CRISP ®/32
UTILITIES
Reference Manual

---

§ CRISP *Software* *Products*

SQUARE D
GROUPE SCHNEIDER

CRISP®/32
Utilities Reference Manual
Document number: 500 002 - 005, Rev. 6

Document History

Revision	Date	Pages affected/Description of change
1	8/6/91	Initial Release. ECN # 3947
2	8/31/91	Update. Replace Appendix B. ECN #3958
3	9/6/91	Reformat Entire Document. ECN # 3964
4	6/26/92	Changes to CRMON, and Appendix D. ECN # 4115
5	12/16/93	Update Document per ECN# 4335
6	4/28/94	Change CRMON per ECN #4398

Software Version CRISP/32 Rev. 3.0 and Later

Table of Contents

Introduction

The CRISP system Utilities Manual describes utility programs that support the use of the CRISP/32 and I/ONYX products.

Before entering the CRISP utility commands, the user must be logged on to the operating system via the appropriate log-in procedure.

This manual is broken down into the following sections.

Section	Description
ARBSTATS *(page 5)*	Displays the arbiter statistics.
CLN *(page 9)*	Cleans up an abnormally exited process from the Software Bus.
CRISP	Compiles a CRISP source code file. This command is typically executed via the LGBUILD command. Refer to CRISP/32 Language Reference Manual.
CRISP_CONFIG *(page 11)*	Configures the CRISP system.
CRISP_SETUP *(page 17)*	Sets up the system to run CRISP/32.
CRMON *(page 19)*	Monitors the health of selected CRISP processes.
CRSTART *(page 25)*	Starts the CRISP system.
CRSTOP *(page 27)*	Stops the CRISP system.
DBINSTALL *(page 29)*	Installs a CRISP database.
DBTCMD *(page 33)*	Generates a database transfer list for the CLE transfer utility.
DDMON *(page 35)*	Displays a list of installed databases.

(Continued on next page.)

Section	Description
directoryDEF *(page 37)*	Selects a specific subdirectory of [CRISP].
ICF	Defines the I/ONYX hardware (refer to the I/ONYX Configurator Reference Manual).
IDCCMP *(page 39)*	Generates a database transfer list for the Inter-Database Communications (IDC) facility.
LDCLN *(page 41)*	Cleans up the logic directory.
LDMON *(page 43)*	Monitors the logic directory.
LGBUILD *(page 45)*	Compiles and links a CRISP logic program and database.
LGCONFIG *(page 49)*	Sets up the database and logic definition files.
LGINSTALL *(page 51)*	Starts, stops, and/or assigns priorities to the CRISP logic.
PDMON *(page 57)*	Displays the processes connected to the Software Bus.

(Continued on next page.)

Section	Description
Appendix A - Database Transfer Timing *(page A-1)*	Defines important timing considerations when using DBT type database transfers.
Appendix B - IDC Configuration *(page B-1)*	Describes the IDC commands and parameters.
Appendix C - CRT Display File Translation *(page C-1)*	Describes the procedures required to perform CRT file translation.
Appendix D - Trend Configuration *(page D-1)*	Describes the Trend process configuration.

NOTE

The CRISP Utility Commands can also come from a command file; such that, when the CRISP system is started, the commands are automatically executed.

Notes:

Description

The user enters the ARBSTATS command to display the statistics of one or all of the arbiters in the system. This utility enables the user to monitor the data transmission between the CRISP System and the destination nodes.

Format

The command format is as follows.

ARBSTATS [num]

Where **[num]** is the number of the arbiter to display (1-6). The default is to display all arbiters in the system.

Operation

The ARBSTATS command displays the communication and arbitration statistics of all arbiters installed in the system. The statistics are cleared, except for the revision level, every time this utility is run. Statistics two through seventeen are 16-bit, unsigned integer accumulations. The 17 statistics are as follows.

(1) Arbiter Board Revision Level - This is the revision level of the firmware installed in the arbiter board.

(2) Invalid Commands for Arbitration - Count of any invalid commands given to the arbitration CSR of the arbiter.

(3) Good Packets Received - Count of communication packets received error free at the arbiter.

(4) Transmit with Receive Packets - Count of transmit-with-receive commands given to the transmit CSR of the arbiter. This type of transmit command requires that the destination node respond with data.

(5) Data Link Acks Received - Count of communication packets received as a data link acknowledgment. The data transmitted reached the destination node successfully and more data will be forthcoming.

(6) Data Link Nacks Received - Count of communication packets received as a negative data link acknowledgment. Most likely, the destination did not properly receive the data; therefore, the arbiter will retry the same data packet.

(7) Network Timeout - Count of number of timeouts on data reception from the destination node. In other words, the destination node did not respond with data in the allotted time. The timeout mechanism in the arbiter automatically goes through two retries before posting the timeout status.

(Continued on next page.)

Operation (cont)

(8) Bad Checksums Received - Count of communication packets received with a bad checksum.

(9) Re-Sync Packets Received - Count of communication packets received with a re-sync. This means the arbiter was not in sync with the destination node due to a power fail, reset, or switchover. The arbiter and destination node then sync up and the data is transmitted again.

(10) Bad Destination Address (Receive) - Count of communication packets received with an invalid address (i.e., the address is outside the range of 0 to 17).

(11) Invalid Commands for Receive - Count of any invalid commands given to the receive CSR of the arbiter.

(12) Xmt/Rcv Err -- Idle Side (Dst 0) - Count of transmit and receive commands given to the arbiter on a Standby CPU. These commands are rejected by the arbiter if the destination address is not the Active CPU.

(13) Transmit-only Packets - Count of transmit-only commands given to the transmit CSR of the arbiter. This type of transmit command requires no response from the destination node.

(14) Bad Destination Address (Transmit) - Count of transmit-only commands given to the arbiter with an invalid address (i.e., the address is outside the range of 0 to 17).

(15) Invalid Commands for Transmit - Count of any invalid commands given to the transmit CSR of the arbiter.

(16) Xmt-only Error -- Idle Side (Dst 0) - Count of transmit-only commands given to the arbiter on a Standby CPU. These commands are rejected by the arbiter if the destination address is not the Active CPU.

(17) Active/Idle Switchovers - Count of Active-to-Standby and Standby-to-Active switchovers.

Example

```
$ ARBSTATS 1

   Arbiter 1 Statistics

           6  Arbiter Board Revision Level (1)
           0  Invalid Commands for Arbitration (2)
       54606  Good Packets Received (3)
       56507  Transmit with Receive Packets (4)
           0  Data Link Acks Received (5)
           0  Data Link Nacks Received (6)
           0  Network Timeout (7)
           0  Bad Checksums Received (8)
          37  Re-Sync Packets Received (9)
           0  Bad Destination Address (Receive) (10)
           0  Invalid Commands for Receive (11)
           0  Xmt/Rcv Err -- Idle Side (Dst  0) (12)
           0  Transmit-only Packets (13)
           0  Bad Destination Address (Transmit) (14)
           0  Invalid Commands for Transmit (15)
           0  Xmt-only Error -- Idle Side (Dst  0) (16)
           1  Active/Idle Switchovers (17)

$
```

Notes

Description

The user enters the CLN command to clean up an exited process that is still connected to the Software Bus. This utility may need to be invoked in the event a process is terminated abnormally.

Format

To execute the CLN command, enter the following.

CLN [entry [entry . . .]]

entry This specifies the name of the process to clean. A process entry number can be specified as /nnn, where nnn is the number of the entry. If no argument is specified, the user is prompted for a name.

Operation

Occasionally, a process may exit improperly without disconnecting from the software bus. If that process is restarted, it may get the "Already Connected" error when it tries to reconnect. When this occurs, the CLN utility may be used to clean out the specified process directory entry, making it available again.

The PDMON command may be used to display the processes still connected to the Software Bus. This display shows all of the processes connected to the Software Bus. From this display, the user can obtain the process name or process directory entry number for use by the CLN command.

Using CLN on a process that still exists may cause unusual operation of CRISP/32. To stop a process, CRISP/32 `CRSTOP` or VMS `STOP` must be used. Verify that the process is truly gone using VMS `SHOW SYSTEM` before using CLN. Logics may also require the use of LDCLN.

Example

```
$ PDMON

CRISP/32 Process Directory Monitor                      Max. entries:  100

Entry  ProcName    PID     Health  Messages         Status
-----  --------  --------  ------  ----------  --------------------------
   0   CLE       00000056       8           0  Rsv All Val
   1   DBCTRL    00000051       0           1  Rsv All Val DBA
   2   CRISPMON  00000052       0           1  Rsv All Val DBA
   3   ICC                      0           0  Rsv
   4   FILES     00000055       0           0  Rsv All Val
   5   CRMON                    0           0  Rsv
   6   DBINSTAL  0000005A       0           0  Rsv All Val
   7   LGINSTAL                 0           0  Rsv
   8   CRKILL                   0           0  Rsv
   9   CASRV                    0           0  Rsv
  10   ICM                      0           0  Rsv

$ CLN DBINSTAL


$ PDMON

CRISP/32 Process Directory Monitor                      Max. entries:  100

Entry  ProcName    PID     Health  Messages         Status
-----  --------  --------  ------  ----------  --------------------------
   0   CLE       00000056       8           0  Rsv All Val
   1   DBCTRL    00000051       0           1  Rsv All Val DBA
   2   CRISPMON  00000052       0           1  Rsv All Val DBA
   3   ICC                      0           0  Rsv
   4   FILES     00000055       0           0  Rsv All Val
   5   CRMON                    0           0  Rsv
   6   DBINSTAL                 0           0  Rsv
   7   LGINSTAL                 0           0  Rsv
   8   CRKILL                   0           0  Rsv
   9   CASRV                    0           0  Rsv
  10   ICM                      0           0  Rsv
```

Description

The user executes the CRISP_CONFIG command procedure to define the configuration of the VAX and CRISP/32 system hardware and software. The user typically enters this command after a new CRISP/32 system has been installed or after installing a new release of CRISP/32 on an existing system.

Format

To execute the CRISP_CONFIG command procedure, enter the following.

@CRISP$:CRISP_CONFIG

Operation

The user responds to several prompts regarding the VAX and CRISP/32 system hardware and software. Most of the questions are straightforward; however, the following two require further explanation. The 'system output device' receives the primary output messages from the various CRISP/32 processes. It is usually the device specification of a terminal or printer (e.g., OPA0: or TXA5:). The 'secondary output device' receives a duplicate of the error messages from the CRISP/32 processes. It may be the device specification of another terminal/printer, a disk file specification, or it may be suppressed by specifying the null device (e.g., NLA0:). If the secondary output is directed to a disk file, the user must be careful to keep the file from filling the disk and causing CRISP processes to abort. The only method of starting a new output file is to stop CRISP, rename or delete the old file, and restart CRISP.

If the user is updating a CRISP system, a CONFIG_RESULTS.COM file may exist from the previous installation. If it does, the program will rename the file to CONFIG_RESULTS.OLD and use the answers in it as defaults for creating a new CONFIG_RESULTS.COM (DO NOT ATTEMPT TO USE THE OLD CONFIG_RESULTS WITH THE NEW SYSTEM). CRISP_CONFIG should be executed each time the CRISP software is upgraded with a new release or when the configuration of the VAX or CRISP/32 system hardware/software is changed.

Configuration Directory

The CONFIG_RESULTS.COM file and other system-specific configuration files are written to the directory pointed to by the logical name CRISP$CFG. The definition of this logical name depends on the SYSGEN parameter SCSNODE. If SCSNODE is undefined (null), CRISP$CFG will be `disk:[CRISP.CFG]`; otherwise, it will be `disk:[CRISP.CFG.scsnode]`. This is done to allow for VAXcluster systems with common disks.

The directory is created automatically during the CRISP installation procedure based on the value of SCSNODE at that time. CRISP$CFG is defined by CRISP_SETUP. If the value of SCSNODE is changed, CRISP_CONFIG will automatically create the appropriate directory and redefine the CRISP$CFG logical. However, all files will remain in the old directory. If desired, these files may be moved to the new directory with the DCL RENAME command. The empty directory may then be deleted.

Example

```
$ @crisp_config

                          +----------------------------------------+
                          | CRISP Software Configuration Procedure |
                          +----------------------------------------+
                          | Copyright 1987-94, Square D Company    |
                          | All rights reserved                    |
                          +----------------------------------------+

%C32CONFIG-I-CPUTYP, Configuring CRISP on a VAXstation 4000-60

Is the target system a dual CPU . . . . . [NO]?
Enter the system output device  . . . . [OPA0:]: TTA2:
Enter the secondary output device . . . [NLA0:]:
Does this system have I/Onyx  . . . . . [NO]? YES

            Classic I/O was used with CRISP/16 systems running on PDP-11
            computers.  This consists of a set of open-rack panels connected
            by three 40-wire ribbon cables terminated in the computer.  These
            CRISP/16 I/O systems may be used on CRISP systems with restrictions.
            Hardware requirements are (2) DRV11 parallel interfaces, a KWV11
            clock, and a CRISP ICC panel.

Does this system have Classic I/O . . . . [NO]?

            Classic I/O may be run from remote Classic I/O servers similar
            to the I/Onyx module servers (CC servers).  The I/O hardware
            requirements are the same as for the directly connected Classic
            I/O.  The CC servers are placed on the Ethernet for communications.
            CC servers may NOT be connected by SSI (Arbitor board) communication.

Does this system have CC servers  . . . . . [NO]?

            The IEEE 802 SAP for the Database Access Server (DBASRV) must be
            the same for all nodes in a network.  How the SAP is specified has
            changed from earlier versions of CRISP.  The following answers
            produce identical SAPs across different versions of CRISP:

               SAP  4   V2.3
               SAP  8   V2.4, V2.4A, V2.4B
               SAP 12   V2.5 and later

What should the SAP be on this node [12]?

            The number of Database Access Server (DBASRV) read-ahead buffers
            should be one greater than the number of clients (i.e., workstations
            or trend processes) that will communicate with this system via
            Ethernet simultaneously.  Note that a workstation with an annunciator
            panel counts as two clients.
```

Example (cont)

```
How many read-ahead buffers should be allocated [4]?

            The CRISP Access Server (CASRV) SAP must be the same for all CRISP
            nodes on the network.

            Note:  The SAP used must not be the same as that specified for
                   the Database Access Server (DBASRV).

What should the SAP be on this node [4]?
How many requests can be pending simultaneously [32]?
How many requests can be processed simultaneously [5]?
At what priority should requests be executed [4]?

            The CRISP Access Server (CASRV) may be used by CRISP PC Workstations
            (PCWS) to access a CWS display file stored on the VAX (remote screen
            mode).  The next question configures the file specification for that
            display file.  It has no effect on the Color Workstation (CWS).

What is the PCWS display filespec . . [CRISP$CWS:USER.CWS]?

            The "CRISPconnect Server for NetDDE" may be used to access CRISP
            Real-Time data by Microsoft Windows applications running on PCs
            that are DDE aware.

            To successfully run, this product requires its license to be
            registered and loaded and also requires the installation of
            "NetDDE for VMS" from Wonderware Software Development Corporation.

            If not yet installed, the product license and the Wonderware
            product can be installed after the CRISP configuration procedure
            is complete.

Do you want the CRISPconnect Server for NetDDE . . . [NO]? YES
Enter directory of "NetDDE for VMS" product [SYS$SYSDEVICE:[NETDDE]]:
At what priority should NetDDE Server requests be executed [20]?

            The message throughput of NetDDE can be boosted by configuring the
            NetDDE Server to send a burst of messages that do not require client
            acknowledgement.  After a burst of "n" messages, the "nth" message
            sent requires client acknowledgement, thereby ensuring that the
            message processsing in the PC client catches up with the NetDDE Server.

Enter max messages to send before requiring client acknowledgement [20]?

            The "CRISPconnect Server for @aGlance/IT" may be used to access
            CRISP Real-Time data and CRISP Historical data by client applications
            running on other nodes and platforms that are @aGlance/IT compliant.

            To successfully run, this product requires its license to be
            registered and loaded and also requires the installation of two
            other products, which are 1) @aGlance/IT and 2) DEC ACA Services.
```

Example (cont)

```
            If not yet installed, the product license and the other products can
            be installed after the CRISP configuration procedure is complete.

Do you want the CRISPconnect Server for @aGlance/IT . . . [NO]? YES

            The @aGlance Server (CSRAAG) to access "real-time" data must be
            configured for the maximum number of @aGlance client applications
            that will concurrently establish sessions to access "real-time" data.

Enter the maximum number of CSRAAG concurrent client sessions [32]?
At what priority should CSRAAG requests be executed [20]?

            The @aGlance Server (CSHAAG) to access "historical" data must be
            configured for the maximum number of @aGlance client applications
            that will concurrently establish sessions to access "historical" data.

Enter the maximum number of CSHAAG concurrent client sessions [32]?
At what priority should CSHAAG requests be executed [15]?
How many requests will CSHAAG clients issue simultaneously [1]?

How many arbiters do you have . . . . . [1]? 0

            Two methods of performing database transfers are available -- The CRISP
            Logic Executive (CLE) and the Inter-Database Communications facilities.
            The older CLE method is limited to transfers within the same CPU.  IDC
            is new with V2.7 and can perform transfers to both local and remote
            databases.  See the V2.7 release notes for more information.  In the
            next question you can select to run the IDC process or not.  You should
            answer no if you are not performing database transfers or if you are
            only performing transfers between local databases.

Do you want the IDC process . . . . . [NO]? YES
At what priority should IDC execute . [19]?

            The CRISP Historian System consists of 3 processes for collecting
            and storing historical process data.

Do you want to enable the Historian System [NO]? YES
    Enter historian disk name [CRISP$DEVICE:]:
    Enter historian initialization filespec [CRISP$CFG:USER.HST]:
    Enter historian logger qualifiers  . . . []: /INTERVAL=60
    Enter historian separator qualifiers . . []: /WAKE=HOURS=1

            To reduce file fragmentation, each point file will be created at a
            specific size.  Whenever a new version of a point file is created,
            the old file will be truncated to actual size.  Users with limited
            disk space may want to specifiy a smaller value, while users who
            are collecting large amounts of data may want a larger value.

    Enter historian allocation size . . . . [100]:
```

Example (cont)

```
Creating CRISP$CFG:CONFIG_RESULTS.COM

            The CRISP Basic Workstation displays a CRISP prompt on startup and
            has no pixel graphics capability.  The CRISP Color Workstation
            (CWS) displays a menu on startup and has pixel graphics support.

CRISP Basic Workstation files were not installed

Writing CRT portion of CONFIG_RESULTS.COM

            No CRISP/32 Workstations have been configured.  If you are using
            networked CRISP Workstations, you may request that the trend process
            be executed to collect trend data.

Do you want the trend process . . [NO]? YES
Enter trend control file spec [CRISP$CFG:USER.TRC]:

Writing CWS portion of CONFIG_RESULTS.COM
Writing CRISP$CFG:EXECUTE_CWSTND.COM
Completing CONFIG_RESULTS.COM

%C32CONFIG-I-PATCH, Patching configuration into CRISP system database

            Patching the CRISP configuration
  PATCH  Version 5-05    20-June-1991
%PATCH-I-NOLCL, image does not contain local symbols
old:    CONFIG_B_DUALCPU:  00
new:    CONFIG_B_DUALCPU:  00
old:    CONFIG_B_CLE_DBT:  00
new:    CONFIG_B_CLE_DBT:  00
old:    DBASRV_W_BASE_SAP:  12
new:    DBASRV_W_BASE_SAP:  12
old:    DBASRV_W_READAHEAD:  4
new:    DBASRV_W_READAHEAD:  4
old:    CASRV_W_SAP:  4
new:    CASRV_W_SAP:  4
old:    CASRV_W_HISTORIAN_PRIORITY:  4
new:    CASRV_W_HISTORIAN_PRIORITY:  4
old:    CASRV_W_QSIZE:  32
new:    CASRV_W_QSIZE:  32
old:    IONYX_B_IN_SYSTEM:  00
new:    IONYX_B_IN_SYSTEM:  01
old:    CC_B_IN_SYSTEM:  00
new:    CC_B_IN_SYSTEM:  00
old:    CIO_B_IN_SYSTEM:  00
new:    CIO_B_IN_SYSTEM:  00
old:    ICC_B_ARB_PATH_ENABLED:  00
new:    ICC_B_ARB_PATH_ENABLED:  00
old:    ICC_B_DBASRV_PATH_ENABLED:  00
new:    ICC_B_DBASRV_PATH_ENABLED:  00
%PATCH-I-WRTFIL, updating image file DISK$USER:[CRISP.EXE]CRISP.EXE;1


%C32CONFIG-S-DONE, The CRISP Configuration procedure is finished

$
```

Notes:

Description

The CRISP_SETUP command procedure performs steps necessary to setup the system to run CRISP/32. This includes defining the CRISP$xxx logical names, installing device drivers (if required), installing CRISP run-time libraries, and setting up local workstation keyboard ports.

Format

To execute the CRISP_SETUP procedure interactively, enter the following.

@device:[CRISP]**CRISP_SETUP**

Operation

The setup command file should be executed by the VMS system startup command procedure when the system is booted, but the command file may be executed any time CRISP is not running, if necessary. CRISP_SETUP checks for the CONFIG_RESULTS.COM file and will, if the file is not found, print a warning message indicating the file does not exist and CRISP will not run.

When executing the CRISP_SETUP command procedure, the CRISP system looks for command procedures generated by the user to be executed each time the CRISP_SETUP command procedure is entered. These command procedures should be placed in the CRISP$CFG: directory and should be named USER_SETUP_xxx.COM; where, 'xxx' is any meaningful alphanumeric string (i.e., USER_SETUP_MINE.COM).

The VMS system startup command procedure is named SYS$MANAGER:SYSTARTUP_V5.COM for VAX/VMS V5 systems and is named SYS$MANAGER:SYSTARTUP_VMS.COM for Open VMS V6 systems.

Example

A line similar to the following should be added at or near the bottom of the VMS system startup command procedure. The device name DISK$USER should be a logical name pointing to the actual disk device on which CRISP was installed.

```
$ @DISK$USER:[CRISP]CRISP_SETUP
```

Notes:

Description

CRMON is the user interface to the CRISPmon process monitoring process. It sends Software Bus messages to CRISPmon in response to user commands allowing the user to tell CRISPmon how to monitor the 'health' of the CRISP system software. CRMON commands enable the user to request that processes on the Software Bus be monitored or no longer monitored, to specify which processes should be considered critical or noncritical, or to obtain a display of the processes currently being monitored.

Format

The command format is as follows.

CRM◎ℕ [command]

The following is a list of CRMON commands.

Command	Description
HELP	Displays a list of the available commands within CRMON.
MONITOR	Requests that CRISPmon monitor a process.
REMOVE	Requests that CRISPmon remove a process from being monitored.
SHOW	Requests a display of the processes currently being monitored.
SWAP SARB	Holds the active CPU during Serial Arbitration unit maintenance.
EXIT	Used to exit CRMON and return to VMS.

The following keys may also be used to exit CRMON and return to VMS.

Ctrl/Z or **F10**

CRSTART will automatically request monitoring of CRISP system processes such as the Database Access Server. User logic programs will be monitored according to the information specified during logic configuration (LGCONFIG). Additional CRMON commands may be placed in a DCL command file, CRISP$CFG:USER_START_xxx.COM. Then, when the CRISP system is started, the additional processes will also be monitored.

Monitor Command

The MONITOR command requests monitoring of the specified process. It has the following format.

MONITOR process_name [/CRITICAL] [/COUNT=nn] [/NOCRITICAL]

Default Parameters:

/NOCRITICAL
/COUNT=50 (12.5 seconds)

Parameters:

process_name -	CRISP Software Bus process name.
/CRITICAL -	Makes process a critical process.
/NOCRITICAL -	Makes process a noncritical process.
/COUNT=nn -	Timeout time (nn = increments of 250 ms)

Remove Command

The REMOVE command terminates monitoring of the specified process. It has the following format.

REMOVE process_name

Parameters:

process_name -	CRISP Software Bus process name.

Show Command

The SHOW command requests a display of the processes which are currently being monitored. It has the following format.

SHOW [/ALL] [/CRITICAL] [/NOCRITICAL]

Parameters:

/ALL (default**) -**	Show all processes being monitored .
/CRITICAL -	Show only the critical processes.
/NOCRITICAL -	Show only the noncritical processes.

Swap Command

The SWAP SARB command forces the Active CPU to remain Active even if there is a failure of a critical process. This feature is intended only for use during online replacement of the Serial Arbitration unit itself. More information on this may be found in the Serial Arbitration Application Note. This command has the following format.

SWAP SARB

Operation

The CRISPmon process performs certain operations every quarter second (250 ms) to monitor the health of the CRISP system. For all system types, this includes decrementing the health count of each process that is being monitored. If any health count reaches zero, that process is considered to be stopped and the system is no longer considered healthy. Each process to be monitored must reset its counter periodically. For user-written programs, the SWB$RESET_HEALTH_COUNT function may be used to perform this operation. The health count specified on the MONITOR command must be determined by the frequency that the counter is reset by the process.

If a process being monitored fails, CRISPmon detects the failure and sends a message to the CRISP$TT device. If the process is critical, a message is also placed into the NETMON database so it can be copied into a user database via the READ_NETMON_MSG logic call. When a critical process failure is detected in a redundant CRISP system, CRISPmon causes a switchover by immediately forcing that system to the Standby state. This allows the other system to become Active. If the Standby CPU also has a critical process failure, the Active CPU will remain Active until the Standby CPU recovers.

For redundant CRISP systems, CRISPmon also refreshes the arbitration watchdog timer every 250 ms. Redundant operation is described more fully below.

Every 5 minutes, CRISPmon reviews the list of processes it is monitoring and signals (on the CRISP$TT device) a message for each process that is still stopped.

There are two types of processes that can be monitored, critical and noncritical. The difference between the two is that failure of a critical process on a redundant CRISP system causes a switchover to the Standby CPU. If the system is not redundant, there is no difference between critical and noncritical processes.

Every 5 minutes, CRISPmon cycles through the list of processes it is monitoring and signals the ones that are still stopped.

Redundant Systems

CRISP systems can have redundant (dual) processors. When a redundant configuration exists, the processor controlling the plant process is called the Active host. The processor that is not in control is called the Standby host. If the Active processor fails, the system will automatically switch to the Standby machine.

The two hosts in a redundant configuration communicate their health to an independent intelligent device. The intelligent device assigned to each host system also communicates to the device assigned to the other host. This communication is called arbitration. When a host fails to communicate with its arbitration device, the watchdog timer expires and causes a switchover.

Redundant Systems (cont)

The CRISP programmer can determine the current status of the system from the first word of logicals in a bit called Active. This bit will be True if the system is currently active. For C or FORTRAN programs, the functions DBA$GET_CRISP_DUALCPU, DBA$GET_CRISP_ACTIVE, and DBA$GET_CRISP_STANDBY may be used to determine the configuration and status of the system.

CRISP supports two different hardware configurations for this arbitration. VAX processors with a Q-bus may have a CRISP Arbiter board plugged into each system's backplane to perform arbitration. Any pair of CRISP systems with available serial ports may be connected to a Serial Arbitration box. In either case, the independent intelligence of the arbitration devices ensures proper switchover even in the case of complete failure of the Active host and also ensures that only one host will be Active at any point in time.

Arbiter Board

The CRISP Arbiter board is an intelligent Q-bus board that provides arbitration between two host processors. It also provides an RS-485 communication path between the machines for software synchronization (ICC) via the SENICC logic call.

The Arbiters of CPU A and CPU B are connected by an Inter-Arbiter Communication Cable. This polarized cable determines which host is CPU A and which is CPU B. This cable carries the following information between hosts:

- Local Arbiter Active/Standby status
- Remote Arbiter Active/Standby status
- Arbiter hardware address (1 or 2)
- Arbiter health status (MALFunction line)
- RS-485 serial data line.

The Arbiter address (1 or 2) is not the logical addressing used in the RS-485 protocol. It is strictly an address used by the hardware to determine whether the host is CPU A or CPU B.

The Active/Standby status is the arbitration line between Arbiter boards. Both Arbiters attempt to achieve Active status at power up, but the first one to assert Active becomes the Active machine. If they both assert simultaneously, the even machine (2) delays for a short period and the odd machine (1) becomes the Active machine.

CRISPmon also checks the Arbiter cable every 250 ms. If the cable is not connected, the Arbiter boards maintain their current state in both CPUs until the Arbiter cable is reconnected, at which time redundancy is restored.

The only condition that will cause a system to switchover is if the Active Arbiter goes to Standby. The Standby Arbiter senses that the other Arbiter is no longer Active and then asserts itself as the Active Arbiter.

Serial Arbitration

The Serial Arbitration unit is a standalone box containing two small intelligent devices which perform the arbitration functions. The box is connected to the host CPUs via a serial port for each CPU. ICC communications is not supported by this unit, although the SENICC call may still be used with the Database Access Server communication path.

The Serial Arbitration unit is connected to the host processors by standard null modem cables. The host serial ports may be either RS-232 or RS-423 (MMJ) ports on the processor itself or, if necessary, the serial ports may be in one or a pair of (Ethernet) terminal servers. The most reliable case, of course, uses serial ports that are internal to the host processors. CPU A and CPU B are determined by which host is connected to which port of the Serial Arbitration unit, however, it is completely arbitrary unless the optional switchcard unit is being used. Additional information about this unit may be found in the Serial Arbitration Note (DM-500 504-001).

Example

```
$ CRMON
CRMON> MON CLE /CRIT /COUNT=10
CRMON> MON ICC /CRIT /COUNT=10
CRMON> MON H1 /NOCRIT /COUNT=40
CRMON> MON H2
CRMON> SHOW

        CRISPmon Critical Process List
 Proc Name      Reset Count    Current Count
 ──────────────────────────────────────────
  CLE                 10              9
  ICC                 10              9

        CRISPmon NonCritical Process List
 Proc Name      Reset Count    Current Count
 ──────────────────────────────────────────
  H1                  40             39
  H2                  50             46

CRMON> REM ICC
CRMON> REM H2
CRMON> SHOW

        CRISPmon Critical Process List
 Proc Name      Reset Count    Current Count
 ──────────────────────────────────────────
  CLE                 10              9

        CRISPmon NonCritical Process List
 Proc Name      Reset Count    Current Count
 ──────────────────────────────────────────
  H1                  40             39

CRMON> EXIT
$
```

Description

The user enters the CRSTART command to start the CRISP system. This command is entered after the CRISP system has been installed and configured and the user databases and logics have been defined and compiled. This command is also used when the CRISP system has been stopped and is now ready to be restarted.

Format

To execute the CRSTART command, enter the following.

CRSTART { [[COLD] [RESTORE]] | [[name]] }

COLD This optional parameter requests the **system** databases be replaced with fresh (as linked) copies. The default is to use the previously installed system databases (i.e., a hot start).

RESTORE This optional parameter requests the running **user** databases be used. The default is to replace the running user databases with fresh copies.

name This optional parameter specifies a process to be started. Only one process can be specified. If CRISP is running, only the named process will be started. If CRISP is not running, this parameter is ignored. Valid names are as follows.

CASRV	CRISP Access Server
CDTXCUT	Print Converter
CLE	CRISP Logic Executive
CSAAG	CRISPconnect Servers for @aGlance
CSRDDE	CRISPconnect Server for NetDDE
CWSTND	CRISP Server Trend
DBASRV	Database Access Server
FILES	Files Server
HIST	Historian
HISTLOG	Historian Logger
HISTSEP	Historian Separator
ICC	Inter-Computer Communications
IDC	Inter-Database Communications
NETMON	Network Monitor
PRINT	Print Server

The COLD and RESTORE parameters cannot be specified in this case.

Operation

The optional argument 'cold' indicates how the CRISP system databases should be installed. (System databases are not user databases.) Normally, after the CRISP system has been started once, the CRISP system databases remain installed even after CRISP is stopped. When CRISP is restarted, it is said to be starting 'hot'. This means the CRISP system databases do not need to be reinstalled. The user can force the reinstallation of the CRISP system databases by entering the optional argument 'cold', which forces a cold start. The first time CRISP is started after VMS has been booted is always a cold start.

Operation (cont)

A 'hot' start is always performed if there are any processes connected to the Software Bus when the CRSTART command is invoked.

NOTE

The system database (CRISP) and the CRISP shareable global section are always installed 'cold'.

The optional argument 'restore' indicates how user databases should be reinstalled. When the system is restarted, the CRISP system recopies the new database files to the running database directory. You can override this by indicating you want to restore the old databases. In this case, the copy operation is eliminated and the old database is reused with the variable values that existed in the database when the process was stopped.

NOTE

There is no way to select which user database should be restored. They are either all fresh or all restored.

When executing the CRSTART command, the CRISP system looks for command procedures generated by the user to be executed each time the CRSTART command is entered. These command procedures should be placed in the CRISP$CFG: directory and should be named USER_START_xxx.COM; where, 'xxx' is any meaningful alphanumeric string (i.e., USER_START_MINE.COM).

Examples

`$ CRSTART<ret>`	Defaults to a hot start if possible, the system databases are reused and fresh user databases are used.
`$ CRSTART COLD<ret>`	Fresh system databases only.
`$ CRSTART RESTORE<ret>`	The system databases and user databases are reused.
`$ CRSTART COLD RESTORE<ret>`	Fresh system databases are used, but user databases are reused.
`$ CRSTART DBASRV<ret>`	Start just the Database Access Server process.
`$ @CRISP$CFG:START_CWS<ret>`	Restarts the CWS processes.

Description

The user enters the CRSTOP command to stop execution of the CRISP processes.

Format

To execute the CRSTOP command, enter the following.

CRSTOP [name [name ...]]

Operation

If name is not specified, the CRSTOP command sends a Software Bus 'Exit' message to every process, including user processes, that is connected to the Software Bus. It is the responsibility of each process to receive that message, perform process-specific exit procedures, disconnect from the Software Bus and exit. After these messages have been sent, the CRISP system looks for command procedures generated by the user to be executed each time the CRSTOP command is entered. These commands could be used to ensure user programs exit when CRISP is stopped. These command procedures should be placed in the CRISP$CFG: directory and should be named USER_STOP_xxx.COM; where, 'xxx' is any meaningful alphanumeric string (i.e., USER_STOP_MINE.COM).

The optional argument [name] is used to specify an individual process that is to be sent an 'Exit' message. In this case only the named process is stopped, not the entire CRISP system. Up to eight names may be specified if separated by spaces. Use PDMON to view the named processes on the Software Bus.

User-written C or FORTRAN programs will also exit if they connect to the Software Bus and handle the 'Exit' message. Refer to the SWB_EXAMPLE programs in CRISP$HLP:.

⚠ CAUTION

Indiscriminately using CRSTOP on an individual process may cause unusual operation of CRISP/32.

Examples

```
$ CRSTOP<ret>
```

```
$ CRSTOP CRT001<ret>
```

Notes:

Description

DBINSTALL is a utility that allows the user to install a CRISP database without stopping the execution of the CRISP system. This command is entered after compiling and linking the new database. There cannot be an old version of the database installed when this is attempted.

Format

The DBINSTALL utility is started by the following command.

DBINSTALL [command]

If DBINSTALL is entered with no command, the prompt `DBINSTALL>` appears on the screen. At the prompt, enter any valid DBINSTALL command. If DBINSTALL followed by a complete command line is entered at the system prompt, DBINSTALL will exit after completing that command.

The following is a list of DBINSTALL commands.

Command	Description
INSTALL	Used to install a CRISP database.
SAVE	Used to save the installed database to its image on disk. This is the same function as the SAVEDB logic call.
EXIT	Used to exit DBINSTALL and return to DCL.

The following keys may also be used to exit DBINSTALL and return to VMS:

Ctrl/Z or **F10** (key)

Install Command

The INSTALL command format is shown below, followed by a description of its arguments.

INSTALL filespec [/qualifier /qualifier ...]

filespec The file specification 'filespec' identifies the CRISP database. The default location is the current default directory and the default file type is .DBE. Note, however, that the database file is not normally installed in the same directory as the logic source. Instead, the .DBE file should be copied to the [CRISP.DB] directory and that copy of the file should be installed. This is the default behavior of LGCONFIG.

(Continued on next page.)

Install Command (cont)

/qualifier The following qualifiers control the definition of the specified CRISP database.

/DBNAME=name This specifies the database name. If not specified, the first six characters of the filename are used.

/IO This specifies that an I/O process will be performed with the database.

/LOCAL This specifies that the database is a local database. This is the default condition.

/PRIVATE This specifies that the database is a private database.

/PUBLIC This specifies that the database is a public database. This is the default condition.

/READ=n This specifies the read priority that the database is assigned in CRISP. The priority number 'n' may be an integer from 1 to 15. The default is 5.

/SYSTEM This specifies that the database is a system database.

/WRITE=n This specifies the write priority that the database is assigned in CRISP. The priority number 'n' may be an integer from 1 to 15. The default is 5.

Operation

The INSTALL command is used to install a CRISP database file (dbname.DBE) for use by CRISP logic. The INSTALL command enables the user to define the following characteristics when saving the database:

- Database location
- The CRISP database name
- Access restrictions
- Read and write priorities.

Example

The following example shows valid command syntax.

Example	Description
DBINSTALL> INSTALL ASHDLR /PRI /REA=5 /W=5	This installs the database file ASHDLR.DBE with the following characteristics: Database name = ASHDLR Private Read Priority = 5 Write Priority = 5

Notes

Description

The user enters the DBTCMD command to generate a database transfer list.

Format

The command format is as follows.

DBTCMD filespec

Operation

A database transfer description file must be built via a text editor before executing the DBTCMD command. The database transfer description text file is specified as an argument to the DBTCMD command. DBTCMD converts the description file into a binary file that can be used by the CRISP/32 system software.

DBT can transfer data locally. To transfer data to another machine, refer to the IDCCMP section of this manual.

NOTE

If the CRISP system is configured to support IDC, the database transfer list generated by using DBTCMD will be ignored. Therefore, the transfer identified in the DBTCMD transfer description file will not occur.

The database transfer description file is a text file composed of one or more transfer command lines. Each transfer command line specifies a range of source variables, an initial destination variable (the first, in an array of variables, into which the source variable values will be written) and a conditional 'trigger' variable. When the trigger variable changes from FALSE to TRUE, the values in the source variables are transferred to the corresponding destination variables.

(Continued on next page.)

Operation (cont)

The syntax for the transfer command line is as follows.

TRANSFER /SOURCE=(db:var1,var_n) /TRIGGER=db:cond /DEST=db:var1

Where TRANSFER, /SOURCE=, /TRIGGER=, and /DEST= are required keywords of the command line. Items shown in lower case must be substituted with the actual database and variable names of your system. Array references with constant subscripts are supported; however, the `db:var` or `var` must then be enclosed in quotes (see example). The END keyword may optionally be included in the file to terminate command processing. If the END keyword is not used, the [End of file] will be recongized. However, if the END keyword is used, comments could be included in the file after the END keyword. These comments would have no affect on the creation of the binary file.

/SOURCE=(db:var1,var_n) Indicates a range of data to be moved from database `db1`. All variables between `var1` and `var_n` are transferred. All variables in a single transfer command line must be of the same variable type. The variable `var1` must occur before `var_n` in the database.

/TRIGGER=db:cond Specifies the name of the variable that initiates the transfer. The transfer begins when `db:cond` changes from FALSE to TRUE.

/DEST=db:var1 Specifies the name of the variable into which the value of the first source variable is written. The next source variable is written into the next destination variable until the transfer is complete.

Comments may be placed in the database transfer description file delimited by an exclamation mark (`!`) at the beginning and a carriage return at the end.

Refer to Appendix A for important timing considerations when using DBT type database transfers.

Example

A typical database transfer description file is shown in the following.

```
! TRANSFER VARIABLES FROM THE STACKING PROCESS
transfer /source=(db11:bit1,bit3) /trigger=db10:bit101 /dest=db10:bit1
transfer /source=(db11:num1,num3) /trigger=db10:bit101 /dest=db10:num1
transfer /source=(db11:long1,long3) /trigger=db10:bit101 /dest=db10:long1
transfer /source=(db11:flt1,flt3) /trigger=db10:bit101 /dest=db10:flt1
transfer /source=("db11:nums(5)","nums(8)") /trigger=db10:bit102 /dest="db10:nums(5)"
end
```

Description

The user enters the DDMON command to display a list of the currently installed databases. This command can be used to verify that a database has been installed properly.

Format

The command format is as follows.

DDM◎ℕ

Operation

The DDMON command lists the statistics of all databases installed in the system. The database statistics listed are defined as follows:

(1) Ent - This is the entry number in the database directory of the database. The entry numbers shall range from 0 to 59.

(2) DBname - This is the database name.

(3) Creation Date - This is the date that the database was created.

(4) Rd/Wrt - This is the read/write priority indicator. A user database priority number may be an integer from 1 to 15 (refer to the DBINSTALL command). System databases are installed with priorities of 5/5.

(5) Attributes - This statistic indicates the status of the database; where,

Res (reserved) indicates that the database is reserved a slot on the Software Bus.

All (allocated) indicates the database is using a Software Bus slot.

Val (valid) confirms of a successful connection to the Software Bus.

Sys (system) indicates that the database is a system database.

Loc (local) indicates that the database is a local database.

Pub (public) indicates that the database is a public database.

I/O indicates that there is an I/O process used in connection with the database.

Prm (permanent) indicates that the database remains installed after CRISP is stopped. `CRSTART COLD` is required to install a new version of this database.

Example

```
$ DDM

CRISP/32 Database Directory Monitor                        Max. entries:  60

Ent  DBname    Creation Date     Rd/Wrt        Attributes
---  ------ -------------------- ------ ------------------------------------
  0   CRISP 22-JAN-1991 13:54:18  5/ 5  Res All Val Sys Loc Pub Prm
  1  TSKDIR 29-MAY-1991 13:47:09  5/ 5  Res All Val Sys Loc Pub Prm
  2   DBDIR  8-JUN-1991 14:31:24  5/ 5  Res All Val Sys Loc Pub Prm
  3  CRTDIR 10-JUL-1990 15:03:10  5/ 5  Res All Val Sys Loc Pub Prm
  4  LOGDIR  7-MAY-1991 17:08:53  5/ 5  Res All Val Sys Loc Pub Prm
  5  NETMON 22-JAN-1991 08:40:57  5/ 5  Res All Val Sys Loc Pub
  6   WSDIR  7-JUN-1991 13:50:41  5/ 5  Res All Val Sys Loc Pub
 10  GEMINI 18-JUN-1991 15:21:41  5/ 5      All Val     Loc Pub     I/O
```

Description

The user enters the directoryDEF commands to select a specific directory within the CRISP system as the new default directory. These commands are defined by CRISP_LOGIN.COM and may be used instead of the VMS SET DEFAULT or CRISP SD commands.

Format

Enter the following command to select the appropriate directory/subdirectory.

ROOTDEF

ALTHISTDEF

CWSDEF

DBDEF

DOCDEF

DSPDEF

EXEDEF

HISTDEF

HLPDEF

IONYXDEF

LIBDEF

LOGDEF

UTLDEF

Operation

The CRISP system contains one main directory and several subdirectories. These subdirectories each contain specific types of data for use by specific functions in the CRISP system. At login, the CRISP system checks the CRISP directory to determine which subdirectories exist and creates the directoryDEF commands based on the existing subdirectories.

The CRISP system normally consists of the following directories.

(Continued on next page.)

Directory	Description
CRISP	The CRISP directory [CRISP], which is the CRISP system main directory and the CRISP user log-in default directory.
ALTHIST	The directory [CRISP.ALTHIST], which is the alternate historical data directory.
CFG	The directory [CRISP.CFG], which **may** be the system-specific configuration directory (refer to CRISP_CONFIG). This directory should only be accessed via the logical name CRISP$CFG. Not accessible via CFGDef.
CWS	The directory [CRISP.CWS], which is the Color Workstation (CWS) directory (optional).
DB	The directory [CRISP.DB], which is the running user database directory.
DOC	The directory [CRISP.DOC], which is the documentation directory.
DSP	The directory [CRISP.DSP], which is the primary Basic Workstation displays directory (optional).
EXE	The directory [CRISP.EXE], which is the executable images directory.
HIST	The directory [CRISP.HIST, which is the historical data directory.
HLP	The directory [CRISP.HLP], which is the help libraries directory.
IONYX	The directory [CRISP.IONYX], which is the I/ONYX related data directory.
LIB	The directory [CRISP.LIB], which is the object, text, and macro libraries directory.
LOG	The directory [CRISP.LOG], which is the logic source directory.
UTL	The directory [CRISP.UTL], which is the utilities directory.

Example

```
$ ROOTD
$ SHOW DEFAULT
  CRISP$DEVICE:[CRISP]
```

Description

The user enters the IDCCMP command to generate a database transfer list when using the Inter-Database Communications facility (IDC) to perform data transfers between databases.

Format

The command format is as follows.

IDCCMP filespec

Operation

A database transfer description file must be built via a text editor before executing the IDCCMP command. The default file type is .IDC. The database transfer description text file is specified as an argument to the command. IDCCMP converts the description file into a binary file (`CRISP$CFG:IDCBIN.DAT`) that can be used by the CRISP/32 system software. The description file format and IDC functionality are described in more detail in Appendix B.

IDC can transfer data locally or across the network to another machine. Refer also to the DBTCMD command for local-only transfers.

NOTE

If the CRISP system is configured to support IDC, the database transfer list generated by using DBTCMD will be ignored. Therefore, the transfer identified in the DBTCMD transfer description file will not occur.

Example

A typical IDC database transfer description file is shown in the example in Appendix B.

Notes:

Description

The user enters the LDCLN command to clean up an exited logic that is still listed in the logic directory. This utility may need to be invoked in the event a CRISP logic process has terminated abnormally.

Format

The command format is as follows.

LDCLN [entry [entry . . .]]

entry This specifies the name of the logic to clean. A logic entry number can be specified as /nnn, where nnn is the number of the entry. If no argument is specified, the user is prompted for a name.

Operation

Occasionally, a logic may exit improperly without being removed from the logic directory by CLE. If that logic is re-installed, CLE may not locate the correct process ID. If this occurs, the LDCLN utility may be used to clean out the specified logic directory entry and make it available again.

The LDMON command may be used to display all installed logics entered in the logic directory. From this display, the user can obtain the logic name or logic directory slot number for use by the LDCLN command.

Using LDCLN on a logic that still exists may cause unusual operation of CRISP/32. To stop a logic, CRISP/32 `LGINSTALL` or VMS `STOP` must be used. Verify that the logic is truly gone using VMS `SHOW SYSTEM` before using LDCLN. Using CLN may also be required.

Example

```
$ LDMON

LOGIC DIRECTORY MONITOR                                  Max. entries:  50

Slot ProcName   PID    Pri      Interval    Passcount Exe(ms)    Entry Status
---- -------- -------- --- --------------- ---------- -------  ----------------
  0  TEST     0000009B  18   0 00:00:01.00     219312      20  All Val Done
  1  MISC     0000009C  18   0 00:00:01.00      17863      20  All Val Done

$ LDCLN MISC

$ LDMON

LOGIC DIRECTORY MONITOR                                  Max. entries:  50

Slot ProcName   PID    Pri      Interval    Passcount Exe(ms)    Entry Status
---- -------- -------- --- --------------- ---------- -------  ----------------
  0  TEST     0000009B  18   0 00:00:01.00     219316      20  All Val Done
```

Notes:

Description

The user enters the LDMON command to display the logic directory. This command is used to monitor user logic processes.

Format

The command format is as follows.

LDM◎ℕ [process_name]

Operation

The LDMON command lists the status of all user logic processes installed in the system. The items listed in the logic directory are defined as follows.

Slot This is the slot number in the logic directory of the process. The slot numbers shall range from 1 to 50.

ProcName This is the name assigned to the user process.

PID This is the process identifier, which identifies active and running processes.

Pri This is the logic priority level.

Interval This is the interval at which the logic is scheduled to run.

Passcount This is the number of passes the logic has completed (longword value).

Exe This is the execution time in milliseconds for the most recently completed logic pass. This value is always a multiple of 10 milliseconds.

Entry Status This statistic indicates the status of the logic; where, All(ocated) indicates the process is using a Software Bus slot, Val(id) is a confirmation of a successful connection to the Software Bus, and Done indicates the process has completed a pass and is not currently executing.

Example

```
$ LDMON

CRISP/32 LOGIC DIRECTORY MONITOR                              Max. entries:  50

Slot ProcName   PID    Pri      Interval    Passcount Exe(ms)    Entry Status
---- -------- -------- --- --------------- ---------- -------  ----------------
  1  TEST     00000120  18   0 00:00:01.00        185      10  All Val Done
  2  CALLS    00000122  18   0 00:00:01.00        184      10  All Val Done
```

Notes:

Description

The user executes the LGBUILD command to compile and link user databases and logics to make them ready for the CRISP system at startup. This command may be used at any time after the user has configured the CRISP/32 system. It may be used either before or after the databases and logics have been defined with LGCONFIG.

Format

To execute the LGBUILD command, enter the following.

LGBUILD [src_filespec [compile [build_db [assemble [build_lg [quals]]]]]]

Operation

The user is prompted for specific data to properly create the CRISP application programs. After the user responds to all the prompts, the requested actions will be performed automatically.

If the compile is not successful, the procedure will halt and the user should search the name.LIS file for specific CRISP error messages. A useful SEARCH command is as follows:

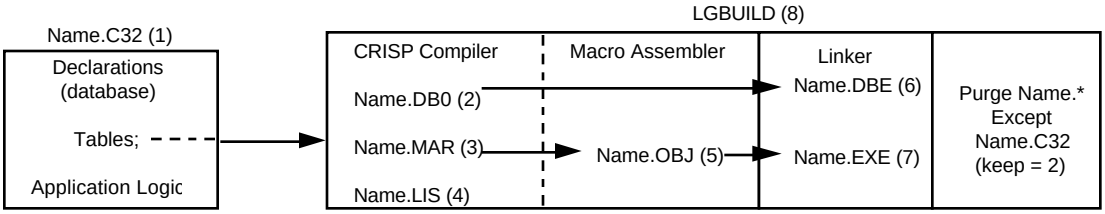
```
$ SEARCH /EXACT /WIN=(1,0) name.LIS %CRISP
```

If the entire operation is successful, all files related to the specified logic program except the source (.C32) file will be purged to the latest two versions.

For more details concerning the compiler, refer to the CRISP/32 Language Reference Manual. The following diagram shows the flow of the LGBUILD command.

When upgrading from a version of CRISP/32 prior to V2.5, it is necessary to translate existing CRISP/32 logics using the qualifier `/TRANS=V20`, which will create a new *.C32 file with the new operands used by the latest version of the CRISP compiler. Refer to the CRISP/32 Language Reference Manual for a list of the CRISP compiler qualifiers.

Operation (cont)



(1) User's logic source file created using the TPU or other editor.

(2) Database declarations in object form created only when a yes is given to the question, Do you want to build the (Name) database?

(3) Application logic rewritten in VMS Macro created only when a yes is given to the question, Do you want to assemble the (Name) logic?

(4) Optional list file. Errors from the CRISP compiler are reported in this file as '%CRISP-...'. This file is created by the CRISP compiler..

(5) Macro code in object form. This file is created by the Macro assembler. This file is created when yes is specified for the question, Do you want to assemble the (Name) logic?

(6) The database in an installable image file. Created when yes is specified for the question, Do you want to build the (Name) database? This file is created by the linker.

(7) The macro code in executable form. This file is created when a yes is specified for the question, Do you want to build the (Name) logic?

(8) An interactive command invoked by the user, which executes the compiler, macro, and linker commands based on yes or no answers to the questions.

Examples

```
$ LGBUILD

       +----------------------------------------+
       | CRISP/32 user logic build procedure    |
       +----------------------------------------+

Enter logic filespec . . . . . [AI1010]:
  Do you want to compile AI1010 . . . . . [Yes]?  ‹
  Do you want to build the AI1010 database  . . [Yes]?  ‹
  Do you want to assemble the AI1010 logic  . . [Yes]?  ‹
  Do you want to build the AI1010 logic . . . . [Yes]?  ‹
  Enter CRISP compiler qualifiers:

%LGBUILD-I-START, CRISP logic build started at dd-mmm-yyyy hh:mm:ss.ss

$ Crisp /list  DISK$USER:[CRISP.LOG]AI1010.C32;
$ Macro AI1010
$ Link /share=AI1010.DBE AI1010.DBO
$ Link /map /full AI1010

›

%LGBUILD-I-DONE, CRISP logic build is finished at dd-mmm-yyyy hh:mm:ss.ss
```

‹ - If no is entered, the corresponding step will be skipped.

› - Errors listed after the **Link** command may indicate that the user has used an invalid function call name.

```
$ LGBUILD XYZ
```

This command performs a complete compile and build as if all default answers were used.

```
$ LGBUILD XYZ Y N
$ LGBUILD XYZ "" N
```

These commands compiles XYZ.C32, assembles XYZ.MAR, and builds the logic image file (XYZ.EXE), but does not build a new database.

Notes:

Description

The user executes the LGCONFIG command to create the database and the application program definition files for the CRISP startup.

Format

To execute the LGCONFIG command, enter the following.

LGCONFIG

Operation

The user is prompted to define all databases and all logics that run against the databases. The 'new' database filespec is the database that results from compiling and building the database. Usually, this is the file `[CRISP.LOG]name.DBE`.

The new database is never modified by the system. It is copied to [CRISP.DB] and then installed from there. That database file is modified when the installed database is 'saved' or when the CRISP system is shut down.

When the CRISP system is restarted, it copies the new database files to the running database directory. You can override this by indicating you want to restore the old database. In this case, the copy operation is eliminated and the old database is reused (the command used to perform this is: CRSTART RESTORE).

LGCONFIG writes two command files in CRISP$CFG: for later use. START_USER_DB.COM is used by CRSTART to copy new databases to running databases and install the databases. START_USER_LG.COM is also used by CRSTART to start the user logics associated with the databases. It also creates CRISP$CFG:LGCONFIG_RESULTS.COM, which contains the answers to the logic configuration questions. The next time LGCONFIG is executed, these saved answers will be used as default answers.

In order to remove a database and corresponding logic from a previous configuration, at the prompt `Enter new database filespec` for the appropriate database, enter a minus (-).

Example

```
$ LGCONFIG

        +----------------------------------------+
        | CRISP/32 Logic Configuration procedure  |
        +----------------------------------------+

Enter new database filespec . .[]: A2
  What is the database name . . . . . [A2]?<Ret>
  Is A2 an I/O database . . . . . .[No]? Y<Ret>
  Is there Logic for the A2 database . . . [Yes]?<Ret>
    Enter logic filespec . . . [DISK$USER:[CRISP.LOG]A2]:<Ret>
    Enter the logic name . . . . . [A2]:<Ret>
    At what priority should A2 run . . . .  [18]?<Ret>
    At what time interval should A2 run [0 00:00:01.00]?<Ret>

        The SAP for a particular logic must be the same for both CPUs in a
        redundant pair and may not be used by any other process in the same
        CPU.  By default, we start at 128 and progress up by 4 for each I/O
        logic.  This value is only used for 802.3 (Ethernet) I/ONYX I/O --
        zero may be specified if the logic uses arbiter I/O.

    What SAP should A2 use for I/O . . . .  [128]?<Ret>
    Should A2 be monitored by CRISPmon . . [Yes]?<Ret>
      Is A2 a critical process . . . . . [Yes]?<Ret>
      Enter the countdown value . . . . . [10]:<Ret>

Enter new database filespec . .[]: B1
  What is the database name . . . . . [B1]?<Ret>
  Is B1 an I/O database . . . . . .[No]? Y<Ret>
  Is there Logic for the B1 database . . . [Yes]?<Ret>
    Enter logic filespec . . . [DISK$USER:[CRISP.LOG]B1]:<Ret>
    Enter the logic name . . . . . [B1]:<Ret>
    At what priority should B1 run . . . .  [18]?<Ret>
    At what time interval should B1 run [0 00:00:01.00]?<Ret>

        The SAP for a particular logic must be the same for both CPUs in a
        redundant pair and may not be used by any other process in the same
        CPU.  By default, we start at 128 and progress up by 4 for each I/O
        logic.  This value is only used for 802.3 (Ethernet) I/ONYX I/O --
        zero may be specified if the logic uses arbiter I/O.

    What SAP should B1 use for I/O . . . .  [132]?<Ret>
    Should B1 be monitored by CRISPmon . . [Yes]?<Ret>
      Is B1 a critical process . . . . . [Yes]?<Ret>
      Enter the countdown value . . . . . [10]:<Ret>

Enter new database filespec . .[]:<Ret>

Writing CRISP$CFG:LGCONFIG_RESULTS.COM
Writing CRISP$CFG:START_USER_DB.COM
Writing CRISP$CFG:START_USER_LG.COM
```

Description

LGINSTALL is a utility that allows the user to start and stop CRISP logic processes and to modify CRISP logic process parameters.

Format

The LGINSTALL program is accessed by entering the following command.

LGINSTALL [command]

If LGINSTALL is entered with no command, the prompt `LGINSTALL>` appears on the screen. At the prompt, the user can enter any of the following commands. If LGINSTALL followed by a complete command is entered at the system prompt, LGINSTALL will exit after completing that command.

The following is a list of LGINSTALL commands.

Command	Description
HELP	Used to display on-line help for the LGINSTALL commands.
START	Used to start CRISP logic.
STOP	Used to stop CRISP logic.
MODIFY	Used to modify execution parameters of a running logic.
EXIT	Used to exit LGINSTALL and return to DCL.

The following keys may also be used to exit LGINSTALL and return to VMS.

Ctrl/Z or **F10** (key)

Start Command

The START command format is shown below, followed by a description of its arguments.

START filespec /qualifier [/qualifier]...

filespec The file specification 'filespec' identifies the executable CRISP logic file. The default file type is .EXE and the default location is the current default directory.

/qualifier The following qualifiers control the execution of the specified CRISP logic.

/NAME=name This specifies a process name to be assigned to the executing CRISP logic, where 'name' is the process name. The 'name' is truncated to 8 characters. If a process name is specified that is currently assigned to an existing process or to a process connected to the software bus, the logic will not execute.

The logic filename becomes the process name by default if /NAME is not specified.

/DBNAME=name This specifies a CRISP database name. The 'name', is truncated to six characters. If a database name is specified that does not exist, the logic will not execute.

The logic filename becomes the database name by default if /DBNAME is not specified.

/PRIORITY=n This specifies the priority at which the logic executes. The priority number 'n' may be an integer from 1 to 25. For complete information on process priorities, refer to VMS documentation.

A default priority of 18 is assigned to the CRISP logic process if the /PRIORITY qualifier is not specified.

(Continued on next page.)

Start Command (cont)

/SAP=n This specifies the SAP used for IEEE 802 (Ethernet) communications to I/ONYX clusters. If zero or omitted, a SAP will be selected. This is preferable for non-redundant host systems. For redundant hosts, this must be specified as an integer multiple of 4, between 4 and 252, and must be the same on each host for the same logic. Different logics on the same host must have different SAPs. This is ignored by logics that are not using Ethernet communications.

/INTERVAL="time" This specifies the time interval between successive executions of the CRISP logic process. For complete information on wakeups and hibernation of processes, refer to VMS documentation ($HIBER system service). The time interval must be enclosed in quotation marks (") and must be expressed in the following format:

"d hh:mm:ss.cc"

where d=days, hh=hours, mm=minutes, ss=seconds, and cc=hundredths. A maximum of 366 days and a minimum of 0.2 seconds may be specified.

Note that the scheduling interval should be greater than the logic execution time as shown in the logic directory (refer to the appropriate system display or LDMON).

A default interval of 1 second is assigned if /INTERVAL is not specified.

Stop Command

The STOP command format and a description of its argument are defined in the following.

STOP procname

procname This specifies the process name of the CRISP logic process to be terminated.

Modify Command

The MODIFY command format and a description of its arguments are defined in the following.

MODIFY procname /qualifier [/qualifier]...

procname This specifies the process name of the CRISP logic process to be modified.

/qualifier The following qualifiers control the execution of the specified CRISP logic.

/PRIORITY=n This specifies the priority at which the logic executes. The priority number 'n' may be an integer from 1 to 25. For complete information on process priorities, refer to VMS documentation.

If the /PRIORITY qualifier is not specified, the logic priority will be unchanged.

/INTERVAL="time" This specifies the time interval between successive executions of the CRISP logic process. For complete information on wake ups and hibernation of processes, refer to VMS documentation ($HIBER). The time interval must be enclosed in quotation marks (") and must be expressed in the following format:

"d hh:mm:ss.cc"

where d=days, hh=hours, mm=minutes, ss=seconds, and cc=hundredths. A maximum of 366 days and a minimum of 0.2 seconds may be specified.

Note that the scheduling interval should be greater than the logic execution time as shown in the logic directory (refer to the appropriate system display or LDMON).

If /INTERVAL is not specified, it will not change.

Operation

LGINSTALL interprets commands entered by the user and formats messages to the CRISP Logic Executive (CLE), which, in turn, affects execution of a CRISP logic process.

(Continued on next page.)

Operation (cont)

The START command is used to cause an assembled and linked CRISP logic file (filename.EXE) to execute as a process. The START command allows you to determine the following characteristics at startup time:

- process name
- the CRISP database to associate with the application program.
- process priority
- hibernation wake-up interval

The MODIFY command is used to alter the priority of a currently executing logic process.

The STOP command is used to terminate the execution of a process that is running a CRISP application program.

Example

The following examples show valid command syntax.

Example	Description
START ASHDLR /P=5 /I="0 to::10"	This starts the logic image file ASHDLR.EXE running as a process with the following characteristics: Process name = ASHDLR Priority = 5 Wake-up interval = 10 sec. Database name = ASHDLR.DBE

Example	Description
START TUPELO101	This starts the logic file TUPELO101 running as a process with the following characteristics: Process name = TUPELO10 Database name = TUPELO Priority = 18 Wake-up interval = 1 second

Notes:

Description

The user enters the PDMON command to display the processes connected to the Software Bus.

Format

The command format is as follows.

PDM◎ℕ [process_name]

Operation

The PDMON command lists the statistics of all processes connected to the Software Bus. The process statistics listed are defined as follows:

Entry	This is the process entry number in the Software Bus. The entry numbers range from 0 to 129. Entries 0 through 19 are reserved for use by CRISP/32.
ProcName	This is the process name.
PID	This is the VMS process identifier that identifies active and running processes.
Health	This is the health counter. This indicator is a count of the messages sent to the system monitor by CRISPmon.
Messages	This indicates the number of Software Bus messages that have been received by the process.
Status	This statistic indicates the status of the process; where, **Rsv** (reserved) is an indication that the process is reserved a slot on the Software Bus. **All** (allocated) indicates the process is using a Software Bus slot. **Val** (valid) is a confirmation of a successful connection to the Software Bus. **DBA** (database access) is an indication that the process is accessing a database.

Example

```
$ PDMON

CRISP/32 Process Directory Monitor                    Max. entries:  130

Entry ProcName   PID    Health  Messages        Status
----- -------- -------- ------ ----------  --------------------------
   0  CLE      254006B6      7          0  Rsv All Val DBA
   1  DBCTRL   2540062C      0          5  Rsv All Val DBA
   2  CRISPMON 254006AE      0          2  Rsv All Val DBA
   3  DBASRV   2540066D     23       3680  Rsv All Val DBA
   4  FILES    254006F3      0          0  Rsv All Val
   5  PRINT    254006F2      0          0  Rsv All Val
   6  CDTXCVT  254006B1      0          0  Rsv All Val
   7  ICC                    0          0  Rsv
   8  ICM                    0          0  Rsv
   9  IDC                    0          0  Rsv
  10  CASRV    25400542      0          0  Rsv All Val DBA
  11  CWSTND   254005B4      0       3681  Rsv All Val
  12  NETMON   254005F0      0          0  Rsv All Val DBA
  13  CRKILL                 0          0  Rsv
  14  CRMON                  0          0  Rsv
  15  DBINSTAL               0          0  Rsv
  16  LGINS_0                0          0  Rsv
  17  ICF                    0          0  Rsv
  18  CSRDDE                 0          0  Rsv
  19  CSRAAG                 0          0  Rsv
  20  CSHAAG                 0          0  Rsv
  21  TEST     00000120      7          1      All Val DBA
  22  CALLS    00000122      7          1      All Val DBA
  23  HIST     00000123      0          0      All Val DBA
  24  HISTLOG  00000124      0          0      All Val
  25  HISTSEP  00000125      0          0      All Val
$
```

Appendix A
Database Transfer Timing

General

There are important timing considerations when using DBT type database transfers.

Constraints

If the following constraints are not followed, the requested transfers may not occur as expected.

- The trigger specification is sampled every 90 ms for a FALSE to TRUE transition by CLE.
  - The trigger variable must be FALSE for at least 90 ms prior to its transition to TRUE.
  - The trigger variable must be TRUE for at least 90 ms for CLE to recognize the trigger.
- Triggered transfers are placed in a queue that is processed at the end of logic execution for the logic associated with the source database. If there is no logic associated with the source database, the transfer is placed in a queue that gets serviced once a second.
- In order for the transfer to occur at the end of the current pass of logic, the trigger must have been enabled at least 90 ms prior to the end of logic sot that the entry can be placed in the queue prior to the end of logic. Otherwise the transfer will not occur until the end of the next pass of logic.
- The source data must remain unchanged until after the transfer has occurred.
  - The time for this to occur can be as little as a few milliseconds after the end of logic if the trigger is seen more than 90 ms before the end of logic.
  - The time can be as long as one logic pass time plus a few milliseconds if the trigger is not set before the last 90 ms of logic execution.
  - If there is no logic associated with the source database, the time from trigger to transfer can vary from a few milliseconds to one second plus a few milliseconds.

Notes:

General

The Inter-Database Communications (IDC) facility provides for the transfer of data from a local CRISP/32 source database to either another local CRISP/32 destination database or a remote CRISP/32 destination database. A remote database resides on a different VAX system connected to the same IEEE 802.3 (Ethernet) network as the local VAX. If the destination database exists on both members of a redundant pair of VAXs, transfer may be made to both.

An IDC transfer description source file specifies these transfers in terms of the following four types of named entities.

Variable Blocks — Specify groups of variables that may be either the source or destination for IDC transfers.

Transfer Specifications — Specify source-destination variable block pairs.

Groups — Consist of lists of either variable blocks or transfer specifications.

Lists — Provide a means of specifying IDC transfers in terms of source-destination variable name pairs.

The Source database must be in the local node, but the destination database may be in the same or a different node as long as the database names are unique on the network (except in the case of redundant node pairs and in the case of specifying an explicit destination node name.

The IDC source file consists of a series of DCL-like commands, which define the variable blocks, transfer specifications, etc., and is processed by a compiler (IDCCMP) to generate a binary file (normally IDCBIN.DAT in `CRISP$CFG:`) to be read by the running CRISP system. The compiler also recognizes a command that allows trigger conditions for the transfers specified by the various transfer specifications to occur. All DCL syntax rules concerning spacing, line continuation, list construction, comments, etc. apply.

Commands

DECLARE BLOCK /LIMITS=(var1,var2) blockname

This command declares a variable block by starting and ending variable names. All variables starting with **var1** up to and including **var2** comprise a block named **blockname**. This block may be used as either a source or destination block in an IDC transfer. For each array variable in the range of the block, the entire array is included in the block. Both **var1** and **var2** must be the same data type.

DECLARE BLOCK /START=var /COUNT=nn blockname

This command declares a variable block by count. The next **nn** variables starting with **var** comprise a block named **blockname**. This block may be used as either a source or destination block in an IDC transfer. For each array variable in the range of the block, the entire array is included in the block.

(Continued on next page.)

Commands (cont)

DECLARE BLOCK /START=var blockname

This command declares a destination variable block. This block, beginning with the variable **var** comprises a block named **blockname**. It may only be used to define a destination block. Its length is determined at run time to be the same as the length of the source block.

```
DECLARE BLOCK /LIST blockname
   VARIABLE  var1, var2, ...
   VARIABLE  vara, varb, ...
      .
      .
      .
   VARIABLE  varx, ...
END_LIST
```

This sequence of commands declares a variable block by means of a list. The block consists of variables **var1**, **var2**, etc. Its name is **blockname**. This block may be used as either a source or destination block in an IDC transfer. Variables may be of mixed type, but corresponding source and destination types of each of the included variables must agree. Also, variables may be declared in any order independent of the order in which they occur in the database.

In the case of timer and counter variables, a reference to the variable name specifies that all elements of the timer or counter variable be transferred.

In the case of array variables, the specification may consist of the name only or the name with either one or two constant subscripts.

Note that when a subscript or subscript range is specified, the specification must be enclosed in quotation marks. Refer to the following example.

```
VARIABLE    "ARRAY_1(1:10)", "ARRAY_Z(5)"
```

The following formats are accepted.

- ARRAY (array name only) - The entire array is specified.
- "ARRAY(m)" - Specifies only the mth element of ARRAY is specified.
- "ARRAY(m:n)" - Specifies elements m through n, inclusive, are specified.
- "ARRAY(:n)" - Specifies elements 0 through n, inclusive, are specified.
- "ARRAY(m:)" - Specifies elements m through the end of the array are specified.

(Continued on next page.)

Commands (cont)

DECLARE TRANSFER_SPECIFICATION -
/SOURCE=(DATABASE=srcdbnam,BLOCK=srcblockname) -
[/FROM=srcspec] -
[/DUPLICATE_SOURCE] -
[/DESTINATION=(DATABASE=dstdbnam[,BLOCK=dstblockname]) -
[/TO=dstspec] [/EXCLUDE=dstspec] -
xfername

The transfer specification declaration is used to define source-destination block pairs. It specifies that data be copied from the database and variables specified by the **/SOURCE** qualifier to a specified destination database and variables. ***srcdbname*** *is the name of a database and* ***srcblockname*** *is either the name of a variable block specification or the name of a variable block specification group.* The destination may be specified either explicitly by means of the **/DESTINATION** qualifier or by means of the **/DUPLICATE_SOURCE** qualifier. If the destination is specified by the **/DESTINATION** qualifier, data is copied to the variables specified by **dstblockname** in the database specified by **dstdbname**. **dstblockname** may be either the name of a variable block specification or the name of a variable block specification group. If the **/DUPLICATE_SOURCE** qualifier is used, the variables specified by the **srcblockname** specification are used for the destination. If the **/DUPLICATE_SOURCE** qualifier is used, the **/DESTINATION** qualifier must be used to specify the destination database only (not a destination variable block name). If the **/DESTINATION** qualifier is missing, the name of the source database is also used for the destination.

It should be noted that the **/DUPLICATE_SOURCE** qualifier does not cause the source variable block specification to be used for the the destination, but causes the list of variable names obtained by resolving the source specification at run time to be used for the destination. A significant distinction occurs, for example, if the source block is specified by starting and ending variables (**DECLARE BLOCK/LIMITS**). In this case, the source and destination databases could each contain both the starting and ending variables, but with a completely different list of variables in between. If the source specification were simply used as the destination specification, transfers would be made between variables of different names. However, transfers are in fact made to destination variables with the same names as the source variables.

It should be noted that not all data specified by the source qualifier will necessarily be transferred to the destination. The following rules apply.

(Continued on next page.)

Commands (cont)

1. Transfers between the source and destination are made on a one-to-one basis by variable. That is, the first source variable is transferred to the first destination variable, the second source variable to the second destination variable, etc.

2. If a variable in either the source specification or the destination specification or both is undefined or there is a type mismatch between corresponding source and destination variables, no data is transferred between the pair. Data transfer still takes place between other corresponding pairs that are both defined and have matching types. For example, if the third variable in the destination specification is not defined, no data is transferred from the third variable in the source specification. However, data from the fourth source variable in the source specification may still be transferred to the fourth variable in the destination specification.

3. For the purposes of forming the one-to-one correspondence between source and destination variables, arrays are treated as single entities.

4. In transferring data between arrays, the first element specified for the source array is transferred to the first element specified for the destination array and so forth in sequence. For transfers to a local destination database, if the subscript range specified for the source array is greater than the subscript range specified for the destination array, the transfer is truncated to fit the destination range. For transfers to remote destinations, the second subscript in a range specification, if present, is ignored. That is, a specification of the form `"ARRAY(m:)"`. In either case, if the number of elements specified by the source subscript range would overrun the destination array, the transfer is truncated to fit.

5. If subscripts fall outside of array bounds, results are unpredictable. However, data is never transferred outside of array bounds (at either the source or destination).

6. Transfers between scalars and arrays are not permitted except for transfers between single array elements (for example, `"ARRAY(m)"`) and scalars.

7. If more variables are specified for the source than for the destination, the transfer is truncated at the destination count.

8. If fewer variables are specified for the source than for the destination, only source count variables will be transferred.

(Continued on next page.)

Commands (cont)

In addition to the **/SOURCE**, the **/DESTINATION** and the **/DUPLICATE_SOURCE** qualifiers, the following three qualifiers also apply.

- **/FROM**
- **/TO**
- **/EXCLUDE**

The **/FROM** qualifier affects the source specification; the **/TO** and **/EXCLUDE** qualifiers affect the destination specification.

The **/FROM** qualifier, if present, requires a keyword list, **srcspec**. This list consists of either the keyword, **ACTIVE_SOURCE**, the keyword **STANDBY_SOURCE**, or both keywords separated by a comma and enclosed in parentheses. If the **ACTIVE_SOURCE** keyword is used by itself, transfers are made only if the source processor is active. If the **STANDBY_SOURCE** keyword is used by itself, transfers are made only when the source processor is in the standby mode. If both keywords are present, transfers are made regardless of whether the source processor is in the active or standby node. If the **/FROM** qualifier is not present, transfers are made only from the active processor.

The **/TO** qualifier also requires a keyword value list (**dstspec**). Some combination of the following must be present: **REMOTE_DEST**, **LOCAL_DEST**, **ACTIVE_DEST**, **STANDBY_DEST**, and **NODE_DEST=nodename**. If the **/TO** qualifier is present, transfers are made only to the specified list of destinations. If the **/TO** qualifier is absent, transfers are made to all destinations except **NODE_DEST=nodename.**

The **NODE_DEST=nodename** keyword and value enable a user to transfer data to identically named databases on remote non-redundant nodes.

The **/EXCLUDE** qualifier takes the same list as the **/TO** qualifier, but causes the listed destinations to be excluded from a transfer.

The parameter **xfername** assigns a name to the transfer specification by which it may subsequently be referenced in a trigger specification or a transfer spec group specification.

In source and destination variable block specifications, variable types may be mixed; however, corresponding source and destination types must agree. If variables in either the source or destination specification are undefined at run time or the types do not agree, no transfer is made for those variables. Other variables are still transferred.

(Continued on next page.)

Commands (cont)

```
DECLARE LIST
      /SOURCE_DATABASE=srcdb [/FROM=srcspec] -
      /DESTINATION=dstdb [/TO=dstspec] [/EXCLUDE=dstspec] -
      xfername
   COPY vara var1
   COPY varb var2
      .
      .
      .
   COPY varx var24
END_LIST
```

The transfer list declaration requires a series of commands and is the equivalent of a source block declaration, a destination block declaration, and a transfer specification declaration. It specifies that the variables **vara**, **varb**, … residing in database **srcdb** be copied to the variables **var1**, **var2**, … in database **dstdb**. It allows source-destination variable pairs to be listed individually. The transfer list has the name **xfername**. The qualifiers **/FROM**, **/TO** and **/EXCLUDE** have the same function as they do in a transfer specification. A reference to a transfer list (in a trigger specification or a group declaration) is logically equivalent to a reference to a transfer specification.

```
DECLARE GROUP /BLOCK  groupname
   ELEMENT ele_1, ele_2, ...
   ELEMENT ele_a, ele_b, ...
      .
      .
      .
   ELEMENT ele_x, ...
END_LIST
```

The variable block group declaration requires a series of commands. It declares the list **ele_1**, **ele_2**, **ele_a**, … to comprise a variable block group named **groupname**. A reference to **groupname** is logically equivalent to a reference to a variable block name. Name lists may include references to other variable block groups.

(Continued on next page.)

Commands (cont)

```
DECLARE GROUP /TRANSFER_SPEC  groupname
   ELEMENT  ele_1, ele_2,...
   ELEMENT  ele_a, ele_b,...
      .
      .
      .
   ELEMENT  ele_x,...
END_LIST
```

The transfer specification group declaration requires a series of commands. It declares the list **ele_1**, **ele_2**, **ele_a**, … to comprise a transfer specification group named **groupname**. A reference to **groupname** is logically equivalent to a reference to a transfer specification name. Name lists may include references to other other transfer specification groups.

TRIGGER [/UPDATE_TIME=updtim] [/TRIGGER_VARIABLE=trigdb:trigvar[(n)]] -
[/[NO]EDGE] [/[NO]RESET] [/[NO]COMPLEMENT] -
/TRANSFER_SPEC=xfername

The trigger command specifies the trigger conditions for which a transfer will occur for a given transfer specification (**xfername**). Trigger conditions may be specified in terms of either a VMS absolute time or a VMS delta time interval (**updtim**), in terms of a trigger variable (**trigdb:trigvar**), or in terms of both a time and a trigger variable. The trigger must be a CRISP Logical variable or array variable.

If an absolute update time is specified and fields are omitted from the time specification, it is treated as follows: If fields of less significance than the most significant field that is present are omitted, the minimum permissible values for the fields are used by default (that is, JAN for the month, 1 for the date and, 0 for hours, minutes, and seconds). A trigger condition is then considered to exist each time the fields from the most significant field present down to the least significant field match the corresponding fields in the system absolute time.

If an update time is specified but no trigger variable is specified, triggering occurs each time the update time condition is met. Update time granularity is 1 second.

(Continued on next page.)

Commands (cont)

Trigger variables are specified as a database name followed by the name of a CRISP logical variable and separated by a colon. If a trigger variable is specified as a trigger condition, the transfer may be edge-triggered (**/EDGE**) or level-triggered (**/NOEDGE**). Also, it may be specified that the variable be reset (**/RESET**) after each transfer or not (**/NORESET**) and that the complement (**/COMPLEMENT**) of the value of the variable be used rather than the value itself (**/NOCOMPLEMENT**). The defaults are **/NOEDGE**, **/NORESET**, and **/NOCOMPLEMENT**.

If an update time specification is given, the trigger variable will be tested each time the update time condition is met. If no update time specification is given, the trigger variable is tested at each 'end-of-logic' for the logic corresponding to the database containing the trigger variable. If the database containing the trigger variable does not have logic, an update time specification must be given.

The trigger variable is processed as follows.

If edge triggering is specified, transfers will occur on each false-to-true transition of the trigger variable unless the **/COMPLEMENT** qualifier is present, in which case transfers occur on true-to-false transitions. If edge-triggering is not specified, transfers occur as long as the trigger variable is true unless the **/COMPLEMENT** qualifier is present, in which case transfers occurs as long as the trigger variable is false. If reset is specified, the variable will be set false after the transfer has been initiated unless the **/COMPLEMENT** qualifier is present, in which case it will be set true after each transfer has been initiated.

TRANSFER /SOURCE=(dbname_a:var_a1,var_a2) [/FROM=srcspec] -
/DESTINATION=dbname_b:varb1 [/TO=dstspec] -
[/EXCLUDE=dstspec] [/UPDATE_TIME=updtim] -
[/TRIGGER_VARIABLE=trigdb:trigvar] -
/[NO]EDGE /[NO]RESET /[NO]COMPLEMENT

The single-command transfer specification is equivalent to a source variable block declaration, a destination variable block declaration, a transfer specification and a trigger command. In addition to providing a means for specifying everything by means of a single command, it also provides for compatibility with the CLE database coupling source file format (DBTCMD). (Note that, for **TRANSFER** to be fully equivalent to the database coupling functionality, the qualifiers **/FROM=(active,standby)**, **/TO=local**, and **/EDGE** must be present.) The source specification is equivalent to the 'limits' block specification, while, destination specification is equivalent to 'starting variable name' specification. Trigger conditions are the same as with the **TRIGGER** command. The **/FROM**, **/TO**, and **/EXCLUDE** qualifiers work as in the transfer specification.

General Considerations

Forward references to variable block names, transfer specification names, or group names are not permitted. If a reference to a block name, a transfer specification name, or a group name is made before the name is declared, it will be treated as undefined by the compiler.

The same transfer may be triggered by a number of conditions by making reference to the same transfer list in multiple **TRIGGER** commands. For example, transfer at several specific times of day as well as a number of trigger variable transitions.

Both trigger variables and source databases must be resident in the local node. The destination databases may be resident on the local node or in a remote node as long as the database names are unique over the network or an explicit destination nodename is specified.

IDC attempts to transfer data to up to two destination databases each having the same database name. This provides for updating databases in both the active and standby members of a redundant pair. In searching for databases of a specified name, IDC first determines if the database exists on the local system. If so, it is used as one of the destinations and an attempt is made to locate one remote database of the same name. If the database does not reside on the local CPU, IDC attempts to locate two remote databases with that name. In either case, the two databases must reside in a redundant pair of machines, otherwise transfers will be made only to the first database located.

Data is always extracted from source databases synchronously with the logic (if any) associated with each database. For local destination databases, data is also placed in the databases synchronously with logic associated with each database. If a destination database is in a remote node, however, the transfers to that database are not synchronized with the destination logic. The changes will occur when the data message is received regardless of the state of the running logic.

If a block is specified such that the starting and ending variables span declarations for variables of a different data type, the intervening variables not of the same type will not be included in the transfer.

Example

The following is an example of a typical IDC database transfer description file.

```
!-----------------------------------------------------------------------------
! Variable block declarations: Declare blocks of variables, the values of which
! are to be copied from one database to another.
!-----------------------------------------------------------------------------

!---------------------------------------------------------------------
! Declare a block of variables named "BIT_BLOCK_1" starting at "BIT1" up
! to and including "BIT30". Also declare two additional blocks.
!---------------------------------------------------------------------

DECLARE BLOCK /LIMITS=(BIT1,BIT30)     BIT_BLOCK_1
DECLARE BLOCK /LIMITS=(NUM400,NUM450)  NUM_BLOCK_2
DECLARE BLOCK /LIMITS=(FLT55,FLT121)   FLT_BLOCK_3

!---------------------------------------------------------------------
! Declare a block of 200 variables named "NUM_BLOCK_4" starting with "NUM1"
! and including the next 199 variables in the order declared. Also, declare
! a block named "LNG_BLOCK_5".
!---------------------------------------------------------------------

DECLARE BLOCK /START=NUM1 /COUNT=200   NUM_BLOCK_4
DECLARE BLOCK /START=LONG5 /COUNT=50   LNG_BLOCK_5

!---------------------------------------------------------------------
! Declare a block of variables named "GEN_BLOCK_6" by specifying each variable
! explicitly.
!---------------------------------------------------------------------

DECLARE BLOCK /LIST                          GEN_BLOCK_6
    VARIABLE  BIT29, BIT50, NUM10, FLT42
    VARIABLE  NUM100, "NUM_ARRAY1(2:5)"
    VARIABLE  CNT005, TIM20, STRING_20
    VARIABLE  "STRING_ARRAY1(2)", NUM201
END_LIST

!-----------------------------------------------------------------------------
! Variable block group declarations: These declarations group variable blocks
! into larger groups.
!-----------------------------------------------------------------------------

!---------------------------------------------------------------------
! Declare a block group named "BLOCK_GROUP_A" consisting of all of the variables
! in variable blocks "BIT_BLOCK_1", "NUM_BLOCK_2" and "FLT_BLOCK_3".
!---------------------------------------------------------------------

DECLARE GROUP /BLOCK                   BLOCK_GROUP_A
    ELEMENT  BIT_BLOCK_1, NUM_BLOCK_2
    ELEMENT  FLT_BLOCK_3
END_LIST
```

(Continued on next page.)

Example (cont)

```
!-------------------------------------------------------------------------------
! Declare a block group named "BLOCK_GROUP_B" consisting of all of the variables
! in variable blocks "NUM_BLOCK_4" and "LNG_BLOCK_5".
!-------------------------------------------------------------------------------

DECLARE GROUP /BLOCK                    BLOCK_GROUP_B
    ELEMENT  NUM_BLOCK_4
    ELEMENT  LNG_BLOCK_5
END_LIST

!-------------------------------------------------------------------------
! Declare a block group named "BLOCK_GROUP_ALL" consisting of all of the
! variables in variable block groups "BLOCK_GROUP_A" and "BLOCK_GROUP_B"
! plus the variables in variable block "GEN_BLOCK".
!-------------------------------------------------------------------------

DECLARE GROUP /BLOCK                    BLOCK_GROUP_ALL
    ELEMENT  BLOCK_GROUP_A
    ELEMENT  BLOCK_GROUP_B
    ELEMENT  GEN_BLOCK_6
END_LIST

!-------------------------------------------------------------------------
! Transfer specifications: These declarations associate variable blocks or
! variable block groups with source and destination databases.
!-------------------------------------------------------------------------

!-------------------------------------------------------------------------
! Declare a transfer specification to copy data from the variables specified
! in "BLOCK_GROUP_A" in database "DB1" to the variable having the same name
! in database DB2 in both the local node and in its redundant partner. The
! transfer only occurs if the source processor is active. The transfer
! specification is name "TRSPEC_A".
!-------------------------------------------------------------------------

DECLARE TRANSFER_SPECIFICATION          TRSPEC_A -
    /SOURCE=(DATABASE=DB1, BLOCK=BLOCK_GROUP_A) /FROM=ACTIVE -
    /DESTINATION=(DATABASE=DB2) /DUPLICATE_SOURCE

!-------------------------------------------------------------------------
! Declare a transfer specification to copy data from the variables specified
! in "BLOCK_GROUP_B" in database "DB1" to the variable having the same name
! in database "DB3" which resides in each menber of a remote redundant pair.
! The transfer only occurs if the source processor is active. The transfer
! specification is name "TRSPEC_B".
!-------------------------------------------------------------------------

DECLARE TRANSFER_SPECIFICATION          TRSPEC_B -
    /SOURCE=(DATABASE=DB1, BLOCK=BLOCK_GROUP_B) /FROM=ACTIVE -
    /DESTINATION=(DATABASE=DB3) /DUPLICATE_SOURCE -
    /TO=(REMOTE, ACTIVE, STANDBY)
```

(Continued on next page.)

Example (cont)

```
!------------------------------------------------------------------------
! Declare a transfer specification to copy data from all specified source
! variables ("BLOCK_GROUP_ALL") in database "DB1" to the variables having the
! same name in database "DB4". The transfer will only be made if the source
! processor is active and will only be made to the active member of the remote
! redundant pair. The transfer specification is named "TRSPEC_C".
!------------------------------------------------------------------------

DECLARE TRANSFER_SPECIFICATION          TRSPEC_C -
    /SOURCE=(DATABASE=DB1, BLOCK=BLOCK_GROUP_ALL) /FROM=ACTIVE -
    /DESTINATION=(DATABASE=DB3) /DUPLICATE_SOURCE /TO=(REMOTE, ACTIVE)

!-------------------------------------------------------------------------------
! Declare a transfer specification to copy data between explicitly-specified
! source-destination variable pairs. The specified transfer will be made
! if the source processor is active and will be made to all possible local
! and remote destinations. The transfer specification is named "TRSPEC_D".
!-------------------------------------------------------------------------------

DECLARE LIST                            TRSPEC_D -
    /SOURCE_DATABASE=DB7 /FROM=ACTIVE /DESTINATION_DATABASE=DB8
    COPY  NUM1_SRC        NUM1_DST
    COPY  NUM2_SRC        NUM2_DST
    COPY  "FLT10(2:10)"   "FLT10(4:12)"
    COPY  TIM50           TIM50
    COPY  STRING_00       STRING_01
END_LIST

!--------------------------------------
! Declare transfer specification groups.
!--------------------------------------

!------------------------------------------------------------------------
! Declare a group consisting of transfer specifications "TRSPEC_A" and
! "TRSPEC_B". The transfer specification group is named "TRSPEC_GROUP_1".
!------------------------------------------------------------------------

DECLARE GROUP /TRANSFER_SPECIFICATION   TRSPEC_GROUP_1
    ELEMENT  TRSPEC_A, TRSPEC_B
END_LIST

!------------------------------------------------------------------------
! Declare a group consisting of transfer specifications "TRSPEC_C" and
! "TRSPEC_D". The transfer specification group is named "TRSPEC_GROUP_2".
!------------------------------------------------------------------------

DECLARE GROUP /TRANSFER_SPECIFICATION   TRSPEC_GROUP_2
    ELEMENT  TRSPEC_C
    ELEMENT  TRSPEC_D
END_LIST
```

(Continued on next page.)

Example (cont)

```
!-------------------------------------------------------------------------
! Specify trigger conditions: These specifications determine the conditions
! under which the transfer specified by given transfer specifications are to
! occur.
!-------------------------------------------------------------------------

!-----------------------------------------------------------------------
! Make trigger specification so that TRSPEC_A is executed every half hour
! on the half hour and on the hour, while TRSPEC_B is executed every hour
! on the hour only.
!-----------------------------------------------------------------------

TRIGGER /UPDATE_TIME="-- :30:00" /TRANSFER_SPEC=TRSPEC_A
TRIGGER /UPDATE_TIME="-- :00:00" /TRANSFER_SPEC=TRSPEC_B

!-----------------------------------------------------------------------
! Make a trigger specification so that TRSPEC_GROUP_2 is executed every 10
! seconds as long as the variable DB2:bit50 is false.
!-----------------------------------------------------------------------

TRIGGER /UPDATE_TIME="0 00:00:10" /TRANSFER_SPEC=TRSPEC_GROUP_2 -
    /TRIGGER_VARIBLE=DB2:BIT50 /COMPLEMENT

END
```

Notes:

Appendix C
CRT Display File Translation

Description

This section contains.the procedures used to translate CRT files when upgrading from CRISP/16 to CRISP/32 or when converting from the CRISP/32 Basic Workstation to the Color Workstation. The following are included in this appendix.

Section	Description
CRTL	This subsection contains information required to translate CRISP/16 Basic Workstation files to CRISP/32 Basic Workstation files.
CWSXLATE	This subsection contains information required to translate CRISP/32 Basic Workstation files to CRISP/32 Color Workstation files.

Notes:

Description

The CRT Translator (CRTL) program is used to translate CRISP/16 display (CRT) files into CRISP/32 display files.

Format

To execute the CRTL command, enter the following.

RUN CRISP$EXE:CRT_TRANSLATOR

Operation

Both CRISP/16 and CRISP/32 Basic Workstation display files bear the '.CRT' file type.

Multiple CRISP/16 displays are stored in a single file, but CRISP/32 displays are stored in a separate file for each 24-line display or display half.

CRISP/32 display filenames contain the display number preceded by the letter 'D' (i.e., `Dxxxxx.CRT` - where `xxxxx` is the display number without leading zeros). For example, display 1 is named `D1.CRT` and display 243 is named `D243.CRT`. See Figure C-1.

Database ID Description File

Before executing the CRTL program, the user must create a Database ID Description File in the same directory that contains the CRISP/16 display file. This file is a listing of names for each database that is to be translated. Each database name (up to six characters) must be on a separate line. Refer to the following example.

NOTE

If no file is specified, the database names will default to B0, B1, B2, ...etc.

```
<start of file>
TP
CC
PB
TEST1
ABC123
<end of file>
```

The order must be the same as installed on the CRISP/16 system.

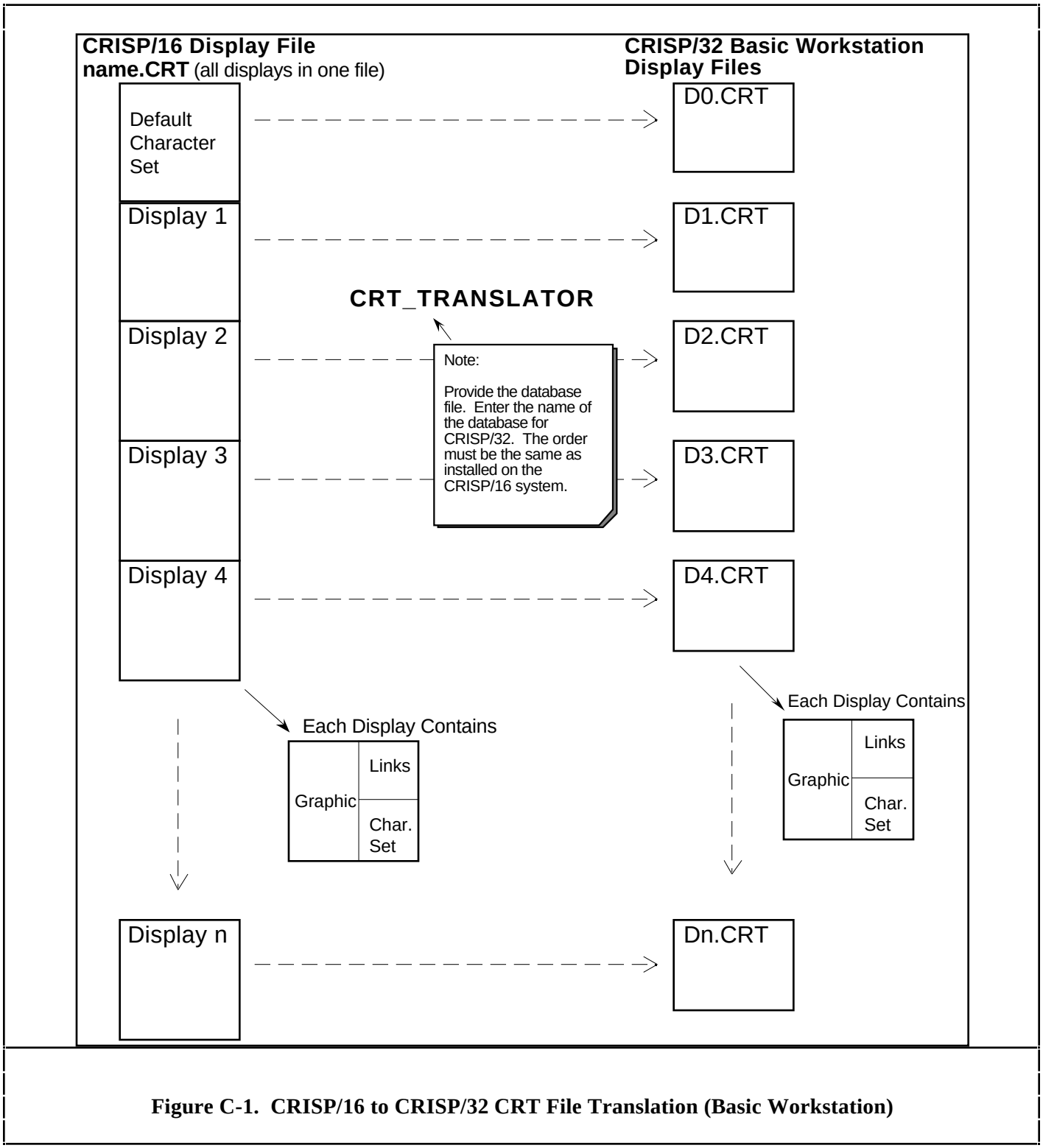


Figure C-1. CRISP/16 to CRISP/32 CRT File Translation (Basic Workstation)

Database ID Description File (cont)

In the previous example, database 0 is named TP, database 1 - CC, database 2 - PR, database 3 - TEST1, database 4 - ABC123, and databases 5 through 7 would be named B5 through B7, respectively. This order must be the same as installed on the CRISP/16 system. Use the Q Display or `LINRUN.CMD` to verify the order on the CRISP/16 system. If the CRISP/16 database is not to be used, use a dummy name. If the CRISP/16 database is to be combined, repeat the name.

CRISP/16 2-Character Ident as installed on running system, verified by Q Display.				Database ID Descriptor File new name for the database on the CRISP/32 system.
BA	Ý	Becomes	Ý	BALARM
BR	Ý	Becomes	Ý	BREC
BC	Ý	Becomes	Ý	BCONTR

The only restriction (other than the 6 character limit) is that the following database names cannot be used (reserved for CRISP Automation use).

- CRISP
- TSKDIR
- DBDIR
- CRTDIR
- LOGDIR
- NETMON
- PRCDIR

CRTL Procedures

Perform the following steps to translate a CRISP/16 display file into CRISP/32 Basic Workstation display files. The CRTL program is invoked from the same directory that contains the CRISP/16 display file.

Step 1. Move the display file

Move the CRISP/16 display file from the PDP to the `disk$USER:[CRISP.DSP]` directory on the VAX using a TK50 tape or other appropriate method.

Step 2. Execute the CRTL program.

Enter the the following to execute the program.

```
RUN CRISP$EXE:CRT_TRANSLATOR
```

Step 3. Enter the CRISP/16 Display filename.

The user is prompted to enter the CRISP/16 display filename.

When entering the CRISP/16 display filename, enter the name only. Do not enter the file type (i.e., .CRT).

(Continued on next page.)

CRTL Procedures (cont)

Step 4. Enter the Database ID description filename.

The user is prompted to enter the description filename (refer to the Database ID Description File Section).

The default file type for the Database ID description file is .DAT.

Step 5. Enter the starting and ending Display file number.

The user is prompted to enter the 'start' display number, and the 'end' display number.

To translate the entire display file, enter zero (0) or just <RETURN> for both the START and END display file number. To translate a series of displays in the file, enter the starting and ending (inclusive) display number. To translate one (1) display, enter that display number for both START and END.

Refer to the following example.

```
$ RUN CRISP$EXE:CRT_TRANSLATOR <ret>
File: name[/switch/switch...] <ret>
Database ID description file: name.ext <ret>
Start: <ret>
End: <ret>
```

NOTE

If a Database ID description file is not specified, the database names will default to B0, B1, B2, etc.

Step 6. Relink all displays that were translated.

All displays must be relinked after translation. Failure to link the displays may result in a fatal error in the Basic Workstation process.

Description

CWSXLATE is used to translate one or more display screens from the Basic Workstation format into display screens that can be displayed on a CRISP/32 Color Workstation or PC Workstation. This program can create a new Color Workstation display file or add Basic Workstation screens to an existing Color Workstation display file. See Figure C-2.

Format

RUN CRISP$EXE:**CWSXLATE**

Operation

After entering the previous command,

- The CRISP/32 file spec. Enter the name of the Basic Workstation display file(s) to be translated. Wildcards are acceptable and the default file type is '.crt'. The following is the recommended specification for input files.

```
INPUT: D%.CRT, D%%.CRT, D%%%.CRT
```

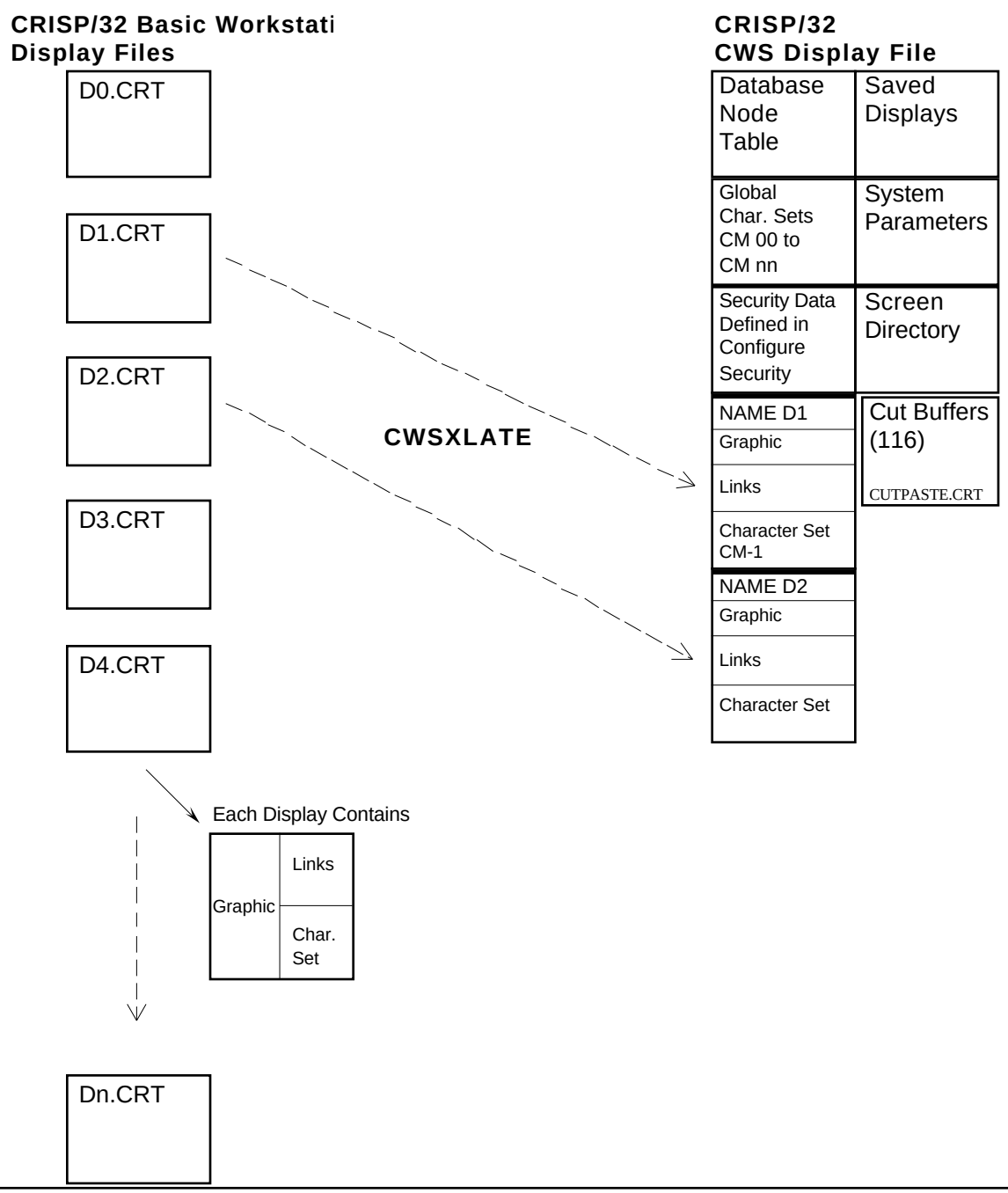


Figure C-2. CRISP/32 Basic Workstation to CWS File Translation

Operation (cont)

The symbol '%' is a place-holding wildcard, and ensures that the input files will be read into the Color Workstation File in *numeric* order rather than *alpha-numeric* order (i.e. 1, 2, 3, …; rather than 1, 10, 2, 3, …).

- The output file spec. This is the name of the Color Workstation display file to be produced. If you do not provide an extension for this file spec, it will assign an extension of '.cws'.

- The number of saved displays. This allows you to establish a maximum number of display 'snapshots' that may be saved using the <GOLD><DO> key sequence at run time. Saved displays are stored in the display file along with display screens. A default is supplied which you may accept. If you have executed this command before, the previous response will be offered as the default. This value is stored as the symbol MAX_DISPLAY and may optionally be entered as a symbol.

- The number of global character sets. This allows you to establish a maximum number of global character sets used on the displays. (Each screen has its own character set—but these are not global character sets. Global character sets are CM00, CM01, CM02, etc.) A default is supplied which you may accept. If you have executed this command before, the previous response will be offered as the default. This value is stored as the symbol MAX_CSET and may optionally be entered as a symbol.

- The number of links per screen. This allows you to establish a maximum number of links allowed on each screen (permissible range is 183 to 600 links per screen). A default is supplied which you may accept. If you have executed this command before, the previous response will be offered as the default. This value is stored as the symbol MAX_LINKS and may optionally be entered as a symbol.

If you have not specified a sufficient number of links, CWSXLATE will issue a message to that effect and continue processing the next input file without processing the links for the errant Basic Workstation display file.

All links are translated except softkey links where keys have been assigned the function of bringing up displays so you should record these links prior to running CWSXLATE.

The translation occurs in two phases: translation and linking. During the first phase, the graphics information in the Basic Workstation files is translated into the Color Workstation display format and a translation table is built which is used during the second phase to resolve the individual key links. This two phase approach is necessary since the screen numbers change during the translation.

'D0.CRT' is considered to be a special file and only the global character set is translated from this file.

Procedures

Perform the following steps to translate CRISP/16 Basic Workstation display files into CRISP/32 Color Workstation display files.

Step 1. Translate from CRISP/16 to CRISP/32

Translate from C/16 to C/32 Basic Workstation by using CRTL.

Step 2. Translate from Basic to Color

Translate from C/32 Basic Workstation to C/32 Color Workstation (refer to CWSXLATE for more detailed information).

Step 3. Relink the file

Relink the new Color Workstation file (Resolve Screens).

Appendix D
Trend Configuration

General

During the CRISP configuration, the user may specify a trend control file specification. This file is used to configure the maximum number of trends and the variables that should always be trended (permanent trends). Note that this does not apply to systems using only the Basic Workstation.

Procedures

To modify the default trend configuration, create a file with the same name as that specified during the CRISP configuration (the default filespec is `CRISP$CFG:USER.TRC`) using the TPU or EDT text editor.

The first line of this file must contain the maximum number of concurrent trends. The default value is 32 trends, and the maximum value is 2048 trends. If a client requests a new trend when the trend process is currently at the maximum number of trends, the oldest non-permanent trend will be replaced by the new trend.

Subsequent lines may contain entries describing the permanently trended variables in the format `db:name,z,a`. The `db:name` portion is the name of the database and the name of the variable to be trended. The `a` indicates average (number of samples to average for trend), and `z` indicates sample interval (sample interval in seconds for trend).

Example

The following is an example of a trend configuration file.

```
40
XYZ:FIRSTVAR,1,1
XYZ:NEXTVAR,10,8
```

This file specifies a maximum of 40 concurrent trends with two permanently trended variables. The first trend will sample `XYZ:FIRSTVAR` at 1-second intervals and store each sample in the trend. The second trend will sample `XYZ:NEXTVAR` every 10 seconds (`Z`) and average eight samples (`A`) into a single value stored in the trend. Thus, each trend point for `NEXTVAR` will represent an 80-second interval.

Notes: