = ALARM_INDEX + 1

JUMP; NEXT_ANALOG, ALARM_INDEX < NUMBER_OF_ANALOG_ALARMS
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.6 A (cont)

```
!--------------------------------------------------------------
!    Logic to disable alarms for all points in the system.
!--------------------------------------------------------------
JUMP;  SKIP_DISABLE_ALL_ALARMS, ~DISABLE_ALL_ALARMS

       INDEX = 0

       LABEL; NEXT_ALARM_TO_DISABLE

              ALARM_DISABLED(INDEX) = TRUE
              INDEX = INDEX + 1

       JUMP; NEXT_ALARM_TO_DISABLE, INDEX < NUMBER_OF_ANALOG_ALARMS

LABEL; SKIP_DISABLE_ALL_ALARMS



!---------------------------------------------------------------
!     Logic to enable alarms for all points in the system.
!---------------------------------------------------------------
JUMP;  SKIP_ENABLE_ALL_ALARMS, ~ENABLE_ALL_ALARMS

       RECALL; BZERO, , ALARM_DISABLED(0), NUMBER_OF_ANALOG_ALARMS

LABEL; SKIP_ENABLE_ALL_ALARMS


!----------------------------------------------------------------
!       This DISPLAY_MESSAGES Call will have a scrolling
!       region linked to display disabled alarms.  This
!       provides the operator with a list of the points
!       are alarm disabled, including the time&date the
!       point was put into that state.
!----------------------------------------------------------------
RECALL; DISPLAY_MESSAGES, ,\
        DISPLAY_RETURN_STATUS, \
        DSP_MSG_MASK(0), \
        , \
        , \
        NUM_LINES, \
        ALM_DISPLAY_STR(0), \
        , \
        , \
        , \
        , \
        NUMBER_OF_ANALOG_ALARMS, \
        ALARM_DISABLED(0), \
        LABEL_STR(0), \
        ID_NAME_STR(0)
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.6 A (cont)

```
!-------------------------------------------------------
!   Reset enable_all/disable_all buttons.
!-------------------------------------------------------
RECALL; MAKZERO, ,ENABLE_ALL_ALARMS, DISABLE_ALL_ALARMS

END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Solution B

For a system with analog points, this task can be accomplished by using the DEV_ALARM_1 Function Call.

Explanation B

The DEV_ALARM_1 Function Call has an argument ENABLE, which enables alarm checking. If this logical is true, the corresponding element in the VALUE array is checked. If false, no alarm checking is done for the corresponding VALUE.

NOTE

The DEV_ALARM_1 Function Call checks all elements of the arrays, each time the Call is executed.

An Alarms-Disabled scrolling region is implemented to indicate those alarms that are disabled. This logic supports an ENABLED ALL button, and a DISABLED ALL button for the Alarms-Disabled display.

NOTE

The ENABLE argument is optional in the DEV_ALARM_1 Function Call. If omitted, all array elements are checked.

Example 2.6 B

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to perform alarm enabling/disabling. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.6 B (cont)

```
!***************************************************************************
!*      Sample B - code for Example 2.6                                    *
!*                                                                         *
!*      Task 2.6 - Example B of Alarm Enable/Disable.                      *
!***************************************************************************

!---------------------------------------------------------------------
!   This is an example of a system containing (100) analog points.
!---------------------------------------------------------------------

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------

!-------------------------------------------------------------------------
!             The following 16 Logicals are used by the CRISP
!             system.  Do not place any other Declarations
!             before these.
!-------------------------------------------------------------------------
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, DEBUG , DEBUG_BREAK ,

!-------------------------------------------------------------------------
!               The follow 32 logical make up the "Message Mask".
!               These are variables that represent each potential print
!               device or file were messages can be routed.
!-------------------------------------------------------------------------
LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , PRINTER1    , PRINTER2    , DECTALK_DEV , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23


NUMERIC;        ZERO:0,            NUMBER_OF_ANALOG_ALARMS:100,\
                ALARM_INDEX,    INDEX

!-----------------------------------------------------------
!             Variables used in alarm checking
!-----------------------------------------------------------
LOGICAL;        ALARM_ENABLED(100),\
                NORMAL(100),\
                ALARM(100),\
                HIGH_ALARM(100),\
                LOW_ALARM(100),\
                NEW_NORMAL,\
                NEW_ALARM
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 2.6 B (cont)

```
FLOAT;          DEADBAND_LIMITS(100),\
                LO_ALM_DEV(100),\
                HI_ALM_DEV(100),\
                ANALOG_TEMP(100),\
                SETPOINT(100)


!------------------------------------------------------------
!             Variables used to enable/disable alarms
!------------------------------------------------------------
LOGICAL;        ALARM_DISABLED(100),\
                ENABLE_ALL_ALARMS,\
                DISABLE_ALL_ALARMS

!------------------------------------------------------------------------
!      Size of scrolling region for displaying points that are
!      alarm disabled.
!------------------------------------------------------------------------
NUMERIC;      NUM_LINES:100

!------------------------------------------------------------------------
!      Option mask to be used in the DISPLAY_MESSAGES Function Call.
!------------------------------------------------------------------------
LOGICAL;      DSP_MSG_MASK(32)

!---------------------------------------------------------------
!        Variables used in the DISPLAY_MESSAGES call
!---------------------------------------------------------------
STRING;         LABEL_STR(100)[8],\
                ID_NAME_STR(100)[10]

!--------------------------------------------------------------------
!      String declarations for Display_Message scrolling region.
!--------------------------------------------------------------------
STRING;ALM_DISPLAY_STR(100)[75]


!---------------------------------------------------------------------
!         Return Status for the DISPLAY_MESSAGES Function Call.
!         The statuses and their meanings are listed in the
!        documentation for this call.
!---------------------------------------------------------------------
LONG;         DISPLAY_RETURN_STATUS


!----------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!----------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.6 B (cont)

```
TABLES;

!---------------------------------------------------------------------------
!     Put date and time in front of each message in scroll message region.
!---------------------------------------------------------------------------
DSP_MSG_MASK(0) = TRUE

!---------------------------------------------------------------------------
!     Initialize LO_ALM_DEV and HI_ALM_DEV for the DEV_ALARM_1
!     Function Call.  Both arguments must be greater then zero.
!---------------------------------------------------------------------------
INDEX = 0
LABEL; INIT_LO_HI_ALM_DEV
   LO_ALM_DEV(INDEX) = 1
   HI_ALM_DEV(INDEX) = 1

   INDEX = INDEX + 1
JUMP;  INIT_LO_HI_ALM_DEV, INDEX < NUMBER_OF_ANALOG_ALARMS

!----------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;

!-----------------------------------
!      Alarm Checking for ANALOGS
!-----------------------------------

!-----------------------------------------------------------------
!       This call will perform the alarm checking for the
!       entire array in one pass of logic, however if the
!       variable ALARM_ENABLED for any one array element
!       is FALSE, no alarm checking is done for that
!       correspoinding ANALOG_TEMP.
!-----------------------------------------------------------------
    RECALL; DEV_ALARM_1, ,\
            NUMBER_OF_ANALOG_ALARMS,\
            NEW_NORMAL,\
            NEW_ALARM,\
            ANALOG_TEMP(0),\
            SETPOINT(0),\
            LO_ALM_DEV(0),\
            HI_ALM_DEV(0),\
            DEADBAND_LIMITS(0),\
            LOW_ALARM(0),\
            HIGH_ALARM(0),\
            NORMAL(0),\
            ALARM(0),\
            ALARM_ENABLED(0)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.6 B (cont)

```
ALARM_INDEX = ZERO
LABEL; NEXT_ANALOG


!*************************
!*                       *
!*  Alarm Handling Code   *
!*                       *
!*************************



!-----------------------------------------------------------------
!       Update trigger for displaying disabled alarms.
!-----------------------------------------------------------------
ALARM_DISABLED(ALARM_INDEX) = ~ALARM_ENABLED(ALARM_INDEX)



ALARM_INDEX = ALARM_INDEX + 1

JUMP; NEXT_ANALOG, ALARM_INDEX < NUMBER_OF_ANALOG_ALARMS



!---------------------------------------------------------------
!    Logic to disable alarms for all points in the system.
!---------------------------------------------------------------
JUMP;  SKIP_DISABLE_ALL_ALARMS, ~DISABLE_ALL_ALARMS

       RECALL; BZERO, , ALARM_ENABLED(0), NUMBER_OF_ANALOG_ALARMS

LABEL; SKIP_DISABLE_ALL_ALARMS


!----------------------------------------------------------------
!     Logic to enable alarms for all points in the system.
!----------------------------------------------------------------
JUMP;  SKIP_ENABLE_ALL_ALARMS, ~ENABLE_ALL_ALARMS

       INDEX = 0

       LABEL; NEXT_ALARM_TO_ENABLE

              ALARM_ENABLED(INDEX) = TRUE
              INDEX = INDEX + 1

       JUMP; NEXT_ALARM_TO_ENABLE, INDEX < NUMBER_OF_ANALOG_ALARMS

LABEL; SKIP_ENABLE_ALL_ALARMS
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 2.6 B (cont)

```
!----------------------------------------------------------------
!       This DISPLAY_MESSAGES Call will have a scrolling
!       region linked to display disabled alarms.  This
!       provides the operator with a list of the points
!       are alarm disabled, including the time&date the
!       point was put into that state.
!----------------------------------------------------------------
RECALL; DISPLAY_MESSAGES, ,\
        DISPLAY_RETURN_STATUS, \
        DSP_MSG_MASK(0), \
        , \
        , \
        NUM_LINES, \
        ALM_DISPLAY_STR(0), \
        , \
        , \
        , \
        , \
        NUMBER_OF_ANALOG_ALARMS, \
        ALARM_DISABLED(0), \
        LABEL_STR(0), \
        ID_NAME_STR(0)


!------------------------------------------------------
!   Reset enable_all/disable_all buttons.
!------------------------------------------------------
RECALL; MAKZERO, ,ENABLE_ALL_ALARMS, DISABLE_ALL_ALARMS


END;
```


© 1991 Square D All Rights Reserved

General

This section defines how to use the CRISP SCAN_HIST function call, including data point edit, and how to write the recalled data to an ASCII file for import to another application (e.g., a spreadsheet package).

Solution

The user can accomplish the task of HISTORIAN data Recall by using the Function call SCAN_HIST, which also provides Data Point Edit functionality. The VALUE_C16 Function Call can optionally be used to write the recalled data to an ASCII file.

Explanation

The SCAN_HIST call is provided in a working example to follow. It is assumed for this example that HISTORIAN Data files already exist. A display must also be created to allow the user to enter the start_date, start_time, end_date, end_time, interval, and a tag_name. The display would have a button to set the logical condition for executing the SCAN_HIST call.

SCAN_HIST Edit Function

The SCAN_HIST function call permits the user to read data, edit it and replace it in HISTORIAN point files. This can be done only to existing point file records; new records cannot be inserted into point files using this functionality.

To illustrate the use of this feature, we will begin by working with a single variable. The user would enter the tag name, the start date and time, the number of points to be recovered, and would specify a time interval of 00:00:00, i.e., recover each point that was recorded for the specified tag name. Supplying an interval other than 00:00:00 causes interpolated data points to be returned which do not actually exist in the point files. After the data is displayed on the screen, the user can modify those values (not the date/time fields), set the edit data logical and trigger the call again and the revised data will be written back to the point file.

If more than one tag name is specified when the call is first executed, all points and date/time stamps are recovered for the first tag name and then the recovered time/stamp array is used to try and recover data for each of the other tag names specified. If the call fails to locate a record in a point file for any of the variables at any of the dates and times in the array, the call will return an error code indicating that specified records were not located (-3). If the call completes with no errors, the user can edit the data fields, set the edit logical and trigger the call again to write the data back to the point files. If this feature is to be used for editing several variables that are to be retrieved together, the data should be saved originally using an event trigger so that all points will be written to the point files with the same date/time stamp.

© 1991 Square D All Rights Reserved

Example 3.1

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to use the SCAN_HIST call. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

In the example to follow, writting the HISTORIAN data to an ASCII file is demonstrated by using the VALUE_C16 call.

The Example Declarations and Logic follows.

```
!**************************************************************************
!*      Sample code for Example 3.1                                       *
!*                                                                        *
!*      Task 3.1 - How to use the SCAN_HIST call for Historian Data Recall.*
!**************************************************************************

!------------------------------------------------------------------------
!       Code for the Declaration Section (before "TABLES;")
!------------------------------------------------------------------------

!------------------------------------------------------------------------
!              The following 16 Logicals are used by the CRISP
!              system.  Do not place any other Declarations
!              before these.
!------------------------------------------------------------------------
LOGICAL;       NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
               ICCDSA, PAUSE, , ,

!------------------------------------------------------------------------
!               The follow 32 logical make up the "Message Mask".
!               These are variables that represent each potential print
!               device or file were messages can be routed.
!------------------------------------------------------------------------
LOGICAL;       CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
               CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
               CRISP_DEV00  , CRISP_DEV01 , CRISP_DEV02 , CRISP_DEV03 , \
               CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
               CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
               CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
               CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
               CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23

LOGICAL;       DDMLM_01     , DDMLM_02    , DDMLM_03    , DDMLM_04    , \
               DDMLM_05     , DDMLM_06    , DDMLM_07    , DDMLM_08    , \
               DDMLM_09     , DDMLM_10    , DDMLM_11    , DDMLM_12    , \
               DDMLM_13     , DDMLM_14    , DDMLM_15    , DDMLM_16    , \
               DDMLM_17     , DDMLM_18    , DDMLM_19    , DDMLM_20    , \
               DDMLM_21     , DDMLM_22    , DDMLM_23    , DDMLM_24    , \
               DDMLM_25     , DDMLM_26    , DDMLM_27    , DDMLM_28    , \
               DDMLM_29     , DDMLM_30    , DDMLM_31    , DDMLM_32
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 3.1 (cont)

```
!----------------------------------------------------------
!      Array size for recovered variables
!----------------------------------------------------------
CONSTANT;      SCAN_SIZE:60

!----------------------------------------------------------
!      Logicals for recovering and printing data
!----------------------------------------------------------
LOGICAL;       GET_HIST_DATA:FALSE,  PRINT_HIST_DATA:FALSE,              \
               EDIT_MODE:FALSE,      SCAN_DONE:FALSE

LONG;          SCAN_STATUS,          INDEX
NUMERIC;       RECS_PER_SCAN:60,     NO_OF_PTS:60,  \
               NO_OF_VARIABLES

LONG;          QTY_RECS_FOUND

NUMERIC;       ONE:1, POINTER
LONG;          PAD:65536
STRING;START_DATE[11]:"13-NOV-1992",\
               START_TIME[8]:"12:00:00",\
               END_DATE[11]:"13-NOV-1992",\
               END_TIME[8]:"12:10:00",\
               INTERVAL[8]:"00:00:00",\
               BLANK_STRING[30]:"                              ",\
               TAG_NAME_1[30]:"TAG_NAME_1",\
               TAG_NAME_2[30]:"TAG_NAME_2",\
               DATE_TIME(SCAN_SIZE)[20],\
               TERMINATOR[]:"@M@J",\
               SPACE[]:" ",\
               COMMA[]:",",\
               VARVAL[]:"\\\\\\.\\"

!------------------------------------------------------------
!      Search variables for first tag name
!------------------------------------------------------------
NUMERIC;       INITIAL_CODE
NUMERIC;       FINAL_CODE

FLOAT;         INITIAL_VALUE
FLOAT;         FINAL_VALUE

!------------------------------------------------------------
!      Destination arrays for recovered variables
!------------------------------------------------------------
FLOAT;         DEST_1(SCAN_SIZE),\
               DEST_2(SCAN_SIZE)

LONG;          TIME_STAMP(SCAN_SIZE)
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 3.1 (cont)

```
NUMERIC;       TEST_1,\
               TEST_2

!-----------------------------------------------------------------
!      Test and statistical arrays for all variables
!-----------------------------------------------------------------
FLOAT;         LOW_1            , HIGH_1           , MIN_1            , \
               MAX_1            , SUM_1
FLOAT;         LOW_2            , HIGH_2           , MIN_2            , \
               MAX_2            , SUM_2

!-----------------------------------------------------------
!      Name of file to receive printed output
!-----------------------------------------------------------
STRING;HIST_OUTPUT_FILE_SPEC[50]: "[CRISP.LOG]HIST_RPT.DAT"

LONG;          RMS_OPTIONS:1



!----------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!----------------------------------------------------------------------
TABLES;

RECALL; SETMSG_C16, , CRISP_TT



!----------------------------------------------------------------------
!      The Logic between the Key Word RESTART; and the Key Word
!      END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;



!-------------------------------------------------------------------------------
!   Process Logic - SCAN_HIST"
!
!      Returns data from Point_files created by the Historian data collection
!      system.
!-------------------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 3.1 (cont)

```
CALL; SCAN_HIST,               \
                GET_HIST_DATA,      \  ! Conditional expression
                START_DATE,         \  ! Date to start scan
                START_TIME,         \  ! Time to start scan
                END_DATE,           \  ! Date to end scan
                END_TIME,           \  ! Time to end scan
                INTERVAL,           \  ! Time interval between points
                INITIAL_VALUE,      \  ! Initial value for first variable
                INITIAL_CODE,       \  ! Initial test condition for first variable
                FINAL_VALUE,        \  ! Final value for first variable
                FINAL_CODE,         \  ! Final test condition for first variable
                NO_OF_PTS,          \  ! Number of points to be returned
                SCAN_STATUS,        \  ! Return status from call
                EDIT_MODE,          \  ! Write variables back to files
                SCAN_DONE,          \  ! Completion bit from call
                RECS_PER_SCAN,      \  ! Number of records per execution of call
                QTY_RECS_FOUND,     \  ! Number of records found
                TIME_STAMP(0),      \  ! 4-byte "c" based time stamp
                DATE_TIME(0),       \  ! ASCII date/time string
                TAG_NAME_1, TEST_1, LOW_1, DEST_1(0), \  ! Variable 1 params
                TAG_NAME_2, TEST_2, LOW_2, DEST_2(0)     ! Variable 2 params

LET; SCAN_DONE, GET_HIST_DATA = FALSE

JUMP; SKIP_PRINT, (~PRINT_HIST_DATA | (QTY_RECS_FOUND == 0))

!-------------------------------------------------------
!      Setup print file to receive output
!-------------------------------------------------------
CALL; NEW_DESTINATION, , CRISP_DEV07, HIST_OUTPUT_FILE_SPEC, RMS_OPTIONS

INDEX = 0
NO_OF_VARIABLES = 0

!-------------------------------------------------------------------------
!      Pad names with trailing spaces
!      Compare to a blank string
!      If not blank, increment no_of_variables
!      Use to determine correct print statements to use
!
!      For comma delimited print outs, replace SPACE with COMMA in
!      VALUE_C16 calls
!-------------------------------------------------------------------------
    RECALL; STR_EDIT, , TAG_NAME_1, PAD, TAG_NAME_1
    RECALL; CMPRS, ,TAG_NAME_1, BLANK_STRING, ONE, POINTER
    LET; ( POINTER <> 0 ), NO_OF_VARIABLES = NO_OF_VARIABLES + 1

    RECALL; STR_EDIT, , TAG_NAME_2, PAD, TAG_NAME_2
    RECALL; CMPRS, ,TAG_NAME_2, BLANK_STRING, ONE, POINTER
    LET; ( POINTER <> 0 ), NO_OF_VARIABLES = NO_OF_VARIABLES + 1
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 3.1 (cont)

```
CRISP_DEV07 = TRUE
CRISP_TT = FALSE

INDEX = 0
LABEL; NXT_POINT
    RECALL; VALUE_C16, (NO_OF_VARIABLES == 1), \
            DATE_TIME(INDEX), SPACE, \
            VARVAL, DEST_1(INDEX), SPACE, \
            TERMINATOR

    RECALL; VALUE_C16, (NO_OF_VARIABLES == 2), \
            DATE_TIME(INDEX), SPACE, \
            VARVAL, DEST_1(INDEX), SPACE, \
            VARVAL, DEST_2(INDEX), SPACE, \
            TERMINATOR


    INDEX = INDEX + 1
JUMP; NXT_POINT, (INDEX < QTY_RECS_FOUND)

PRINT_HIST_DATA = FALSE

QTY_RECS_FOUND = 0

LABEL; SKIP_PRINT

CRISP_DEV07 = FALSE
CRISP_TT = TRUE

END;
```


© 1991 Square D All Rights Reserved

General

This section defines how to communicate with the various types of devices available on CRISP/32.

This section is broken down into the following subsections.

Subsection	Description
Redundant PLC Interface *(page 5-3)*	Defines how to implement a redundant PLC configuration.
Multi-Drop Interface *(page 5-11)*	Defines how to handle communications with multi-drop devices.

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Redundant PLC Interface

To eleminate a single point of failure in a PLC environment, it is sometimes necessary to implement a redundant PLC configuration where inputs and outputs are shared by the redundant PLCs and other data is transferred to the secondary PLC from the primary PLC at end of scan. In this scenario, it may also be necessary to use two serial ports on the host computer to handle communications with the PLCs, resulting in four potential communication paths.

Example 4.1

The following example includes all the necessary declarations and logic to function correctly with a corresponding UCF for the IDI product. The example is provided only to help the user understand how to implement a redundant PLC/serial port configuration.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample Code for Example 4.1                                        *
!*                                                                         *
!*      Task 4.1 - How to implement a redundant PLC system                 *
!***************************************************************************

!---------------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------------

TITLE;"        INTERMEDIATE VARIABLE DECLARATIONS"
!
LOGICAL;       NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
               ICCDSA, PAUSE, , ,
!
LOGICAL;       CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
               CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
               CRISP_DEV00  , CRISP_DEV01 , CRISP_DEV02 , CRISP_DEV03 , \
               CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
               CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
               CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
               CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
               CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23
!
LOGICAL;       DDMLM_01     , DDMLM_02    , DDMLM_03    , DDMLM_04    , \
               DDMLM_05     , DDMLM_06    , DDMLM_07    , DDMLM_08    , \
               DDMLM_09     , DDMLM_10    , DDMLM_11    , DDMLM_12    , \
               DDMLM_13     , DDMLM_14    , DDMLM_15    , DDMLM_16    , \
               DDMLM_17     , DDMLM_18    , DDMLM_19    , DDMLM_20    , \
               DDMLM_21     , DDMLM_22    , DDMLM_23    , DDMLM_24    , \
               DDMLM_25     , DDMLM_26    , DDMLM_27    , DDMLM_28    , \
               DDMLM_29     , DDMLM_30    , DDMLM_31    , DDMLM_32
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 4.1 (cont)

```
!--------------------------------------------------------------------------
!      Number of registers to read
!--------------------------------------------------------------------------

CONSTANT;      PLC_REGISTERS:128

NUMERIC;       PLC_INPUTS(PLC_REGISTERS)
NUMERIC;       PLC_OUTPUTS(PLC_REGISTERS)

NUMERIC;       PLC_STATUS_REGISTER

!--------------------------------------------------------------------------
!      Array of digitals that first register will be unpacked into
!      First bit contains a status bit
!--------------------------------------------------------------------------

LOGICAL;       D_I(16)

!--------------------------------------------------------------------------
!
!      For the following variables, the two digit suffixes represent:
!             11 - Communications via serial port 1 to plc number 1
!             12 - Communications via serial port 1 to plc number 2
!             21 - Communications via serial port 2 to plc number 1
!             22 - Communications via serial port 2 to plc number 2
!
!--------------------------------------------------------------------------

LOGICAL;       GET_11,              GET_12,              \
               GET_21,              GET_22,              \
               GET_IF11,                   GET_IF12,                   \
               GET_IF21,                   GET_IF22

LOGICAL;       PUT_IF11,                   PUT_IF12,                   \
               PUT_IF21,                   PUT_IF22

LOGICAL;       ACTIVE11,            SLOW_POLL11,         OFFLINE11,    \
               ACTIVE12,            SLOW_POLL12,         OFFLINE12,    \
               ACTIVE21,            SLOW_POLL21,         OFFLINE21,    \
               ACTIVE22,            SLOW_POLL22,         OFFLINE22

LOGICAL;       DROP11_INIT,                DROP12_INIT,                \
               DROP21_INIT,                DROP22_INIT

!--------------------------------------------------------------------------
!      The following counters are used in controlling determination of PLC
!      path to use in communications.
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 4.1 (cont)

```
!          PLC_LOOP_COUNT - used to determine which path to use for
!             communications:
!                 if greater than 3 * plc_loop_count, use path 1
!                 if between 2 & 3 * plc_loop_count, use path 2
!                 if between 1 & 2 * plc_loop_count, use path 3
!                 if less than plc_loop_count, use path 4
!          PLC_RETRY_COUNT - number of logic passes spent trying to
!             communicate on a given path (initialized to 8).
!-----------------------------------------------------------------------------

NUMERIC;      PLC_LOOP_COUNT,             PLC_RETRY_COUNT:8

!-----------------------------------------------------------------------------
!      Bits in status register for plc 1
!-----------------------------------------------------------------------------

LOGICAL;      PLC1_HALTED,                PLC1_OUTPUTS_DISABLED,       \
              PLC1_RUNNING,               PLC1_MEMORY_LED_ON,          \
              PLC1_FORCE_LED_ON,          PLC1_I_O_LED_ON,             \
              PLC1_KEY_IN_HALT,           PLC1_KEY_DISABLE_OUT, \
              PLC1_KEY_IN_RUN,            PLC1_WRITE_PROTECT,          \
              PLC1_BATTERY_LOW_LED, PLC1_FIELD_12,       \
              PLC1_FIELD_13,       PLC1_FIELD_14,       \
              PLC1_FIELD_15,       PLC1_FIELD_16

!-----------------------------------------------------------------------------
!      Bits in status register for plc 2
!-----------------------------------------------------------------------------

LOGICAL;      PLC2_HALTED,                PLC2_OUTPUTS_DISABLED,       \
              PLC2_RUNNING,               PLC2_MEMORY_LED_ON,          \
              PLC2_FORCE_LED_ON,          PLC2_I_O_LED_ON,             \
              PLC2_KEY_IN_HALT,           PLC2_KEY_DISABLE_OUT, \
              PLC2_KEY_IN_RUN,            PLC2_WRITE_PROTECT,          \
              PLC2_BATTERY_LOW_LED, PLC2_FIELD_12,       \
              PLC2_FIELD_13,       PLC2_FIELD_14,       \
              PLC2_FIELD_15,       PLC2_FIELD_16

!-----------------------------------------------------------------------------
!      Logicals used in determining primary plc
!-----------------------------------------------------------------------------

LOGICAL;      LTI11_PRIMARY,       LTI21_PRIMARY,       \
              LTI12_PRIMARY,       LTI22_PRIMARY,       \
              PLC1_PRIMARY,               PLC2_PRIMARY,                \
              PLC1_HISTORY,               PLC2_HISTORY

!-----------------------------------------------------------------------------
!      Variables used in calls
!-----------------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 4.1 (cont)

```
NUMERIC;      ZERO:0,              ONE_TWENTY_EIGHT:128

!-----------------------------------------------------------------------------
!      All logic placed between the Key Words TABLES;  and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization code.
!-----------------------------------------------------------------------------

TABLES;

RECALL; SETMSG_C16, , CRISP_TT

RESTART;

!-----------------------------------------------------------------------------
! Get data on PLC status
!-----------------------------------------------------------------------------

!-----------------------------------------------------------------------------
!      Zero plc1_primary if plc1 is not active plc
!-----------------------------------------------------------------------------

RECALL; MAKZERO, (~ACTIVE11 & ~ACTIVE21), PLC1_PRIMARY

!-----------------------------------------------------------------------------
!      Unpack status register from plc1 (register 8186 in SQUARE D Model 400
!      Processors)
!-----------------------------------------------------------------------------

RECALL; BIT_PACK, ((GET_11 & ACTIVE11) | (GET_21 & ACTIVE21)), \
        PLC_STATUS_REGISTER, PLC1_HALTED

!-----------------------------------------------------------------------------
!      Zero plc2_primary if plc2 is not active plc
!-----------------------------------------------------------------------------

RECALL; MAKZERO, (~ACTIVE12 & ~ACTIVE22), PLC2_PRIMARY

!-----------------------------------------------------------------------------
!      Unpack status register from plc2
!-----------------------------------------------------------------------------

RECALL; BIT_PACK, ((GET_12 & ACTIVE12) | (GET_22 & ACTIVE22)), \
        PLC_STATUS_REGISTER, PLC2_HALTED

!-----------------------------------------------------------------------------
!      For SQUARE D Model 400 processors, ladder logic in the plc is used to
!      transfer register 8161, bit 20 to the first bit of the first input
!      register so we can determine if the plc is primary.  This bit cannot
!      be read directly from the plc.
!-----------------------------------------------------------------------------
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 4.1 (cont)

```
RECALL; BIT_PACK, , PLC_INPUTS(0), D_I(0)

!------------------------------------------------------------------------------
!      Determine which LTI bit is set to primary
!------------------------------------------------------------------------------

LTI11_PRIMARY = FALSE
LTI21_PRIMARY = FALSE
LTI12_PRIMARY = FALSE
LTI22_PRIMARY = FALSE

LET; GET_11 & ~PLC1_HALTED, LTI11_PRIMARY = D_I(0)
LET; GET_21 & ~PLC1_HALTED, LTI21_PRIMARY = D_I(0)
LET; GET_12 & ~PLC2_HALTED, LTI12_PRIMARY = D_I(0)
LET; GET_22 & ~PLC2_HALTED, LTI22_PRIMARY = D_I(0)

!------------------------------------------------------------------------------
!      Determine if plc1 is the primary plc
!------------------------------------------------------------------------------

PLC1_PRIMARY =  PLC1_RUNNING & \
                ~PLC1_HALTED & \
                ~PLC1_KEY_DISABLE_OUT & \
                ~PLC1_KEY_IN_HALT & \
                (ACTIVE11 | ACTIVE21) & \
                (LTI11_PRIMARY | LTI21_PRIMARY)

!------------------------------------------------------------------------------
!      Determine if plc2 is the primary plc
!------------------------------------------------------------------------------

PLC2_PRIMARY =  PLC2_RUNNING & \
                ~PLC2_HALTED & \
                ~PLC2_KEY_DISABLE_OUT & \
                ~PLC2_KEY_IN_HALT & \
                (ACTIVE12 | ACTIVE22) & \
                (LTI12_PRIMARY | LTI22_PRIMARY)

!------------------------------------------------------------------------------
!      Transfer input registers to output registers so data can be written
!      back to plc write registers.
!------------------------------------------------------------------------------

RECALL; TRANSFER, (PLC1_PRIMARY | PLC2_PRIMARY), PLC_INPUTS(0), ZERO, \
        PLC_OUTPUTS(0), ZERO, ONE_TWENTY_EIGHT

!------------------------------------------------------------------------------
!      This code is used in determining which port/plc combination we should
!      be trying to talk to.  It utilizes a round robin approach, trying first
!      one combination and then the next if the first is not successful.
!------------------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 4.1 (cont)

```
!---------------------------------------------------------------------------
!      Wait until IDI finishes initializing ports before attempting to
!      communicate with a PLC
!---------------------------------------------------------------------------

JUMP; WAIT_AWHILE, ~DROP11_INIT
JUMP; WAIT_AWHILE, ~DROP21_INIT
JUMP; WAIT_AWHILE, ~DROP12_INIT
JUMP; WAIT_AWHILE, ~DROP22_INIT

!---------------------------------------------------------------------------
!      Skip determination of path to use if everything is OK
!---------------------------------------------------------------------------

JUMP; WAIT_AWHILE, PLC1_PRIMARY & GET_11 & ~SLOW_POLL11
JUMP; WAIT_AWHILE, PLC1_PRIMARY & GET_21 & ~SLOW_POLL21
JUMP; WAIT_AWHILE, PLC2_PRIMARY & GET_12 & ~SLOW_POLL12
JUMP; WAIT_AWHILE, PLC2_PRIMARY & GET_22 & ~SLOW_POLL22

!---------------------------------------------------------------------------
!      Find correct port
!---------------------------------------------------------------------------

!---------------------------------------------------------------------------
!      PLC_RETRY_COUNT is used for setting how long the logic should try to
!      communicate with a given port before moving on to the next one.
!
!      if ( PLC_LOOP_COUNT >  (3 * PLC_RETRY_COUNT)), use Port 1 PLC 1 path
!
!      if ( PLC_LOOP_COUNT >  (2 * PLC_RETRY_COUNT)) & \
!         ( PLC_LOOP_COUNT <= (3 * PLC_RETRY_COUNT)), use Port 1 PLC 2 path
!
!      if ( PLC_LOOP_COUNT >  (    PLC_RETRY_COUNT)) & \
!         ( PLC_LOOP_COUNT <= (2 * PLC_RETRY_COUNT)), use Port 2 PLC 1 path
!
!      if ( PLC_LOOP_COUNT <  (    PLC_RETRY_COUNT)), use Port 2 PLC 2 path
!---------------------------------------------------------------------------

!---------------------------------------------------------------------------
!      If we have been through all four paths, reset counter and start again
!---------------------------------------------------------------------------

        LET; (PLC_LOOP_COUNT <= 0), \
        PLC_LOOP_COUNT = PLC_RETRY_COUNT * 4

!---------------------------------------------------------------------------
!      Decrement counter if we don't have a primary PLC.
!---------------------------------------------------------------------------

LET; (~PLC1_PRIMARY & ~PLC2_PRIMARY & (PLC_LOOP_COUNT > 0)), \
        PLC_LOOP_COUNT = PLC_LOOP_COUNT - 1
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 4.1 (cont)

```
!-----------------------------------------------------------------------------
!      If selected path is in slow poll, decrement counter.
!-----------------------------------------------------------------------------

LET; (PLC1_PRIMARY & GET_11 & SLOW_POLL11), \
        PLC_LOOP_COUNT = PLC_LOOP_COUNT - 1
LET; (PLC1_PRIMARY & GET_21 & SLOW_POLL21), \
        PLC_LOOP_COUNT = PLC_LOOP_COUNT - 1
LET; (PLC2_PRIMARY & GET_12 & SLOW_POLL12), \
        PLC_LOOP_COUNT = PLC_LOOP_COUNT - 1
LET; (PLC2_PRIMARY & GET_22 & SLOW_POLL22), \
        PLC_LOOP_COUNT = PLC_LOOP_COUNT - 1

!-----------------------------------------------------------------------------
!      1) Select path depending on value of PLC_LOOP_COUNT
!      2) Zero all other path variables
!      3) Reset loop count if find valid path
!-----------------------------------------------------------------------------

LET; (PLC_LOOP_COUNT > (3 * PLC_RETRY_COUNT)), GET_11 = TRUE
CALL; MAKZERO, GET_11, \
        GET_IF12, GET_IF21, GET_IF22, \
        LTI12_PRIMARY, LTI21_PRIMARY, LTI22_PRIMARY, \
        GET_12, GET_21, GET_22, PUT_IF12, PUT_IF21, PUT_IF22, PLC2_PRIMARY
LET; (PLC1_PRIMARY & GET_11 & ~SLOW_POLL11), PLC_LOOP_COUNT = 4 * PLC_RETRY_COUNT

LET; ( (PLC_LOOP_COUNT > (2 * PLC_RETRY_COUNT)) & \
        (PLC_LOOP_COUNT <= (3 * PLC_RETRY_COUNT)) ), GET_12 = TRUE
CALL; MAKZERO, GET_12, \
        GET_IF11, GET_IF21, GET_IF22, \
        LTI11_PRIMARY, LTI21_PRIMARY, LTI22_PRIMARY, \
        GET_11, GET_21, GET_22, PUT_IF11, PUT_IF21, PUT_IF22, PLC1_PRIMARY
LET; (PLC2_PRIMARY & GET_12 & ~SLOW_POLL12), PLC_LOOP_COUNT = 3 * PLC_RETRY_COUNT

LET; ( (PLC_LOOP_COUNT > PLC_RETRY_COUNT)  & \
        (PLC_LOOP_COUNT <= (2 * PLC_RETRY_COUNT)) ), GET_21 = TRUE
CALL; MAKZERO, GET_21, \
        GET_IF11, GET_IF12, GET_IF22, \
        LTI11_PRIMARY, LTI12_PRIMARY, LTI22_PRIMARY, \
        GET_11, GET_12, GET_22, PUT_IF11, PUT_IF12, PUT_IF22, PLC2_PRIMARY
LET; (PLC1_PRIMARY & GET_21 & ~SLOW_POLL21), PLC_LOOP_COUNT = 2 * PLC_RETRY_COUNT

LET; (PLC_LOOP_COUNT <= PLC_RETRY_COUNT), GET_22 = TRUE
CALL; MAKZERO, GET_22, \
        GET_IF11, GET_IF12, GET_IF21, \
        LTI11_PRIMARY, LTI12_PRIMARY, LTI21_PRIMARY, \
        GET_11, GET_12, GET_21, PUT_IF11, PUT_IF12, PUT_IF21, PLC1_PRIMARY
LET; (PLC2_PRIMARY & GET_22 & ~SLOW_POLL22), PLC_LOOP_COUNT = PLC_RETRY_COUNT

LABEL; WAIT_AWHILE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 4.1 (cont)

```
!---------------------------------------------------------------------------
!      Set up history bits so we can detect a change in primary PLC
!---------------------------------------------------------------------------

PLC1_HISTORY = PLC1_PRIMARY
PLC2_HISTORY = PLC2_PRIMARY

!
! READ PORT 1 PLC 1
!
LET; GET_11, GET_IF11 = TRUE
LET; GET_11, PUT_IF11 = TRUE

!
! READ PORT 1 PLC 2
!
LET; GET_12, GET_IF12 = TRUE
LET; GET_12, PUT_IF12 = TRUE

!
! READ PORT 2 PLC 1
!
LET; GET_21, GET_IF21 = TRUE
LET; GET_21, PUT_IF21 = TRUE

!
! READ PORT 2 PLC 2
!
LET; GET_22, GET_IF22 = TRUE
LET; GET_22, PUT_IF22 = TRUE

END;
```


© 1991 Square D All Rights Reserved

Multi-drop Interface

In a multi-drop environment, the most efficient way to handle communications is by looping through the devices, reading from each one in turn until all have been processed. However, since the primary reason for communicating with the devices is to be able to remotely send setpoints to the controllers, writes need to take precedence over the reads. If all reads are suspended when a write must be made, it is possible that the controllers at the end of the multi-drop may seldom get processed. The use of arrays enables the user to suspend the read on only the device that must be written to.

Example 4.2

The following example includes all the necessary declarations and logic to function correctly with a corresponding UCF for the IDI product. The example is provided only to help the user understand how to implement a multi-drop configuration.

The Example Declarations and Logic follows.

```
!***************************************************************************
!*      Sample Code for Example 4.2                                        *
!*                                                                         *
!*      Task 4.2 - How to implement a multi-drop PLC system                *
!***************************************************************************

!---------------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------------

TITLE;"        INTERMEDIATE VARIABLE DECLARATIONS"
!
LOGICAL;       NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
               ICCDSA, PAUSE, DEBUG, DEBUG_BREAK,

LOGICAL;       CRISP_TT:TRUE, CRISP_LP     , CRISP_CL    , CRISP_SS    , \
               CRISP_LOGFIL , CRISP_ALMFIL , CRISP_OUTPUT, CRISP_ERROR , \
               CRISP_CSR_DLY, CRISP_REPORT , CRISP_CASTER, CRISP_DEV03 , \
               CRISP_DEV04  , CRISP_DEV05  , CRISP_DEV06 , CRISP_DEV07 , \
               CRISP_DEV08  , CRISP_DEV09  , CRISP_DEV10 , CRISP_DEV11 , \
               CRISP_DEV12  , CRISP_DEV13  , CRISP_DEV14 , CRISP_DEV15 , \
               CRISP_DEV16  , CRISP_DEV17  , CRISP_DEV18 , CRISP_DEV19 , \
               CRISP_DEV20  , CRISP_DEV21  , CRISP_DEV22 , CRISP_DEV23

LOGICAL;       DDMLM_01     , DDMLM_02    , DDMLM_03    , DDMLM_04    , \
               DDMLM_05     , DDMLM_06    , DDMLM_07    , DDMLM_08    , \
               DDMLM_09     , DDMLM_10    , DDMLM_11    , DDMLM_12    , \
               DDMLM_13     , DDMLM_14    , DDMLM_15    , DDMLM_16    , \
               DDMLM_17     , DDMLM_18    , DDMLM_19    , DDMLM_20    , \
               DDMLM_21     , DDMLM_22    , DDMLM_23    , DDMLM_24    , \
               DDMLM_25     , DDMLM_26    , DDMLM_27    , DDMLM_28    , \
               DDMLM_29     , DDMLM_30    , DDMLM_31    , DDMLM_32
```

(Continued on next page.)

Example 4.2 (cont)

```
!---------------------------------------------------------------------------
!      Number of SLCs being polled and number of last one (one less than
!      number of SLCs since arrays start at element 0)
!---------------------------------------------------------------------------

CONSTANT;      NUMBER_OF_SLCS:8,     FINAL_INDEX:7

!---------------------------------------------------------------------------
!      Loop counter
!---------------------------------------------------------------------------

NUMERIC;       SLC_INDEX

!---------------------------------------------------------------------------
!      Input and output variables
!---------------------------------------------------------------------------

FLOAT;         SLC_INPUTS(NUMBER_OF_SLCS),  \
               SLC_OUTPUTS(NUMBER_OF_SLCS)

!---------------------------------------------------------------------------
!      Read logicals
!---------------------------------------------------------------------------

LOGICAL;       GET_IF(NUMBER_OF_SLCS)

!---------------------------------------------------------------------------
!      Read logicals history states - used to enable IDI to resume reading
!      with the SLC that was next to be read prior to processing a write
!---------------------------------------------------------------------------

LOGICAL;       GET_IF_HISTORY(NUMBER_OF_SLCS)

!---------------------------------------------------------------------------
!      Write logicals
!---------------------------------------------------------------------------

LOGICAL;       PUT_IF(NUMBER_OF_SLCS)

!---------------------------------------------------------------------------
!      The minimum port scan time required to process all of the devices
!      on a multi-drop line.  If the actual scan time is less than this
!      variable, IDI will delay until this time is reached before starting
!      the next scan.  If set too low, CPU usage for this process will
!      increase since IDI will continue to scan as fast as possible.
!      For this example, the scan time will be set to 8 seconds (800).
!---------------------------------------------------------------------------

NUMERIC;       MIN_UPDATE_TIME:800
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 4.2 (cont)

```
!---------------------------------------------------------------------------
!      The normal value for MIN_UPDATE_TIME when only reads are being
!      processed.  The scan time will be shortened when writes are required.
!---------------------------------------------------------------------------

NUMERIC;       NORMAL_UPDATE_TIME

!---------------------------------------------------------------------------
!      The following variable is true if a write to at least one device is
!      required.
!---------------------------------------------------------------------------

LOGICAL;       PUT_IF_REQUIRED

!---------------------------------------------------------------------------
!      This variable will be true if writes were required and have not yet
!      been complete.
!---------------------------------------------------------------------------

LOGICAL;       PUT_IF_TO_DO

!---------------------------------------------------------------------------
!      All logic placed between the Key Words TABLES;  and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization code.
!---------------------------------------------------------------------------

TABLES;

RECALL; SETMSG_C16, , CRISP_TT

RESTART;

!---------------------------------------------------------------------------
!      SLC Polling Logic
!---------------------------------------------------------------------------

!---------------------------------------------------------------------------
!      If there was no PUT to do from the previous pass of logic, skip this
!      part of the code (PUT_IF_REQUIRED is false).
!---------------------------------------------------------------------------

JUMP; CHECK_FOR_PUTS, ~PUT_IF_REQUIRED

!---------------------------------------------------------------------------
!      Otherwise see if it has completed.
!---------------------------------------------------------------------------

PUT_IF_TO_DO = TRUE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 4.2 (cont)

```
RECALL; OR_C16, , PUT_IF(0), PUT_IF_TO_DO, NUMBER_OF_SLCS

JUMP; CHECK_FOR_PUTS, PUT_IF_TO_DO

!--------------------------------------------------------------------------
!      If PUTS completed, recover GET_IF array and resume polling
!      where we left off.
!--------------------------------------------------------------------------

RECALL; TRANSFER, , GET_IF_HISTORY(0), ZERO, GET_IF(0), ZERO, NUMBER_OF_SLCS

!--------------------------------------------------------------------------
!      Return port scan time to its normal interval until next PUT must
!      be processed.
!--------------------------------------------------------------------------

MIN_UPDATE_TIME = NORMAL_UPDATE_TIME

LABEL; CHECK_FOR_PUTS

!--------------------------------------------------------------------------
!      Determine if any PUTS need to be made
!--------------------------------------------------------------------------

PUT_IF_REQUIRED = FALSE

RECALL; OR_C16, , PUT_IF(0), PUT_IF_REQUIRED, NUMBER_OF_SLCS

!--------------------------------------------------------------------------
!      If no PUTs need to be processed, skip the following code and continue
!      to read data from the SLCs
!--------------------------------------------------------------------------

JUMP; SKIP_PUTS, ~PUT_IF_REQUIRED

!--------------------------------------------------------------------------
!      Save GET_IF array if PUT_IF_REQUIRED is true
!--------------------------------------------------------------------------

RECALL; TRANSFER, , GET_IF(0), ZERO, GET_IF_HISTORY(0), ZERO, NUMBER_OF_SLCS

!--------------------------------------------------------------------------
!      Set the GET_IF elements which correspond to the PUT_IF elements so
!      that the writes will be followed immediately by reads.
!--------------------------------------------------------------------------

SLC_INDEX = 0

LABEL;  GET_IF_LOOP
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 4.2 (cont)

```
!-----------------------------------------------------------------------------
!      Set the GET_IF bits which correspond to any PUT_IF bits
!-----------------------------------------------------------------------------

    GET_IF(SLC_INDEX) = PUT_IF(SLC_INDEX)

!-----------------------------------------------------------------------------
!      Increment the loop counter
!-----------------------------------------------------------------------------

    SLC_INDEX = SLC_INDEX + 1

!-----------------------------------------------------------------------------
!      If all SLCs have not been serviced, do the next one
!-----------------------------------------------------------------------------

JUMP;   GET_IF_LOOP, (SLC_INDEX < NUMBER_OF_SLCS)

!-----------------------------------------------------------------------------
!      Set port scan time to 0 if a PUT is to be done.  This will cause IDI
!      to begin the next scan as soon as the current one completes.
!-----------------------------------------------------------------------------

MIN_UPDATE_TIME = 0

LABEL; SKIP_PUTS

!-----------------------------------------------------------------------------
!      Skip initializing the GET_IFs if PUT_IF_REQUIRED is true or if the
!      GET has not completed on the last SLC on the drop.
!-----------------------------------------------------------------------------

JUMP; SKIP_GETS, (PUT_IF_REQUIRED | GET_IF(FINAL_INDEX))

!-----------------------------------------------------------------------------
!      Initialize loop counter
!-----------------------------------------------------------------------------

SLC_INDEX = 0

LABEL;  POLLING_LOOP

!-----------------------------------------------------------------------------
!      Set GET_IF logicals to true.
!-----------------------------------------------------------------------------

    GET_IF(SLC_INDEX) = TRUE
```

(Continued on next page.)

Example 4.2 (cont)

```
!---------------------------------------------------------------------------
!      Increment the loop counter
!---------------------------------------------------------------------------

    SLC_INDEX = SLC_INDEX + 1

!---------------------------------------------------------------------------
!      If all SLCs have not been serviced, do the next one
!---------------------------------------------------------------------------

JUMP;   POLLING_LOOP, (SLC_INDEX < NUMBER_OF_SLCS)

LABEL;  SKIP_GETS

END;
```

Sample UCF Code

Some sample code from a UCF in the following shows how the previous code would be implemented for two multi-dropped devices. Additional devices would be added by duplicating the code and modifying the device numbers and array elements. Without writes to perform, this UCF would sequentially read from all of the SLCs every 8 seconds (the default port scan time). If a write is required, the port scan time is set to zero and GETs are suspended until the write completes, at which time polling resumes with the SLC it was last preparing to read. The exception to this is that all writes to a SLC are immediately followed by a read of that particular device.

```
LOG_PORT>                                    ;1            ! TXnn:

LOG_PORT_DROP>        ;1                    ! SLC 1

DEVICE_MEMORY_TYPE>                          ;FLOAT
DEVICE_MEMORY_ADDRESS>                       ;0
DATUM_COUNT>                                 ;1
ADB_LOCATION>                                ;"SLC_OUTPUTS(0)"
PUT_AND_CLR_TRIGGER_IF>             PUT_IF(0);0
!
DEVICE_MEMORY_TYPE>                          ;FLOAT
DEVICE_MEMORY_ADDRESS>                       ;0
DATUM_COUNT>                                 ;1
ADB_LOCATION>                                ;"SLC_INPUTS(0)"
GET_AND_CLR_TRIGGER_IF>             GET_IF(0);0
!
LOG_PORT_DROP>        ;2                    ! SLC 2

DEVICE_MEMORY_TYPE>                          ;FLOAT
DEVICE_MEMORY_ADDRESS>                       ;0
DATUM_COUNT>                                 ;1
ADB_LOCATION>                                ;"SLC_OUTPUTS(1)"
PUT_AND_CLR_TRIGGER_IF>             PUT_IF(1);0
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Sample UCF Code (cont)

```
!
DEVICE_MEMORY_TYPE>                          ;FLOAT
DEVICE_MEMORY_ADDRESS>                       ;0
DATUM_COUNT>                                 ;1
ADB_LOCATION>                                ;"SLC_INPUTS(1)"
GET_AND_CLR_TRIGGER_IF>              GET_IF(1);0
!
SET_MIN_PORT_SCAN_TO> MIN_UPDATE_TIME;      ! Minimum scan interval
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

General

This section defines how to open, read, write and close small and large files. The following subsections define various methods of performing these functions.

Solution A

Often a CRISP Developer must save a portion of the database to an RMS disk file for later recall. The CRISP calls provided to accomplish this task are asynchronous, meaning they do not complete their function prior to the logic continuing with the next logic step. This action requires some attention to program timing to insure that the file data is in sync with the rest of the realtime data.

The following example shows how to perform basic file operations for a small file (50 records).

Explanation A

This program uses the FIL_OPEN call to test for an existing file on disk. If located, the open operation is completed and all of the records are read into the database using a single FIL_READ call. If the file is not located, a second FIL_OPEN is executed to create the file. A single FIL_WRITE call is used to initialize each record for later reading. Finally, a FIL_CLOSE is demonstrated for closing the file.

Example 5.1 A

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to implement file input/output. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 5.1 A                                      *
!*                                                                         *
!*      Task 5.1 - How to implement File Operations                        *
!***************************************************************************

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------

LOGICAL;       NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
               ICCDSA, PAUSE, , ,

CONSTANT;      ZERO:0, ONE:1
STRING;        PAD[25]:"-------------------------"
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 A (cont)

```
LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , CRISP_DEV01 , CRISP_DEV02 , CRISP_DEV03 , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23

!----------------------------------------------------------------------
!       Return status codes
!----------------------------------------------------------------------
CONSTANT;      SUCCESS:1, REC_NOT_FOUND:98994, FILE_NOT_FOUND:98962,     \
               CHAN_IN_USE:-4

CONSTANT;      FILE1_DIM:51,                 \
               FILE1_SIZE:50,        \
               NEG_FILE1_SIZE:-50

!----------------------------------------------------------------------
!       DCL path and file name definition
!
!       If subdirectory [CRISP.DATA] does not exist on your system,
!       either create it using the following DCL command:
!
!       CREATE /DIR DISK$USER:[CRISP.DATA]
!
!       or revise this declaration to an existing subdirectory
!----------------------------------------------------------------------
STRING;        FILE1_NAME[]:"DISK$USER:[CRISP.DATA]FILE1.DAT"

       !-------------------------------------------------------------
       !                     NOTE
       !-------------------------------------------------------------
       !      Assign a unique channel number to each file or allow
       !      CRISP to perform the assignment by declaring channel
       !      number = 0
       !-------------------------------------------------------------
LONG;          FILE1_CHANNEL:10

!----------------------------------------------------------------------
!       FIL_OPEN Options
!
!       Check = 4 (shared read/write) + 16 (file must exist) = 20
!       RW    = 4 (shared read/write) + 1  (create file)     = 5
!
!----------------------------------------------------------------------
LONG;          FILE1_CHECK_OPTION:20,       \
               FILE1_RW_OPTION:5
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 A (cont)

```
!----------------------------------------------------------------------
!       FIL_LOG_MESSAGE Options (DEBUG)
!
!       Log   =  4 (print good reads) + 16 (print good writes) +
!               32 (print records not found during reads)     = 52
!
!       Set FILE1_DEBUG = TRUE to prior to starting CRISP to enable
!       log message operations
!----------------------------------------------------------------------
LONG;          FILE1_LOG_OPTION:52
LOGICAL;       FILE1_DEBUG:FALSE

LONG;          FILE1_REC_ID:1,       \
               FILE1_REC_LEN:0

LONG;          FILE1_DEFINE_STATUS, FILE1_OPEN_STATUS1,   \
               FILE1_CLOSE_STATUS, FILE1_READ_STATUS,     \
               FILE1_WRITE_STATUS, FILE1_OPEN_STATUS2,    \
               FILE1_LOG_STATUS

LOGICAL;       FILE1_DEFINE_DONE, FILE1_OPEN_DONE1,       \
               FILE1_CLOSE_DONE, FILE1_READ_DONE,         \
               FILE1_WRITE_DONE, FILE1_OPEN_DONE2,        \
               FILE1_LOG_DONE

LOGICAL;       FILE1_DEFINE_OK, FILE1_OPEN_OK,            \
               FILE1_CLOSE_OK, FILE1_READ_OK,             \
               FILE1_WRITE_OK, FILE1_CREATE_OK

!----------------------------------------------------------------------
!      CRISP table into which file records are read/written to
!----------------------------------------------------------------------
STRING;        STR(FILE1_DIM)[25]
LONG;          LNG(FILE1_DIM)
LOGICAL;       BIT(FILE1_DIM)
FLOAT;         FLT(FILE1_DIM)

!----------------------------------------------------------------------
!      These variables initiate each of the FIL calls
!----------------------------------------------------------------------
LOGICAL;       FILE1_OPEN_TRIG, FILE1_READ_TRIG, FILE1_WRITE_TRIG,\
               FILE1_CREATE_TRIG, FILE1_CLOSE_TRIG

LOGICAL;       FILE1_OPEN_ENABLE, FILE1_READ_ENABLE, FILE1_WRITE_ENABLE, \
               FILE1_STILL_OPEN
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 A (cont)

```
!-------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!-------------------------------------------------------------------
TABLES;

RECALL; SETMSG_C16, , CRISP_TT

!-------------------------------------------------------------------
!      Define record for future file operations
!      The call returns a value for record length
!-------------------------------------------------------------------

RECALL; FIL_DEFINE_REC, , \
       FILE1_DEFINE_STATUS, FILE1_REC_ID, FILE1_REC_LEN,\
       STR(1),LNG(1),BIT(1),FLT(1),        \
       STR(2),LNG(2),BIT(2),FLT(2),        \
       STR(3),LNG(3),BIT(3),FLT(3),        \
       STR(4),LNG(4),BIT(4),FLT(4),        \
       STR(5),LNG(5),BIT(5),FLT(5),        \
       STR(6),LNG(6),BIT(6),FLT(6),        \
       STR(7),LNG(7),BIT(7),FLT(7),        \
       STR(8),LNG(8),BIT(8),FLT(8),        \
       STR(9),LNG(9),BIT(9),FLT(9),        \
       STR(10),LNG(10),BIT(10),FLT(10),           \
       STR(11),LNG(11),BIT(11),FLT(11),           \
       STR(12),LNG(12),BIT(12),FLT(12),           \
       STR(13),LNG(13),BIT(13),FLT(13),           \
       STR(14),LNG(14),BIT(14),FLT(14),           \
       STR(15),LNG(15),BIT(15),FLT(15),           \
       STR(16),LNG(16),BIT(16),FLT(16),           \
       STR(17),LNG(17),BIT(17),FLT(17),           \
       STR(18),LNG(18),BIT(18),FLT(18),           \
       STR(19),LNG(19),BIT(19),FLT(19),           \
       STR(20),LNG(20),BIT(20),FLT(20),           \
       STR(21),LNG(21),BIT(21),FLT(21),           \
       STR(22),LNG(22),BIT(22),FLT(22),           \
       STR(23),LNG(23),BIT(23),FLT(23),           \
       STR(24),LNG(24),BIT(24),FLT(24),           \
       STR(25),LNG(25),BIT(25),FLT(25),           \
       STR(26),LNG(26),BIT(26),FLT(26),           \
       STR(27),LNG(27),BIT(27),FLT(27),           \
       STR(28),LNG(28),BIT(28),FLT(28),           \
       STR(29),LNG(29),BIT(29),FLT(29),           \
       STR(30),LNG(30),BIT(30),FLT(30),           \
       STR(31),LNG(31),BIT(31),FLT(31),           \
       STR(32),LNG(32),BIT(32),FLT(32),           \
       STR(33),LNG(33),BIT(33),FLT(33),           \
       STR(34),LNG(34),BIT(34),FLT(34),           \
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 A (cont)

```
        STR(35),LNG(35),BIT(35),FLT(35),            \
        STR(36),LNG(36),BIT(36),FLT(36),            \
        STR(37),LNG(37),BIT(37),FLT(37),            \
        STR(38),LNG(38),BIT(38),FLT(38),            \
        STR(39),LNG(39),BIT(39),FLT(39),            \
        STR(40),LNG(40),BIT(40),FLT(40),            \
        STR(41),LNG(41),BIT(41),FLT(41),            \
        STR(42),LNG(42),BIT(42),FLT(42),            \
        STR(43),LNG(43),BIT(43),FLT(43),            \
        STR(44),LNG(44),BIT(44),FLT(44),            \
        STR(45),LNG(45),BIT(45),FLT(45),            \
        STR(46),LNG(46),BIT(46),FLT(46),            \
        STR(47),LNG(47),BIT(47),FLT(47),            \
        STR(48),LNG(48),BIT(48),FLT(48),            \
        STR(49),LNG(49),BIT(49),FLT(49),            \
        STR(50),LNG(50),BIT(50),FLT(50)

!----------------------------------------------------------------------
!       Log file messages to console printer (DEBUG)
!----------------------------------------------------------------------
RECALL; FIL_LOG_MESSAGE, FILE1_DEBUG, \
        FILE1_LOG_STATUS, FILE1_LOG_DONE, FILE1_LOG_OPTION


!----------------------------------------------------------------------
!       The Logic between the Key Word RESTART; and the Key Word
!       END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;


!----------------------------------------------------------------------
!       Open File on successful FIL_DEFINE
!
!       This logic will execute if CRISP is started WARM
!       or with the RESTORE option
!----------------------------------------------------------------------
SET; (FILE1_DEFINE_STATUS == SUCCESS), FILE1_OPEN_TRIG = TRUE

SET; FILE1_OPEN_TRIG, FILE1_OPEN_ENABLE = TRUE

!
!>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE1_READ_LOGIC, ~FILE1_OPEN_ENABLE
!>>>>>>>>>>>>>>>>>>>>>
!

!----------------------------------------------------------------------
!       Check for existing file
!----------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 A (cont)

```
CALL; FIL_OPEN, FILE1_OPEN_TRIG, \
       FILE1_CHANNEL, FILE1_OPEN_STATUS1, FILE1_OPEN_DONE1, \
       FILE1_NAME, FILE1_REC_LEN, FILE1_SIZE, FILE1_CHECK_OPTION

FILE1_OPEN_OK = (FILE1_OPEN_STATUS1 == SUCCESS) & FILE1_OPEN_DONE1
FILE1_STILL_OPEN = (FILE1_OPEN_STATUS1 == CHAN_IN_USE) & FILE1_OPEN_DONE1

FILE1_OPEN_TRIG = FALSE

!----------------------------------------------------------------------
!      Initiate immediate FIL_READ if file opened successfully
!----------------------------------------------------------------------
SET; FILE1_OPEN_OK, FILE1_READ_TRIG = TRUE
SET; (FILE1_OPEN_OK | FILE1_STILL_OPEN) & FILE1_OPEN_DONE1, \
       FILE1_OPEN_ENABLE = FALSE

!----------------------------------------------------------------------
!      Create and Open File if it does not exist
!----------------------------------------------------------------------
SET; (FILE1_OPEN_STATUS1 == FILE_NOT_FOUND) & FILE1_OPEN_DONE1, \
       FILE1_CREATE_TRIG = TRUE

CALL; FIL_OPEN, FILE1_CREATE_TRIG, \
       FILE1_CHANNEL, FILE1_OPEN_STATUS2, FILE1_OPEN_DONE2, \
       FILE1_NAME, FILE1_REC_LEN, FILE1_SIZE, FILE1_RW_OPTION

FILE1_CREATE_OK = (FILE1_OPEN_STATUS2 == SUCCESS) & FILE1_OPEN_DONE2

!----------------------------------------------------------------------
!      Fill in String arrays for first file write
!----------------------------------------------------------------------
CALL; SCOPY, FILE1_CREATE_OK,\
       PAD, ZERO, STR(1), ZERO, NEG_FILE1_SIZE

MESSAGE; FILE1_OPEN_DONE2 & FILE1_CREATE_OK,\
       "@M@J***  FILE 1 WAS CREATED  ***@M@J"
MESSAGE; FILE1_OPEN_DONE2 & ~FILE1_CREATE_OK,\
       "@M@J***  FILE 1 NOT OPENED SUCCESSFULLY  ***@M@J"

FILE1_CREATE_TRIG = FALSE

!----------------------------------------------------------------------
!      Initiate immediate FIL_WRITE if file created successfully
!----------------------------------------------------------------------
SET; FILE1_CREATE_OK, FILE1_WRITE_TRIG = TRUE
SET; FILE1_OPEN_DONE2, FILE1_OPEN_ENABLE = FALSE

!
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 5.1 A (cont)

```
!<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE1_READ_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<
!
!----------------------------------------------------------------------
!      Read File on demand or on file open
!----------------------------------------------------------------------
SET; FILE1_READ_TRIG, FILE1_READ_ENABLE = TRUE

!
!>>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE1_WRITE_LOGIC, ~FILE1_READ_ENABLE
!>>>>>>>>>>>>>>>>>>>>>>
!

CALL; FIL_READ, FILE1_READ_TRIG, \
       FILE1_CHANNEL, FILE1_READ_STATUS, FILE1_READ_DONE, \
       ONE, FILE1_REC_ID

FILE1_READ_OK = (FILE1_READ_STATUS == SUCCESS) & FILE1_READ_DONE

SET; FILE1_READ_DONE, FILE1_READ_ENABLE = FALSE

MESSAGE; FILE1_READ_DONE & ~FILE1_READ_OK, \
       "@M@J***  FILE 1 READ NOT SUCCESSFUL  ***@M@J"

FILE1_READ_TRIG = FALSE

!
!<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE1_WRITE_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<
!
!----------------------------------------------------------------------
!      Write File on demand or on creation of new file
!----------------------------------------------------------------------
SET; FILE1_WRITE_TRIG, FILE1_WRITE_ENABLE = TRUE

!
!>>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE1_CLOSE_LOGIC, ~FILE1_WRITE_ENABLE
!>>>>>>>>>>>>>>>>>>>>>>
!

CALL; FIL_WRITE, FILE1_WRITE_TRIG, \
       FILE1_CHANNEL, FILE1_WRITE_STATUS, FILE1_WRITE_DONE, \
       ONE, FILE1_REC_ID

FILE1_WRITE_OK = (FILE1_WRITE_STATUS == SUCCESS) & FILE1_WRITE_DONE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 A (cont)

```
SET; FILE1_WRITE_DONE, FILE1_WRITE_ENABLE = FALSE

MESSAGE; FILE1_WRITE_DONE & ~FILE1_WRITE_OK,\
       "@M@J***  FILE 1 WRITE NOT SUCCESSFUL  ***@M@J"

FILE1_WRITE_TRIG = FALSE

!
!<<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE1_CLOSE_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<<
!
!----------------------------------------------------------------------
!      Close file on demand and clear status/state varables
!----------------------------------------------------------------------
CALL; FIL_CLOSE, FILE1_CLOSE_TRIG, \
       FILE1_CHANNEL, FILE1_CLOSE_STATUS, FILE1_CLOSE_DONE

CALL; MAKZERO, FILE1_CLOSE_DONE, \
       FILE1_CLOSE_TRIG, FILE1_DEFINE_STATUS, FILE1_OPEN_STATUS1,       \
       FILE1_CLOSE_STATUS, FILE1_READ_STATUS, FILE1_WRITE_STATUS,       \
       FILE1_OPEN_STATUS2, FILE1_LOG_STATUS, FILE1_DEFINE_DONE,         \
       FILE1_OPEN_DONE1, FILE1_CLOSE_DONE, FILE1_READ_DONE,             \
       FILE1_WRITE_DONE, FILE1_OPEN_DONE2, FILE1_LOG_DONE,              \
       FILE1_DEFINE_OK, FILE1_OPEN_OK, FILE1_READ_OK, FILE1_WRITE_OK,   \
       FILE1_CREATE_OK

END;
```


© 1991 Square D All Rights Reserved

Solution B

The following example shows one method to perform basic file operations for a medium to large file (1000 records).

Explanation B

This program uses the FIL_OPEN call to test for an existing file on disk. If located, the open operation is completed and all of the records are read into the database using a program loop and a FIL_READ call. The read is performed in blocks less than the total record count in order not to overload the file process. This block size may be optimized (increased/decreased) for the individual system.

If the file is not located, a second FIL_OPEN is executed to create the file. A second program loop is used along with a FIL_WRITE call to initialize each record for later reading. Finally, a FIL_CLOSE is demonstrated for closing the file.

Example 5.1 B

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to implement file input/output. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 5.1 B                                      *
!*                                                                         *
!*      Task 5.1 - How to implement File Operations                        *
!***************************************************************************

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, , ,

CONSTANT;       ZERO:0, ONE:1
STRING;         PAD[25]:"-------------------------"

LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , CRISP_DEV01 , CRISP_DEV02 , CRISP_DEV03 , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
!---------------------------------------------------------------------
!      Return status codes
!---------------------------------------------------------------------
CONSTANT;       SUCCESS:1, REC_NOT_FOUND:98994, FILE_NOT_FOUND:98962,   \
                CHAN_IN_USE:-4

LONG;           FILE2_BLK_DIM:101,         \
                FILE2_BLK_LMT:100

CONSTANT;       FILE2_DIM:1001,            \
                FILE2_REC_LMT:1000,        \
                NEG_FILE2_REC_LMT:1000

!---------------------------------------------------------------------
!      DCL path and file name definition
!
!      If subdirectory [CRISP.DATA] does not exist on your system,
!      either create it using the following DCL command:

!
!      CREATE /DIR DISK$USER:[CRISP.DATA]
!
!      or revise this declaration to an existing subdirectory
!---------------------------------------------------------------------
STRING;         FILE2_NAME[]:"DISK$USER:[CRISP.DATA]FILE2.DAT"

       !------------------------------------------------------------
       !                    NOTE
       !------------------------------------------------------------
       !      Assign a unique channel number to each file or allow
       !      CRISP to perform the assignment by declaring channel
       !      number = 0
       !------------------------------------------------------------
LONG;           FILE2_CHANNEL:20

!---------------------------------------------------------------------
!      FIL_OPEN Options
!
!      Check = 4 (shared read/write) + 16 (file must exist) = 20
!      RW    = 4 (shared read/write) + 1  (create file)     = 5
!
!---------------------------------------------------------------------
LONG;           FILE2_CHECK_OPTION:20,     \
                FILE2_RW_OPTION:5

!---------------------------------------------------------------------
!      FIL_LOG_MESSAGE Options (DEBUG)
!
!      Log   =  4 (print good reads) + 16 (print good writes) +
!               32 (print records not found during reads)     = 52
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
!
!      Set FILE1_DEBUG = TRUE to prior to starting CRISP to enable
!      log message operations
!---------------------------------------------------------------------
LONG;           FILE2_LOG_OPTION:52
LOGICAL;        FILE2_DEBUG:FALSE

!---------------------------------------------------------------------
!      Record ID is zero since read/writes define records as part of
!      there argument list (see FIL_READ, FIL_WRITE calls)
!---------------------------------------------------------------------
LONG;           FILE2_REC_ID:0,                         \
                FILE2_REC_LEN:0

LONG;           FILE2_READ_PTR(FILE2_DIM),              \
                FILE2_WRITE_PTR(FILE2_DIM)
LONG;           FILE2_READ_STATUS(FILE2_DIM),           \
                FILE2_WRITE_STATUS(FILE2_DIM)

LONG;           FILE2_DEFINE_STATUS, FILE2_OPEN_STATUS1,  \
                FILE2_CLOSE_STATUS, FILE2_OPEN_STATUS2,   \
                FILE2_LOG_STATUS

LOGICAL;        FILE2_READ_DONE(FILE2_DIM),       \
                FILE2_WRITE_DONE(FILE2_DIM)

LOGICAL;        FILE2_DEFINE_DONE, FILE2_OPEN_DONE1,     \
                FILE2_CLOSE_DONE, FILE2_OPEN_DONE2,      \
                FILE2_LOG_DONE

LOGICAL;        FILE2_DEFINE_OK, FILE2_OPEN_OK,          \
                FILE2_CLOSE_OK, FILE2_CREATE_OK

LONG;           FILE2_BLK_PTR, FILE2_REC_PTR
LONG;           FILE2_READ_PASS_CTR, FILE2_WRITE_PASS_CTR

!---------------------------------------------------------------------
!      CRISP table into which file records are read/written to and
!      used by logic Application
!---------------------------------------------------------------------
STRING;         STR(FILE2_DIM)[25]
LONG;           LNG(FILE2_DIM)
LOGICAL;        BIT(FILE2_DIM)
FLOAT;          FLT(FILE2_DIM)

!---------------------------------------------------------------------
!      These variables initiate each of the FIL calls
!---------------------------------------------------------------------
LOGICAL;        FILE2_OPEN_TRIG, FILE2_READ_TRIG, FILE2_WRITE_TRIG,\
                FILE2_CREATE_TRIG, FILE2_CLOSE_TRIG
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
LOGICAL;        FILE2_OPEN_ENABLE, FILE2_READ_ENABLE, FILE2_WRITE_ENABLE, \
                FILE2_READ_ABORT, FILE2_WRITE_ABORT, FILE2_STILL_OPEN,    \
                FILE2_READ_BLK_ON, FILE2_WRITE_BLK_ON

!----------------------------------------------------------------------
!       All logic placed between the Key Words TABLES; and
!       RESTART;  will be executed only on the first pass of
!       logic.  All code in this area is Initialization Code.
!----------------------------------------------------------------------
TABLES;

RECALL; SETMSG_C16, , CRISP_TT

!----------------------------------------------------------------------
!       Log file messages to console printer (DEBUG)
!----------------------------------------------------------------------
RECALL; FIL_LOG_MESSAGE, FILE2_DEBUG, \
        FILE2_LOG_STATUS, FILE2_LOG_DONE, FILE2_LOG_OPTION

!----------------------------------------------------------------------
!       Define record for file open operations only.
!       The call returns a value for record length
!----------------------------------------------------------------------
RECALL; FIL_DEFINE_REC, , \
        FILE2_DEFINE_STATUS, 2, FILE2_REC_LEN,\
        STR(0),LNG(0),BIT(0),FLT(0)

!----------------------------------------------------------------------
!       The Logic between the Key Word RESTART; and the Key Word
!       END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;


!----------------------------------------------------------------------
!       Open File on successful FIL_DEFINE
!
!       This logic will execute if CRISP is started WARM
!       or with the RESTORE option
!----------------------------------------------------------------------
SET; (FILE2_DEFINE_STATUS == SUCCESS), FILE2_OPEN_TRIG = TRUE

SET; FILE2_OPEN_TRIG, FILE2_OPEN_ENABLE = TRUE

!
!>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE2_READ_LOGIC, ~FILE2_OPEN_ENABLE
!>>>>>>>>>>>>>>>>>>>>>
!
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
!----------------------------------------------------------------------
!      Check for existing file
!----------------------------------------------------------------------
CALL; FIL_OPEN, FILE2_OPEN_TRIG, \
        FILE2_CHANNEL, FILE2_OPEN_STATUS1, FILE2_OPEN_DONE1, \
        FILE2_NAME, FILE2_REC_LEN, FILE2_DIM, FILE2_CHECK_OPTION

FILE2_OPEN_OK = (FILE2_OPEN_STATUS1 == SUCCESS) & FILE2_OPEN_DONE1
FILE2_STILL_OPEN = (FILE2_OPEN_STATUS1 == CHAN_IN_USE) & FILE2_OPEN_DONE1

FILE2_OPEN_TRIG = FALSE

!----------------------------------------------------------------------
!      Initiate immediate FIL_READ if file opened successfully
!----------------------------------------------------------------------
SET; FILE2_OPEN_OK, FILE2_READ_TRIG = TRUE
SET; (FILE2_OPEN_OK | FILE2_STILL_OPEN) & FILE2_OPEN_DONE1, \
        FILE2_OPEN_ENABLE = FALSE


!----------------------------------------------------------------------
!      Create and Open File if it does not exist
!----------------------------------------------------------------------
SET; (FILE2_OPEN_STATUS1 == FILE_NOT_FOUND) & FILE2_OPEN_DONE1, \
        FILE2_CREATE_TRIG = TRUE

CALL; FIL_OPEN, FILE2_CREATE_TRIG, \
        FILE2_CHANNEL, FILE2_OPEN_STATUS2, FILE2_OPEN_DONE2, \
        FILE2_NAME, FILE2_REC_LEN, FILE2_DIM, FILE2_RW_OPTION

FILE2_CREATE_OK = (FILE2_OPEN_STATUS2 == SUCCESS) & FILE2_OPEN_DONE2

!----------------------------------------------------------------------
!      Fill in String arrays for first file write
!----------------------------------------------------------------------
CALL; SCOPY, FILE2_CREATE_OK,\
        PAD, ZERO, STR(1), ZERO, NEG_FILE2_REC_LMT

MESSAGE; FILE2_OPEN_DONE2 & FILE2_CREATE_OK,\
        "@M@J***  FILE 1 WAS CREATED  ***@M@J"
MESSAGE; FILE2_OPEN_DONE2 & ~FILE2_CREATE_OK,\
        "@M@J***  FILE 1 NOT OPENED SUCCESSFULLY  ***@M@J"

FILE2_CREATE_TRIG = FALSE

!----------------------------------------------------------------------
!      Initiate immediate FIL_WRITE if file created successfully
!----------------------------------------------------------------------
SET; FILE2_CREATE_OK, FILE2_WRITE_TRIG = TRUE
SET; FILE2_OPEN_DONE2, FILE2_OPEN_ENABLE = FALSE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
!
!<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE2_READ_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<
!

!---------------------------------------------------------------------
!      Read File on demand or on file open
!---------------------------------------------------------------------
SET; FILE2_READ_TRIG, FILE2_READ_ENABLE = TRUE
SET; FILE2_READ_ENABLE, FILE2_REC_PTR = 1
SET; FILE2_READ_ENABLE, FILE2_READ_PASS_CTR = 0

!---------------------------------------------------------------------
!      Restart Read loop logic when last READ_DONE bit is TRUE
!      or on start of Read
!---------------------------------------------------------------------
FILE2_READ_BLK_ON = FILE2_READ_DONE(FILE2_BLK_LMT) | FILE2_READ_TRIG


LET; FILE2_READ_BLK_ON, FILE2_BLK_PTR = 1
LET; FILE2_READ_BLK_ON, FILE2_READ_PTR(FILE2_BLK_PTR) = FILE2_REC_PTR
LET; FILE2_READ_BLK_ON, FILE2_READ_DONE(FILE2_BLK_LMT) = FALSE
FILE2_READ_TRIG = FALSE

!
!>>>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE2_WRITE_LOGIC, ~FILE2_READ_ENABLE | ~FILE2_READ_BLK_ON
!>>>>>>>>>>>>>>>>>>>>>>>
!
!
!<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE2_READ_LOOP
!<<<<<<<<<<<<<<<<<<<<<<

!---------------------------------------------------------------------
!      Queue up a block of FIL_READs and pass status, done and
!      record pointers as array values. This allows the call to
!      correctly return the results of all reads
!---------------------------------------------------------------------

       !-------------------------------------------------------------
       !                    NOTE
       !-------------------------------------------------------------
       !      The file record definition is part of the call argument
       !      list. This allows data to be read directly into the
       !      final database destination (FILE2_REC_ID = 0)
       !-------------------------------------------------------------
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
RECALL; FIL_READ, FILE2_READ_BLK_ON,                                    \
        FILE2_CHANNEL, FILE2_READ_STATUS(FILE2_BLK_PTR),                \
        FILE2_READ_DONE(FILE2_BLK_PTR), FILE2_READ_PTR(FILE2_BLK_PTR), \
        FILE2_REC_ID, STR(FILE2_REC_PTR), LNG(FILE2_REC_PTR),           \
        BIT(FILE2_REC_PTR), FLT(FILE2_REC_PTR)

!----------------------------------------------------------------------
!      Increment pointers and conditionally return to top of read loop
!----------------------------------------------------------------------
LET; FILE2_READ_BLK_ON, FILE2_BLK_PTR = FILE2_BLK_PTR + 1
LET; FILE2_READ_BLK_ON, FILE2_REC_PTR = FILE2_REC_PTR + 1
LET; FILE2_READ_BLK_ON, FILE2_READ_PTR(FILE2_BLK_PTR) = FILE2_REC_PTR
FILE2_READ_BLK_ON = FILE2_READ_BLK_ON & (FILE2_BLK_PTR <= FILE2_BLK_LMT)
FILE2_READ_ENABLE = FILE2_READ_ENABLE & (FILE2_REC_PTR <= FILE2_REC_LMT)

!
!>>>>>>>>>>>>>>>>>>>>
JUMP; FILE2_READ_LOOP, FILE2_READ_BLK_ON & FILE2_READ_ENABLE
!>>>>>>>>>>>>>>>>>>>>
!

!
!<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE2_WRITE_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<
!

!
!----------------------------------------------------------------------
!      Write File on demand or on creation of new file
!----------------------------------------------------------------------
SET; FILE2_WRITE_TRIG, FILE2_WRITE_ENABLE = TRUE
SET; FILE2_WRITE_ENABLE, FILE2_REC_PTR = 1
SET; FILE2_WRITE_ENABLE, FILE2_WRITE_PASS_CTR = 0

!----------------------------------------------------------------------
!      Restart Write loop logic when last WRITE_DONE bit is TRUE
!      or on first pass
!----------------------------------------------------------------------
FILE2_WRITE_BLK_ON = FILE2_WRITE_DONE(FILE2_BLK_LMT) | FILE2_WRITE_TRIG

LET; FILE2_WRITE_BLK_ON, FILE2_BLK_PTR = 1
LET; FILE2_WRITE_BLK_ON, FILE2_WRITE_PTR(FILE2_BLK_PTR) = FILE2_REC_PTR
LET; FILE2_WRITE_BLK_ON, FILE2_WRITE_DONE(FILE2_BLK_LMT) = FALSE
FILE2_WRITE_TRIG = FALSE

!
!>>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE2_CLOSE_LOGIC, ~FILE2_WRITE_ENABLE | ~FILE2_WRITE_BLK_ON
!>>>>>>>>>>>>>>>>>>>>>>
!
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
!
!<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE2_WRITE_LOOP
!<<<<<<<<<<<<<<<<<<<<<<
!

!---------------------------------------------------------------------
!      Queue up a block of FIL_WRITEs and pass status, done and
!      record pointers as array values. This allows the call to
!      correctly return the results of all writes
!---------------------------------------------------------------------

       !------------------------------------------------------------
       !                    NOTE
       !------------------------------------------------------------
       !      The file record definition is part of the call argument
       !      list. This allows data to be directly written from the
       !      original database destination (FILE2_REC_ID = 0)
       !------------------------------------------------------------


RECALL; FIL_WRITE, FILE2_WRITE_BLK_ON,                                 \
        FILE2_CHANNEL, FILE2_WRITE_STATUS(FILE2_BLK_PTR),              \
        FILE2_WRITE_DONE(FILE2_BLK_PTR), FILE2_WRITE_PTR(FILE2_BLK_PTR), \
        FILE2_REC_ID, STR(FILE2_REC_PTR), LNG(FILE2_REC_PTR),          \
        BIT(FILE2_REC_PTR), FLT(FILE2_REC_PTR)

!---------------------------------------------------------------------
!      Increment pointers and conditionally return to top of read loop
!---------------------------------------------------------------------
LET; FILE2_WRITE_BLK_ON, FILE2_BLK_PTR = FILE2_BLK_PTR + 1
LET; FILE2_WRITE_BLK_ON, FILE2_REC_PTR = FILE2_REC_PTR + 1
LET; FILE2_WRITE_BLK_ON, FILE2_WRITE_PTR(FILE2_BLK_PTR) = FILE2_REC_PTR
FILE2_WRITE_BLK_ON = FILE2_WRITE_BLK_ON & (FILE2_BLK_PTR <= FILE2_BLK_LMT)
FILE2_WRITE_ENABLE = FILE2_WRITE_ENABLE & (FILE2_REC_PTR <= FILE2_REC_LMT)

!
!>>>>>>>>>>>>>>>>>>>>
JUMP; FILE2_WRITE_LOOP, FILE2_WRITE_BLK_ON & FILE2_WRITE_ENABLE
!>>>>>>>>>>>>>>>>>>>>
!

!
!<<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE2_CLOSE_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<<
!
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 5.1 B (cont)

```
!----------------------------------------------------------------------
!      Close file on demand and clear status/state varables
!----------------------------------------------------------------------
CALL; FIL_CLOSE, FILE2_CLOSE_TRIG, \
      FILE2_CHANNEL, FILE2_CLOSE_STATUS, FILE2_CLOSE_DONE

CALL; MAKZERO, FILE2_CLOSE_DONE, \
      FILE2_CLOSE_TRIG, FILE2_DEFINE_STATUS, FILE2_OPEN_STATUS1, \
      FILE2_CLOSE_STATUS, FILE2_OPEN_STATUS2, FILE2_DEFINE_DONE, \
      FILE2_OPEN_DONE1, FILE2_CLOSE_DONE, FILE2_OPEN_DONE2,      \
      FILE2_LOG_DONE, FILE2_DEFINE_OK, FILE2_OPEN_OK, FILE2_CREATE_OK

END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Solution C

The following example shows a second way to perform basic file operations for a medium to large file (1000 records).

Explanation C

This program uses the FIL_OPEN call to test for an existing file on disk. If located, the open operation is completed and all of the records are read into the database using a program loop and a FIL_READ call. The read is performed using record blocks greater than the logical record size in order to maximize the amount of data read with each file access. This block size may be optimized by redefining the record definition (limit is approximately 250 elements.)

If the file is not located, a second FIL_OPEN is executed to create the file. A second program loop is used along with a FIL_WRITE call to initialize each record for later reading. The same physical record blocking arrangement for reading is repeated. Finally, a FIL_CLOSE is demonstrated for closing the file.

Example 5.1 C

The following example includes all the necessary declarations and logic to function correctly. The example is provided only to help the user understand how to implement file input/output. The user must develop code that follows the principles described in the example. However, the user also must develop code that meets the requirements of the specific application being developed.

The example declarations and logic follows.

```
!***************************************************************************
!*      Sample code for Example 5.1 C                                      *
!*                                                                         *
!*      Task 5.1 - How to implement File Operations                        *
!***************************************************************************

!---------------------------------------------------------------------
!      Code for the Declaration Section (before "TABLES;")
!---------------------------------------------------------------------
LOGICAL;        NEW_DB:TRUE, NEW_CLE, ACTIVE, CACDSA, , , , , , , , \
                ICCDSA, PAUSE, , ,

CONSTANT;       ZERO:0, ONE:1
STRING;         PAD[25]:"-------------------------"

LOGICAL;        CRISP_TT:TRUE, CRISP_LP    , CRISP_CL    , CRISP_SS    , \
                CRISP_LOGFIL , CRISP_ALMFIL, CRISP_OUTPUT, CRISP_ERROR , \
                CRISP_DEV00  , CRISP_DEV01 , CRISP_DEV02 , CRISP_DEV03 , \
                CRISP_DEV04  , CRISP_DEV05 , CRISP_DEV06 , CRISP_DEV07 , \
                CRISP_DEV08  , CRISP_DEV09 , CRISP_DEV10 , CRISP_DEV11 , \
                CRISP_DEV12  , CRISP_DEV13 , CRISP_DEV14 , CRISP_DEV15 , \
                CRISP_DEV16  , CRISP_DEV17 , CRISP_DEV18 , CRISP_DEV19 , \
                CRISP_DEV20  , CRISP_DEV21 , CRISP_DEV22 , CRISP_DEV23
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
!-----------------------------------------------------------------------
!      Return status codes
!-----------------------------------------------------------------------

CONSTANT;       SUCCESS:1, REC_NOT_FOUND:98994, FILE_NOT_FOUND:98962,   \
                CHAN_IN_USE:-4

CONSTANT;       FILE3_BLK_DIM:51,   \
                FILE3_BLK_SIZE:50,  \
                FILE3_DIM:1001,            \
                FILE3_SIZE:1000,    \
                NEG_FILE3_SIZE:-1000

CONSTANT;       FILE3_BLK_LMT:20

!-----------------------------------------------------------------------
!      DCL path and file name definition
!
!      If subdirectory [CRISP.DATA] does not exist on your system,
!      either create it using the following DCL command:
!
!      CREATE /DIR DISK$USER:[CRISP.DATA]
!
!      or revise this declaration to an existing subdirectory
!-----------------------------------------------------------------------
STRING;         FILE3_NAME[]:"DISK$USER:[CRISP.DATA]FILE3.DAT"

       !--------------------------------------------------------------
       !                    NOTE
       !--------------------------------------------------------------
       !      Assign a unique channel number to each file or allow
       !      CRISP to perform the assignment by declaring channel
       !      number = 0
       !--------------------------------------------------------------
LONG;           FILE3_CHANNEL:30

!-----------------------------------------------------------------------
!      FIL_OPEN Options
!
!      Check = 4 (shared read/write) + 16 (file must exist) = 20
!      RW    = 4 (shared read/write) + 1  (create file)     = 5
!
!-----------------------------------------------------------------------
LONG;           FILE3_CHECK_OPTION:20,     \
                FILE3_RW_OPTION:5

!-----------------------------------------------------------------------
!       FIL_LOG_MESSAGE Options (DEBUG)
!
!       Log   =  4 (print good reads) + 16 (print good writes) +
!               32 (print records not found during reads)     = 52
```

(Continued on next page.)

 © 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
!
!       Set FILE1_DEBUG = TRUE to prior to starting CRISP to enable
!       log message operations

!----------------------------------------------------------------------
LONG;           FILE3_LOG_OPTION:52
LOGICAL;        FILE2_DEBUG:FALSE

LONG;           FILE3_REC_ID:1,         \
                FILE3_REC_LEN:0

LONG;           FILE3_DEFINE_STATUS, FILE3_OPEN_STATUS1,\
                FILE3_CLOSE_STATUS, FILE3_READ_STATUS,  \
                FILE3_WRITE_STATUS, FILE3_OPEN_STATUS2, \
                FILE3_LOG_STATUS

LOGICAL;        FILE3_DEFINE_DONE, FILE3_OPEN_DONE1,    \
                FILE3_CLOSE_DONE, FILE3_READ_DONE,      \
                FILE3_WRITE_DONE, FILE3_OPEN_DONE2,     \
                FILE3_LOG_DONE

LOGICAL;        FILE3_DEFINE_OK, FILE3_OPEN_OK,\
                FILE3_CLOSE_OK, FILE3_READ_OK, \
                FILE3_WRITE_OK, FILE3_CREATE_OK

LONG;           FILE3_BLK_INDEX, FILE3_BLK_PTR
LONG;           FILE3_READ_PASS_CTR, FILE3_WRITE_PASS_CTR

!----------------------------------------------------------------------
!      CRISP table into which file records are read/written to
!----------------------------------------------------------------------
STRING;         STR_BLK(FILE3_BLK_DIM)[25]
LONG;           LNG_BLK(FILE3_BLK_DIM)
LOGICAL;        BIT_BLK(FILE3_BLK_DIM)
FLOAT;          FLT_BLK(FILE3_BLK_DIM)

!----------------------------------------------------------------------
!      CRISP table used by logic Application
!----------------------------------------------------------------------
STRING;         STR(FILE3_DIM)[25]
LONG;           LNG(FILE3_DIM)
LOGICAL;        BIT(FILE3_DIM)
FLOAT;          FLT(FILE3_DIM)

!----------------------------------------------------------------------
!      These variables initiate each of the FIL calls
!----------------------------------------------------------------------
LOGICAL;        FILE3_OPEN_TRIG, FILE3_READ_TRIG, FILE3_WRITE_TRIG,\
                FILE3_CREATE_TRIG, FILE3_CLOSE_TRIG

LOGICAL;        FILE3_OPEN_ENABLE, FILE3_READ_ENABLE, FILE3_WRITE_ENABLE, \
                FILE3_READ_ABORT, FILE3_WRITE_ABORT, FILE3_STILL_OPEN
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
!---------------------------------------------------------------------
!      All logic placed between the Key Words TABLES; and
!      RESTART;  will be executed only on the first pass of
!      logic.  All code in this area is Initialization Code.
!---------------------------------------------------------------------
TABLES;

RECALL; SETMSG_C16, , CRISP_TT

!---------------------------------------------------------------------
!      Log file messages to console printer (DEBUG)
!---------------------------------------------------------------------
RECALL; FIL_LOG_MESSAGE, FILE3_DEBUG, \
        FILE3_LOG_STATUS, FILE3_LOG_DONE, FILE3_LOG_OPTION


!---------------------------------------------------------------------
!      Define record for future file operations
!      The call returns a value for record length
!---------------------------------------------------------------------
RECALL; FIL_DEFINE_REC, , \
        FILE3_DEFINE_STATUS, FILE3_REC_ID, FILE3_REC_LEN,\
        STR_BLK(1),LNG_BLK(1),BIT_BLK(1),FLT_BLK(1),  \
        STR_BLK(2),LNG_BLK(2),BIT_BLK(2),FLT_BLK(2),  \
        STR_BLK(3),LNG_BLK(3),BIT_BLK(3),FLT_BLK(3),  \
        STR_BLK(4),LNG_BLK(4),BIT_BLK(4),FLT_BLK(4),  \
        STR_BLK(5),LNG_BLK(5),BIT_BLK(5),FLT_BLK(5),  \
        STR_BLK(6),LNG_BLK(6),BIT_BLK(6),FLT_BLK(6),  \
        STR_BLK(7),LNG_BLK(7),BIT_BLK(7),FLT_BLK(7),  \
        STR_BLK(8),LNG_BLK(8),BIT_BLK(8),FLT_BLK(8),  \
        STR_BLK(9),LNG_BLK(9),BIT_BLK(9),FLT_BLK(9),  \
        STR_BLK(10),LNG_BLK(10),BIT_BLK(10),FLT_BLK(10),  \
        STR_BLK(11),LNG_BLK(11),BIT_BLK(11),FLT_BLK(11),  \
        STR_BLK(12),LNG_BLK(12),BIT_BLK(12),FLT_BLK(12),  \
        STR_BLK(13),LNG_BLK(13),BIT_BLK(13),FLT_BLK(13),  \
        STR_BLK(14),LNG_BLK(14),BIT_BLK(14),FLT_BLK(14),  \
        STR_BLK(15),LNG_BLK(15),BIT_BLK(15),FLT_BLK(15),  \
        STR_BLK(16),LNG_BLK(16),BIT_BLK(16),FLT_BLK(16),  \
        STR_BLK(17),LNG_BLK(17),BIT_BLK(17),FLT_BLK(17),  \
        STR_BLK(18),LNG_BLK(18),BIT_BLK(18),FLT_BLK(18),  \
        STR_BLK(19),LNG_BLK(19),BIT_BLK(19),FLT_BLK(19),  \
        STR_BLK(20),LNG_BLK(20),BIT_BLK(20),FLT_BLK(20),  \
        STR_BLK(21),LNG_BLK(21),BIT_BLK(21),FLT_BLK(21),  \
        STR_BLK(22),LNG_BLK(22),BIT_BLK(22),FLT_BLK(22),  \
        STR_BLK(23),LNG_BLK(23),BIT_BLK(23),FLT_BLK(23),  \
        STR_BLK(24),LNG_BLK(24),BIT_BLK(24),FLT_BLK(24),  \
        STR_BLK(25),LNG_BLK(25),BIT_BLK(25),FLT_BLK(25),  \
        STR_BLK(26),LNG_BLK(26),BIT_BLK(26),FLT_BLK(26),  \
        STR_BLK(27),LNG_BLK(27),BIT_BLK(27),FLT_BLK(27),  \
        STR_BLK(28),LNG_BLK(28),BIT_BLK(28),FLT_BLK(28),  \
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
        STR_BLK(29),LNG_BLK(29),BIT_BLK(29),FLT_BLK(29),  \
        STR_BLK(30),LNG_BLK(30),BIT_BLK(30),FLT_BLK(30),  \
        STR_BLK(31),LNG_BLK(31),BIT_BLK(31),FLT_BLK(31),  \
        STR_BLK(32),LNG_BLK(32),BIT_BLK(32),FLT_BLK(32),  \
        STR_BLK(33),LNG_BLK(33),BIT_BLK(33),FLT_BLK(33),  \
        STR_BLK(34),LNG_BLK(34),BIT_BLK(34),FLT_BLK(34),  \
        STR_BLK(35),LNG_BLK(35),BIT_BLK(35),FLT_BLK(35),  \
        STR_BLK(36),LNG_BLK(36),BIT_BLK(36),FLT_BLK(36),  \
        STR_BLK(37),LNG_BLK(37),BIT_BLK(37),FLT_BLK(37),  \
        STR_BLK(38),LNG_BLK(38),BIT_BLK(38),FLT_BLK(38),  \
        STR_BLK(39),LNG_BLK(39),BIT_BLK(39),FLT_BLK(39),  \
        STR_BLK(40),LNG_BLK(40),BIT_BLK(40),FLT_BLK(40),  \
        STR_BLK(41),LNG_BLK(41),BIT_BLK(41),FLT_BLK(41),  \
        STR_BLK(42),LNG_BLK(42),BIT_BLK(42),FLT_BLK(42),  \
        STR_BLK(43),LNG_BLK(43),BIT_BLK(43),FLT_BLK(43),  \
        STR_BLK(44),LNG_BLK(44),BIT_BLK(44),FLT_BLK(44),  \
        STR_BLK(45),LNG_BLK(45),BIT_BLK(45),FLT_BLK(45),  \
        STR_BLK(46),LNG_BLK(46),BIT_BLK(46),FLT_BLK(46),  \
        STR_BLK(47),LNG_BLK(47),BIT_BLK(47),FLT_BLK(47),  \
        STR_BLK(48),LNG_BLK(48),BIT_BLK(48),FLT_BLK(48),  \
        STR_BLK(49),LNG_BLK(49),BIT_BLK(49),FLT_BLK(49),  \
        STR_BLK(50),LNG_BLK(50),BIT_BLK(50),FLT_BLK(50)

!----------------------------------------------------------------------
!       The Logic between the Key Word RESTART; and the Key Word
!       END; will be executed during each pass of Logic.
!----------------------------------------------------------------------
RESTART;


!----------------------------------------------------------------------
!       Open File on successful FIL_DEFINE
!
!       This logic will execute if CRISP is started WARM
!       or with the RESTORE option
!----------------------------------------------------------------------
SET; (FILE3_DEFINE_STATUS == SUCCESS), FILE3_OPEN_TRIG = TRUE

SET; FILE3_OPEN_TRIG, FILE3_OPEN_ENABLE = TRUE

!
!>>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE3_READ_LOGIC, ~FILE3_OPEN_ENABLE
!>>>>>>>>>>>>>>>>>>>>>>
!

!----------------------------------------------------------------------
!       Check for existing file
!----------------------------------------------------------------------
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
CALL; FIL_OPEN, FILE3_OPEN_TRIG, \
        FILE3_CHANNEL, FILE3_OPEN_STATUS1, FILE3_OPEN_DONE1, \
        FILE3_NAME, FILE3_REC_LEN, FILE3_DIM, FILE3_CHECK_OPTION

FILE3_OPEN_OK = (FILE3_OPEN_STATUS1 == SUCCESS) & FILE3_OPEN_DONE1
FILE3_STILL_OPEN = (FILE3_OPEN_STATUS1 == CHAN_IN_USE) & FILE3_OPEN_DONE1

FILE3_OPEN_TRIG = FALSE

!----------------------------------------------------------------------
!      Initiate immediate FIL_READ if file opened successfully
!----------------------------------------------------------------------
SET; FILE3_OPEN_OK, FILE3_READ_TRIG = TRUE
SET; (FILE3_OPEN_OK | FILE3_STILL_OPEN) & FILE3_OPEN_DONE1, \
        FILE3_OPEN_ENABLE = FALSE



!----------------------------------------------------------------------
!      Create and Open File if it does not exist
!----------------------------------------------------------------------
SET; (FILE3_OPEN_STATUS1 == FILE_NOT_FOUND) & FILE3_OPEN_DONE1, \
        FILE3_CREATE_TRIG = TRUE

CALL; FIL_OPEN, FILE3_CREATE_TRIG, \
        FILE3_CHANNEL, FILE3_OPEN_STATUS2, FILE3_OPEN_DONE2, \
        FILE3_NAME, FILE3_REC_LEN, FILE3_DIM, FILE3_RW_OPTION

FILE3_CREATE_OK = (FILE3_OPEN_STATUS2 == SUCCESS) & FILE3_OPEN_DONE2

!----------------------------------------------------------------------
!      Fill in String arrays for first file write
!----------------------------------------------------------------------
CALL; SCOPY, FILE3_CREATE_OK,\
        PAD, ZERO, STR(1), ZERO, NEG_FILE3_SIZE

MESSAGE; FILE3_OPEN_DONE2 & FILE3_CREATE_OK,\
        "@M@J***  FILE 1 WAS CREATED  ***@M@J"
MESSAGE; FILE3_OPEN_DONE2 & ~FILE3_CREATE_OK,\
        "@M@J***  FILE 1 NOT OPENED SUCCESSFULLY  ***@M@J"

FILE3_CREATE_TRIG = FALSE

!----------------------------------------------------------------------
!      Initiate immediate FIL_WRITE if file created successfully
!----------------------------------------------------------------------
SET; FILE3_CREATE_OK, FILE3_WRITE_TRIG = TRUE
SET; FILE3_OPEN_DONE2, FILE3_OPEN_ENABLE = FALSE

!
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
!<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE3_READ_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<
!

!----------------------------------------------------------------------
!       Read File in blocks of size FILE3_BLK_DIM
!       on demand or after opening of file
!----------------------------------------------------------------------
SET; FILE3_READ_TRIG, FILE3_READ_ENABLE = TRUE
SET; FILE3_READ_ENABLE, FILE3_BLK_PTR = 0

!
!>>>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE3_WRITE_LOGIC, ~FILE3_READ_ENABLE
!>>>>>>>>>>>>>>>>>>>>>>>
!

!----------------------------------------------------------------------
!       Restart Read loop logic when READ_DONE bit is TRUE
!       or on first Read pass
!----------------------------------------------------------------------
FILE3_READ_OK = (FILE3_READ_STATUS == SUCCESS) & FILE3_READ_DONE | \
                FILE3_READ_TRIG

!----------------------------------------------------------------------
!       Exit loop early if bad read status is returned
!----------------------------------------------------------------------
SET; FILE3_READ_DONE & ~FILE3_READ_OK, FILE3_READ_ABORT = TRUE

LET; FILE3_READ_OK, FILE3_BLK_PTR = FILE3_BLK_PTR + 1
FILE3_READ_ENABLE = (FILE3_BLK_PTR <= FILE3_BLK_LMT) & ~FILE3_READ_ABORT

RECALL; FIL_READ, FILE3_READ_OK & FILE3_READ_ENABLE, \
        FILE3_CHANNEL, FILE3_READ_STATUS, FILE3_READ_DONE, \
        FILE3_BLK_PTR, FILE3_REC_ID

MESSAGE; FILE3_READ_ABORT,\
        "@M@J***  FILE 1 READ NOT SUCCESSFUL  ***@M@J"

!----------------------------------------------------------------------
!       Unload block record into CRISP table
!----------------------------------------------------------------------
FILE3_BLK_INDEX = 1 + ((FILE3_BLK_PTR - 1) * FILE3_BLK_SIZE)

RECALL; SCOPY, FILE3_READ_OK,\
        STR_BLK(1), ZERO, STR(FILE3_BLK_INDEX), ZERO, FILE3_BLK_SIZE
RECALL; TRANSFER, FILE3_READ_OK,\
        LNG_BLK(1), ZERO, LNG(FILE3_BLK_INDEX), ZERO, FILE3_BLK_SIZE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
RECALL; TRANSFER, FILE3_READ_OK,\
        BIT_BLK(1), ZERO, BIT(FILE3_BLK_INDEX), ZERO, FILE3_BLK_SIZE
RECALL; TRANSFER, FILE3_READ_OK,\
        FLT_BLK(1), ZERO, FLT(FILE3_BLK_INDEX), ZERO, FILE3_BLK_SIZE

FILE3_READ_TRIG = FALSE

!
!<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE3_WRITE_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<
!

!----------------------------------------------------------------------
!      Write File in blocks of size FILE3_BLK_DIM
!      on demand or on creation of new file
!----------------------------------------------------------------------
SET; FILE3_WRITE_TRIG, FILE3_WRITE_ENABLE = TRUE
SET; FILE3_WRITE_ENABLE, FILE3_BLK_PTR = 0

!
!>>>>>>>>>>>>>>>>>>>>>>
JUMP; FILE3_CLOSE_LOGIC, ~FILE3_WRITE_ENABLE
!>>>>>>>>>>>>>>>>>>>>>>
!

!----------------------------------------------------------------------
!      Restart Read loop logic when READ_DONE bit is TRUE
!      or on first Read pass
!----------------------------------------------------------------------
FILE3_WRITE_OK = (FILE3_WRITE_STATUS == SUCCESS) & FILE3_WRITE_DONE | \
        FILE3_WRITE_TRIG
SET; FILE3_WRITE_DONE & ~FILE3_WRITE_OK, FILE3_WRITE_ABORT = TRUE

LET; FILE3_WRITE_OK, FILE3_BLK_PTR = FILE3_BLK_PTR + 1

FILE3_WRITE_ENABLE = (FILE3_BLK_PTR <= FILE3_BLK_LMT) & ~FILE3_WRITE_ABORT

!----------------------------------------------------------------------
!      Load block record from CRISP table
!----------------------------------------------------------------------
FILE3_BLK_INDEX = 1 + ((FILE3_BLK_PTR - 1) * FILE3_BLK_SIZE)

RECALL; SCOPY, FILE3_WRITE_OK,\
        STR(FILE3_BLK_INDEX), ZERO, STR_BLK(1), ZERO, FILE3_BLK_SIZE
RECALL; TRANSFER, FILE3_WRITE_OK,\
        LNG(FILE3_BLK_INDEX), ZERO, LNG_BLK(1), ZERO, FILE3_BLK_SIZE
RECALL; TRANSFER, FILE3_WRITE_OK,\
        BIT(FILE3_BLK_INDEX), ZERO, BIT_BLK(1), ZERO, FILE3_BLK_SIZE
```

(Continued on next page.)

© 1991 Square D All Rights Reserved

Example 5.1 C (cont)

```
RECALL; TRANSFER, FILE3_WRITE_OK,\
        FLT(FILE3_BLK_INDEX), ZERO, FLT_BLK(1), ZERO, FILE3_BLK_SIZE

RECALL; FIL_WRITE, FILE3_WRITE_OK, \
        FILE3_CHANNEL, FILE3_WRITE_STATUS, FILE3_WRITE_DONE, \
        FILE3_BLK_PTR, FILE3_REC_ID

MESSAGE; FILE3_WRITE_DONE & ~FILE3_WRITE_OK,\
        "@M@J***  FILE 1 WRITE NOT SUCCESSFUL  ***@M@J"

FILE3_WRITE_TRIG = FALSE

!
!<<<<<<<<<<<<<<<<<<<<<<<<
LABEL; FILE3_CLOSE_LOGIC
!<<<<<<<<<<<<<<<<<<<<<<<<
!

!--------------------------------------------------------------------
!      Close file on demand and clear status/state varables
!--------------------------------------------------------------------
CALL; FIL_CLOSE, FILE3_CLOSE_TRIG, \
        FILE3_CHANNEL, FILE3_CLOSE_STATUS, FILE3_CLOSE_DONE

CALL; MAKZERO, FILE3_CLOSE_DONE, \
        FILE3_CLOSE_TRIG, FILE3_READ_ABORT, FILE3_WRITE_ABORT,          \
        FILE3_DEFINE_STATUS, FILE3_OPEN_STATUS1, FILE3_CLOSE_STATUS,    \
        FILE3_READ_STATUS, FILE3_WRITE_STATUS, FILE3_OPEN_STATUS2,      \
        FILE3_LOG_STATUS, FILE3_DEFINE_DONE, FILE3_OPEN_DONE1,          \
        FILE3_CLOSE_DONE, FILE3_READ_DONE, FILE3_WRITE_DONE,            \
        FILE3_OPEN_DONE2, FILE3_LOG_DONE, FILE3_DEFINE_OK,              \
        FILE3_OPEN_OK, FILE3_READ_OK, FILE3_WRITE_OK, FILE3_CREATE_OK

END;
```

© 1991 Square D All Rights Reserved

Notes:

© 1991 Square D All Rights Reserved

Index

A
ABSALM_C16 3-15
ack button 3-35
Alarm Display 3-29
Alarm file 3-3, 3-11
Alarm Handling 3-43
Alarm message 3-3, 3-15, 3-21
alarm trigger 3-21

B
blink option 3-15
BZER0 3-29

C
CMPRS 3-35
command procedure 3-11
[CRISP] directory 2-5
CRISPWIN 2-5
CRISPwindows process 2-5, 2-7

D
date/time stamp 4-1
DEFAULT security 2-3, 2-15
DEV_ALARM_1 3-49
DISPLAY_MESSAGES 3-15, 3-21

F
FIL_CLOSE 6-1, 6-9, 6-19
FIL_OPEN 6-1, 6-9, 6-19
FIL_READ 6-1, 6-9, 6-19
FIL_WRITE 6-1, 6-9, 6-19
FIXDAT 3-3

G
graphic screen 2-5

H
HISTORIAN data 4-1, 4-2
HISTORIAN point file 4-1
home directory 2-25

I
IDI 5-3, 5-11

J
JUMP statement 3-43

L
Link Level 2-3
log out 2-25
logging alarms 3-11

M
Main Menu 2-15
Message Mask 3-3, 3-4, 3-15, 3-21
multi-drop 5-11, 5-16

N
NEW_DESTINATION 3-3, 3-4
Node Database Table 2-3

P
PLC 5-3
point file 4-1
polling 5-16
primary PLC 5-3

R
redundant PLC 5-3
Remote Security 2-3
Remote Security Table 2-3
RMS disk file 6-1

S
SCAN_HIST 4-1, 4-2
Screen Security 2-3
scrolling region 3-15, 3-21, 3-29, 3-35, 3-43, 3-49
secondary PLC 5-3
security 2-6, 2-9, 2-15, 2-19, 2-25
security file 2-5, 2-6, 2-7
security level 2-3, 2-7, 2-9, 2-19
security mask 2-6, 2-7
security table 2-9, 2-19
SETMSG_C16 3-3
setpoint 5-11
SLC 5-16
SMERGE_C16 3-35
string substitution 3-15
STR_MERGE 3-3
SUBMIT_BATCH 3-11

T
time and date stamp 3-15
TIME_AND_DATE 3-3

U
UCF 5-3, 5-11, 5-16
unacknowledged alarm 3-29, 3-35
User Lock Level 2-9, 2-19
User Lock Range 2-9, 2-19

V
VALUE_C16 3-3, 3-4, 4-1, 4-2

© 1991 Square D All Rights Reserved

© 1991 Square D All Rights Reserved