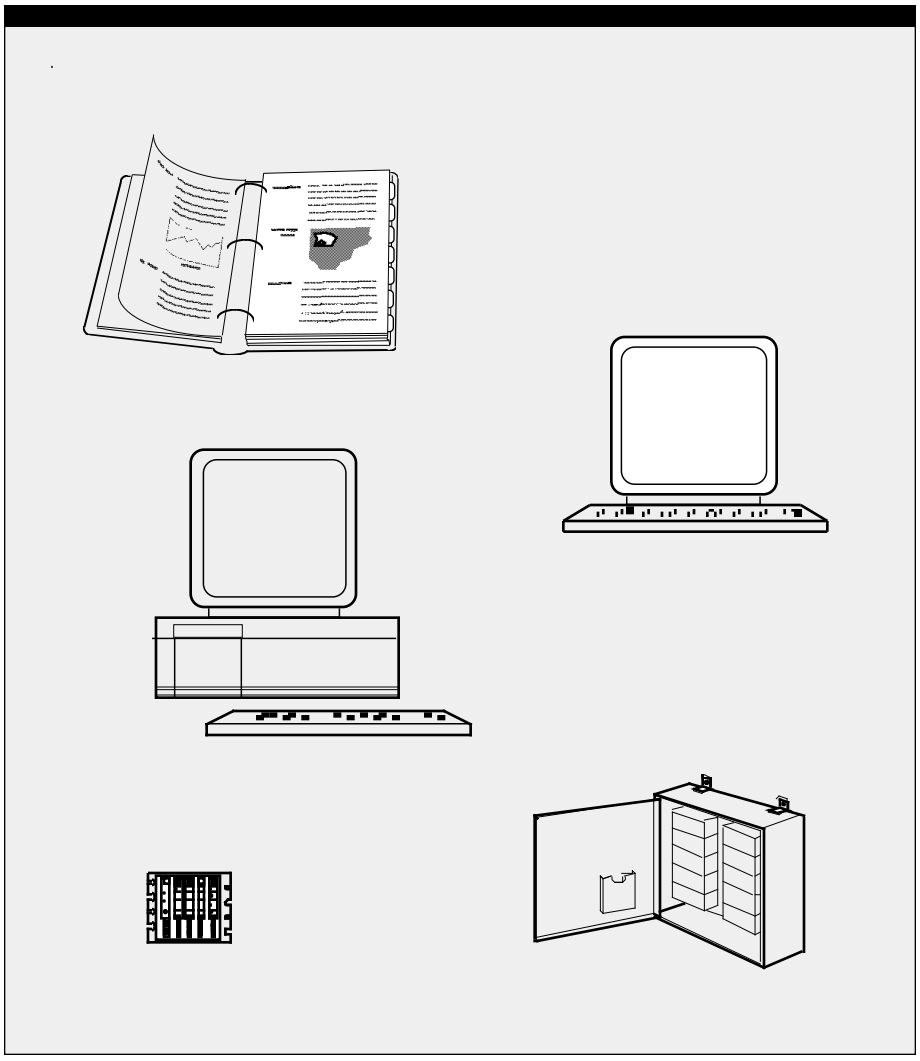
STREAMLINE ™

Software

Reference Documents

SQUARE D COMPANY
GROUPE SCHNEIDER



STREAMLINE ™

SOFTWARE

Reference

Documents

Introduction

General

This manual describes the format and syntax of the text file that describes the relationship between a Host logic and a Classic I/O subsystem.

This manual is divided into the following sections.

Section	Description
CC Interface Overview *(page 3)*	This section describes the functionality, requirements, and operation of the CC Interface.
CCF File *(page 7)*	This section describes the relationship between the Classic I/O system and the CRISP® logic database.
CCF File Compilation *(page 13)*	This section describes compilation and error messages.
Function Calls *(page 17)*	This section describes the three logic calls which form the interface to CC Servers.

Notes:

CC Interface Overview

General

The purpose of the CC Interface (CRISP®/32 to Classic I/O Subsystem Interface) is to provide a method of connecting CRISP Classic I/O to CRISP/32 systems via an Ethernet LAN. This product is complimentary to the locally connected Classic I/O interface. CRISP Classic I/O was originally used with systems based on one of the several models of the PDP-11 computer. With this interface, existing CRISP customers using CRISP/16 or DRM computers may upgrade to VAX computers using CRISP/32.

The CC Interface system consists of the following items:

- A Classic I/O system with either the two-bus or three-bus configuration.
- The CC Server Backplane Assembly which will Accommodate a single or dual CC Server Module.
- One or more CC Servers which are the intelligent devices that perform the actual I/O operations and communications to and from the host CPU(s).
- A single or dual CPU CRISP/32 system.
- CRISP/32 software, Version 3.0-10 or later.
- A single or dual Ethernet network connecting the CC Servers to the host CPU(s).

The CC Server module is based on the microprocessor hardware and software used in the CRISP I/ONYX® Module Server. An Parallel Interface board in the CC Server connects to the ABUS, DBUSI and DBUSO cables of the Classic I/O cabinets.

CC Servers may be installed as single or dual on the CC Server Backplane Assembly. The dual configuration provides the capability of automatic switchover of CC Servers in case of server or communications failure.

The Classic I/O cabinets and cabling are configured in the same way as currently used with a CRISP/16 or DRM system. Cable length limitations and termination requirements are the same.

The CC Interface may be simultaneously operated in the same VAX system with the following CRISP I/O interface products:

- I/ONYX®
- Locally connected Classic I/O
- IDI

CPU Requirements

Any VAX computer configured to run CRISP/32 may be used to connect to Classic I/O using the CC Interface. The CRISP/32 system may be a single or dual computer. A dual computer system may use the Q-bus Arbiter board or the Serial Arbitration unit for switchover control. (The serial unit allows the use of busless computers such as the MicroVAX 3100 Model 90.)

Ethernet Requirements

Any valid Ethernet configuration may be used to connect the VAX host computers running CRISP/32 to the CC Server(s). This may include bridges and repeaters but not routers because the CC Interface system uses IEEE 802.3 messages which are not routable.

Classic I/O Hardware

The Classic I/O hardware beyond the CC Server must conform to the rules for cable length and card loading that have always applied to Classic I/O.

The CC Interface supports the following:

- Digital inputs using CR486 panels with momentary pushbutton latching
- Digital outputs using CR194 or CR587 panels
- Analog inputs using CR786 panels
- Analog outputs using CR587 panels
- Pulse inputs using CR486 panels in the 8-bit or 4-bit counter mode

The CC server interface does NOT support the following:

- Analog inputs using CR188 panels
- Pulse outputs (CR272 cards)
- Motor start staggering
- Smoothing of analog input values

Any of the Classic I/O simulator panels may be in the configuration.

I/O Configuration File

The user specifies the connection between the Classic I/O system and the CRISP/32 system with a configuration file for each CRISP/32 logic that is to perform I/O functions to the Classic I/O system.

Each CRISP/32 logic may perform I/O to up to eight CC Server pairs, each connected to its own Classic I/O system.

Logic Calls

The following CRISP/32 logic calls are provided:

- Read input data from the Classic I/O system
- Write output data to the Classic I/O system
- Read the current status of the CC Interface system

Operation

When the first logic call to read input data is executed, the compiled configuration file for that logic is read. The appropriate data structures are built and the specified CC Servers are downloaded with the required structures needed to perform scanning of the Classic I/O system. On every successive logic call to read input data, each specified CC Server is polled to obtain the current input tables scanned from the Classic I/O system and these tables are transferred to the CRISP/32 logic database.

On each logic call to write output data, the data from the CRISP/32 logic database is transferred to output tables and sent to the specified CC Servers for subsequent copying to the Classic I/O system.

Operation (cont)

If the CC Server Backplane Assembly is configured as a dual unit, an interconnecting cable and hardware logic causes one of the CC Servers to be designated as *active* and one as *idle*. The *active* unit performs the actual scanning of the Classic I/O system and communicates the data to the host VAX computer running CRISP/32. If the *active* CC Server fails or communications is lost to the host computers, the pair will switch over and the previously *idle* CC server will become *active*. As in CRISP/16, a switchover of the device that is scanning the Classic I/O system may cause loss of pulse input totalization or normally closed contact latching.

A monitor program in the host CRISP/32 system continuously polls both of the CC Servers and insures that the *idle* CC Server is always available for use if necessary. This monitor program also reports all changes in the CC Server state to the CRISP$TT: device.

Notes:

General

The CCF file defines the relationship between the Classic I/O system and the CRISP logic database. Each logic that uses the CC Interface requires one of these files. These files are named CRISP$CFG:dbname.CCF. See the section CCF File Compilation later in this manual for more information about CCF file processing.

All error messages associated with the Classic I/O run-time system are sent to the standard CRISP error handler and are date-time stamped.

The general .CCF format rules are as follows:

- The .CCF file is an RMS sequential variable file suitable for editing with TPU or other editor.
- Each line is a logical entity without relationship to any other line.
- Comment lines may be placed anywhere and start with exclamation point.
- There are 7 types of definition lines:

  ESERVER - even-numbered CC Server
  OSERVER - odd-numbered CC Server
  INPUT - digital input points
  OUTPUT - digital output points
  ANAIN - analog input points
  ANAOUT - analog output points
  PLSIN4, PLSIN8 - pulse input points

- A definition line consists of a keyword followed by a space followed by one or more parameters separated by spaces.
- All parameters are position dependent and all must be specified. There are no default values. For timeout values, a specified value of zero will have defaults in the CC Server.
- Pulse input types may be specified as 4-bit counters or 8-bit counters but not both for the same CC Server or Server pair.
- Each ESERVER keyword declares the creation of a new CCDB to perform independent I/O in the specified CC Server. There may be no more than 8 such declarations for a single logic.
- Each I/O type may be specified only once per CCDB. Up to 8 CCDB's may be specified in a single .CCF file.

NOTE

Comments in the dbname.CCF file are preceded by an exclamation point (`! comment`).

Definition Lines

Definition lines in the CCF file have syntax as follows.

> **NOTE**
>
> **For variable declarations, the variable name must be enclosed in quotation marks ("START_VAR").**

ESERVER

Defines a CC Server to be used for this logic and create the CCDB for it.

ESERVER ***addr net auto_enable scan_time timeout_time clear_tmo_cnt***

Where:

addr	The Ethernet address of the desired CC Server in the hexadecimal dashed form 02-00-2B-01-mm-nn. The value "mm" is normally 00, but another value may be set in the CC Server at the factory. The low byte value "nn" is determined by the switches on the CC Server and must be an even value. If there are two CC Servers, they must have even/odd addresses.
net	The network number to use (0 to the highest network number available as determined by the CRISP$NET* logical names).
auto_enable	The AUTO_ENABLE keyword indicates whether the software should automatically enable a pair of CC Servers after being disabled. The choices are AUTO_ENABLE or NOAUTO_ENABLE.
scan_time	The time in milliseconds between successive I/O bus scan cycles (normally set to 100 ms). The maximum value is 65535.
timeout_time	The time in of ms for the host to wait for response from this CC Server pair. The maximum value is 65535.
clear_tmo_cnt	The number of ***scan_time*** intervals that the CC Server will wait for lack of I/O read and write messages before asserting the failsafe value (CLEAR). This value is specified in SCAN_TIME counts. This value is also used for disabling and marking a CCDB as "ready for removal" when no more I/O-type messages have been received by the CC Server. The failsafe value is asserted after SCAN_TIME*CLEAR_TMO_CNT milliseconds (normally a CLEAR_TMO_CNT set to 50 counts; default failsafe is 100ms* 50 or 5 seconds). The maximum value is 65535.

ESERVER (cont)

The following default values are assumed if zero is specified:

scan_time	100 ms
timeout_time	50 ms
clear_tmo_cnt	50 counts.

OSERVER

Defines a second CC Server to define a dual pair.

OSERVER *net*

Where:

net The network number to use (0 to the highest network number available as determined by the CRISP$NET* logical names).

Note that the Ethernet address of this odd CC Server is the above specified even CC Server address with the low order bit set to one.

INPUT

Defines the digital input array.

INPUT *"start_var" start_addr number_of_points*

Where:

"start_var" The first CRISP LOGICAL variable; it receives the first digital input point. This must be the first element of an array or list of CRISP LOGICAL variables of size NUMBER_OF_POINTS.

start_addr The first Classic I/O mux address to be used to read the physical input points. This value is expressed in octal notation. Due to the hardware addressing method, this value must be between 0 and 17740 (octal) and must be a multiple of 40.

number_of_points The number of digital input points to be read. This value must be a multiple of 16 as the digital input hardware can only be read in groups of 16 points.

OUTPUT

Defines the digital output array.

OUTPUT *"start_var" start_addr number_of_points*

Where:

"start_var" The first CRISP LOGICAL variable; it contains the first digital output point. This must be the first element of an array or list of CRISP LOGICAL variables of size NUMBER_OF_POINTS.

OUTPUT (cont)

start_addr The first Classic I/O mux address to be used to write the physical output points. This value is expressed in octal notation. Due to the hardware addressing method, this value must be between 0 and 17740 (octal) and must be a multiple of 40.

number_of_points The number of digital output points to be written. This value must be a multiple of 16 as the digital output hardware can only be written in groups of 16 points.

ANAIN

Defines the analog input array.

ANAIN ***"start_var" start_addr number_of_points***

Where:

"start_var" The first CRISP NUMERIC or LONGWORD variable; it receives the first analog input point. This must be the first element of an array or list of CRISP NUMERIC or LONGWORD variables of size NUMBER_OF_POINTS. Analog values from the hardware are signed 16-bit quantities. These are copied to CRISP NUMERIC variables or sign extended to 32-bit LONGWORD variables.

start_addr The first Classic I/O mux address to be used to read the physical input points. This value is expressed in octal notation. Due to the hardware addressing method, this value must be between 0 and 17740 (octal) and must be a multiple of 40.

number_of_points The number of analog input points to be read.

ANAOUT

Defines the analog output array.

ANAOUT ***"start_var" start_addr number_of_points***

Where:

"start_var" The first CRISP NUMERIC or LONGWORD variable; it contains the first analog output point value. This must be the first element of an array or list of CRISP NUMERIC or LONGWORD variables of size NUMBER_OF_POINTS. Analog values to the hardware are signed 16-bit quantities. The signed 16-bit NUMERIC variables are copied to the hardware, the values of the signed 32-bit LONGWORD variables are truncated to 16-bit values and copied to the hardware.

ANAOUT (cont)

start_addr	The first Classic I/O mux address to be used to write the physical output points. This value is expressed in octal notation. Due to the hardware addressing method, this value must be between 0 and 17740 (octal) and must be a multiple of 40.
number_of_points	The number of analog output points to be written.

PLSIN4

Defines the 4-bit pulse counter input array.

PLSIN4 ***"start_var" start_addr number_of_points***

Where:

"start_var"	The first CRISP NUMERIC or LONGWORD variable; it receives the first 4-bit pulse counter input point. This must be the first element of an array or list of CRISP NUMERIC or LONGWORD variables of size NUMBER_OF_POINTS. These CRISP variables receive the pulse totalizations for their respective input points. For a scan time of 100 ms, the maximum pulse frequency is 150 pps, otherwise pulses will be lost.
start_addr	The first Classic I/O mux address to be used to read the physical input points. This value is expressed in octal notation. Due to the hardware addressing method, this value must be between 0 and 17740 (octal) and must be a multiple of 40.
number_of_points	The number of 4-bit pulse counter input points to be read. This value must be a multiple of 4 as the 4-bit pulse counter hardware can only be read in groups of 4 points.

PLSIN8

Define the 8-bit pulse counter input array.

PLSIN8 ***"start_var" start_addr number_of_points***

Where:

"start_var"	The first CRISP NUMERIC or LONGWORD variable; it receives the first 8-bit pulse counter input point. This must be the first element of an array or list of CRISP NUMERIC or LONGWORD variables of size NUMBER_OF_POINTS. These CRISP variables receive the pulse totalizations for their respective input points. For a scan time of 100 ms, the maximum pulse frequency is 2550 pps, otherwise pulses will be lost.

PLSIN8 (cont)

start_addr	The first Classic I/O mux address to be used to read the physical input points. This value is expressed in octal notation. Due to the hardware addressing method, this value must be between 0 and 17740 (octal) and must be a multiple of 40.
number_of_points	The number of 8-bit pulse counter input points to be read. This value must be a multiple of 2 as the 8-bit pulse counter hardware can only be read in groups of 2 points.

General

In order to be used by the CCIO routines, the ASCII CCF file must first be compiled. This also provides immediate error checking of the configuration in that file. The CCF file must be put in the directory accessed via the CRISP$CFG logical name. The command to compile the CCF file `TEST` would be as shown below.

```
$ CCC TEST
```

where:

CRISP$CFG:TEST.CCF is the ASCII (input) CCF file and CRISP$CFG:TEST.CFDAT is the compiled (output) file.

The compiled configuration file is then automatically read the first time the logic program `TEST` executes the CCREAD call. If a logic calls CCREAD and there is no corresponding .CFDAT file, then the logic will complain once on the CRISP$TT device and continue without any I/O to the Classic I/O subsystem. In this case, all status variables passed to the CCSTAT call will indicate the `No CC Server defined` state.

When a .CFDAT file is read by CCREAD, a global section dbname_CCDB is created and filled from the contents of the file. The appropriate configuration information is later sent to the CC Server to define the hardware configuration and communications parameters as originally declared in the .CCF file.

If CCC detects an error in the CCF file, a message will be displayed indicating the problem and the line where the error is detected. No new .CFDAT file will be created, but the previous file(s) will remain on the disk.

Error Messages

The possible error messages and their meaning is shown below. In the sample messages, *XXX* is the CCF filename, *NN* is the line number within that file, and (*ARG*) is the name of the definition argument in which the error was found. Where applicable, the problem line or additional information will be displayed beneath the error message.

`CCF file` *XXX* `error: CANNOT OPEN INPUT FILE`

CCC was unable to open the indicated CCF file. Only the filename should be specified on the command line; do not enter CRISP$CFG: or .CCF. The file must reside in the CRISP$CFG: directory.

`CCF file` *XXX* `error: CANNOT CREATE OUTPUT FILE`

CCC was unable to create the compiled output file. Check that there is available space on the disk on which CRISP has been installed using the command `SHOW DEVICE CRISP$:.`

`CCF file` *XXX* `error: CANNOT CLOSE OUTPUT FILE`

CCC was unable to close the compiled output file. Check for problems on the disk on which CRISP has been installed.

Error Messages (cont)

`CCF file XXX error at line NN: INVALID DEFINITION COMMAND`

The command keyword in the indicated definition line is not a valid CCC command. Correct the statement and recompile. If the line was intended to be a comment, make sure it begins with an exclamation point (!).

`CCF file XXX error at line NN: MISSING ESERVER SPECIFICATION`

The first non-comment line in a CCF file must be an ESERVER definition. Each additional ESERVER definition line begins the configuration of a new I/O cluster (CCDB).

`CCF file XXX error at line NN: DUPLICATE SECTION`

There can only be one of an OSERVER or I/O type definition per CCDB. Remove the extra line(s) between ESERVER definition lines.

`CCF file XXX error at line NN: MISSING OR INVALID PARAMETER(S)`

The indicated definition line is missing one or more required arguments. This can also occur if a CRISP variable array reference is not enclosed in double quotes ("). Check the definition syntax elsewhere in this manual and make sure that all values are supplied correctly.

`CCF file XXX error at line NN (ARG): MISSING OR INVALID PARAMETER(S)`

The indicated argument of the indicated line has an illegal character such as an alphabetic character in a numeric argument. Check the definition syntax elsewhere in this manual and make sure that a valid value is supplied.

`CCF file XXX error at line NN (ARG): INVALID VALUE`

The indicated argument of the indicated line has an illegal character such as an 8 or 9 in a start address. Check the definition syntax elsewhere in this manual and make sure that a valid value is supplied.

`CCF file XXX error at line NN (ARG): VALUE OUT OF RANGE`

The indicated argument of the indicated line has a value that is outside the allowable range. For string arguments such as the variable name, this means that the string is too long. Check the definition syntax elsewhere in this manual and make sure that a valid value is supplied.

Error Messages (cont)

`CCF file` *`XXX`* `error at line` *`NN`* (*`ARG`*)`: INVALID MULTIPLE`

The indicated definition argument of the indicated line has an value that is not an exact multiple of a required value (i.e., it is not a multiple or 2 or 16 or whatever). Check the definition syntax elsewhere in this manual and make sure that a valid value is supplied.

`CCF file` *`XXX`* `error at line` *`NN`*`: INVALID NETWORK NUMBER`

The network number in the ESERVER or OSERVER definition must be an integer value between 0 and the highest available network number on your system. For systems with dual Ethernets, the network number must be either 0 or 1. You can check the number of available networks with the command `SHOW LOGICAL CRISP$NET*.`

Notes:

General

The two CRISP logic calls CCREAD and CCWRITE cause the logic to perform I/O operations to the specified CC Servers as declared in the .CCF file dbname.CCF. The CRISP logic call CCSTAT returns the communications status of the CC Servers used with this logic.

The CCREAD call performs the input functions (digital inputs, analog input, pulse inputs) and the CCWRITE call performs the output functions (digital output, analog output).

It is recommended that the CCREAD call be placed near the beginning of the logic and the CCWRITE call be placed near the end of the logic.

```
RECALL; CCREAD, conditional, VALID1, VALID2, VALID3, ...
```

VALIDn are CRISP logical variables, one for each CCDB specified in the .CCF file for this logic / database. The VALIDn value is true if a complete set of input data was received from the corresponding CC Server.

```
RECALL; CCWRITE, conditional,

RECALL; CCSTAT, conditional, STAT1, STAT2, STAT3, ...
```

The returned status values are copied from the status word of the corresponding CCDB. If the call specifies more STATn variables than there are CCDBs, then the trailing STATn variables are filled with the "no CC Server defined" flag. If there are not enough STATn words for the number of CCDBs, they are filled to the end of the argument list.

Refer to the following Function Calls for more detail on the call syntax.

CCREAD

CCREAD is called by the logic to do a scan of the CC Servers.

Format

[RE]CALL; CCREAD, `expr`**, *validn*,...**

Arguments

validn	**Usage:**	The ***n*** th valid bit is set to TRUE if the input point read completed without error, Else it is set to FALSE.
	Type:	LOGICAL.
	Access:	Write only.

Operation

On each pass, CCREAD checks initialization, checks the status of the network, verifies that CCDBs are up to date and sequences through the CCDB list sending a message to each CC server and waiting for a response.

If the response indicates that valid data has been received, it sprays that data into the logic database and sets the corresponding VALIDn bit.

Example

```
RECALL; CCREAD , , VALID1, VALID2
```

CCSTAT

CCSTAT is called by the logic to return status of CC Servers.

Format

[RE]CALL; *ccstat*, , *statn*,...

Arguments

statn

Usage: The ***n*** th status longword is copied from the status longword in the nth CCDB structure as defined by the .CCF file.

Type: LONGWORD.

Access: Write only.

Operation

On each call, CCSTAT copies the status longword from the successive CCDBs and returns that longword to logic. The bits in the status longword are encoded as follows:

bit num	mask	use
0	1	Output data valid, one pass of logic has occurred
1	2	At least one physical I/O scan has occurred
2	4	Analog input data is valid
3	8	Digital input data is valid (at least one physical scan since the last CCREAD call)
4	16	Pulse input data is valid
5	32	CC server pair is "disabled"
10	1024	CC server not defined, .CCF file entry missing

Example

```
RECALL; CCSTAT, , SRVSTAT1, SRVSTAT2
```

CCWRITE

CCWRITE is called by the logic to send outputs to the CC Servers.

Format **[RE]CALL; CCWRITE, ,**

Arguments NONE

Operation On each pass, CCWRITE checks initialization, checks the status of the network, verifies that CCDBs are up to date and sequences through the CCDB list sending the output data to each CC server in the list.

Example

```
RECALL; CCWRITE, ,
```

Notes: