KineticSystems Company, LLC
Model AD4Z-NPGZ
2115 PCI Serial Highway
NT Device Driver & API Library
Version 3.4

July 14, 1999

NOTICE

The 2x15 device may have trouble loading in some Pentium based computers.

If you experience trouble loading the KSC device driver:

1. Reboot
2. Enter the CMOS setup program
3. Enable the option asking if you are using a P&P operating system
4. Reboot the machine
5. Load the driver

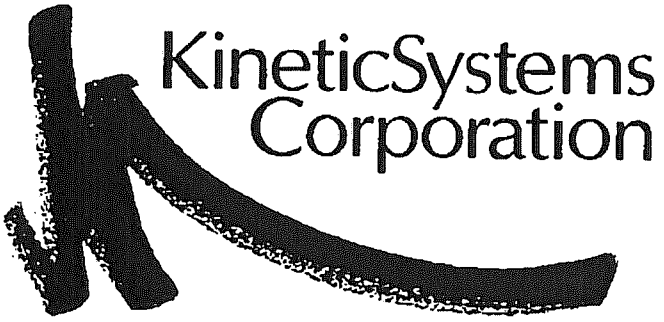
This should take care of any loading problems.

KSC Support Staff

Kinetic Systems Corporation

NT Device Driver &
API Libraries

2115 PCI Serial Highway



900 N. State Street, Lockport, Illinois 60441 (815) 838.0005 (815) 939.4424

NT Device Driver/API

2115 PCI Serial Highway

Document Revision: July 14, 1999

Software Version: 3.4

Operating System: Microsoft NT Version 4.0 (service pack 2)

July 14, 1999

Kinetic Systems Corporation makes no representations that the use of its products in manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of license to make, use, or sell equipment or software in accordance with the description.

Document Name: NT2115_main

1. INTRODUCTION **7**

1.1 2115 PCI CAMAC SERIAL ADAPTER 7
1.2 NT DEVICE DRIVER 7
1.2.1 Clocked Mode of Operation *8*
1.2.2 Multi-buffered Mode of Operation *8*
1.2.3 Demand Messages *9*
1.3 DEVICES 9
1.4 PROGRAMMING SUPPORT 10

2. KSC CAMAC SOFTWARE LIBRARIES **11**

2.1 SUPPORT LIMITATIONS 11
2.2 CAMAC LIST BUILDING 12
2.3 CAMAC LIBRARY ROUTINES 12
2.4 PERFORMANCE CONSIDERATIONS 12
2.4.1 CAMAC Library Call Summary *13*
2.4.2 FORTRAN Language Interface *14*
2.4.3 C Language Interface *14*
2.4.4 Initialization Calls *15*
2.4.5 Single-Action Data Transfer Calls *15*
2.4.6 Block Transfer Calls *15*
2.4.7 Enhanced Serial Highway Operations *16*
2.4.8 Control and Status Calls *17*
2.4.9 Error Status Considerations *18*
2.4.10 Asynchronous Event Handling (LAMS) *19*
2.5 CAMAC ROUTINES 20
2.6 ERROR CODES 20
2.7 LINKER REQUIREMENTS 20
2.8 CAB16 21
2.9 CAB16E 24
2.10 CAB24 27
2.11 CAB24E 30
2.12 CACLOS 33
2.13 CACTRL 35
2.14 CALAM 37
2.15 CAM16 42
2.16 CAM24 45
2.17 CAMSG 48
2.18 CAOPEN 49
2.19 CCSTAT 51
2.20 CXLAM 53

3. CAMAC LIST BUILDING ROUTINES **58**

3.1 CABLK 59
3.2 CAEBLK 62
3.3 CAEXEC 65
3.4 CAEXEW 67
3.5 CAHALT 69
3.6 CAINAF 70
3.7 CAINIT 73
3.8 CANAF 77

4. KSC LIST GENERATION INTERFACE LIBRARY **80**

4.1 LIBRARY USAGE 80
4.2 COMMAND LIST MACROS 87

4.2.1 Command List Types *88*
4.2.2 Start List *88*
4.2.3 End List Macros *89*
4.3 CAMAC COMPATIBILITY 89
4.4 KSC_BDCAST_TRIGGER 92
4.5 KSC_CASE 93
4.6 KSC_DUMP_LIST 95
4.7 KSC_ELSE 97
4.8 KSC_END_LIST 98
4.9 KSC_END_SUBLIST 99
4.10 KSC_ENDCASE 100
4.11 KSC_ENDIF 101
4.12 KSC_FINISH 102
4.13 KSC_GEN_DEMAND 103
4.14 KSC_IF 104
4.15 KSC_INIT_LIST 106
4.16 KSC_LOAD_TEST_VAL 108
4.17 KSC_MARK_LIST 110
4.18 KSC_STORE_FLAG 111
4.19 KSC_SWITCH 112
4.20 KSC_TIME_STAMP 114

5. CAMAC COMMAND LINE UTILITIES 115

5.1 COMMAND SUMMARY 115
5.1.1 CACTRL CAMAC Utility *115*
5.1.2 CAM CAMAC Utility *116*
5.1.3 CCSTAT CAMAC Utility *117*

6. KSC API LIBRARY 119

6.1 API USAGE 119
6.2 API HANDLE 119
6.3 COMMAND LIST GENERATION 120
6.4 PARTITION CONTENTION 120
6.5 TEST_API PROGRAM 120
6.6 KSC_DEMAND_READ 121
6.7 KSC_DISPLAY_PARTITIONS 123
6.8 KSC_ENABLE_DEMAND 125
6.9 KSC_EXEC_CLOCKED_LIST 128
6.10 KSC_EXEC_RLIST 131
6.11 KSC_EXEC_WLIST 133
6.12 KSC_GET_FAILURE 135
6.13 KSC_INIT 137
6.14 KSC_LASTERROR 139
6.15 KSC_LOADGO 140
6.16 KSC_LOAD_CMDLIST 142
6.17 KSC_MBUF_DONE 145
6.18 KSC_PRINT_SYMBOLIC 147
6.19 KSC_READ_CMDLIST 148
6.20 KSC_READ_COUNTERS 151
6.21 KSC_READ_MULTIBUF 153
6.22 KSC_SET_PARTITIONS 155
6.23 KSC_SET_TIMEOUTS 157
6.24 KSC_STOP_MBUF 158
6.25 API AND DRIVER ERRORS 159

7. DEMANDS **160**

7.1 THE DEMAND PROCESS 160
7.2 DEMAND CONFIGURATION FILE 160
7.2.1 Application Registration for Demands *162*
7.2.2 Demand Processing *162*
7.3 USER APPLICATION PROGRAM 164
7.4 DEMAND PROCESS DATAFLOW 164
7.5 DEMAND UTILITIES 166
7.5.1 Program DMDSTS *166*

8. NT KCDRIVER **168**

8.1 DRIVER INTERFACE 168
8.2 NT DEVICES 168
8.3 DEVICEIOCONTROL FUNCTIONS 169
8.3.1 ReadFile and WriteFile Operations *170*
8.3.2 Buffers *170*
8.3.3 KSC_PARTITION- Set the partition table *171*
8.3.4 KSC_TIMEOUT- Set the time out for a partition *171*
8.3.5 KSC_TIMERSET- Set the device internal timer *171*
8.3.6 KSC_2115 RESET- Reset the device *171*
8.3.7 KSC_ID- Return the current release of the driver *171*
8.3.8 KSC_COUNTERS- Return counters for the driver *171*
8.3.9 KSC_RDPARTABLE- Read the current partition table *171*
8.3.10 KSC_ERRREG[1-8]- Read the last status and error information for a partition *171*
8.3.11 KSC_DMDREAD- Read any demands currently in the device adapter *171*
8.3.12 KSC_BUFCOMPLETE- Read any buffer completion flags *172*
8.3.13 KSC_ACKBUFCOMPETE- Acknowledge the processing of the buffer completion *172*
8.3.14 DMA Considerations *172*
8.4 STATUS RETURNS 172
8.5 DEMANDS AND LAMS 173
8.6 MULTIBUFFER CONSIDERATIONS 173
8.7 NT LIMITATIONS 174

9. CAMAC ERROR CODES **175**

10. INSTALLATION **183**

10.1 DIRECTORY STRUCTURE 183
10.1.1 Include Files *183*
10.2 POST INSTALLATION 184
10.3 PROGRAM GROUPS 185

1. Introduction

This document documents the application programming libraries and the NT device driver for the Kinetic System's 2115 PCI CAMAC Serial Highway adapter. Although a cursory overview of the 2115 is provided, the user should reference the 2115 Hardware Manual for more details.

1.1 2115 PCI CAMAC Serial Adapter

The 2115 provides the host application programs the ability to address CAMAC chassis on the CAMAC Serial data highway. The actual accessing of the modules within the crate is via command lists. These command lists must first be loaded into the command list memory of the 2115 and then requested to execute. The 2115 supports command lists containing CAMAC commands (CNAF) plus additional command lists unique to the 2115.

All data transfer operations to or from the 2115 requires a data buffer (except for a command list containing all inline writes). A command list when executed by the 2115 may either supply data or sink data but not both (e.g. a block write and a block read in the same command list will cause an error). Special command list instructions are provided that allow the ability to store data within the list itself. These special instructions can be used for setting up crate registers prior to a read or a write.

The 2115 has the ability to trigger lists by either an internal or external clock. When the 2115 is loaded with a command list the list will begin execution when the clock period expires. Prior to the clock expiring, there must be a data buffer available for the transfer or the clock trigger will be lost.

The 2115 supports multi-buffering using the memory within the host processor. Once set up a multi-buffer interrupt occurs when ever a fixed number of transfers occurs. The number of buffers can be from two to four. The host processor must have all of the buffer mapped for DMA transfer prior to the execution of the list. The list must be first loaded into the 2115 command list memory and then triggered (normally by the internal or external clock).

1.2 NT Device Driver

The 2115 device driver for NT (INTEL Platform only) provides a standard NT interface to the 2115 CAMAC Serial Highway adapter.

The command memory of the 2115 is partitioned into eight partitions (numbered 1 to 8). The size of each of the partition is user selectable via an NT deviceIOControl service call or by using appropriate API call. The driver defaults the size of the all the partitions to start at zero and to contain the complete command list memory. This means, in effect, all partitions are using the same command list memory.

The CAMAC library functions make use of command list number one. The simple single CAMAC commands, such as CAM16 and CAM24 only use about eight memory locations in the 2115. The CAMAC list building routines can generate larger lists which may not fit in partition one. The standard CAMAC list building routines did not provide the ability to specify a partition which may limit the minimum size the end user may want to select for the first partition.

The 2115 device driver supports the following functions of the 2115:

- Command List processing
- Clocked Command list processing
- Multi-buffered Command List processing
- Demand Message processing

It should be noted that command list execution in different modes are not all supported due to limitations of the 2115 hardware. The following are not supported:

- If the current IO request is a command list which is clocked, other lists or requests to the driver are queued until the completion of the clocked list. For example, if the current clocked rate is fifteen seconds, the IO request that is using the clock will monopolize the 2115 until it is triggered and completes. After the list has executed, other lists can be done as long as a new request is presented to the driver before the next clock interval. This management is left up to the user. Therefore, a user may see a delay in the list execution until after the clock has triggered if the process that is doing the clocked request does not wait before posting the next clocked IO request.
- If a multi-buffering command list is executing, it is the only list that can be executing.

Demand interrupts are supported concurrently for all modes.

1.2.1 Clocked Mode of Operation

A new request for the same partition that is not clocked will disable the clock. A change in the frequency of the clock will reset the clock and load the new frequency.

1.2.2 Multi-buffered Mode of Operation

The multi-buffer capabilities of the 2115 allows a user buffer to be partitioned into two to four different equal segments that can be used as a circular buffer. If multi-buffering is enabled, then this function will monopolize the 2115 (the list execution may be clocked). Another device unit (kca02:) is provided to handle the notification of the completion of each buffer segment.

A command list running in multi-buffer mode has the ability to reset the command list memory address to zero or to execute a halt. This ability is a not a normal CAMAC list instruction and is only available using the 2115 specific list building functions. If the command list is reset to zero, the command list continues execution at the first command in the command list memory regardless of the partitioning of the command list memory by the 2115 device driver. The partitioning of the command list memory is managed simply as a table by the 2115 device driver. If a halt instruction is executed, the driver reloads the command list address to the beginning of the particular command list partition.

1.2.3 Demand Messages

The driver will only dequeue the demands from the device when a process posts a read for the demands. The read will complete when one or more demands are present in the FIFO of the 2115. The user will receive from one to the number which can be placed into user's buffer. If there are demands present when the user posts the read, as many as possible are immediately returned to the user. The user's read request will wait only if there are currently no demands in the demand FIFO of the 2115. The maximum number of demands that are dequeued is also limited by the driver as they must be unloaded under interrupt lockout. If the 2115 is reset due to an error, the user will be notified with a status indicating that a reset has occurred. The user should be aware that demands may have been lost.

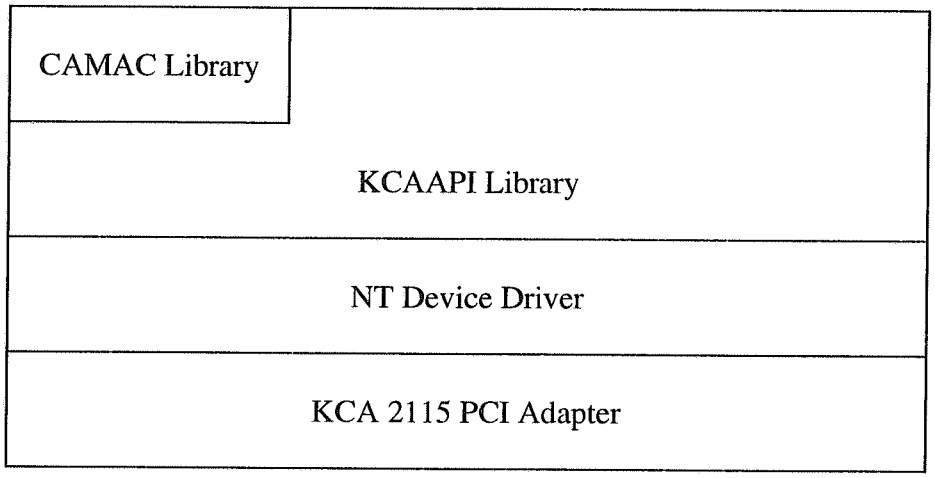
A special demand process server is provided with the driver to help users integrate demands into their application. The Demand Process is required for LAM support in the Kinetic Systems CAMAC library. LAMS (CAMAC Look At Me) is a subset of the demand types handled by the 2115 device driver.

1.3 Devices

There are twenty devices (kca00 to kca19 first controller) created for each 2115 device that is loaded into the system. The user should read the driver chapter for the use of each device that is created. Access to the different devices are through normal NT system service calls.

1.4 Programming Support

An API (Application Programming Interface) is provided with the 2115 CAMAC Serial device driver. The user should use these routines to provide more portability and reduce system dependencies. Kinetic Systems also provides a list building support for the 2115 specific command lists and for the 3972 CAMAC crate controller. The following describes the provided software layers:



2. KSC CAMAC Software Libraries

This chapter explains the use of the existing CAMAC software library available from Kinetic Systems for CAMAC crates. The CAMAC library routines eventually call the KSC API for the 2115. Users who want to use the extended list building capabilities of the 2115 must use the KSC API list building routines.

2.1 Support Limitations

The existing KSC CAMAC routines were originally developed for RSX11M Plus and OpenVMS/VAX environments using older KSC serial highway and bus technologies. These older CAMAC adapters did not support the ability of placing the CAMAC command execution list completely within the adapter as the 2115 does.

The execution of the command list within the adapter results in the inability to return certain status information to the user, No-X, NO-Q, etc., on a per command instance. Every attempt was made to support the existing CAMAC installed base from VMS. The CAMAC library maintained a header which pointed to five different buffers. The CAMAC compatibility library supplied with the 2115 does not use all of these buffers. The main reason for this change is the design of the 2115 and the changes in the NT device driver. Because the command list is actually loaded into the 2115 and then executed by the 2115, the device driver can only report the following information:

1. Number of bytes actually transferred
2. The location in the command list after last command executed
3. The highway status of the last command
4. Controller CSR
5. The method of LAM handling is different than previous releases of the software

The earlier CAMAC serial highway drivers executed a single CAMAC instruction and the results of each instruction was available to the caller. The CAMAC compatibility library uses the three items above to populate the status array, however, the QXE buffer and the Word Count buffer are not supported. The compatibility library maps the 2115 NT error codes to the existing error codes in the CAMAC library.

The CAMAC library required a channel number passed for each call to the CAMAC library routines. **This variable was a sixteen bit word value. This value has been replaced by a thirty-two bit long word value. This variable no longer contains the channel number but is a pointer to an allocated structure.**

All buffers used by the CAMAC library must be long word (32 bits) aligned. Additionally, all buffers that are used for write functions must contain an **additional eight, thirty-two bit long words** at the end. These requirements are a result of the DMA pipelining of the 2115.

Users who wish to use the clocked lists or the multi-buffer support of the 2115 must use the KSC API as there is no equivalent function in the existing CAMAC library.

The routine CXLMST is no longer supported. The ASTs which were triggered by the LAMs are passed with different arguments.

2.2 CAMAC list building

The CAMAC library provided software to build CAMAC lists. This structure has been changed by adding a new longword. If existing users have followed the CAMAC library standards, the size of the header array will be increased with a new compile. This new addition allows for the ability to use either the existing CAMAC list building along with the 2115 specific list building routines found in the KSC API library.

2.3 CAMAC library routines

The High Level Language CAMAC Library Routines supplied in the CAMAC library can be used in conjunction with FORTRAN, C, and other high level languages. These routines may also be used with any language that follows the calling standard. The documentation explains each call in a language independent manner.

The routines in this chapter are simple to understand and use. In general, the specified CAMAC I/O operation will be executed before control is returned to the user process, and each call corresponds to a basic CAMAC I/O operation (either adding to the command list being built or actually performing the operation).

> **Note: Before attempting to issue any CAMAC commands to a Serial Crate the user must insure the crate is on-line. This can be accomplished by a call to CACTRL specifying the 'ONLINE' function.**

2.4 Performance Considerations

Frequently CAMAC applications may involve time critical program segments. By default NT is a time sharing system and as a result can lead to very disappointing performance in some time critical situations. The user may need to lock down all data buffers to improve performance. The loading of the 2115 with the command list can also be done once and left within the 2115 for later execution. The software driver provides the ability to partition the command list memory into eight different partitions. All CAMAC routines use the first partition. More control of the 2115 requires the user use the KSC API calls.

The 2115 also supports multi-buffer memory and triggers which can improve performance of lists that are executed more than once. This is achieved by locking the buffer into memory (the I/O request never completes) and loading the command list only once and providing the I/O completion notification via another I/O device.

2.4.1 CAMAC Library Call Summary

The standard CAMAC library routines provide you with a simple direct set of calls to perform I/O operations to CAMAC. The calls are divided into six groups:

Initialization calls

CAOPEN (chan,device,StatusArray)
CACLOS (chan,StatusArray)

Single-Action Data Transfer Calls

CAM16 (chan,C,N,AF,data,StatusArray)
CAM24 (chan,C,N,AF,data,StatusArray)

Block Transfer Calls

CAB16 (chan,C,NA,F,mode,DataArray,TransCount,StatusArray)
CAB24 (chan,C,NA,F,mode,DataArray,TransCount,StatusArray)

Enhanced Serial Highway Block Transfer Calls

CAB16E (chan,C,N,A,F,mode,DataArray,TransCount,StatusArray)
CAB24E (chan,C,NA,Fmode,DataArray,TransCount,StatusArray)

Status and Control Calls

CACTRL (chan,C,func,StatusArray)
CCSTAT (chan,C,CrateStat,StatusArray)
CAMSG (StatusArray)

LAM or Asynchronous Calls

CXLAM (chan,C,LAMid,Type,Prio,ASTadr,StatusArray)
CALAM
(handle,C,lam_id,lam_type,priority,ast_addr,user_parm,ClrN,ClrA,ClrF,DsbN,DsbA,DsbF,error)

The CAMAC library routines are called either as a subroutine (FORTRAN):

```
CALL camroutine(arguments...),
```

or as an INTEGER*4 function subroutine (Fortran, C):

```
INTEGER*4 camroutine,IERROR
IERROR = camroutine(arguments...),
```

where camroutine is one of the CAMAC routines defined in this manual. In the case of the Function subroutine, the function returns the error status. The error status follows NT conventions and is always odd if the operation was successful. The Function subroutine simplifies the checking of the success or failure of a CAMAC I/O operation, since the call and the test are made in the same line as follows:

```
if((camroutine(args ... ) .and. 1) .eq. 1) THEN "success" ELSE "fail"
```

Examples in this manual use the CALL form, but the Function form can also be used as appropriate.

2.4.2 FORTRAN Language Interface

The CAMAC library can be called from FORTRAN as either a CALL or as a function subroutine reference. When called as a function the library routines return the error status making it easy to perform a CAMAC operation and test the result in a single statement. All parameters are passed by reference.

To simplify your task, the FORTRAN Include File CAUSER.INC is provided. This file defines various parameters which are used with the FORTRAN interface. When this file is added to your FORTRAN program through the Include *CAUSER.INC* statement, it is possible to symbolically reference parameters. For example, the Q-Stop block transfer mode can be represented as *"QSTP"* rather than needing to remember that Mode '0' is the Q-Stop mode. This file also declares all FORTRAN entry points as Integer*4.

2.4.3 C Language Interface

The CAMAC library is shared between all languages and generally follows FORTRAN conventions. When using this interface with C, there are three considerations:

- FORTRAN passes string variables differently than C.
- FORTRAN indexes arrays from one, C indexes from zero.
- FORTRAN passes all variables by reference while C passes arrays by reference and variables by value.

The second item affects pointers returned by the driver and library routines when using the *Advanced CAMAC Routines* which are used to build and execute command lists. All pointers are returned as the FORTRAN index into the indicated arrays. When these pointers are used from C their value must be reduced by one since arrays in C are indexed from zero rather than one as in FORTRAN.

The third item affects how parameters are passed from a C program. Since C passes variables by value and arrays by reference, all variable parameters to the CAMAC library must be preceded by the *address-of* operator (&) as follows:

```
status = caml6 ( &chan, &crate, &a, &f, &f, &data, statusarray );
```

To simplify your task, C Include File KCAUSER.H is provided. This file defines various parameters which are used with the C language interface. When this file is added to your C program through the *#include KSCUSER.H* statement, it is possible to symbolically reference parameters. For example, the Q-Stop block transfer mode can be represented as *"QSTP"* rather than needing to remember that Mode "0" is the Q-Stop mode. This file also declares C language ANSI prototypes for all entry points (See Appendix). The kscuser.h file also contains function prototypes. The Microsoft Visual C++ compiler will not allow the programmer to specify a constant as an address (e.g., &30 for a crate number).

2.4.4 Initialization Calls

The initialization calls provide a mechanism to open the CAMAC device for I/O by a program. Subroutine CAOPEN should be called once for each CAMAC interface (2115) to be accessed by the program and should not be called again until the channel has been closed. Previous users will note that the CAOPEN call no longer returns the channel number, but a thirty bit longword pointer. The pointer points to a KSC API handle which is allocated when the 2115 is opened.

2.4.5 Single-Action Data Transfer Calls

The single-action data transfer calls are simple to use. Each call results in a single CAMAC operation and the appropriate data transfer. Two versions of the single-action routines are provided, CAM16 for 16-bit transfers and CAM24 for full 24-bit transfers. These routines are appropriate for applications where single I/0 operations are required or for short blocks of data where the overhead of program-transfer operations can be tolerated. For large blocks of data, the CAMAC block transfer routines are recommended; they take full advantage of the hardware DMA features and only incur the setup overhead once for the entire operation.

2.4.6 Block Transfer Calls

The CAMAC block transfer calls move blocks of data to or from modules in a single operation using the DMA features of the 2115 Serial Highway Driver. Use these routines for reading or writing blocks of data between Alpha memory and transient digitizers, FIFO modules, display modules, etc., for repeated operations to a single module; and for reading or writing a group of modules in a CAMAC crate. Even for a modest-size data block, these routines have less overhead than the equivalent number of single-action calls because, they transfer the data block at a DMA rate and incur the software setup overhead only once for the entire operation.

2.4.7 Enhanced Serial Highway Operations

The CAMAC Enhanced Serial Highway Block Transfer ESHBT calls move blocks of data to or from CAMAC modules in a single operation using the Enhanced Block Transfer features of the 2115 Serial Highway Driver. Use these routines for reading or writing large blocks of data in applications requiring maximum throughput.

The Enhanced Serial Highway achieves significantly higher throughput by employing a more efficient protocol which transmits the C,N,A,F header once for the entire block rather than once for each dataway operation as illustrated in Figure 1.6. This reduces the number of bytes that must be transmitted over the serial highway for the block to approximately one fourth that needed for the standard protocol. Also, the enhanced mode uses pipelined techniques so that once the block transfer gets started delays in the serial loop do not further degrade throughput.

The Enhanced Serial Highway modes are summarized in Table below.

QSTP (mode=0)	Performs a Q-Stop CAMAC block transfer operation. This mode continues to transfer the block of data until the Transfer Count is exhausted or a NO-Q is received.
QIGN (mode=8)	Performs a Q-Ignore CAMAC block transfer operation. This mode transfers the block of data until the Transfer Count is exhausted. The Q response is ignored.
QRPT (mode=16)	Performs a Q-Repeat CAMAC block transfer operation. This mode transfers the block of data until the Transfer Count is exhausted or a (hardware or software) time-out occurs. Whenever a Q=0 response is received during the block, the Dataway operation is repeated and the data array address pointer is not incremented.
QSCN (mode=24)	Performs a Q-Scan CAMAC block transfer operation. This mode transfers a block of data until the Transfer Count is exhausted or N>23. **A** represents the starting subaddress and **N** represents the initial station number for the scan operation. Note that the ending values of **A** and **N** are not returned.

The first two enhanced modes, EQSTP and EQIGN, are supported in hardware by the 2115 driver and the 3952-ZIE, -ZIF L2 crate controllers. The last two enhanced modes, LSQRPT and LSQIGN, require the 3830 List Sequencer Module option. Note the Enhanced Serial Highway mode is only supported by the Kinetic Systems 2115-series drivers and the 3952-ZIE, -ZIF L2 crate controllers. Enhanced mode operations may be used on serial loops which include other L2 crate controllers in the serial loop as long as all Enhanced mode operations are addressed to crates with the 3952-ZIE or -ZIF crate controllers.

When using the enhanced modes the following considerations apply:

- Since pipelining techniques are used it is generally not possible to use some of the CAMAC error recovery techniques (e.g. the reread register in the L2 crate controller) because the error may not be detected and a response generated by the 2115 until the end of the block or until additional CAMAC operations have occurred due to transactions already in the pipeline. In either case the data in the re-read register will no longer be the data associated with the error condition.

- Not all standard CAMAC Block Modes are available in the Enhanced Mode because of the pipelining. For example, Q-repeat write operation is not feasible because the crate controller may not be able to accept the next data item when it appears in the serial stream.

- Software written for the Enhanced Mode is not transparent on other CAMAC hardware configurations

- Auxiliary Controller Lockout (ACL) is asserted by an addressed crate for the duration of the Enhanced Block Transfer. This blocks other Auxiliary Controllers from performing CAMAC I/O for the duration of the Enhanced Block Transfer. This is only a consideration if an Enhanced Block transfer is addressed to a crate with Auxiliary Crate Controllers.

2.4.8 Control and Status Calls

With the control and status calls, you can Initialize or Clear a crate change the state of crate Inhibit, read crate status, and read the status of the last CAMAC operation.

All of the library calls return a status array. This array contains information on the last call to the CAMAC routines. At the simplest level, it indicates whether the I/O request was successfully performed. If StatusArray(ERR) is odd, it indicates a successful completion of the I/O operation (no errors). Additional information on the success or failure of the I/O request in the status array is indicated the following table. The error codes follow the NT standard for error codes as well. The odd codes were selected as successful status as well such that users migrating from OpenVMS would not need to modify their software if they tested for odd status. Specific return error codes are listed in Appendix. Note that the Subroutine CAMSG can be used to decode the returned error number into ASCII text. Even though the simple CAMAC routines build the lists for the user, the last four items are returned for all CAMAC routines. The symbolic name for the status array element is shown along with its decimal FORTRAN array index.

STATUS ARRAY

1 ERR	Error Status: Contains the returned error code. An odd returns status indicates a successful transfer. Any other value indicates an error or warning.
2 CSR	Control and Status Register: Contains the state of the 2115 Control and Status register. This is copied from the I/O status block. See the I/O status block for the 2115 device driver and the 2115 hardware manual for a more complete description.
3 ERS	Error Status Register: Contains the state of the 2115 Error Status Register. This is copied from the I/O status block. See the I/O status block for the 2115 device driver and the 2115 hardware manual for a more complete description.
4 LCSR	List Status Register: Offset within list of the fault or halt instruction.
5 QXSUM	Q and X Sum- The 2115 returns NO-Q for Q Ignore as well.
6 TC	Bytes not transferred in list.
7	CMA within partition.
8	Byte index into data buffer of last word written (also byte count).
9	Total number of word count errors (not supported on 2115).
10	Total number of QXE buffer errors (not supported on 2115).

2.4.9 Error Status Considerations

There are many places where status information is provided. For compatibility reasons, status information is translated to other existing error codes. However, sometimes this manipulation of the status occludes the real reason of the fault. NT, the 2115 driver, the API library, or the CAMAC library may all return error status.

The 2115 device driver returns its status via the I/O status block to the API level into the structure pointed to by the both the CAMAC library (CAOPEN) and the API library (KSC_INIT). Typically the NT status is returned in the same structure as well. The CAMAC library will attempt to translate this error code to one of the existing CAMAC error codes.

In summary, to acquire the most detailed status information, the user should call the KSC_PRINT_SYMBOLIC passing the address returned from CAOPEN or KSC_INIT. The CAMAC library can be called as function values. While trying to support both the existing CAMAC error codes and still trying to provide as much information as possible, the function value returned and the first word of the status array may be different in the event there is an error status to be returned. A test of odd on either error codes indicates success.

The error status array has been made consistent for all CAMAC library calls. This array is a nine long word array. Some of the entries are not populated for some of the CAMAC calls where a list is not used.

2.4.10 Asynchronous Event Handling (LAMS)

In many real-time applications it is necessary to handle asynchronous events such as events which occur outside the computer and sometimes outside of the CAMAC front-end. For example, an application may require notification when a discrete input from some device changes state, when some amount of data has been stored in a FIFO memory in a module, or when a transient recorder has completed recording a wave form. The LAM or Look-At-Me is the CAMAC mechanism for signaling of asynchronous events. The CAMAC LAM is delivered to the host computer system as a hardware interrupt.

In the computer, the application software must receive notification of the asynchronous event. The operating system mechanism for asynchronous event notification is the Asynchronous Procedure Call (APC). The CXLAM routine is provided to notify the CAMAC driver of the module and crate that will be generating LAMs and the operating system of the address of the routine to be dispatched when the event occurs.

> *The CXLAM routine with LAM- Types 2 and 3 are new with this release of the driver and is the preferred LAM handling mechanism. The CALAM routine with LAM_Types 0 and I continue to be supported for compatibility with previous releases. LAM_Types 2 and 3 are more powered and can handle most modules whose design conform to the IEEE 538 CAMAC Standard. LAM-Types 0 and 1 can only handle LAMs from modules which provide a single control commands to clear LAM and disable LAM. Refer to Appendix E on Driver LAM Handling.*

In developing software employing LAMs some special care needs to be observed:

1. LAMs typically signal asynchronous real-time events which in turn trigger execution of time critical application software.

2. FORTRAN is not re-entrant. Subroutines written in FORTRAN cannot be shared between an APC routine and the main program.

3. FORTRAN I/O is not re-entrant. FORTRAN READ and WRITE statements should not be included in an APC routine.

4. The CAMAC library is written in the C language and is re-entrant so calls to the library may be made from both the APC routine and the main program.

5. The operating system can only handle a limited number of outstanding (undelivered) APCs at any given time. The delivery of the LAMs to the user process is done with the use of the DEMAND process and NT pipes. All LAMs which are expected to be processed must be configured by the Demand process. The Demand Process is responsible for enabling LAM recognition for any CAMAC crates that are to process LAMs. The actual enabling of a particular CAMAC device in a crate is the responsibility of the user.

6. For recurring LAMs, the demand process will queue LAM messages to the user process as long as the pipe is not full.

2.5 CAMAC Routines

The remainder describes each of the supported CAMAC routines. The following are the data types with respect to NT/AXP. Some of the routines use FORTRAN notation due to the previous installed base.

Generic	FORTRAN	Field size
Quadword	Integer*8	64 bits
Longword	Integer*4	32 bits
Word	Integer*2	16 bits
Descriptor	Character	NT Descriptor 64 bit structure

2.6 Error Codes

Error codes are documented in the appendix. Those errors denoted as "KSC_xxxxx" where xxxxx is a symbolic string are from the KSC API library or the NT device driver. The error codes of the form: "ERRnnn" where nnn is an integer are from the CAMAC library. Errors may also be returned by the NT itself. The function return value and the ERR entry of the status array should both be examined when the value of the function is even. Either may be used for success status (an odd status is successful).

2.7 Linker Requirements

All of the CAMAC and KSC API library routines are provided in the library: KCAAPI.LIB object library. This library file was created with Microsoft Visual C++ ™ version 4.2. Any applications that use the CAMAC or KSC API library routines will need to link to this library.

Include files necessary for application building are detailed in the Installation chapter of this document.

2.8 CAB16

Perform a 16 bit block transfer.

FORMAT

CAB16 (handle, C, N, A, F, mode, DataArray, TransCount, StatusArray)

RETURNS

NT usage: status
Type: longword
Mechanism: by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle
Type: long word
Access: write
Mechanism: by reference

Handle returned by CAOPEN.

C
Type: word
Access: readonly
Mechanism: by reference

The number of the CAMAC crate to be selected.

N
Type: word
Access: readonly
Mechanism: by reference

The Station number of the CAMAC module to be selected.

A
Type: word
Access: readonly
Mechanism: by reference

The Subaddress to be selected within the CAMAC module.

F

Type:	word
Access:	readonly
Mechanism:	by reference

The CAMAC Function Code to be performed. If F is in the range of 0 to 7, a CAMAC Read operation is performed. If F is 16 to 23, a Write operation is selected. If F is 8 to 15 or 24 to 31 (a dataless Command operation), the results are unpredictable; this subroutine prevents such an operation and returns an error condition.

mode

Type:	integer*2
Access:	read/write
Mechanism:	by reference

The type of CAMAC block transfer to be performed. The modes are found in Include File CAUSER.INC and KCAUSER.H.

DataArray

Type:	integer*2
Access:	read
Mechanism:	by reference

DataArray is an array containing the data to be read or written by the CAMAC block or transfer operation. For a block CAMAC returned Read operation, DataArray is returned. For a block CAMAC Write operation, the data in DataArray is written to CAMAC. Although the array is made up of 16 bit words, the array must be on a long word boundary. If the operation is a write operation, 32 additional data bytes must be available at the end of the buffer for the pipeline of the 2115. If the user specifies an odd number of transfers, an extra 16 bit transfer is always added.

TransCount

Type:	longword
Access:	read
Mechanism:	by reference

The number of CAMAC transfers (16 bit words) to be performed by the block operation. If the number of transfers is not even for a read (a transfer from the crate to the Alpha), an extra transfer of 16 bits of zeros is added. This is a requirement of the 2115.

StatusArray

Type:	long word
Access:	write
Mechanism:	by reference

Returns status.

DESCRIPTION

Subroutine CAB16 performs block transfers of 16-bit data words to and from CAMAC. Four types of block transfers are possible: Q-Stop, QRepeat, Q-Scan, and Q-Ignore. The type of transfer is specified by the mode argument. The Include File CAUSER.INC or KSCUSER.H defines these Qmode arguments. For this I/O operation, only the lower 16 bits of each 24-bit CAMAC word are transferred between the CAMAC module(s) and the data array in the data array. If the number of transfers is odd, the user must allocate an additional 16 bit word which will be filled in with a zero. This is a requirement for DMA completion on the 2115.

CONDITION VALUES RETURNED

	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.
ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.9 CAB16E

Perform a 16 bit enhanced block transfer.

FORMAT

CAB16E (handle, C, N, A, F, mode, DataArray, TransCount, StatusArray)

RETURNS

NT Usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	long word
Access:	write
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	readonly
Mechanism:	by reference

The number of the CAMAC crate to be selected.

N

Type:	word
Access:	readonly
Mechanism:	by reference

The Station number of the CAMAC module to be selected.

A

Type:	word
Access:	readonly
Mechanism:	by reference

The Subaddress to be selected within the CAMAC module.

F
Type: word
Access: readonly
Mechanism: by reference

The CAMAC Function Code to be performed. If F is in the range of 0 to 7, a CAMAC Read operation is performed. If F is 16 to 23, a Write operation is selected. If F is 8 to 15 or 24 to 31 (a dataless Command operation), the results are unpredictable; this subroutine prevents such an operation and returns an error conciliation.

mode
Type: integer*2
Access: read/write
Mechanism: by reference

The type of CAMAC block transfer to be performed. The modes are found in Include File CAUSER.INC and KCAUSER.H.

DataArray
Type: integer*2
Access: read
Mechanism: by reference

DataArray is an array containing the data to be read or written by the CAMAC block or transfer operation. For a block CAMAC returned Read operation, DataArray is returned. For a block CAMAC Write operation, the data in DataArray is written to CAMAC.

TransCount
Type: longword
Access: read
Mechanism: by reference

The number of CAMAC transfers to be performed by the block operation.

StatusArray
Type: long word
Access: write
Mechanism: by reference

Returns status.

DESCRIPTION

Subroutine CAB16E performs ESHBT of 16-bit data words to and from CAMAC. Four types of block transfers are possible: Q-Stop, Q-Ignore, List Sequencer Q-Repeat (read only), and List Sequencer Q-Ignore. The type of transfer is specified by the mode argument. Note CAMAC control functions (F8-15, F24-31) are not valid for Enhanced Block Operations The Include Files CAUSER.INC and KSCUSER.H defines these arguments. For this I/O operation, only the lower 16 bits of each 24-bit CAMAC word are transferred between the CAMAC module(s) and the data array in the Alpha.

If the operation is a write operation (from the Alpha to the CAMAC crate) then the data array must have an additional 32 bytes of space for the DMA pipelining of the 2115. If the number of transfers is odd, the user must allocate an additional 16 bit word which will be filled in with a zero. This is a requirement for DMA completion on the 2115.

CONDITION VALUES RETURNED

	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.
ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR703	
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.10 CAB24

Perform a 24 bit block transfer.

FORMAT

CAB24 (handle, C, N, A, F, mode, DataArray, TransCount, StatusArray)

RETURNS

NT Usage: status
Type: longword
Mechanism: by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle
Type: long word
Access: write
Mechanism: by reference

Handle returned by CAOPEN.

C
Type: word
Access: readonly
Mechanism: by reference

The number of the CAMAC crate to be selected.

N
Type: word
Access: readonly
Mechanism: by reference

The Station number of the CAMAC module to be selected.

A
Type: word
Access: readonly
Mechanism: by reference

The Subaddress to be selected within the CAMAC module.

F
Type: word
Access: readonly
Mechanism: by reference

The CAMAC Function Code to be performed. If F is in the range of 0 to 7, a CAMAC Read operation is performed. If F is 16 to 23, a Write operation is selected. If F is 8 to 15 or 24 to 31 (a dataless Command operation), the results are unpredictable; this subroutine prevents such an operation and returns an error condition.

mode
Type: word
Access: readonly
Mechanism: by reference

The type of CAMAC block transfer to be performed. The modes are found in Include File CAUSER.INC and KCAUSER.H.

DataArray
Type: longword
Access: read/write
Mechanism: by reference

DataArray is an array containing the data to be read or written by the CAMAC block or transfer operation. For a block CAMAC returned Read operation, DataArray is returned. For a block CAMAC Write operation, the data in DataArray is written to CAMAC.

TransCount
Type: longword
Access: read
Mechanism: by reference

The number of CAMAC transfers to be performed by the block operation.

StatusArray
Type: long word
Access: write
Mechanism: by reference

Returns status.

DESCRIPTION

Subroutine CAB24 performs block transfers of 24-bit data words to and from CAMAC. When using this subroutine, the 24-bit CAMAC word is transferred into the lower 24 bits of a 32-bit word. The upper byte is set to 'zero' for CAMAC Read operations. Four types of block transfers are possible: Q-Stop, Q-Repeat, Q-Scan, and Q-Ignore. The type of transfer is specified by the mode argument. **The arguments for CAB24 are the same as the arguments for CAB16 with the exception of DataArray, which is an array of 32 bit longwords instead of 16 bit words. Also the transfer count variable contains the number of 32 bit longwords to be read or written.**

CONDITION VALUES RETURNED

	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.
ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.11 CAB24E

Perform a 24 bit extended block transfer.

FORMAT

CAB24E (handle, C, N, A, F, mode, DataArray, TransCount, StatusArray)

RETURNS

NT Usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	long word
Access:	write
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	readonly
Mechanism:	by reference

The number of the CAMAC crate to be selected.

N

Type:	word
Access:	readonly
Mechanism:	by reference

The Station number of the CAMAC module to be selected.

A

Type:	word
Access:	readonly
Mechanism:	by reference

The Subaddress to be selected within the CAMAC module.

F

Type:	word
Access:	readonly
Mechanism:	by reference

The CAMAC Function Code to be performed. If F is in the range of 0 to 7, a CAMAC Read operation is performed. If F is 16 to 23, a Write operation is selected. If F is 8 to 15 or 24 to 31 (a dataless Command operation), the results are unpredictable; this subroutine prevents such an operation and returns an error conciliation.

mode

Type:	word
Access:	readonly
Mechanism:	by reference

The type of CAMAC block transfer to be performed. The modes are found in Include File CAUSER.INC and KCAUSER.H.

DataArray

Type:	longword
Access:	readonly
Mechanism:	by reference

DataArray is an array containing the data to be read or written by the CAMAC block or transfer operation. For a block CAMAC returned Read operation, DataArray is returned. For a block CAMAC Write operation, the data in DataArray is written to CAMAC. If the transfer is a write (from the Alpha to crate) the buffer must be sized with an additional 32 bytes for the DMA pipelining requirements of the 2115.

TransCount

Type:	longword
Access:	readonly
Mechanism:	by reference

The number of CAMAC transfers to be performed by the block operation.

StatusArray

Type:	long word
Access:	write
Mechanism:	by reference

Returns status.

DESCRIPTION

CAB24E performs ESHBT of 24-bit data words to and from CAMAC. When using this subroutine, the 24-bit CAMAC word is transferred into the lower 24 bits of a 32-bit word. Four types of block transfers are possible: Q-Stop, Q-Ignore, List Sequencer QRepeat (read only), and List Sequencer Q-Ignore. The type of transfer is specified by the mode argument. Note CAMAC control functions (F8-15, F24-3 1) are not valid for Enhanced Block Operations. The Include Files CAUSER.INC and KSCUSER.H defines these arguments.

The arguments for CAB24E are the same as the arguments for CAB16E with the exception of DataArray, which is 32 bits instead of 16 bits. The transfer count variable contains the number of 32 bit words to be read or written. If the operation is a write operation (from the Alpha to the CAMAC crate) then the data array must have an additional 32 bytes of space for the DMA pipelining of the 2115.

CONDITION VALUES RETURNED

	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.
ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.12 CACLOS

Close current CAMAC session with CAMAC Serial Device driver.

FORMAT

CACLOS (handle, StatusArray)

RETURNS

NT usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	long word
Access:	write
Mechanism:	by reference

Handle returned by CAOPEN.

StatusArray

Type:	long word
Access:	write
Mechanism:	by reference

Returns status.

DESCRIPTION

Subroutine CACLOS de-assigns a channel from the CAMAC Serial Device and deallocates the per process space for the controller. This routine allows the user to call the CAOPEN routine to assign to a different controller. This routine is not needed if the user is exiting as normal NT rundown will do all clean up for the process.

CONDITION VALUES RETURNED

ERR603	The CACLOS error is unknown to the CAMAC software. See KSC_CLOSE for additional errors and also NT error codes that might returned be in the ERR value of the status array.

2.13 CACTRL

Perform CAMAC Crate Control Operations.

FORMAT

CACTRL (handle, C, func, StatusArray)

RETURNS

NT Usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	longword
Access:	read only
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	read only
Mechanism:	by reference

The number of the CAMAC crate to be selected.

func

Type:	word
Access:	read only
Mechanism:	by reference

The function to be executed at Station N=30. The Include File CAUSER.INC and KCAUSER.H contains the control function names as defined parameters.

StatusArray

Type:	longwords
Access:	write only
Mechanism:	by reference

StatusArray contains information from the last operation performed. It is an array of ten 32 bit words.

DESCRIPTION

Subroutine CACTRL performs crate-wide CAMAC control operations. These operations are addressed to the crate controller by the 2115 with a target Station address of N(30).

CONDITION VALUES RETURNED

	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.
ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.14 CALAM

Register for CAMAC LAM notification.

FORMAT

CALAM(handle,C,lam_id,lam_type,priority,apc_addr,user_parm,ClrN,ClrA,ClrF,DsbN,DsbA,DsbF,error)

RETURNS

Usage:	Completion status or error code from CAMAC library.
Type:	longword
Mechanism:	by value

ARGUMENTS

handle

Type:	longword
Access:	read only
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	read only
Mechanism:	by reference

Crate where the LAM will be sourced from.

lam_id

Type:	word
Access:	read only
Mechanism:	by reference

The Station number of the LAM to be booked.

lam_type

Type:	word
Access:	read only
Mechanism:	by reference

Specifies the kind of LAM handling to be performed. The type can be one of the following:

0: This LAM type, when the LAM occurs, will unbook the LAM and issue a disable LAM and clear LAM CAMAC command.

1: This LAM type, when the LAM occurs, will leave the LAM booked and issue a clear LAM CAMAC command.

Priority

Type:	word
Access:	read only
Mechanism:	by reference

Not supported with the 2115.

apc_addr

Type:	longword
Access:	read only
Mechanism:	by reference

The address of the APC routine to be called when the LAM occurs. For information on the arguments passed to the APC routine, see the description.

user_parm

Type:	longword
Access:	read only
Mechanism:	by reference

A user supplied value passed to the AST routine.

ClrN

Type:	word
Access:	read only
Mechanism:	by reference

The station number (N) of the module for the Clear LAM operation.

ClrA
Type: word
Access: read only
Mechanism: by reference

The subaddress (A) to be selected within the module for the Clear LAM operation.

ClrF
Type: word
Access: read only
Mechanism: by reference

The CAMAC Function Code (F) for the Clear LAM Operation.

DsbN
Type: word
Access: read only
Mechanism: by reference

The Station number (N) of the module for the Disable LAM operation.

DsbA
Type: word
Access: read only
Mechanism: by reference

The subaddress (A) to be selected within the module for the Disable LAM operation.

DsbF
Type: word
Access: read only
Mechanism: by reference

The CAMAC Function Code (F) for the Disable LAM operation.

error
Type: longword
Access: write only
Mechanism: by reference

StatusArray contains information from the I/0 operation performed. It is an array of ten 32 bit words.

DESCRIPTION

The routine **CALAM** requests the Demand Process to service LAMs for this process. When the DEMAND (LAM) pipe message is received, an APC routine contained within this routine is called which disables the LAM and calls the user specified APC.

In general, it is the users responsibility to actually enable the LAM in the module. The command to enable the module LAM should be placed in the CCL somewhere after the command after CALAM is called. The Demand Process will enable LAMs for the crate if they are not already enabled from a previous request for the same crate.

If the module LAM is not enabled by the user, the LAM will never occur. If the user enables the module LAM prior to calling CALAM the LAM could occur before the Demand Process has processed the LAM setup request. This situation should be avoided as the results are hardware and timing dependent. A LAM generated by a module in the CAMAC chassis is the slot number minus one.

The information passed to the APC routine is as follows:

dmd_id

Type:	longword
Access:	read only
Mechanism:	by value

The demand id from the 2115 Demand FIFO Register.

handle

Type:	longword
Access:	read only
Mechanism:	reference

The handle argument given by the CAOPEN routine of the parent program to the AST.

chassis

Type:	longword
Access:	read only
Mechanism:	reference

The chassis number for the generated LAM.

user_arg

Type:	longword
Access:	read only
Mechanism:	user specified

This user_parm argument from the CXLAM call.

RETURNS

	See NT error codes that might returned be in the ERR value of the status array.
ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.15 CAM16

Execute single 16 bit CAMAC operation.

FORMAT

CAM16 (handle, C, N, A, F, data, StatusArray)

RETURNS

NT usage:	status
Type:	long word
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	long word
Access:	readonly
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	readonly
Mechanism:	by reference

The number of the CAMAC crate to be selected.

N

Type:	word
Access:	readonly
Mechanism:	by reference

The Station number of the CAMAC module to be selected.

A

Type:	word
Access:	readonly
Mechanism:	by reference

The Subaddress to be selected within the CAMAC module.

F

Type:	word
Access:	readonly
Mechanism:	by reference

The CAMAC Function Code to be performed. If F is in the range of 0 to 7, a CAMAC Read operation is performed. If F is 16 to 23, a Write operation is selected. If F is 8 to 15 or 24 to 31 (a dataless Command operation), the results are unpredictable; this subroutine prevents such an operation and returns an error conciliation.

data

Type:	word
Access:	read/write
Mechanism:	by reference

Data is a sixteen bit word containing the data to be read or written by the CAMAC transfer operation.

StatusArray

Type:	long word
Access:	write
Mechanism:	by reference

Returns status.

DESCRIPTION

Subroutine CAM16 performs a single 16-bit CAMAC data transfer. This subroutine reads or writes 16 bits of data to or from a CAMAC module. For this I/O operation, the lower 16 bits of the 24-bit CAMAC word are transferred between the CAMAC module and the user long word data variable.

CONDITION VALUES RETURNED	
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR714	Illegal CAMAC crate number.
	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.

2.16 CAM24

Execute single 24 bit CAMAC operation.

FORMAT

CAM24 (handle, C, N, A, F, data, StatusArray)

RETURNS

NT usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	long word
Access:	readonly
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	readonly
Mechanism:	by reference

The number of the CAMAC crate to be selected.

N

Type:	word
Access:	readonly
Mechanism:	by reference

The Station number of the CAMAC module to be selected.

A

Type:	word
Access:	readonly
Mechanism:	by reference

The Subaddress to be selected within the CAMAC module.

F

Type:	word
Access:	readonly
Mechanism:	by reference

The CAMAC Function Code to be performed. If F is in the range of 0 to 7, a CAMAC Read operation is performed. If F is 16 to 23, a Write operation is selected. If F is 8 to 15 or 24 to 31 (a dataless Command operation), the results are unpredictable; this subroutine prevents such an operation and returns an error conciliation.

data

Type:	longword
Access:	read/write
Mechanism:	by reference

Data to be read or written by the CAMAC or transfer operation.

StatusArray

Type:	long word
Access:	write
Mechanism:	by reference

Returns status.

DESCRIPTION

Subroutine CAM24 performs a 24-bit CAMAC operation. This subroutine reads or writes 24 bits of data to or from a CAMAC module. This call is similar to CAM16 except that CAM24 performs a 24-bit transfer instead of the 16 bits transferred in CAM16. The arguments for CAM24 are the same as the arguments for CAM16, except the data variable is 32 bits instead of 16 bits. For CAM24, the full 24-bit CAMAC word is stored in the lower bits of the 32-bit integer data variable.

CONDITION VALUES RETURNED

	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0

	or greater than 15.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.17 CAMSG

Translate CAMAC errors.

FORMAT

CAMSG (error)

RETURNS

NT usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

error

Type:	longword
Access:	read only
Mechanism:	by reference

This argument is the error code returned from a previous CAMAC call which is to be evaluated. The error code is returned as the first 32 bit word of the StatusArray used in each subroutine call. In addition, it can also be obtained as the function value returned by any of the subroutines when the call is used as a function subroutine.

DESCRIPTION

Subroutine CAMSG is used to evaluate the error code which is returned from the CAMAC subroutines. This subroutine will print the appropriate error message to standard output associated with the error code. The printed error may be from the device driver, the CAMAC subroutine, or from NT.

CONDITION VALUES RETURNED

2.18 CAOPEN

Open CAMAC session with CAMAC Serial Device driver.

FORMAT

CAOPEN (handle, Device, StatusArray)

RETURNS

NT usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	long word
Access:	write
Mechanism:	by reference

Returned handle for the all CAMAC operations to the indicated controller by this process.

Device

Type:	character string
Access:	readonly
Mechanism:	by reference

Contains the CAMAC Serial device. Legal device names are: “kca00”, "kcb00", "kcc00", "kcd00”.

StatusArray

Type:	long word
Access:	write
Mechanism:	by reference

Returns status.

DESCRIPTION

CAOPEN assigns a channel to a device and initializes the CAMAC library so that CAMAC operations can be performed. This subroutine must be called once at the start of the program before attempting any CAMAC operations. Once the channel has been opened, CAOPEN should not be called again unless the channel is de-assigned by a call to CACLOS.

CAOPEN initializes the handle parameter. This is a pointer to a process and controller specific region that has been allocated for the user. The CAOPEN should be called as part of the process's initialization. The handle should be passed to all of the remaining CAMAC routines. CACLOS can be called to close the channel and release this per process and controller space. If two 2115s are installed in the system, the user should reserve a handle variable for each of the controllers and use each for the operations directed for each 2115.

CONDITION VALUES RETURNED

KSC_BAD_ARG	One or more of the arguments are not read or writeable. See KSC_INIT for additional errors and also NT error codes that might returned be in the ERR value of the status array.

2.19 CCSTAT

Return Crate Controller Status.

FORMAT

CCSTAT (handle, C, CrateStatus, StatusArray)

RETURNS

NT usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	longword
Access:	read only
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	read only
Mechanism:	by reference

The number of the CAMAC crate to be selected.

CrateStatus

Type:	longword
Access:	write only
Mechanism:	by reference

Contains the status returned from the crate controller. It is an array of four-32 bit words.

StatusArray

Type:	longword
Access:	write only
Mechanism:	by reference

Contains information from the last operation performed. It is an array of ten 32 bit words.

DESCRIPTION

Subroutine CCSTAT returns the crate controller status.

CONDITION VALUES RETURNED

	See KSC_LOADGO for additional errors and also NT error codes that might returned be in the ERR value of the status array.
ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

2.20 CXLAM

Register for CAMAC LAM notification.

FORMAT

CXLAM (handle, C, LamID, Type, Prio, APC_Adr, StatusArray)

RETURNS

NT usage:	status
Type:	longword
Mechanism:	by value

Completion status or error code from CAMAC library.

ARGUMENTS

handle

Type:	longword
Access:	read only
Mechanism:	by reference

Handle returned by CAOPEN.

C

Type:	word
Access:	read only
Mechanism:	by reference

Crate where the LAM will be sourced from.

LamID

Type:	word
Access:	read only
Mechanism:	by reference

The Station number of the LAM to be booked.

Type

Type:	word
Access:	read only
Mechanism:	by reference

Specifies the kind of LAM handling to be performed. The type can be one of the following:

2: This LAM type will support a single LAM event.

3: This LAM type, when the LAM occurs, will leave the LAM set up and allow for the reception of subsequent LAM events.

Prio
Type: word
Access: read only
Mechanism: by reference

This parameter is not supported with the 2115.

APC_Adr
Type: longword
Access: read only
Mechanism: by reference

The address of the APC routine to be called when the LAM occurs. For information on the arguments passed to the APC routine, see the description.

StatusArray
Type: longword
Access: write only
Mechanism: by reference

StatusArray contains information from the I/0 operation performed. It is an array of ten 32 bit words.

DESCRIPTION

The routine CXLAM requests the Demand Process to service LAMS for this process. When the Demand (LAM) pipe message is received, an APC routine contained within this routine is called. This internal APC will then call the user specified APC. Use this routine to enable processing of asynchronous events triggered by a CAMAC LAM. The user should also read the Demand Process chapter of this document.

In general, it is the users responsibility to enable the LAM in the module. The command to enable the module LAM should be placed in the CCL somewhere after the command after CALAM is called. The Demand Process will enable LAMs for the crate if they are not already enabled from a previous request for the same crate.

If the module LAM is not enabled by the user, the LAM will never occur. If the user enables the module LAM prior to calling CXLAM, the LAM could occur before the Demand Process has processed the LAM setup request. This situation should be avoided as the results are hardware and timing dependent. A LAM generated by a module in the CAMAC chassis is the slot number minus one.

The information passed to the APC routine is as follows:

dmd_id
Type: longword
Access: read only
Mechanism: by value

The demand id from the 2115 Demand FIFO Register.

handle
Type: longword
Access: read only
Mechanism: reference

The handle argument given by the CAOPEN routine of the parent program to the AST.

chassis
Type: longword
Access: read only
Mechanism: reference

The chassis number for the generated LAM.

user_arg
Type: longword
Access: read only
Mechanism: user specified

This parameter is undefined for CXLAM.

CONDITION VALUES RETURNED

See NT error codes that might returned be in the ERR value of the status array.

ERR141 Data buffer not long word aligned.

ERR601 An invalid channel number was specified. The passed handle is invalid.

ERR701 An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0

	or greater than 15.
ERR703	
ERR704	An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31.
ERR706	An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30.
ERR714	Illegal CAMAC crate number.

3. CAMAC List Building Routines

This chapter describes the routines that are provided that will allow the user to build CAMAC command lists (CCL). The user must call the caINIT routine to initialize the list building functions.

The size of the Header array as documented by previous of releases of CAMAC list building software has increased by one long word. The new header points to another structure allocated for use by the CAMAC 2115 Specific list building routines. The CAMAC List Building Routines can be used in conjunction with the 2115 specific command list generation routines as well.

Some of the files support returning the location of where the data is expected to be placed for the generated list. With the use of some of the CAMAC 2115 list building functions, this algorithm will be invalid based on the conditional execution of the CAMAC 2115 list functions.

All of the CAMAC 2115 List Building Routines are provided in the KCAPI.OLB object library. True ANSI C prototyping has been added to the kscuser.h file. This may require some type casting to compile programs with out errors.

The 2115 requires that block transfers return multiples of thirty-two bit long words. Therefore, if an user does a block transfer of five sixteen bit words, the list building routines will also store an instruction that will return an additional sixteen zero bits to round the transfer up to a long word boundary. All data and command list buffers must be long word aligned even if their data type is a sixteen bit integer.

3.1 caBLK

The routine caBLK adds a command to the CAMAC Control List which when executed will result in a CAMAC block transfer operation.

FORMAT

caBLK(Header,C,N,A,F,mode,DatCnt,DatInd,Error)

RETURNS

Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type:	integer*4
Access:	modify
Mechanism:	by reference

Header array is the array built by caINIT and contains pointers to the CAMAC Control List and Data buffer. Updated to reflect the addition of this command list entry.

C

Type:	integer*2
Access:	read
Mechanism:	by reference

The number of the crate (C) to be selected.

N

Type:	integer*2
Access:	read
Mechanism:	by reference

The station number (N) of the module to be selected.

A

Type:	integer*2
Access:	read
Mechanism:	by reference

The subaddress (A) to be selected within the module.

F

Type:	integer*2
Access:	read
Mechanism:	by reference

The CAMAC Function Code (F) to be performed.

mode

Type:	integer*2
Access:	read
Mechanism:	by reference

The type of single CAMAC operation to be performed. The mode byte specifies the word size (16-bit or 24-bit), transfer type (Q-Stop, Q-Ignore, Q-Repeat, or Q-Scan), and abort condition.

DatCnt

Type:	
Access:	
Mechanism:	

DatInd

Type:	integer*4
Access:	write
Mechanism:	by reference

The argument DatInd is returned with the index into the Data Buffer marking the starting location for the block of data to be read or written by the CAMAC operation. For CAMAC read operations the index can be used to access the data after the CAMAC Control List has been executed. For CAMAC write operations DatInd can be used to move the data to be written into the data buffer before the CAMAC Control List has been executed. As an example, for a write operation the user must load the data into the Data Buffer beginning with the location Data(DatInd) and continuing through Data(DatInd + DatCnt). This must be accomplished prior to executing the CAMAC Control List.

Error

Type:	integer*4
Access:	write
Mechanism:	by reference

The return error code, a return value of one means no error. The return error code is in NT format. The subroutine, if called as a function will return the same value as Error.

DESCRIPTION

The routine caBLK adds a command to the CAMAC Control List which when executed will result in a CAMAC block transfer operation. This command will allocate two elements within the CCL. In addition, space is allocated from the data buffer based on the parameter DatCnt. Note: CAMAC control functions (F8-15, F24-31) are not valid for block operations. The block transfer can be Q-Stop, Q-Repeat, Q-Scan, or Q-Ignore. The type of transfer and whether the transfers are 16-bit or 24-bit are controlled by the mode argument.

RETURNS

ERR702	Invalid mode byte. The mode byte for the Advanced Fortran routines is invalid.
ERR703	A invalid CAMAC block transfer type was found. The legal block transfer types are QSTP, QIGN, QRPT, and QSCN with corresponding values of 0, 8, 16, and 24, respectively.
ERR715	Direction error, the CAMAC Control List should be only READ or WRITE, no both.

3.2 caEBLK

FORMAT

caEBLK(Header,C,N,A,F,mode,DatCnt,DatInd,Error)

RETURNS

Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type: integer*4
Access: modify
Mechanism: by reference

Header array is the array built by caINIT and contains pointers to the CAMAC Control List and Data buffer. Updated to reflect the addition of this command list entry.

C

Type: integer*2
Access: read
Mechanism: by reference

The number of the crate (C) to be selected.

N

Type: integer*2
Access: read
Mechanism: by reference

The station number (N) of the module to be selected.

A

Type: integer*2
Access: read
Mechanism: by reference

The subaddress (A) to be selected within the module.

F

Type:	integer*2
Access:	read
Mechanism:	by reference

The CAMAC Function Code (F) to be performed.

mode

Type:	integer*2
Access:	read
Mechanism:	by reference

The type of single CAMAC operation to be performed. The mode byte specifies the word size (16-bit or 24-bit), transfer type (Q-Stop, Q-Ignore, Q-Repeat, or Q-Scan), and abort condition.

DatCnt

Type:	integer*4
Access:	read
Mechanism:	by reference

The number of 16-bit words to be read or written by the CAMAC block transfer operation. Note that for 24-bit transfer operations that DatCnt must reflect the fact that each 24-bit CAMAC transfer requires two 16-bit words.

DatInd

Type:	integer*4
Access:	read
Mechanism:	by reference

The argument DatInd is returned with the index into the Data Buffer marking the starting location for the block of data to be read or written by the CAMAC operation. For CAMAC read operations the index can be used to access the data after the CAMAC Control List has been executed. For CAMAC write operations DatInd can be used to move the data to be written into the data buffer before the CAMAC Control List has been executed. As an example, for a write operation the user must load the data into the Data Buffer beginning with the location Data(DatInd) and continuing through Data(DatInd + DatCnt). This must be accomplished prior to executing the CAMAC Control List.

Error

Type:	integer*4
Access:	write
Mechanism:	by reference

The return error code, a return value of one means no error. The return error code is in NT format. The subroutine, if called as a function will return the same value as Error.

DESCRIPTION

The routine caEBLK adds a command to the CAMAC Control List which when executed will result in an Enhanced Serial Highway CAMAC block transfer operation. The command will allocate two elements in the CCL. In addition, space is allocated from the data buffer based on the parameter DatCnt. Note CAMAC control functions (F8-15, F24-31) are not valid for Enhanced Block operations. The enhanced CAMAC block transfer supports Q-Stop and Q-Ignore modes with the 3952-ZlE or -ZlF Serial Crate Controllers. For configurations which include the 3830 List Sequencer module, the List Sequencer Q-Repeat, and List Sequencer Q-Ignore modes are also supported. The enhanced serial highway mode transfers data considerably faster than the Standard Serial Highway Block Modes. In using the enhanced mode, the user should be aware that this mode is pipeline and as a result it is not always possible to determine the actual word count when an exception (Q=0, X=0, or Serial Transmission Error) occurs. The type of transfer and whether the transfers are 16-bit or 24-bit are controlled by the mode argument.

RETURNS

ERR702	Invalid mode byte. The mode byte for the Advanced Fortran routines is invalid.
ERR703	An invalid CAMAC block transfer type was found. The legal block transfer types are QSTP, QIGN, QRPT, and QSCN with corresponding values of 0, 8, 16, and 24, respectively.
ERR715	Direction error, the CAMAC Control List should be only READ or WRITE, no both.

3.3 caEXEC

Load and Execute a Command List to the CAMAC driver (without wait).

FORMAT

caEXEC(Header,handle,error_array,event_flag)

RETURNS

Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type: integer*4
Access: modify
Mechanism: by reference

Header array is the array built by caINIT and contains pointers to the CAMAC Control List and Data buffer. Each call to CAEXEC must use an unique copy of the header.

handle

Type: integer*4
Access: read-only
Mechanism: by reference

This is the handle (or channel) returned by the **caOPEN** function call.

error_array

Type: integer*4 array
Access: write
Mechanism: by reference

StatusArray contains information from the I/0 operation performed. It is an array of ten 32-bit words.

event

Type: HANDLE
Access: read-only
Mechanism: by reference

An NT event to be signaled when the list has completed..

DESCRIPTION

This routine executes the list built by the CAMAC list building routines. Control is returned to the user process after queuing the I/O to the driver. The user must check the **event** to determine when the I/O request is complete.

This form of the Execution Routines is useful when the user needs to make use of multiple buffering techniques, or wishes to overlap computation with I/O execution. Note that if this routine is used for multiple buffering that separate data structures (header) must be specified for each outstanding I/O request as part of the synchronization mechanism is maintained within the header.

RETURNS

ERR143	CAMAC Header not initialized. Access Violation to List memory.
ERR144	CAMAC Header not properly initialized.
ERR601	An invalid channel number was specified.

3.4 caEXEW

Load and Execute a Command List and then Wait.

FORMAT

caEXEW(Header,handle,error_array)

RETURNS

Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type:	integer*4
Access:	modify
Mechanism:	by reference

Header array is the array built by caINIT and contains pointers to the CAMAC Control List and Data buffer. Each call to CAEXEC must use an unique copy of the header.

handle

Type:	integer*4
Access:	read-only
Mechanism:	by reference

This is the handle (or channel) returned by the **caOPEN** function call.

error_array

Type:	integer*4 array
Access:	write
Mechanism:	by reference

StatusArray contains information from the I/O operation performed. It is an array of ten 32-bit words.

DESCRIPTION

This routine executes the list built by the CAMAC list building routines. Control is returned to the user process after queuing the I/O to the driver. Control is not returned to the user process until the I/O operation is complete.

RETURNS

ERR143	CAMAC Header not initialized. Access Violation to List memory.
ERR144	
ERR601	An invalid channel number was specified.

3.5 caHALT

The routine caHALT adds a command to the CAMAC Control List which marks the end of the control list.

FORMAT

caHALT(Header,Error)

RETURNS

Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type:	integer*4
Access:	modify
Mechanism:	by reference

Header array is the array built by caINIT and contains pointers to the CAMAC Control List and Data buffer. Updated to reflect the addition of this command list entry.

Error

Type:	integer*4
Access:	write
Mechanism:	by reference

The return error code, a return value of one means no error. The return error code is in NT format. The subroutine, if called as a function will return the same value as Error.

DESCRIPTION

The routine caHALT adds a command to the CAMAC Control List which marks the end of the control list. This command allocates four elements in the CCL. For proper operation of the driver the CCL must have the list properly terminated.

RETURNS

ERR143	CAMAC Header not initialized. Access Violation to List memory.

3.6 caINAF

The routine caINAF adds a command to the CAMAC Control List which when executed will result in a single CAMAC Write transaction.

FORMAT

caINAF(Header,C,N,A,F,mode,InlDat,Error)

RETURNS

Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type: integer*4
Access: modify
Mechanism: by reference

Header array is the array built by caINIT and contains pointers to the CAMAC Control List and Data buffer. Updated to reflect the addition of this command list entry.

C

Type: integer*2
Access: read
Mechanism: by reference

The number of the crate (C) to be selected.

N

Type: integer*2
Access: read
Mechanism: by reference

The station number (N) of the module to be selected.

A

Type: integer*2
Access: read
Mechanism: by reference

The subaddress (A) to be selected within the module.

F

Type:	integer*2
Access:	read
Mechanism:	by reference

The CAMAC Function Code (F) to be performed.

mode

Type:	integer*2
Access:	read
Mechanism:	by reference

The type of single CAMAC operation to be performed. The mode byte specifies the word size (16-bit or 24-bit), transfer type (Q-Stop, Q-Ignore, Q-Repeat, or Q-Scan), and abort condition.

InlDat

Type:	integer*4
Access:	read
Mechanism:	by reference

The 24 bits of data to be written by the CAMAC operation. If the CAMAC operation is a control function the data is ignored.

Error

Type:	integer*4
Access:	write
Mechanism:	by reference

The return error code, a return value of one means no error. The return error code is in NT format. The subroutine, if called as a function will return the same value as Error.

DESCRIPTION

The routine caINAF adds a command to the CAMAC Control List which when executed will result in a single CAMAC Write transaction. This command will allocate two elements within the CCL. The data to be written is specified in the call and is stored in the CCL as part of the Inline Write command. The purpose of this routine is to allow the user to write control information to a module without having to imbed the control data in the data buffer. It is typically used in Read operations where limited control information must be written to the module to select the data to be read. The data for the CAMAC operation is placed directly in the CAMAC Control List by this routine. Only CAMAC function code for control and write operations are allowed by this routine.

RETURNS

ERR143	CAMAC Header not initialized. Access Violation to List memory.

3.7 caINIT

Initialize CAMAC List building header.

FORMAT

caINIT (Header, CCList, LisMax, Data, DatMax, Status, WC, WCMax, QXE, QXEMax, Error)

RETURNS

NT Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type:	integer*4 array of size hedmax
Access:	write
Mechanism:	by reference

The information in the Header consists of pointers to the other data structures, the sizes and lengths of the other data structures, and Header constants. HedMax is a parameter that is declared in the include file CAUSER.INC and KSCUSER.H and specifies the size of the Header array. This variable is the actual name of the list.

CCList

Type:	integer*4
Access:	read-only
Mechanism:	by reference

The longword array that will hold the CAMAC List. The CAMAC Control List should be declared as a long word array with a size of LisMax. The address of the Control List is initialized into the header.

LisMax

Type:	integer*4
Access:	read-only
Mechanism:	by reference

The number of elements available in the Command List array (CCList above). The Command List should be declared as a long word array with a size of LisMax.

The value of LisMax must be declared by the user to be sufficiently large so that the array List(LisMax) can hold the largest CCL that the user plans to generate. The size of the CCL can be estimated from the number of calls to the List Building Routines.

The driver requires an extra four words beyond the end of the CCL to ensure proper list termination. Thus, LisMax must be at least four words longer than the longest CCL you plan to generate.

Data

Type:	integer*2
Access:	read-only
Mechanism:	by reference

This array will hold the Data for all requests in the associated CCL. The Data Array should Array should be declared as an integer*2 word array with a size of DatMax. The address of this array is initialized into the header. The buffer must be on a long word boundary (e.g., address lower two bits must be zero) and must contain 32 extra bytes for DMA pipelining of the 2115 if the databuffer is sources data for the command list.

DatMax

Type:	integer*2
Access:	read-only
Mechanism:	by reference

Size in integer*2 words (size in bytes divided by two) of the Data Array above. This is used to initialize the header. Sixteen should be added for the thirty-two byte DMA pipelining requirement for buffers that source data for the list.

The value of DatMax must be declared by the user to be sufficiently large so that the array Data(DatMax) can hold the data from all requests in the associated CCL including all block transfer requests

Status
Type:
Access:
Mechanism:

Not supported with the 2115, but must be present. A value of zero may be used.

WC
Type:
Access:
Mechanism:

Not supported with the 2115, but must be present. A value of zero may be used.

WCMax
Type:
Access:
Mechanism:

Not supported with the 2115, but must be present. A value of zero may be used.

QXE
Type:
Access:
Mechanism:

Not supported with the 2115, but must be present. A value of zero may be used.

QXEMax
Type:
Access:
Mechanism:

Not supported with the 2115, but must be present. A value of zero may be used.

Error
Type:
Access:
Mechanism:

The return error code, a return value of one means no error. The return error

code is in NT format. The subroutine, if called as a function will return the value stored in the Error argument.

DESCRIPTION

The routine caINIT is used to initialize the Header and the other data structures. It should be called whenever a new CAMAC Control List is to be built. The Header holds the sizes, lengths, and pointers to the other data structures. The Header is a parameter for most of the other subroutine calls. Arguments to caINIT include user declared arrays and array sizes which will be used by the List Building Routines and the CAMAC driver. These arrays must be declared sufficiently large by the user to hold the needed information. For example, the Data Buffer must be large enough to hold the data for all commands in the list, the CCL to hold the CAMAC Control List, etc. Since these data structures are dynamically allocated, the user need not be concerned if the data structures are larger than required. (The only effect is that the program will require more memory than is required by the commands defined in the list.)

CONDITION VALUES RETURNED

ERR141	Data buffer not long word aligned.
ERR142	Control list buffer not long word aligned.

3.8 caNAF

The routine caNAF adds a command to the CAMAC Control List which when executed will result in a single CAMAC transaction.

FORMAT

caNAF(Header,C,N,A,F,mode,DatInd,Error)

RETURNS

Usage:
Type:
Mechanism:

ARGUMENTS

Header

Type:	integer*4
Access:	modify
Mechanism:	by reference

Header array is the array built by caINIT and contains pointers to the CAMAC Control List and Data buffer. Updated to reflect the addition of this command list entry.

C

Type:	integer*2
Access:	read
Mechanism:	by reference

The number of the crate (C) to be selected.

N

Type:	integer*2
Access:	read
Mechanism:	by reference

The station number (N) of the module to be selected.

A

Type:	integer*2
Access:	read
Mechanism:	by reference

The subaddress (A) to be selected within the module.

F

Type:	integer*2
Access:	read
Mechanism:	by reference

The CAMAC Function Code (F) to be performed.

mode

Type:	integer*2
Access:	read
Mechanism:	by reference

The type of single CAMAC operation to be performed. The mode byte specifies the word size (16-bit or 24-bit), transfer type (Q-Stop, Q-Ignore, Q-Repeat, or Q-Scan), and abort condition.

DatInd

Type:	integer*4
Access:	write
Mechanism:	by reference

The argument DatInd is returned with the index into the Data Buffer marking the location within the Data Buffer for the data to be read or written by the CAMAC operation. For CAMAC read operations the index can be used to access the data after the CAMAC Control List has been executed. For CAMAC write operations DatInd can be used to move the data to be written to buffer before the CAMAC Control List has been executed.

Error

Type:	integer*4
Access:	write
Mechanism:	by reference

The return error code, a return value of one means no error. The return error code is in NT format. The subroutine, if called as a function will return the same value as Error.

DESCRIPTION

The routine caNAF adds a command to the CAMAC Control List which when executed will result in a single CAMAC transaction. This command will allocate one element in the CCL. If the CAMAC operation is a Read (F0-F7) or Write (Fl6-23) operation, then space in the data buffer will also be allocated. The parameter DatInd will be returned with a value corresponding to the FORTRAN index into the data buffer Data where the data is to be located Data(DatInd). Note the data buffer Data is the name of the data buffer passed to the caINIT routine. For write operations the user must place the data in the data buffer prior to executing the CAMAC transfer routine. For Read operations, the data read can be retrieved from the data buffer following the execution of a successful CAMAC transfer operation.

RETURNS

ERR143	CAMAC Header not initialized. Access Violation to List memory.
ERR202	An In-Line CAMAC read was specified. Only CAMAC write and control functions can be specified in an In-Line CAMAC Control List command.

4. KSC List Generation Interface Library

4.1 Library Usage

The List Generation Library is implemented as a set of linkable routines in the KSCAPI library. The list building routines are prototyped in the "kscapi.h" file. Additionally, a set up "C" macros are also available to create inline lists.

The list generation routines are provided to help in the creation of lists using a more structured convention. Creating a list involves first allocating memory to store the list and then calling **KSC_init_list**. This routine will return back a pointer to a structure of type ksc_list that will be used by all of the other list generating routines. If the user is building multiple lists, the user must provide storage for each of the lists and call KSC_init_list for each list. The user may build multiple lists concurrently as all information about the current state of each list is maintained by the structure allocated by KSC_init_list. The list must be allocated on a long word boundary.

The user calls the individual functions to "compile" the instruction list into the user provided list memory. Each callable function in the library is usually associated with one particular command instruction. There exists functions that implement standard IF...ELSE...ENDIF and SWITCH...CASE...ENDCASE properties found in most high-level languages. The list generating library keeps track of calculating offsets and inserting the proper commands into the list, making such IF and CASE blocks much easier to develop.

Upon completion of making a list, **KSC_finish** should be called to clean up the list and check for any possible errors in the list. The routine **KSC_dump_list** can be called to display the compiled list to standard output.

A sample program that creates a list follows. This list *does not* perform any real functionality, and is provided merely as an example for list creation. This list also includes some commands that are only valid for the KSC 2962 Grand Interconnect device. **Do not attempt to actually execute the list!**

```
/*
        TEST PROGRAM
        This program will demonstrates the use of the List
Generation functions
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "../include/ksc_genlist.h"

main()
{
  /* Variable defs */

  short   *mem;            /* Our memory buffer */
  struct ksc_list *list;   /* Our list definition structure
*/
  int              size;   /* Our value of how big list is */
  /*
  * Begin here
  */

  mem = malloc(1024);              /* Allocate a 1024 byte
buffer */
  KSC_init_list(mem,1024,&list);

  /*
  * List code begins here
  /

  KSC_bdcast_trigger(list);
  KSC_block_rw(list,ABORT,WS8,DECADR,15,33,READ,INTERNAL,
               0x7F7F7F,0x252525);

  KSC_if(list,EQ,0xFFFFFF,0x353535);
    KSC_execute_msg_dev(list,0x75,0,1,20,50,"A simple text
block");
    KSC_gen_demand(list,200);
  KSC_endif(list);


KSC_inline_rw(list,ABORT,WS8,DECADR,15,33,WRITE,INTERNAL,0x1
3300);

KSC_inline_w(list,ABORT,WS8,DECADR,15,33,INTERNAL,0x22222,0x
535353);
  KSC_if(list,EQ,0xFFFF,0x616161);
    KSC_load_test_val(list, 15,WS16,0x7A7A7A);
    KSC_mark_list(list);
  KSC_else(list);
    KSC_slave_trigger(list, 33, 1,1,0,1,0);
    KSC_if(list,EQ,0xFFFF,0x616161);
```

```
      KSC_load_test_val(list, 15,WS16,0x7A7A7A);
      KSC_mark_list(list);
    KSC_else(list);
      KSC_slave_trigger(list, 33, 1,1,0,1,0);
      KSC_store_flag(list,0x5050);
    KSC_endif(list);
    KSC_store_flag(list,0x5050);
  KSC_endif(list);

  KSC_time_stamp(list);

  KSC_switch(list,0xFAFAFA);
  KSC_case(list,0x101010);
    KSC_load_test_val(list, 15,WS16,0x7A7A7A);
    KSC_mark_list(list);
    KSC_if(list,EQ,0xFFFFFF,0x353535);
      KSC_execute_msg_dev(list,0x75,0,1,20,50,"A simple text
block");
      KSC_gen_demand(list,200);
    KSC_endif(list);

  KSC_case(list,0x202020);
    KSC_load_test_val(list, 15,WS16,0x717171);
    KSC_mark_list(list);

  KSC_case(list,0x303030);
    KSC_load_test_val(list, 15,WS16,0x2b2b2b);
    KSC_mark_list(list);
  KSC_endcase(list);

  KSC_end_list(list);

  /*
  * List code ends here
  */

  KSC_finish(list);
  /*
  * Write the list out in a symbolic fashion (see following
output)
  */
  KSC_dump_list(mem,0,1);  /* Display the built list */
}
```

This code creates the following output list:

```
LOC  DATA CODE
---- ---- ------------------------------------------------------------
-----------------
0000 8041 BRDCST_TRIG
     0000
     0000
     0000
0008 47AE BLK_RW ab:0 ws:3 am:1 chas_adr:0F adr_mod:21 rw:1
int:1
                addr:007F7F7F tr_cnt:00252525
     C021
     7F7F
     007F
     2525
     0025
0014 8084 IF cond:0 mask:00FFFFFF test:00353535
     0000
     FFFF
     00FF
     3535
     0035
     002A
0022 8090   EXEC_MSG_DEV addr:75 term:0 rply:1 time_out:0014
rply_lng:32
                      cmd_lng:14 [A simple text block]
     8075
     0014
     1432
     2041
     6973
     706D
     656C
     7420
     7865
     2074
     6C62
     636F
     006B
003E 8091   RESUME_MSG_DEV
     0000
0042 8102   GEN_DEMAND pattern:C8
     00C8
0046 8083   END_OF_SUBLIST
     0000
          END_IF
004A 478E INL_RW ab:0 ws:3 am:1 chas_adr:0F adr_mod:21 rw:0
int:1
                addr:00013300
     8021
     3300
```

```
     0001
0052 47CE INLN_W ab:0 ws:3 am:1 chas_adr:0F adr_mod:21 rw:0
int:1
                  addr:00022222 data:00535353
     8021
     2222
     0002
     5353
     0053
005E 8085 IF(ELSE) cond:1 mask:0000FFFF test:00616161
     0001
     FFFF
     0000
     6161
     0061
     0014
006C 8082   LD_TEST_VAL add_mod:0F ws:2 addr:007A7A7A
     800F
     7A7A
     007A
0074 8080   MRK_LST_ADR
     0000
0078 8083   END_OF_SUBLIST
     0000
007C 0042 ELSE
007E 8040   ADDR_SLV_TRIG chas_adr:21 TTL: 1 ECL:1 FP:0
list:1 timst:0
     0021
     1101
     0000
0086 8085   IF(ELSE) cond:1 mask:0000FFFF test:00616161
     0001
     FFFF
     0000
     6161
     0061
     0014
0094 8082     LD_TEST_VAL add_mod:0F ws:2 addr:007A7A7A
     800F
     7A7A
     007A
009C 8080     MRK_LST_ADR
     0000
00A0 8083     END_OF_SUBLIST
     0000
00A4 0012   ELSE
00A6 8040     ADDR_SLV_TRIG chas_adr:21 TTL: 1 ECL:1 FP:0
list:1 timst:0
     0021
     1101
     0000
00AE 80F8     STO_FLG flag:5050
     5050
```

```
00B2 8083      END_OF_SUBLIST
     0000
             END_IF
00B6 80F8    STO_FLG flag:5050
     5050
00BA 8083    END_OF_SUBLIST
     0000
           END_IF
00BE 8002  READ_TIME_STAMP
     0000
00C2 8086  SWITCH mask:00FAFAFA
     007E
     FAFA
     00FA
00CA 1010    CASE test_val:00101010
     0010
     004A
00D0 8082    LD_TEST_VAL add_mod:0F ws:2 addr:007A7A7A
     800F
     7A7A
     007A
00D8 8080    MRK_LST_ADR
     0000
00DC 8084    IF cond:0 mask:00FFFFFF test:00353535
     0000
     FFFF
     00FF
     3535
     0035
     002A
00EA 8090      EXEC_MSG_DEV addr:75 term:0 rply:1
time_out:0014 rply_lng:32
                        cmd_lng:14 [A simple text block]
     8075
     0014
     1432
     2041
     6973
     706D
     656C
     7420
     7865
     2074
     6C62
     636F
     006B
0106 8091      RESUME_MSG_DEV
     0000
010A 8102      GEN_DEMAND pattern:C8
     00C8
010E 8083      END_OF_SUBLIST
     0000
            END_IF
```

```
0112 8083    END_OF_SUBLIST
     0000
0116 2020    CASE test_val:00202020
     0020
     0014
011C 8082    LD_TEST_VAL add_mod:0F ws:2 addr:00717171
     800F
     7171
     0071
0124 8080    MRK_LST_ADR
     0000
0128 8083    END_OF_SUBLIST
     0000
012C 3030    CASE test_val:00303030
     0030
     0000
0132 8082    LD_TEST_VAL add_mod:0F ws:2 addr:002B2B2B
     800F
     2B2B
     002B
013A 8080    MRK_LST_ADR
     0000
013E 8083    END_OF_SUBLIST
     0000
          END_CASE
0142 8081 END_OF_LIST
     0000
0146 8000 HALT
     0000
```

4.2 Command List Macros

The command list macros generate in line code that for some of the common functions of crate controller. These macros generate command lists that can be used with the KSC_loadgo routine directly. The structure of the list is as follows:

Length in bytes of the command list
Command list
Data buffer

To use these macros:

1. Include the "cmdlist.h" definition file.
2. Invoke the definition macro: CMD_LIST_TYPES passing the number of "ints" to be allocated for the command list and data. The complete list need not be completely used allowing the user to create a larger common list.
3. Invoke one or more of the list building macros.
4. Terminate the list with one of the end list macros.
5. Call the KSC_loadgo to execute the list on the host adapter, or use the first "int" as the byte count of the command list, and point to the second "int" as the address of the list.

The following code fragment shows a list that would do a read of the Crate Controller status register (F=1, N=30, A=0, C=1) in crate one. The testing of status was omitted to make the program more readable.

```
/*
* Example program to read 3952 status register
*/
CMD_LIST_TYPES(10);  // Space for the command list and data to be
returned
int n=30;  // Slot 30= Crate controller
int c=1;   // Crate
int a=0;  // Sub Address
int f=1;  // Function
int status;
struct KSC_handle *hdl; // Handle for library
/*
   * CAMAC 24 bit write.  Use an inline write instruction
   * to do this.
   * CAMAC inline write
   * ws_= word size (WS24 OR WS16)
   * qm_= Qmode (Q_STOP, Q_IGNORE, Q_REPEAT, Q_SCAN)
   * tm_= Timing mode (N_CAMAC, E_CAMAC, F_CAMAC)
   * ch_= Chassis number
   * sl_= Slot number
   * sa_= CAMAC sub address
   * fn_= CAMAC function
   * dd_= DATA (always a long word)
   *
   * CAMAC_INLINEWRITE(ws_,qm_,tm_,ch_,sl_,sa_,fn_,dd_)
   */

   START_LIST;
   CAMAC_SINGLE(WS24,Q_IGNORE,N_CAMAC,c,n,a,f);
   HALT;
   END_READ_LIST(1,data);  // Space for data to be returned

   status= KSC_init(hdl,0);  // Open device
   status= KSC_loadgo(hdl,
                      cmd_list__,
                      lgo_size__,
                      0);
printf("\nData Returned: %x\n",*fifo__);
```

4.2.1 Command List Types

CMD_LIST_TYPES macro should be included in the user program's local variable initialization section. It is used to allocate an array of "ints" that will contain the command list and optionally, the data. The macro also defines variables that are used by the macros for building the list. The macro takes a single argument which is the number of "ints" to allocate the command list array for. If the user intends to do a write, the buffer needs to contain 8 "ints" for the DMA pipeline of the CAMAC Serial host adapter.

4.2.2 Start List

The START_LIST macro should be used to begin the generation of a new command list. It may be called more than once if the command list storage is being used more than once. The START_LIST macro does not require any arguments.

4.2.3 End List Macros

There are three macros used to end a list being built. Each of these macros populates the first "int" of the command list with the size of the command list. It then initializes the following variables as:

fifo__ = First "int" in command list for data
lgo_size__= Size of total command list plus read or write data

The END_EXEC_LIST will terminate a list where the list that was created only had in line writes.

The END_READ_LIST ends a list where the list will generate data. The user should specify the number of "ints" that the list will generate. It should be noted that the CAMAC Serial devices must provide full 32 bit transfers. If the user generates a list that generates a single 16 bit word, the list will time-out. The user must read a harmless location to acquire the additional 16 bits.

Syntax: END_READ_LIST(number of "ints");

The END_WRITE_LIST ends a list where the list requires data from the host. This macro will automatically pad the buffer with the eight "ints" required for DMA pipelining. This macro requires a single argument, the number of "ints" of data required by the list. The CAMAC Serial host adapters require that transfers occur in 32-bit "ints". The user should ensure that the list consumes all of the data or the list will time-out. If a 16-bit write operation occurs, typical tricks are to write the location twice, thereby consuming a full 32 bits of data or to use an in-line write for writing a single 16-bit data item.

Syntax: END_WRITE_LIST(number of "ints");

4.3 CAMAC Compatibility

The list generation routines are also compatible with CAMAC list building routines included in the library. The MACROS described in the previous section are for standalone functions only. To use CAMAC calls, you will need to do the following:

1. Use the CAMAC calls to initialize the list memory and data structures via caINIT.
2. When using a KSC routine, you can use the sub-element of the CAMAC header structure as the argument for the KSC header **list_base**.

An example of using CAMAC and KSC list routines follows:

```
/*
* File: Test_list.c
*
* This program will test a sample use of the List Generator functions
* for the CAMAC & KSC application library.
*
* WARNING:  This list does not perform any function.  Do not attempt to
* actually execute this list!
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "camac.h"
#include "kscuser.h"
#include "ksc_genlist.h"
#include "cmdlist.h"

main()
{
  /* Variable defs */

  short *mem;  /* Our memory buffer */
  ksc_list *list;  /* Our list definition structure */
  int size; /* Our value of how big list is */

  int status;
  int mode;

  int *data_buffer;
  int *data_index;

  struct s_header header;

  data_buffer = malloc(1024); /* Allocate a 1024 byte buffer for data */
  mem = malloc(1024);         /* Allocate a 1024 byte buffer for command list */

  printf("\n\n********* CAMAC FUNCTION PATCH CALLS ********\n");

  memset(mem,0,1024);  /* Clear memory */

  cainit(&header,mem,&1024,data_buffer,&1024,&0,&0,&0,&0,&status);
  camsg(&status);

  canaf(&header,&5,&3,&0x25,&0x10,&0x1A,&data_index,&status);
  canaf(&header,&5,&3,&0x25,&0x18,&0x0,&data_index,&status);
  cainaf(&header,&5,&3,&0x25,&0x08,&0x0,&0x37FBE,&status);
  cablk(&header,&5,&3,&0x25,&0x10,&0x0,&0x25,&data_index,&status);
  caeblk(&header,&5,&3,&0x25,&0x1C,&0x0,&0x25,&data_index,&status);
  /*
  * Note the following are KSC list routines built into a CAMAC list.
  * Especially note how the header is passed
  */

  KSC_if(header.gihdr,EQ,0xFFFF,0x616161);

  KSC_load_test_val(header.gihdr, 15,WS16,0x7A7A7A);
  KSC_mark_list(header.gihdr);

  KSC_else(header.gihdr);

  KSC_slave_trigger(header.gihdr, 33, 1,1,0,1,0);
  KSC_store_flag(header.gihdr,0x5050);

  KSC_endif(header.gihdr);

  cahalt(&header,&status);

  /*
  * All done.  Display the list
  */

  KSC_dump_list(mem,0,1);
```

The generated list:

```
LOC  DATA CODE
---- ---- ------------------------------------------------------------
------------
0000 0282 CAMAC_SXFR no-x:0 ws:1(W24) qm:0(Stop) fc:24(Ctrl) sa:5
sn:03
                       ti:0(Norm)
     06B8
0004 02C2 CAMAC_SIW no-x:0 ws:1(W24) qm:0(Stop) fc:08(Ctrl) sa:5
sn:03
                      ti:0(Norm) data:00037fbe
     06A8
     7FBE
     0003
000C 02A4 CAMAC_BXFR no-x:0 ws:2(W16) qm:0(Stop) fc:16(Write)
sa:5 sn:03
                       ti:0(Norm) #xfer:37
     06B0
     FFDB
     FFFF
0014 8085 IF(ELSE) cond:1 mask:0000FFFF test:00616161
     0001
     FFFF
     0000
     6161
     0061
     0014
0022 8082   LD_TEST_VAL add_mod:0F ws:2 addr:007A7A7A
     800F
     7A7A
     007A
002A 8080   MRK_LST_ADR
     0000
002E 8083   END_OF_SUBLIST
     0000
0032 0012 ELSE
0034 8040   ADDR_SLV_TRIG chas_adr:21 TTL: 1 ECL:1 FP:0 list:1
timst:0
     0021
     1101
     0000
003C 80F8   STO_FLG flag:5050
     5050
0040 8083   END_OF_SUBLIST
     0000
          END_IF
0044 8000 HALT
     0000
```

4.4 KSC_bdcast_trigger

This adds a Broadcast Trigger instruction to the passed list.

FORMAT

KSC_bdcast_trigger (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

DESCRIPTION

This routine add the Broadcast Trigger instruction to the end of the list defined by *list_base.*

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.5 KSC_case

This is the main clause used in CASE blocks. It will mark the end of the previous CASE option and signal the start of a new one in an instruction list.

FORMAT

KSC_case (list_base, test_val)

RETURNS

NT Usage: status
Type: int
Mechanism: by value

List generation return status.

ARGUMENTS

list_base
Type: structure ksc_list
Access: read only
Mechanism: by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

test_val
Type: int
Access: read only
Mechanism: by value

This is the actual test value that the current CASE is to check for.

DESCRIPTION

This command signals the start of the next segment in a CASE block. A case block has the format, similar to the C programming language.

By using this routine, the previous **KSC_case** instruction list (if any) is terminated with an End of Sublist instruction, and the next CASE is then initialized. All commands following the **KSC_case** command, up to the next **KSC_case** or **KSC_endcase**, will be executed if the test value matches that provided by the CASE *test_val.*

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.6 KSC_dump_list

This routine displays an already built list in a readable format.

FORMAT

KSC_dump_list (mem, size, dump)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

mem

Type:	pointer value
Access:	read only
Mechanism:	by reference

This should be a pointer to the start of the list to be displayed.

size

Type:	int
Access:	read only
Mechanism:	by value

This is set to the maximum size of the buffer, in bytes. The routine will display all bytes up to and including *mem+size* bytes. If *size* is specified as zero, the routine will display all instructions up to a HALT instruction.

dump

Type:	int
Access:	read only
Mechanism:	by value

This is a flag used to set the display format for the instruction list. If set to a value of zero, the display will only show the beginning of each command. If set to any other value, the display will also include all additional words of data for each command.

DESCRIPTION

This routine will display an already built list stored in memory. The list should end with a HALT instruction.

The display will give for each instruction its location (as a byte offset), instruction code, and the actual instruction and data. If the *data* value is set to a non-zero value, you will also receive each additional word of data for the instruction.

IF and CASE blocks will be indented accordingly. Currently, no nesting of CASE blocks is supported, and up to 10 nested IF blocks are supported.

If the routine encounters an invalid opcode, it will be displayed and the routine will continue, attempting to parse the next word as an opcode.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.

4.7 KSC_else

This marks an optional part of an IF block that starts the ELSE portion of the block.

FORMAT

KSC_else (list_base)

RETURNS

NT Usage: status
Type: int
Mechanism: by value

List generation return status.

ARGUMENTS

list_base
Type: structure ksc_list
Access: read only
Mechanism: by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

DESCRIPTION

This routine will modify the current IF block to become and IF(ELSE) conditional block. It will append an End of Sublist instruction to the end of the previous **KSC_if** list. All instructions following the **KSC_else** (and up to a **KSC_endif**) will be executed if the referenced **KSC_if** results in a false test.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.8 KSC_end_list

This will add an EOL (End of List) instruction to the list.

FORMAT

KSC_end_list (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list**.

DESCRIPTION

This routine will insert a End of List (EOL) instruction at the end of the passed list given by *list_base*.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.9 KSC_end_sublist

This will add an End of Sublist instruction to the list.

FORMAT

KSC_end_sublist (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list**.

DESCRIPTION

This routine will insert a End of Sublist instruction at the end of the passed list given by *list_base*. It is normally called by routines which end sublists, such as **KSC_endif**, **KSC_endcase**, **KSC_else**, and **KSC_case**.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.10 KSC_endcase

This marks the end of a CASE block.

FORMAT

KSC_endcase (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

DESCRIPTION

This routine is called to mark the end of a CASE block. It will terminate the last **KSC_case** instruction list with an End of Sublist command, and then modify the CASE block's offset data.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.11 KSC_endif

This marks the end of an IF block.

FORMAT

KSC_endif (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list**.

DESCRIPTION

This routine is called to mark the end of an IF block. It will terminate the last **KSC_if** (or **KSC_else**) instruction list with an End of Sublist command, and then modify the IF block's offset data.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.12 KSC_finish

End the creation of a list and free allocated list building resources.

FORMAT

KSC_finish (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

DESCRIPTION

This routine should be called at the completion of creating a list. It will check to insure that all IF and CASE blocks are properly completed.

A halt instruction is automatically added to the end of the list and the *list_base* memory is then released back to the system. You *can not* use the *list_base* value after calling this routine.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.13 KSC_gen_demand

This places a Generate Demand instruction into the list.

FORMAT

KSC_gen_demand (list_base, pattern)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

pattern

Type:	int
Access:	read only
Mechanism:	by value

Demand pattern value (0-255).

DESCRIPTION

This routine will insert a Generate Demand instruction at the end of the passed list given by *list_base.*

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.14 KSC_if

This routine marks the beginning of an IF block.

FORMAT

KSC_if (list_base, cond_code, mask, test_value)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

cond_code

Type:	int
Access:	read only
Mechanism:	by value

Type of test. The file: "ksc_genlist.h" defines the following condition code symbols:

EQ	Equal to
GT	Greater than
LT	Less than
GE	Greater than or equal to
LE	Less than or equal to
BT	Bitwise test

mask

Type:	int
Access:	read only
Mechanism:	by value

This mask is applied to the tested value before being compared.

test_value
Type: int
Access: read only
Mechanism: by value

This is the value to test the value with.

DESCRIPTION

This routine marks the start of an IF block or an IF(ELSE) block. All of the following instructions (up to a **KSC_else** or **KSC_endif** instruction) will be executed if the test is evaluated as TRUE.

The IF block must be terminated at the end by a **KSC_endif** call.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.15 KSC_init_list

Prepare allocated memory for list generation.

FORMAT

KSC_init_list (mem_base, size, list_base)

RETURNS

NT Usage: status
Type: int
Mechanism: by value

List generation return status.

ARGUMENTS

mem_base
Type: pointer
Access: read only
Mechanism: by value

This is a pointer to the start of memory where you want the list to be built. The memory must already be allocated.

size
Type: int
Access: read only
Mechanism: by value

This is the size, in bytes, of the allocated memory starting at *mem_base*.

list_base
Type: pointer to pointer of type struct ksc_list
Access: write
Mechanism: by reference

This must be a pointer to a pointer value. It will reflect the allocated space given for the list structure ksc_list.

DESCRIPTION

This routine must be called before using any of the other list generating routines. It will allocate a structure of type ksc_list, initialize it, and then return its location back in *list_base*. You will need this value for calls to any of the other list generating functions.

You may work on more than one list at a time. Each list will have its own *list_base* value.

At the end of creating a list, you must call **KSC_finish** to cleanup the list and remove the allocated structure from memory.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_BAD_ARD	Bad arguments passed.
KSC_NOMEM	Not enough memory for allocation of structure.

4.16 KSC_load_test_val

This places a Load Test Value instruction into the list.

FORMAT

KSC_load_test_val (list_base, addr_mod, ws, address)

RETURNS

NT Usage: status
Type: int
Mechanism: by value

List generation return status.

ARGUMENTS

list_base
Type: structure ksc_list
Access: read only
Mechanism: by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

addr_mod
Type: int
Access: read only
Mechanism: by value

Address modifier, a value from 0-127.

ws
Type: int
Access: read only
Mechanism: by value

Word size. Set this to one of WS16, or WS32.

address
Type: int
Access: read only
Mechanism: by value

Address of the test value.

DESCRIPTION

This routine will insert a Load Test Value instruction at the end of the passed list given by *list_base*.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.17 KSC_mark_list

This places a Mark List Address instruction into the list.

FORMAT

KSC_mark_list (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

DESCRIPTION

This routine will insert a Mark List Address instruction at the end of the passed list given by *list_base.*

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.18 KSC_store_flag

This places a Store Flag instruction into the list.

FORMAT

KSC_store_flag (list_base, flag_word)

RETURNS

NT Usage: status
Type: int
Mechanism: by value

List generation return status.

ARGUMENTS

list_base
Type: structure ksc_list
Access: read only
Mechanism: by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list**.

flag_word
Type: int
Access: read only
Mechanism: by value

Flag word value to store.

DESCRIPTION

This routine will insert a Store Flag instruction at the end of the passed list given by *list_base*.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.19 KSC_switch

This marks the beginning of a CASE block.

FORMAT

KSC_switch (list_base, mask)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

mask

Type:	int
Access:	read only
Mechanism:	by value

This is a mask value that is applied to the value tested before it is compared in a CASE condition.

DESCRIPTION

This routine marks the beginning of a CASE block. It is normally followed by successive calls to **KSC_case** which define blocks of instructions to be performed if the case matches the tested value.

The CASE block must be terminated with a call to **KSC_endcase**.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

4.20 KSC_time_stamp

This places a Time Stamp instruction into the list.

FORMAT

KSC_time_stamp (list_base)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by value

List generation return status.

ARGUMENTS

list_base

Type:	structure ksc_list
Access:	read only
Mechanism:	by reference

Used by all of the List Generation routines. It is created by calling **KSC_init_list.**

DESCRIPTION

This routine will insert a Time Stamp instruction at the end of the passed list given by *list_base.*

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal, successful return.
KSC_NOLISTMEM	Not enough list memory for this instruction.

5. CAMAC Command Line Utilities

This chapter describes the general purpose CAMAC utilities available for simple testing. These utilities may be called from a DOS prompt, a batch file, or from the WINDOWS icon. The commands are simple to understand and use. Features included in the commands are single 24/16-bit CAMAC data transfers, control operations to the crate controller, and return of the crate controller status. Using the commands allow the user to verify that a given CAMAC module can be addressed, and that it is operating properly. In addition, it is a convenient way to become familiar with how the module functions before developing application code which makes use of it.

5.1 Command Summary

Command parameters define what the utility will act upon. All parameters are optional as indicated by brackets "[...]". The user will be prompted for any parameters that are not specified on the command line.

The following describes the execution of the utilities from the DOS prompt or a batch file with parameters.

5.1.1 CACTRL CAMAC Utility

This utility does control functions to CAMAC chassis on the CAMAC Serial highway. CACRTL performs a crate wide CAMAC control operation (i.e., Init, Clear, Set Inhibit, Clear Inhibit, Online).

The syntax for the CACTRL utility is:

CACTRL [/C] [/INIT] [/CLEAR] [/SETINH] [/CLRINH] [/ONLINE]

Note - All parameters may be omitted or when specified may be entered in any order.

Qualifier	Description
C	**Chassis number of the crate (0 to 63). The default value is chassis one.**
INIT	**Assert the init line in the CAMAC chassis**
CLEAR	**Performs a CAMAC clear operation**
SETINH	**Set the dataway inhibit line in the CAMAC chassis**
CLRINH	**Clear the dataway inhibit line in the CAMAC chassis**
ONLINE	**Put the chassis online**

CACTRL Examples

Example 1:
In this example, the first CACTRL command specifies the crate number and performs a control operation (crate online). The second CACTRL command will prompt the user for the crate number and sets the inhibit bit in the crate controller. As a result of the inhibit bit being set the LED on the crate controller is turned on and the inhibit data-way signal true.

```
CACTRL /ONLINE /C=3
CACTRL /SETINH
```

Example 2:
In this example, the first CACTRL command specifies the crate number and performs a control operation (crate online). The second CACTRL command prompts for the crate number and sets the inhibit bit in the crate controller. As a result of the inhibit bit being set the LED on the crate controller is turned on and the inhibit data way signal true. The third CACTRL command prompts for the crate and performs a CAMAC clear operation.

```
CACTRL /ONLINE /C=2
CACTRL /ONLINE /SETINH
CACTRL /CLEAR
```

5.1.2 CAM CAMAC Utility

The CAM utility allows the user to do simple CAMAC operations. This utility should be used with caution as it does commands to the target crate without any regard to the current applications running on the system. The syntax for the CAM utility is:

CAM [/C=] [/N=] [/A=] [/F=] [/DATA=]

Note - All parameters may be omitted or when specified may be entered in any order.

Qualifier	Description
C	**Chassis number of the crate (0 to 63). The default value is chassis one.**
N	**Station number within the CAMAC chassis of the module to be selected.**
A	**Sub address to be selected within the CAMAC module. The default value is zero.**
F	**The CAMAC function code toe be performed to the device. The default value is zero.**
DATA	**Optional Write data if the function requires data. The user may indicate hexadecimal by prepending a "X" to the value.**

CAM executes a single 24-bit CAMAC data transfer. This command reads or writes 24 bits of data to or from a CAMAC module.

CAM Examples

Example 1:

In this example, the first CAMAC command performs a read function, F(0), from sub-address zero, A(0), of crate one, C(1) directed to slot 1, N(l). The second command also performs a read function from the same slot and address, but the user will be prompted for the crate number. The third CAMAC command will prompt the user for all parameters. The output for a read operation displays the data in both decimal and hexadecimal format. Although the output is listed only once in the following example, it would actually be produced by each of the read operations as they were executed.

```
CAM /C=1 /N=l /A=O /F=O
CAM /N=1 /A=O /F=O
CAM
```

Data returned from CAM24 in decimal = 32, in hex = 0x20\n

Example 2:
In this example, the first CAMAC command performs a write function, F(16), to sub-address zero, A(O), with a value of 10 directed to crate 2, slot 3. The second command also performs a write function however the data value is specified in hexadecimal format. The "x" is used to represent hex notation with a value "20".

```
CAM /C=2 /N=3 /A=O /F=16 /DATA=32
CAM /C=2 /N=3 /A=O /F=16 /DATA=x20
```

5.1.3 CCSTAT CAMAC Utility

Displays the crate controller status (i.e., Inhibit status, L-SUM status, LAM register status, Crate Controller Status register, and Error Status register). The first two values are displayed in decimal, the remaining three values are in hexadecimal format. Refer to the crate controller manual for the meaning of the bits in the crate controller registers.

CCSTAT /C

Qualifier	Description
C	**Chassis number of the crate (0 to 63). The default value is chassis one.**

Example 1:
In this example, the first CCSTAT command specifies the crate number and displays all crate controller status registers.

CCSTAT /C=1

Output -

Crate status for crate: 1
Inhibit Status = 1
LSUM status = 0
Lam Register (Box) = 0x40
Crate Controller Status Register = 0x 44
Error Status Register = 0x0

6. KSC API Library

The KSC API (Application Programming Interface) provides all of the functions of the 2115 to the user by actually do the NT system service calls to the NT 2115 device driver. The users are encouraged to use these routines to limit the changes resulting from changes in the Driver interface and the NT operating system.

Besides providing the basic functioning the 2115, the KSCAPI also supports the building of command lists. These command lists are for both VXI and CAMAC chassis. The CAMAC library calls these routines to build its lists and to function the 2115

Programmers that work in "C", may also use command list generation macros written in "C". These macros generate runtime code that initialize a command list. The use of these macros or the command list generation routines are encouraged in the event there are changes to the command list instructions.

6.1 API Usage

The API is implemented as a set of linkable routines in an archive library. User applications may link with this file or access the device driver directly. All of the routines are prototyped in the ksc_api.h file. This file may be included using the following in a normal "C" program:

```
#include <ksc_handle.h>
#include <ksc_api.h>
```

Those user's using FORTRAN:

```
include "ksc_handle.inc"
include "ksc_api.inc"
```

6.2 API Handle

The API allocates a handle and returns its address to the caller when the user calls the KSC_Init interface. All particulars of the API are then maintained within this allocated region. The definition of this handle is maintained in the ksc_handle.h file. The handle is passed to all of the routines (except for the KSC_init routine). The actual byte count completed for a request is returned in the handle element: **"xfsize"**. The device driver status is returned in the handle element: **"status"**. The status should be examined along with the return value from the interface routine. The status may be either a status from the device driver or another error from NT. The routine KSC_print_symbolic will translate and print the returned English status code to standard output. All of the API routines return a 32 bit integer for status similar to the NT system services. If the returned status is odd, then the call completely successfully.

6.3 Command List Generation

There are a set of macros that are provided for the user to create lists which can be executed by the CAMAC Serial Host adapter or a slot zero crate controller. These are documented in the command list macros chapter. Additionally, the user may use the runtime routines to initialize a command list. The runtime routines will function in any language while the macros are only valid for "C" programmers.

6.4 Partition Contention

The CAMAC Serial device driver allows the user to load any of the partitions. The user must use mutexes or semaphore to protect the use of the partitions by multiple threads or processes. The KSC_loadgo always uses partition one and will both load the command partition and execute it autonomously.

6.5 TEST_API Program

The TEST_API program provides a menu to execute many of the KSC routines. Originally its purpose was to test the various KSC routines. It can also be used to control events for debugging application programs and testing hardware configurations. This application is built to test both the 2115 and the 2962. Therefore, v160 routines are not valid.

The menu presented on the screen/window is:

6.5.1.1.1.1.1 KSC Application Library Test Utility

1.	KSC_init	2.	KSC_set_partitions
3.	KSC_display_partitions	4.	KSC_diag
5.	KSC_load_cmdlist	6.	KSC_read_cmdlist
7.	KSC_exec_rlist	8.	KSC_exec_wlist
9.	KSC_read_counters	10.	KSC_reset
11.	KSC_demand_read	12.	
13.	Init command list (read)	14.	KSC_loadgo (write)
15.	Init command list (write)	16.	KSC_loadgo (read)
17.	KSC_get_failure	18.	Change Crate number
19.	KSC_v160_loadcmd (& build)	20.	KSC_v160_trigger
21.	KSC_v160_readcmd	22.	KSC_v160_readbuf
23.	KSC_v160_readreg	24.	KSC_exec_clocked_list
25.	KSC_v160_writereg	26.	KSC_read_multibuf
27.	KSC_mbuf_done	28.	
99.	Exit		

Enter selection [1]:

Always execute selection 1 before any other.

6.6 KSC_demand_read

Post a read for one or more demand commands.

FORMAT

KSC_demand_read (KSC_handle, demand_list, demand_list_size)

RETURNS

NT Usage: status
Type: int
Mechanism: by reference

API return status.

ARGUMENTS

KSC_handle
Type: structure KSC_handle
Access: readonly handle
Mechanism: by reference

demand_list
Type: array of ints
Access: write
Mechanism: by reference

A long word array to receive the demand interrupts. Must be long word aligned.

demand_list_size
Type:
Access:
Mechanism:

Size in bytes of the demand list array.

DESCRIPTION

The driver allows the ability to receive demand messages from the highway and list interrupts while the 2115 is executing a command list. These demands are stored in a demand FIFO and they are dequeued with this read. If there are no demands currently pending, the calling thread will be suspended until such time that a demand message or a list interrupt is processed.

In the event of a device reset this routine will return with status indicating that demand messages might have been lost (those that might have been in the FIFO when the reset was executed).

The host "in list" interrupts can be distinguished from the demand messages from the 2115 by a non-zero upper word. The content of the lower demand message is as it was read from the 2115. The user should reference the 2115 manual for this information.

There is no included support to provide access control for different processes using this call. The Demand Process uses this call to acquire the demands from the driver. This call is provided for users who wish to develop their own demand servicing applications.

This call will not complete until a demand is received. Therefore, it may wait for a very long time. The user is responsible for enabling of the crate such that LAMs can be generated.

CONDITION VALUES RETURNED

KSC_CHASSIS	Chassis number is not a valid chassis number.
KSC_HANDLE	The handle is invalid.
KSC_BUFTOOSMALL	The user buffer must be at least four bytes long.
KSC_ALIGNMENT	The user buffer must be long word aligned.
KSC_OPENERROR	Unable to open a handle to the demand device. Check to make sure driver is loaded.
KSC_READERR	Error while reading the demands. Inspect the "status" variable contained within the handle for condition codes from the driver.

6.7 KSC_display_partitions

Read command list partition table of the KSC 2115.

FORMAT

KSC_display_partitions (KSC_handle, partition_table)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by reference

API return status.

ARGUMENTS

KSC_handle

Type:	structure KSC handle
Access:	readonly
Mechanism:	by reference

Used by all device driver access routines. Returned by KSC_INIT.

partition_table

Type:	structure KSC_driver_ptable
Access:	write
Mechanism:	by reference

Returns the current partition table of the KSC 2115 as maintained by the KSCGI device driver.

DESCRIPTION

This call will call the kscgi device driver to return the current partition table of the KSC 2115. The KSC_driver_ptable structure contains both the starting address and the length of each partition in bytes.

CONDITION VALUES RETURNED		
	KSC_HANDLE	Handle is invalid.
	KSC_NOTOPEN	Handle has not been initialized.
	KSC_IOCTL	Error from device driver or NT. Examine status in KSC_handle.

6.8 KSC_enable_demand

Enable reception of a single demand from a particular chassis.

FORMAT

KSC_enable_demand (KSC_handle, Chassis, DmdId, DmdType, APCAdr, APCPrm)

RETURNS

NT Usage:
Type:
Mechanism:

API return status.

ARGUMENTS

KSC_handle
Type:
Access:
Mechanism:

Chassis

Type:	int
Access:	readonly
Mechanism:	by value

Chassis number that the demand is expected from. Chassis number zero is the host CAMAC Serial adapter.

DmdId

Type:	int
Access:	readonly
Mechanism:	by value

This is the ID of the demand. For command lists that generate the demand, this is the demand id that is coded into the instruction. The demand is a demand from a CAMAC chassis (2952) the demand id is the encoding of the LAM lines or the multi-buffer bit. For CAMAC chassises, the slot number minus one is the demand id.

DmdType
Type: int
Access: readonly
Mechanism: by value

The demands can either be set up to generate a single one shot when the demand occurs or a reoccurring demand. A value of one (1) is a single one shot and a value of two (2) is a repeating demand.

APCAdr
Type: address
Access: readonly
Mechanism: by reference

Asynchronous Procedure routine to be called when the demand occurs. The APC parameter is the demand id and the chassis that generated the demand passed by reference. The demand id is in the lower sixteen bits and the chassis is in the upper sixteen bits.

APCPrm
Type: address
Access: readonly
Mechanism: by reference

A pointer to an user defined value passed along to the Ast Routine.

DESCRIPTION

This routine Enables Demands by sending a message to the Demand Process requesting notification when the particular demand id for the indicated chassis occurs. If the Demand Process is not running a message will be displayed that the demand region could not be found and the demand will not be enabled.

This routine communicates with the demand process using named pipes.

For one shot demands, this routine may be called multiple times to re-enable the one shot. Any process that registers for a demand supersedes any previous process for the particular demand id for a particular chassis.

CONDITION VALUES RETURNED		
	KSC_CHASSIS	Invalid user chassis number entered.
	KSC_NOTCFG	All demands that are to be enabled must be configured. See the Demand Process chapter.
	KSC_DMDTBLFULL	The system allows for a maximum number of demands that single process can connect for. The caller has exceeded this value. Examine the KSC_handle.h for this maximum.

6.9 KSC_exec_clocked_list

Execute a command list off the clock using multi-buffering.

FORMAT

KSC_exec_clocked_list (KSC_handle, partition, buffer, BufLen, TimerValue, MultiBuffFlag)

RETURNS

NT Usage:
Type:
Mechanism:

ARGUMENTS

KSC_handle
Type:
Access:
Mechanism:

partition

Type:	int
Access:	readonly
Mechanism:	by value

Partition which contains the command list to be executed off the clock.

buffer

Type:	int array
Access:	write
Mechanism:	by reference

Address of user buffer to receive the data. The buffer must be long word aligned. If multi-buffering is selected, the buffer must evenly divisible by the number of buffers selected. For example, if the number of buffers is 4, the buffer size must be evenly divisible by sixteen (16= 4*size of (int)).

BufLen
Type: int
Access: readonly
Mechanism: by value

Total size of the buffer in bytes.

TimerValue
Type: int
Access: readonly
Mechanism: by value

Value to be loaded into the host CAMAC Serial Timer register. This is a sixteen bit value passed as an int. See the *2115 Hardware Manual* for the definition of this register. A negative value is considered to be the external clock. If a zero value is provided, the list is triggered once.

MultiBuf
Type: int
Access: readonly
Mechanism: by value

This is the number of multi-buffers to be used. This value must be between one and four.

DESCRIPTION

This API allows the user to specify a partition which contains a command list that when triggered by the external or internal clock to run will provide data into the passed buffer. Only multibuffer transfers are supported. List that are not clocked are initially triggered and the list must continue executing to provide the data.

The user must use the KSC_read_multibuf to determine which buffer has completed. In a multi-buffer situation, this entry maintains an active I/O request on the device until the multi-buffering is disabled or the process exits. This monopolizes the 2115 to the calling process. This call never completes. Normally two threads or processes are created. The first process thread is responsible for calling this routine to set up the transfer. The second process thread is responsible for reading the buffer completions and acknowledging them back to the driver.

CONDITION VALUES RETURNED		
	KSC_ALIGNMENT	User buffer is not long word aligned.
	KSC_BUFTOOSMALL	The user buffer must be at least four bytes.
	KSC_HANDLE	KSC_init has not been called or handle is invalid.
	KSC_NOTOPEN	KSC_init has not been called.
	KSC_PARTIONERR	The partition parameter is out of range.
	KSC_CLKINTERVAL	Clock interval is larger than 0xffffffff and it is not minus one.
	KSC_OPENERROR	Unable to open the kca2: device.
	KSC_MBUFALIGNMENT	The number of multi-buffers specified and the buffer size do not evenly divide.
	KSC_READERR	User list generated a read error.
	KSC_NOTALLXFER	The user list did not completely fill the user buffer.

6.10 KSC_exec_rlist

Execute a read command list.

FORMAT

KSC_exec_rlist (KSC_handle, partition, buf, buf_size)

RETURNS

NT Usage: status
Type: int
Mechanism: by reference

API return status.

ARGUMENTS

KSC_handle
Type: structure KSC handle
Access: readonly
Mechanism: by reference

Used by all device driver access routines. Returned by KSC_INIT.

partition
Type: int
Access: int
Mechanism: by value

Command list partition within the KSC 2115 that is to be executed.

buf
Type: byte
Access: write
Mechanism: by reference

Data to be returned to user as generated by the execution of the command list stored in the indicated partition.

buf_size
Type: int
Access: readonly
Mechanism: by value

DESCRIPTION

The command list currently stored in the KSC 2115 will be executed. The user supplied buffer will receive any data that is sourced by the KSC 2115. If the user buffer is too large, or the KSC 2115 fails to source sufficient data, the request will only complete via a device timeout or an embedded command list interrupt. If the command list contains a list interrupt, the driver will consider such an interrupt as a completion of the list. The kscgi device driver always attempts to store a list completion interrupt at the end of the loaded command list partition.

CONDITION VALUES RETURNED

KSC_HANDLE	Handle is invalid.
KSC_NOTOPEN	Handle has not been initialized.
KSC_READERR	Error from device driver or NT. Examine status in KSC_handle.

6.11 KSC_exec_wlist

Executes a write command list.

FORMAT

KSC_exec_wlist (KSC_handle, buf, buf_size, used_size,partition)

RETURNS

NT Usage: status
Type: int
Mechanism: by reference

API return status.

ARGUMENTS

KSC_handle
Type: structure KSC handle
Access: readonly
Mechanism: by reference

Used by all device driver access routines. Returned by KSC_INIT.

buf
Type: array of words
Access: readonly
Mechanism: by reference

Data source for the command list which was loaded in the indicated partition.

buf_size
Type: int
Access: readonly
Mechanism: by value

Size of the user supplied buffer.

used_size
Type: int
Access: write
Mechanism: by reference

The returned number of bytes that requested by the KSC 2115.

partition

Type:	int
Access:	readonly
Mechanism:	by value

The command list partition that contains the command list which is to be executed.

DESCRIPTION

This call will request the execution of the command list that has already been loaded into the command list partition of the KSC 2115. The actual completion of the request depends on whether the user supplied the correct number of bytes for the KSC 2115. A command list may require more data than what was provided by the user, however, the DMA will complete regardless. If the user embedded a command list interrupt, this will terminate the command list.

CONDITION VALUES RETURNED

KSC_HANDLE	Handle is invalid.
KSC_NOTOPEN	Handle has not been initialized.
KSC_READERR	Error from device driver or NT. Examine status in KSC_handle.

6.12 KSC_get_failure

Retrieve list execution failure from last list execution.

FORMAT

KSC_get_failure (KSC_handle, partition, failure_array)

RETURNS

NT Usage: status
Type: int
Mechanism: by reference

API return status.

ARGUMENTS

KSC_handle
Type: structure KSC handle
Access: readonly
Mechanism: by reference

Used by all device driver access routines. Returned by KSC_INIT.

partition
Type: int
Access: readonly
Mechanism: by reference

Partition to return the last failure. This should be the same as the partition passed to the KSC_exec_list, KSC_exec_rlist, or KSC_exec_wlist.

failure_array

Type:	struct of type KSC_error_exc
Access:	write
Mechanism:	by reference

An array to receive the last failure that the CAMAC Serial device driver encountered. The following are returned in the eight "ints" at the time of the I/O completion. The user should reference the KSC 2115 device manual for a description of the bits contained in the device registers.

[0]- Ending Driver Status
[1]- Ending 2115 CSR
[2]- Ending 2115 ICSR
[3]- Ending 2115 TTCR
[4]- Ending 2115 CMA
[5]- Starting 2115 CSR
[6]- Starting 2115 TTCR
[7]- Starting 2115 ICST

DESCRIPTION

The device driver copies the completion information at the end of each list execution for each command list partition in the driver. This routine will return these values in the integer array.

CONDITION VALUES RETURNED

KSC_HANDLE	Handle is invalid.
KSC_NOTOPEN	Handle has not been initialized.
KSC_READERR	Error from device driver or NT. Examine status in KSC_handle.
KSC_IOCTL	Driver error or NT error. Examine status in KSC_handle.
KSC_PARTIONERR	Desired partition number is not valid.

6.13 KSC_init

Initialize driver access routines.

FORMAT

KSC_init (KSC_handle,ctrl)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by reference

API return status.

ARGUMENTS

KSC_handle

Type:	pointer to structure KSC handle
Access:	readonly
Mechanism:	by reference

Handle used by all of the device driver access routines. Returns address of an allocated KSC_handle structure or a null.

ctrl

Type:	controller number
Access:	read
Mechanism:	by value

Controller number of the KSC 2115 CAMAC Serial (range: 0-4).

DESCRIPTION

This routine initializes the application library. The KSC 2115 devices are opened and a pointer to the KSC_handle is returned to the caller for future calls to the API. The controller number is used to open the "\\ks<ctrl>00" device. Upon a successful call, the KSC_handle will contain a pointer to the API's handle. This should be used for all KSC_API calls.

CONDITION VALUES RETURNED

KSC_ALLOC	Unable to allocate the KSC handle.
KSC_OPENERROR	Unable to open KSC devices. The status of the NT "open" call is returned in handle->status.

KSC_SUCCESS	Successful completion.

6.14 KSC_lasterror

Display the last known API and Driver error conditions.

FORMAT

KSC_lasterror(handle)

RETURNS

Usage:	NT Status
Type:	longword
Mechanism:	by value

ARGUMENTS

handle

Type:	structure KSC_handle
Access:	read only
Mechanism:	by reference

The handle returned by KSC_init.

DESCRIPTION

This routine will display the most recent API and Driver status information. The API status is the last status received from any non-listbuilding API routine. The driver status information is from actual device system service calls. Note that the API error codes will be of the KSC_xxxx type, while the device error codes can be either KSC or NT error codes.

CONDITION VALUES RETURNED

KSC_HANDLE	Handle not initialized. Need to call KSC_init first before using this function.

6.15 KSC_loadgo

Load a command list and execute it.

FORMAT

KSC_loadgo (KSC_handle, partition,list, list_size, buf, buf_size,direction)

RETURNS

NT Usage: status
Type: int
Mechanism: by reference

API return status.

ARGUMENTS

KSC_handle
Type: structure KSC handle
Access: readonly
Mechanism: by reference

Used by all device driver access routines. Returned by KSC_INIT.

buf
Type: array of words
Access: read/write
Mechanism: by reference

Source or sink of the data for the command list loaded into the indicate partition. The buffer contains both the command list and the data buffer. The buffer contains in the first "int" the size of the command list in bytes, followed by the command list and the data buffer. The command list macros provide a convenient way to create this combined buffer.

buf_size
Type: int
Access: read
Mechanism: by value

Size of user buffer.

direction

Type:	int
Access:	readonly
Mechanism:	by value

Direction of the transfer (0= user sources the data, 1= 2115 sources the data for the command list).

DESCRIPTION

This entry provides an ability for the user to both load a command list partition and then execute the loaded command list. The user indicates the direction of the data transfer. This routine simply calls the command partition load function and then either the read or write command list function. The completion of the command list is identical to that for the execute read or write command lists.

Because this routine is completed in a single operation, there is no need to control access to a particular partition. This routine always uses command list partition one of the CAMAC Serial host adapter.

CONDITION VALUES RETURNED

KSC_HANDLE	Handle is invalid.
KSC_NOTOPEN	Handle has not been initialized.
KSC_READERR	Error from device driver or NT. Examine status in KSC_handle.
KSC_PARTIONERR	Desired partition number is not valid.

6.16 KSC_load_cmdlist

Load a KSC 2115 command list into a partition.

FORMAT

KSC_load_cmdlist (KSC_handle, partition, list, list_len)

RETURNS

NT Usage: status
Type: int
Mechanism: by reference

API return status.

ARGUMENTS

KSC_handle
Type: structure KSC handle
Access: readonly
Mechanism: by reference

Used by all device driver access routines. Returned by KSC_INIT.

partition
Type: int
Access: readonly
Mechanism: by value

Which partition the command list should be loaded into.

list
Type: array of ints
Access: readonly
Mechanism: by reference

Command list to be loaded into the indicated partition of the KSC 2115.

list_len
Type: int
Access: readonly
Mechanism: by value

Length of the command list to be loaded into the indicated partition of the KSC 2115. The length of the list must be less than the length of the indicated partition.

This page intentionally left blank.

DESCRIPTION

Before a command list can be executed it must be loaded into the command list memory of the KSC 2115. The programmer should reference the KSC 2115 documentation with regard to the actual content of the command list.

CONDITION VALUES RETURNED

KSC_HANDLE	Handle is invalid.
KSC_NOTOPEN	Handle has not been initialized.
KSC_READERR	Error from device driver or NT. Examine status in KSC_handle.
KSC_IOCTL	Driver error or NT error. Examine status in KSC_handle.
KSC_PARTIONERR	Desired partition number is not valid.

6.17 KSC_mbuf_done

Release multi-buffers buffers.

FORMAT

KSC_mbuf_done (KSC_handle, Done)

RETURNS

NT Usage:
Type:
Mechanism:

ARGUMENTS

KSC_handle
Type:
Access:
Mechanism:

Done

Type:	int
Access:	readonly
Mechanism:	by value

Mask as returned from KSC_read_multibuf indicating which of the multi-buffers have been serviced by the user application.

DESCRIPTION

This routine releases the segments of the user's buffer that have been processed by the user application. The Done parameters should be the value returned by the KSC_read_multibuf routine.

As the user process is informed that segments within the user buffer set up when the clocked multi-buffer was set up (KSC_exec_clocked_list) the driver needs to be informed that they may be reused. Failure to call this routine will result in a multi-buffer overflow condition.

CONDITION VALUES RETURNED

KSC_HANDLE	KSC_init has not been called or handle is invalid.
KSC_NOTOPEN	KSC_init has not been called.
KSC_OPENERROR	Unable to assign a device to KCA3:
KSC_READERR	Bad driver status, see status in the KSC_handle structure.
KSC_SUCCESS	Normal completion.

6.18 KSC_print_symbolic

Convert and print symbolic text for a status code.

FORMAT

KSC_print_symbolic (status)

RETURNS

NT Usage:
Type:
Mechanism:

ARGUMENTS

status
Type:
Access:
Mechanism:

NT, API, or driver status to be converted to text.

DESCRIPTION

This routine calls the sys$getmsg and prints to standard output the symbolic description of the status code. The user must have linked the message file: kgdriver_msg from the library.

CONDITION VALUES RETURNED

6.19 KSC_read_cmdlist

Read a loaded command list from a partition.

FORMAT

KSC_read_cmdlist (KSC_handle, partition, list, list_len)

RETURNS

NT Usage: status
Type: int
Mechanism: by reference

API return status.

ARGUMENTS

KSC_handle
Type: structure KSC handle
Access: readonly
Mechanism: by reference

Used by all device driver access routines. Returned by KSC_INIT.

partition
Type: int
Access: readonly
Mechanism: by value

The desired command list partition to read the contents.

list
Type: array of ints
Access: write
Mechanism: by reference

Buffer to receive the current contents of the indicated command list partition. The number of bytes returned depends on the list size passed and the current size of the requested partition.

list_len
Type: int
Access: read
Mechanism: by reference

list_len bounds the number command list words from the command list

memory that can be returned from the partition. If the size of the partition currently is larger than user's buffer, only list_len bytes will be returned along with a informational status indicating that the user's buffer was too small. The actual size is returned in xsize in the KSC_handle.

DESCRIPTION

This entry can be used to determine if a command list was successfully loaded into the command list partition or to recover a possible faulty list that is not executing correctly. The number of words specified by the list_len is retrieved from the indicated partition. The partition table is maintained by the software kscgi device driver and set by the KSC_set_partitions call. If the user passed buffer does not match the partition size, a informational status is returned. The actual number of words returned is maintained in the KSC_handle. The xsize variable is in bytes and will, therefore, be twice as large as the number of command list words requested. This routine will return the complete partition which may not have been completely loaded by a KSC_load_cmdlist call.

CONDITION VALUES RETURNED

KSC_HANDLE	Handle is invalid.
KSC_NOTOPEN	Handle has not been initialized.
KSC_READERR	Error from device driver or NT. Examine status in KSC_handle.
KSC_PARTIONERR	Desired partition number is not valid.

6.20 KSC_read_counters

Return Driver statistic counters.

FORMAT

KSC_read_counters (KSC_handle, Counters)

RETURNS

NT Usage:
Type:
Mechanism:

ARGUMENTS

KSC_handle
Type:
Access:
Mechanism:

Counters

Type:	struct KSC_counters_gi
Access:	readonly
Mechanism:	by reference

Buffer to receive the counters. The array contains the following in long word integers in order as follows:

Number of timeouts
Number of writes
Number of reads
Number of command list loads
Number of interrupts
Number of list_interrupts
Number of demand_interrupts

DESCRIPTION

This routines returns the current counters from the 2115 device driver. These may be used for user written diagnostic programs.

CONDITION VALUES RETURNED

KSC_HANDLE	KSC_init has not been called or handle is invalid.

6.21 KSC_read_multibuf

FORMAT

KSC_read_multibuf (KSC_handle, Done)

RETURNS

NT Usage:
Type:
Mechanism:

ARGUMENTS

KSC_handle
Type:
Access:
Mechanism:

Done

Type:	int
Access:	write
Mechanism:	by reference

This is a bit word that will contain a bit for each of the multi-buffer segments that have been filled. This value should be passed to KSC_mbuf_done after the user has processed the data in the segment.

DESCRIPTION

This routine posts a read on the KCA3 device. When the 2115 driver receives a multi-buffer interrupt, the read will complete. If at the time of the read post, there has already been a multi-buffer completion, then the read will complete immediately. The user is returned a bit word indicating which segments of the buffer are complete. The user should use the bit word to determine which segment to process the data from. When the processing is done, the bit word should be passed to KSC_mbuf_done to release the buffer segments. It should be noted that more than a single bit may be set.

CONDITION VALUES RETURNED

KSC_SUCCESS	Normal completion.
KSC_HANDLE	KSC_init has not been called or handle is invalid.
KSC_OPENERROR	Unable to open the kca3: device.
KSC_NOTOPEN	KSC_init has not been called.
KSC_READERR	Read of the bit word failed. See status value in the KSC_handle.

6.22 KSC_set_partitions

Load partition boundaries of the KSC 2115.

FORMAT

KSC_set_partitions (ksc_handle, partition_table)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by reference

API return status.

ARGUMENTS

ksc_handle

Type:	structure KSC handle
Access:	readonly
Mechanism:	by reference

Used by all device driver access routines. Returned by KSC_INIT.

partition_table

Type:	structure KSC_partition_table
Access:	readonly
Mechanism:	by reference

The desired partitioning of the command list memory of the 2115. This table contains the starting address in bytes for each of the partitions. A zero terminates the table. Specifying all zeroes will result in the last partition being allocated all of the 32KB of the command list memory. To allocate all of the command list memory to the first partition, specify (0,0x8000,0,0,0,0,0,0).

DESCRIPTION

This routine calls the device driver to partition the 32K command memory of the KSC 2115. The command memory may be divided in up to eight different partitions. Each of the partitions is assigned a pair of devices. Any of the partitions may be from zero to the remaining size of the command list memory. Partitions cannot be overlapped or concatenated..

CONDITION VALUES RETURNED

KSC_HANDLE	Handle is invalid.
KSC_NOTOPEN	Handle has not been initialized.
KSC_IOCTL	Driver error or NT error. Examine status in KSC_handle.

6.23 KSC_set_timeouts

Set partition command list timeout values.

FORMAT

KSC_set_timeouts (KSC_handle, time_array)

RETURNS

NT Usage:	status
Type:	int
Mechanism:	by reference

API return status.

ARGUMENTS

KSC_handle

Type:	structure KSC handle
Access:	readonly
Mechanism:	by reference

Used by all device driver access routines. Returned by KSC_INIT.

time_array

Type:	array of 8 longwords
Access:	readonly
Mechanism:	by reference

This array contains in seconds the timeout for each partition. The minimum value is two seconds as NT driver timeout routines are called only at one second intervals plus or minus one second.

DESCRIPTION

The device driver requires that all lists complete within a specific time limit. This value can be specified for each partition. When the device driver is loaded a default value is used for each partition.

CONDITION VALUES RETURNED

KSC_NOTOPEN	User has not called ksc_init.
KSC_READERR	The driver returned an error. See status variable in the API handle.

6.24 KSC_stop_mbuf

Stop current multi-buffer Read operation in progress.

FORMAT

KSC_stop_mbuf (KSC_handle)

RETURNS

NT Usage:
Type:
Mechanism:

ARGUMENTS

KSC_handle
Type:
Access:
Mechanism:

DESCRIPTION

If a user has called KSC_exec_clocked_list with multi-buffering, the 2115 is monopolized by the user until the process exits, deassigns the channel, or calls this routine. This completes any current multi-buffer reads (KSC_read_multibuf) and completes the I/O request set up by KSC_exec_clocked_list.

CONDITION VALUES RETURNED

KSC_HANDLE	KSC_init has not been called or handle is invalid.

6.25 API and Driver Errors

The API and driver error codes are defined in a later chapter. By convention, all of the interface routines return an odd value if the call was successful and even if the call was not. The error returned by the NT kernel is returned in the status argument of the handle. This status argument is only valid if the interface returns an even error code. The value of the status entry may be from the device driver or from NT. All of the error codes are defined in kscgi_errors.h header file. The user should reference the KSC 2115 hardware documentation for the most of the device driver errors.

The routine KSC_print_symbolic will translate and print the error code to the symbolic English description to the standard output.

7. Demands

7.1 The Demand Process

The Demand Process is a high priority process that acts as a server to application processes for handling demands from the KSC2115 NT device driver. Application processes send registration requests to the Demand Process for all demands received from the CAMAC highway they wish to service. Demands are enabled by the Demand Process for each CAMAC chassis if demands are not currently enabled on the chassis. When the device driver receives demands from the CAMAC highway, the Demand Process immediately dispatches the demand to the registered process. Demands that are received for which there are no processes registered are ignored by the Demand Process (only a statistic is kept).

7.2 Demand Configuration File

On startup, the Demand Process creates a temporary group global section called "DMDREGION". It then populates this region with demand entries read from a configuration file pointed located in the same directory as the Demand Process (DMDPROC.EXE). If the Demand Process receives an error that the region already exists, it knows that another Demand Process is currently servicing demands. The Demand Process exits under these conditions.

The maintenance of this file is through a normal text editor. The information contained in the configuration file is:

Chassis Number	Demand ID	Chassis Type	Demand Queue Length	Description
1 to 63	0 to 255	2	10	English comment

The syntax of the demand configuration file is:

chassis, id, type, qlength, desc

Where:

chassis	Decimal Chassis number on the Grand Interconnect highway
id	Demand Id generated by the Chassis. See the V160 and 3972 slot zero controllers for description.
type	VXI (1) or CAMAC (2)
qlength	Queue length
desc	User description displayed by dmdsts utility

The following is an example configuration file. Any line beginning with an exclamation mark is ignored.

```
! Sample configuration file. This file is input to the demand
process.
! Exclamation points at the beginning of a line denote comment
lines.
! Commas are used to separate the columns of information. The
columns are
! defined below. Commas used in the description field will
simply
! truncate the description at the position of the comma. The use
of spaces
! before and after columns will be considered valid input.
!Chass   Dema    Type    Queue     Description
is       nd Id           Length
!
 1,      1,      2,      11,       Crate 1 / Demand 1
 1,      2,      2,      12,       Crate 1 / Demand 2
 1,      3,      2,      13,       Crate 1 / Demand 3
 1,      4,      2,      14,       Crate 1 / Demand 4
 2,      4,      2,      15,       Crate 2 / Demand 4
 2,      5,      2,      16,       Crate 2 / Demand 5
 2,      6,      2,      17,       Crate 2 / Demand 6
```

7.2.1 Application Registration for Demands

The Demand Process establishes a single system-wide pipe "\\.\pipe\dmdproc". The Demand Process reads registration requests from user processes (see KSC_ENABLE_EVENT). The user receives the status of the Demand notification via a unique pipe created by the user process for the particular demand. The demand must be defined within the demand configuration file prior to the startup of the demand process. Adding new demands requires the restart of the Demand Process and the stopping of all processes currently registered for demands.

7.2.2 Demand Processing

When a demand is received by the Demand Process, the Chassis number is used to traverse the demand entries associated with it. This should reduce the search time for the matching Demand ID. Any demands that are received and are not in the table, will be logged to the Demand Process's log file, and the unknown-demand counter incremented. If the Application process that should receive the demand is no longer present, or if its pipe is closed, the demand event will be logged and the not-registered counter incremented. Otherwise, the Demand Process sends the following information to the registered application:

Function = DEMAND_MSG
Chassis Number of Demand
Demand ID in Chassis
User Index
Time of Demand

If this demand was a one-shot, the demand entry is cleared. When there are no longer any Application processes registered for a Chassis, the Demand Process will disable demand recognition for that Chassis.

It is possible that the process may be still active but the image that requested the demand registration may have been run down. The Demand Process will consider a pipe write error to be the same as a process no longer being available.

Each time a demand is processed, the Demand Process also increments statistic counters and stores the time stamp of the event in the group global region. (The utility DMDSTS can display this information.) If at any time the Demand Process gets a failure writing to a pipe it will disable the demand .

The number of demand messages in the demand FIFO, the frequency at which they arrive, and activity caused by other processes on the highway at the time will influence the speed at which a demand is delivered to an application process. For CAMAC crates, the Demand Process must read the LAM status register within the Crate to determine which of the slots within the Crate are asserting their LAM lines. This required read competes with all other requests to the device driver and will effect the response of the demand delivery to the registered process.

The Demand Process can not determine if multiple LAMs have been presented within a chassis. It is the user's responsibility to determine a redundant demand notification which can be created under the following circumstances:

1. Slot 1 in Crate 1 asserts LAM
2. Demand is sent to 2115
3. Demand process determines that a LAM is present in Crate 1 and read the LAM status register
4. The Demand is sent to the registered user
5. Slot 2 in Crate 1 asserts LAM
6. Demand process determines that a LAM is present in Crate 1 and reads the LAM status register which shows two slots asserting a LAM.
7. The Demand Process sends a redundant demand for slot 1 and the demand for slot 2
8. The requesting process for Slot 1runs and tries to clears the LAM in slot 1
9. The requesting process for Slot 1 receives the redundant Demand
10. The requesting process for Slot 2 receives the LAM

7.3 User Application Program

There may be more than one Application program that receives demands, but a single Demand ID in a Chassis can be registered to only one Application.

All Application programs must contain the following elements (see program \KCA001\EXAMPLES\TEST_DMD.C):

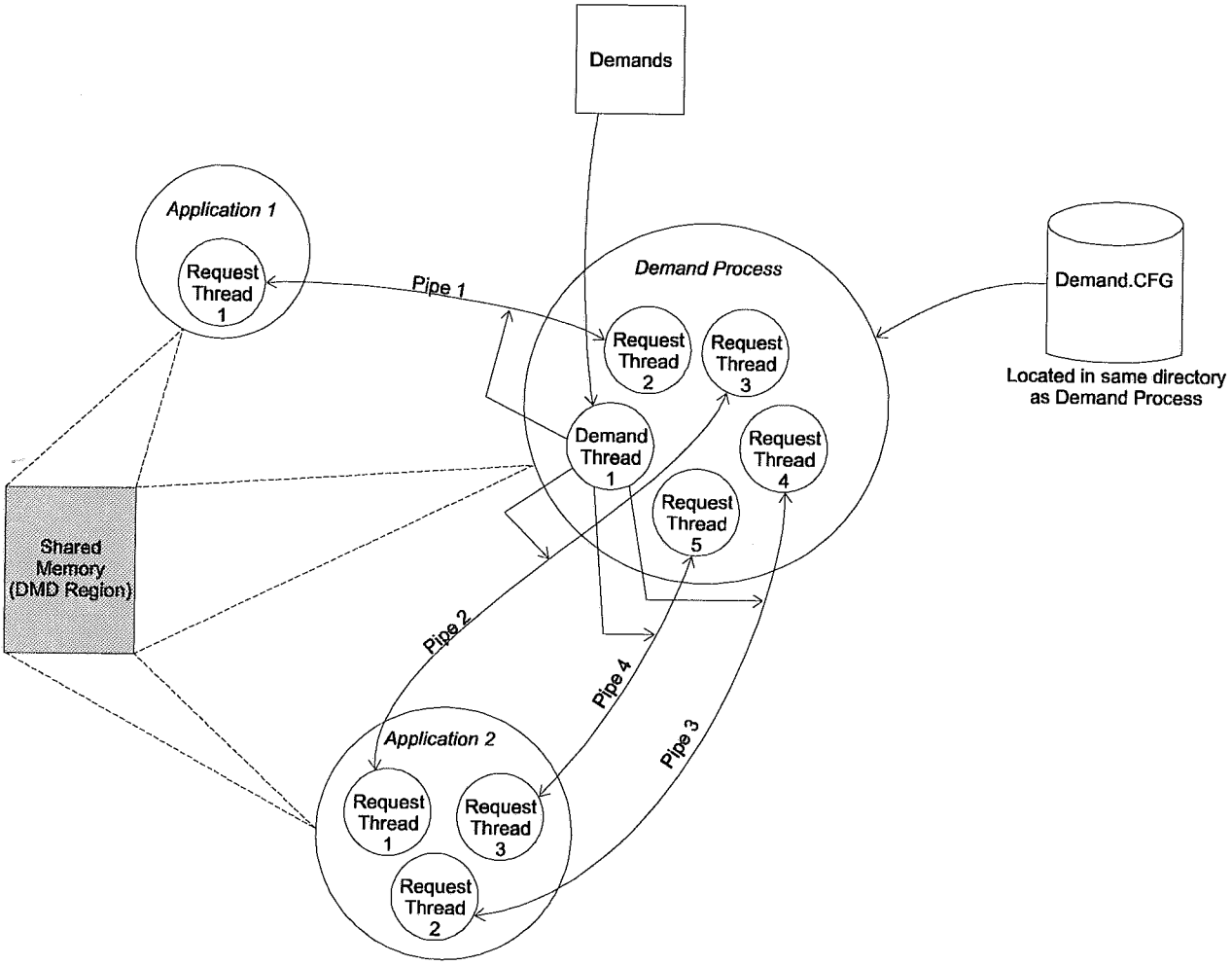
- Call KSC_init to create the structure KSC_handle required by all other KSC and CAM modules.
- Call KSC_enable_demand for each demand to be received. The application maps the demand region to ensure the Demand Process is running, and another process is not currently capturing the demand. This module creates a pipe then sends a registration request to the Demand Process using the Demand Process's registration pipe.. The registration reply is received in the APC routine, which it also sets up. Demands received are dispatched to a user-written APC routine which should appropriately process each demand received. Finally, this module reposts another read on the pipe for the subsequent demand.
- The developer must create a read APC routine to examine each demand received and take appropriate action.

The following diagram shows an overview of the Demand Process, a process registering for demands via the VTL library, and a process registering for LAMS (Demands) using the CAMAC library.

7.4 Demand Process Dataflow

The design of the Demand Message Process is a combination of the OpenVMS, NT and UNIX device drivers for the KSC2962 and KSC2115. This design allows for both CAMAC and VTL (only available on 2962) demands to be supported.

The NT version of the Demand Process utilizes named pipes for its method of communication between application programs and itself. A named pipe is a one-way or two-way pipe for communicating between a server process and one or more client processes. Named pipes allow for multiple instances of a single pipe, however an instance of the pipe may only be opened by one client at a time. Due to the fact that an instance of a pipe may only be opened once, multiple threads are often used to create multiple instances of the pipe to communicate with multiple clients. The following drawing is an overview of demand request flow.



The demand process creates 1 named pipe with the name \\.\PIPE\DMDPROC. The multiple pipes shown in the drawing are multiple instances of the pipe \\.\PIPE\PMDPROC.

The arced lines above each represent an instance of the named pipe \\.\PIPE\DMDPROC. These are two -way pipes. The application program will *send to* the Demand Process a request for a particular demand, and will *receive from* the Demand Process demand information. For every demand requested by an application program a thread and pipe instance will be created by both the application program and Demand Process. There will be one thread and pipe instance per demand request. In addition the Demand Process creates one additional thread (Thread 1 in the drawing above) to read demands from the KSC2962 or KSC2115. Thread 1 will be able to write to Pipes 1, 2, 3, 4 since all it needs is the pipe handle which it will be able to obtain from the shared memory region. The Demand Process is a Windows program with one window to display the error and status messages

7.5 Demand Utilities

7.5.1 Program DMDSTS

The DMDSTS program is a diagnostic that maps the Demand Process's demand global section. It allows a user to display the demand registration, pipes, and demand statistics. DMDSTS has read-only access to the Demand Process's global section.

The program presents information in one of two mutually exclusive modes:

- Continuous mode - Displays and updates every 5 seconds information on currently active demands. Output is only to the CRT screen in 132 column mode.
- Dump mode - Displays various amounts of information to the CRT screen or (optionally) to a file. The amount of information displayed depends on which command line switch is used.

Continuous Mode

Usage:	$ DMDSTS /CONT	Switch explicitly specified
	$ DMDSTS	Invokes/CONT, by default

The screen/window is put into 132 column mode. For each enabled demand found in the system, the following are displayed and updated every 5 seconds.

ACTIVE DEMANDS

chassis	type id	last demand	count	mbx	letters	pid	proc name	image name
1	CAMAC 10	20-FEB-1996 10:03:52.13	12	_MBA275:	0	89	DMD_PROC_1	DKA0:[KGI]DMD_1.EXE;52
1	CAMAC 11	20-FEB-1996 10:07:43.19	432	_MBA275:	0	89	DMD_PROC_1	DKA0:[KGI]DMD_1.EXE;52
2	CAMAC 20	20-FEB-1996 10:05:07.00	53	_MBA318:	4	92	DMD_PROC_2	DKA0:[KGI]DMD_2.EXE;21

To exit this screen, push RETURN. The screen/window should return to the original size (80 or 132 columns).

If the /OUT=file switch is present on the command line with the /CONT switch, it is ignored. Output is only to screen/window.

Dump Mode

Usage:	$ DMDSTS /CHASSIS=x	Region header plus chassis "x", enabled or not
	$ DMDSTS /ENABLED	Region header plus all enabled demands
	$ DMDSTS /CONFIG	Region header plus all configured demands

$ DMDSTS /ALL	Region header plus all demands, even those not enabled or configured.
$ DMDSTS /OUT=file	Output goes to "file", not screen

The first four switches are mutually exclusive, and, if more than one is present on the command line, the one with highest precedence is used.

Switch Precedence:

/ALL	Highest, overrides all below
/CONFIG	Overrides all below
/ENABLED	Overrides all below
/CHASSIS=x	Lowest, overrides no other switches

For each switch, the information displayed is:

Switch	Region Header	Chassis Table	Demand Entry
/ALL	X	All, configured or not	All that are config
/CONFIG	X	Only those configured	All for config
/ENABLED	X	Only those enabled	All for enabled
/CHASSIS=x	X	Only chassis "x", even if not config or not enabled	All for the chassis

The /OUT switch can be used with any of the above four switches to direct output to the indicated "file" instead of to the screen/window.

8. NT KCDRIVER

This chapter describes the implementation of the KSC 2115 PCI adapter under Windows NT version 4.0. The reader should reference the KSC hardware documentation for specifics about this device. Of the functionality provided by the 2115, the following are supported:

- Support for DMA of large memory buffers. The maximum transfer size is 1M bytes (however, depending on configuration, the number of mapping pages given to the device may limit the actual transfer size).
- Storage of command lists within the 2115
- Support for Demands/LAMS
- Support for segmented multibuffers
- Support for the Clock on the 2115 to trigger execution of command lists (Second Release)

The 2115 is a high performance device that differs from other previously manufactured CAMAC adapters from KSC. These differences allow for higher throughput, but provide less knowledge about No-X and No-Q on a per CAMAC instruction execution. Either the list can stop or ignore these conditions with the total status being the or of all of the CAMAC commands contained within the list.

8.1 Driver Interface

The NT driver is called using the following NT native system calls:

- **DeviceIoControl**
- **ReadFile**
- **WriteFile**

The DeviceIoControl service uses IOCTRL codes reserved for user and customer devices. These are defined in the header file provide with the provided kit. The KSC 2115 does not fit well into the NT file and I/O subsystem. It is not a real file structured device. To utilize its functionality and to acquire the maximum benefit from the device, a mapping of functionality was done.

8.2 NT devices

There are twenty (0 through 19) NT devices created when the NT driver is loaded. The purpose of sixteen of the devices is to map the KSC 2115 command list memory into eight partitions. This allows the user to store a command list in any of the partitions and then execute it. The purpose and mapping of each of the devices is as follows:

- KCA00- Used for control functions in particular the setting and reading partitions and status information. It is also used for small buffers that utilized programmed buffered I/O.
- KCA01- Used for reading Demands

- KCA02- Used for getting Buffer Completion flags
- KCA03- For loading command list into partition one
- KCA04- Executing command lists, and normal transfers using command list in partition one
- KCA05- For loading command list into partition two
- KCA06- Executing command lists, and normal transfers using command list in partition two
- KCA07- For loading command list into partition three
- KCA08- Executing command lists, and normal transfers using command list in partition three
- KCA09- For loading command list into partition four
- KCA10- Executing command lists, and normal transfers using command list in partition four
- KCA11- For loading command list into partition five
- KCA12- Executing command lists, and normal transfers using command list in partition five
- KCA13- For loading command list into partition six
- KCA14- Executing command lists, and normal transfers using command list in partition six
- KCA15- For loading command list into partition seven
- KCA16- Executing command lists, and normal transfers using command list in partition seven
- KCA17- For loading command list into partition eight
- KCA18- Executing command lists, and normal transfers using command list in partition eight
- KCA19- Load and Execute a Load and GO command list

8.3 DeviceIoControl functions

The file KSCIOCTL.H contains the IOCTL codes that may be passed to the NT driver. The functionality and buffer contents are defined below. These particular IOCTL codes are only valid for the KCA0, KCA1, and KCA 2 devices. The DeviceIoControl function takes the input buffer and moves it to a system non-paged pool buffer and calls the driver. The driver does the operation and if it is to return data, places the data into another non-paged pool buffer that is then copied to the user buffer.

This buffer copying is not desirable for large high speed transfers but is actually faster for small data transfers as it takes less time to set up the DMA and lock down user buffers than to use those already locked down in the system space. The driver uses this method for programmed I/O of the KSC 2115. In particular the API uses this for single small transfers.

Device KCA00- Control device

- KSC_PARTITION- Set the partition table
- KSC_TIMEOUT- Set the time out for a partition
- KSC_TIMERSET- Sets the timer for a clocked command list
- KSC_2115RESET- Reset the device
- KSC_ID- Return the current release of the driver
- KSC_COUNTERS- Return counters for the driver
- KSC_RDPARTABLE- Read the current partition table
- KSC_ERRREG- Read the last status and error information for a partition

Device KCA01- Demand device

- KSC_DMDREAD- Read any demands currently in the device adapter.

Device KCA02- Buffer Complete device

- KSC_BUFCOMPLETE- Read any buffer completion flags from the driver. The user must have a repeating buffer function active on KCA03.

8.3.1 ReadFile and WriteFile Operations

To transfer large amounts of data, the remaining KCA(03 to 19) devices should be used. These devices are accessed using the NT ReadFile and WriteFile system service calls. The KCA19 device is unique in that it assumes that the user has built a combined buffer that contains the command list and the actual buffer (see earlier description of the LoadGo buffer). Access to these devices results in the user buffer being locked into memory and DMA being transferred to and from the KSC 2115 directly. This method provides the maximum through put for the device. Additionally, it is the only method which will support the segmented buffered mode.

8.3.2 Buffers

All buffers must be long word aligned (e.g. on a 32 bit boundary). Additionally, all output buffer must have space for pipeline requirements of the device (8 long words or 32 bytes).

Some of the IOCTL calls indicate a Read operation when in effect they are being used for a write operation. Since the Read operation means that the driver must be able to write to indicated address space, this of less protection than a read, therefore, the buffer should also be readable by the driver.

The LoadGo buffer is a combined buffer that contains both the command list to be loaded and the buffer. The first long word of the buffer contains the size of the command list in the lower 24 bits and the partition number in the upper 8 bits. If the user does not specify a partition, then partition one is used by the driver. The command list follows the command list size, followed by storage for the actual buffer which is either read from for a write from the host to the 2115 or written to for a read from the 2115.

8.3.3 KSC_PARTITION- Set the partition table

The user buffer will be buffered. The IOCTL dispatch code can simply populate the partition table within the CBF. The buffer will contain an array of eight long words that specify the length of each of the partitions.

8.3.4 KSC_TIMEOUT- Set the time out for a partition

The user buffer will be buffered. The IOCTL dispatch code can simply populate the partition table for the indicated partition. The buffer will contain the timer value and the partition number both as long words.

8.3.5 KSC_TIMERSET- Set the device internal timer

This will set the value to be used when the user uses the internal clock of the device. The buffer contains the timer value to load into the device.

8.3.6 KSC_2115 RESET- Reset the device

This control code contains no data. It simply does a reset on the device. Any outstanding I/O is terminated. This dispatch code will queue the request to the driver as the controller must be acquired.

8.3.7 KSC_ID- Return the current release of the driver

The IOCTL dispatch code will populate the user's buffer with a long word containing the current release of the driver.

8.3.8 KSC_COUNTERS- Return counters for the driver

The IOCTL dispatch code will populate the user's buffer with the current counters from the CBF.

8.3.9 KSC_RDPARTABLE- Read the current partition table

The IOCTL dispatch code will populate the user's buffer with eight long words describing the current partition layout of the command list memory of the device

8.3.10 KSC_ERRREG[1-8]- Read the last status and error information for a partition

The IOCTL dispatch code will populate the user's buffer with the status information maintained for a particular partition.

8.3.11 KSC_DMDREAD- Read any demands currently in the device adapter.

The DeviceIoControl dispatch code will queue the IRP to the particular demand device. The demand device will then execute the STARTIO entry of the demand device. The STARTIO entry of the demand device will then indicate that there is an user buffer for demands and store the size of the user's demand buffer. It will then enable demand interrupts on the device and wait on a semaphore. If there are demands or when demands arrive, a demand interrupt will be triggered. The DPC (Deferred Procedure Call) for the demands will be requested. It will unload as many demands as possible into the user's demand buffer and trigger the semaphore. Due to timing, it is possible that semaphore may be already signaled before the STARTIO routine begins the wait if there were demands already within the device Demand FIFO.

The 2115 only generates an interrupt when the demand FIFO goes from empty to non-empty. Therefore, if the FIFO is not empty, the Demands must be unloaded.

8.3.12 KSC_BUFCOMPLETE- Read any buffer completion flags

The IOCTL dispatch code will queue the IRP to the buffer device if there are no current buffers completed, otherwise the IOCTL dispatch code will return the flags immediately. The buffer device STARTIO entry will then wait for a semaphore to be triggered. The DPC for buffered I/O will then set the semaphore when a new buffer has been filled. Due to timing, it is possible that semaphore may be already signaled before the STARTIO routine begins the wait if a buffered operation fills a new buffer segment.

8.3.13 KSC_ACKBUFCOMPETE- Acknowledge the processing of the buffer completion

The IOCTL dispatch code will queue the IRP to the buffer device. The buffer device STARTIO will raise IPL, capture a device spinlock, and mark the particular buffers as free. It will then do the I/O completion.

8.3.14 DMA Considerations

NT will provide mapping registers to the device driver. The number of map registers available may limit the size of a DMA transfer. Normally a driver would then execute multiple DMA transfers to accommodate the complete buffer. However, due to the design of the device, this is not feasible. Therefore, the user may receive a DMA size too large status code.

For the LoadGo buffers that are sent to KCA19, the buffers are mapped in both system space and mapped for DMA. This is required since the command list is loaded into the device using programmed I/O.

8.4 Status Returns

The NT WriteFile and ReadFile system service calls provide either a TRUE or FALSE return status and a byte count regarding the read or write operation. If an error is encountered, the application thread may call GetLastError to return the status from the device driver. If the user is doing overlapped I/O and has multiple threads, it is unclear if this is sufficient to get the desired error. The driver supports additional calls to get more detailed information from the driver.

For the CAMAC library, if the list generates a fault, the CAMAC library API will attempt to request the driver for the status of the last execution. The CSR and the current memory address of the list when the list faulted can be useful to determine list faults. The 2115 typically points to the next location of the command list after the list instruction that caused the stop. Depending on the coding of the command list, the stop may be a result of a No-Q, No-X, a Halt Instruction, or a faulty list. Because there is not a good way to acquire sufficient status from the driver with each NT native call, the CAMAC APIs must do a second request from the driver to acquire the status. It is possible and likely if more than a single process is using the device that the status will be overwritten by a subsequent request before the extended status is acquired. Therefore, users who have used the Status buffer will find that the buffer will not be accurate. The overall status and the first word of the Status buffer will always be accurate as it is a reflection of the current I/O. This has impact on the CAB16, CAB24, CACTRL, CAM24, CAM16, and their variants.

Lists that generate or sink less data than expected require examination of the command list itself. The 2115 does not give a status buffer which was available on other KSC devices.

8.5 Demands and LAMS

The CAMAC crates on the CAMAC highway have the ability to generate LAMS which are translated to a demand and stored in the Demand FIFO of the 2115. The 2115 can also generate demands as a result of list execution. Because the servicing of a LAM requires that a read be done to determine what has card within the chassis is requesting the LAM, the processing of the LAMs has been migrated to a Demand Process. The user is notified that a LAM is present using a read from an NT Pipe.

The actual read of the demands as documented earlier is done using the device: kca01. The demands are queued within the 2115 until a user process does a device control function. The driver captures the spinlock for the access to the device registers. It then check to see if there are any currently pending. If not, a flag is set indicating that a demand interrupt is expected and the demand interrupt is enabled. If there are demands, then a maximum number of demands will be removed from 2115 and returned to the user. The extraction of the demand interrupts is done with interrupt lockout and therefore, a maximum of twenty five demands will be extracted at any one time such that the system does not experience any degradation.

8.6 MultiBuffer Considerations

The multibuffer functions of the 2115 allow the user to create lists that are can be clocked by either an external or the 2115 internal clock. The advantage of the multibuffer is that the user's DMA buffer need not be locked down more than once. The notification of each segment of the user buffer is via returned to the user via the KCA02 device. As each segment is processed by the user the driver must be informed. Failure to do so, will result in a multibuffer overflow. The driver may report more than a single multibuffer segment completion per read on KCA02 device.

8.7 NT Limitations

When NT delivers an IRP packet to the device driver, the packet is not cancelable. There are two special IOCTRL codes that will cancel either a Demand Read or a Multibuffer read request. The special utility RESETDRIVER is provided to reset the driver and to cancel these types of requests. Users should add this to their process exit handling.

9. CAMAC Error Codes

The driver and language interface routines perform various checks on both the parameters passed by the calling program and the operation of the hardware. When an error is detected, these routines return an error code to the calling program. This appendix contains a list of error numbers and an explanation of the error. Many errors can only be generated by improper calls to the Advanced Fortran Routines. These errors are designated by the phrase (*advanced Fortran routines*).

101. The version number of the driver does not match the version number found in the Header. Check to make sure all software is at the same version number.

102. The length of the Data Buffer is greater than the specified size of the Data Buffer (*advanced Fortran routine*).

103. The Header size does not match the Header size of the current version (*advanced Fortran routines*).

104. The length of the CAMAC Control List is greater than the specified size of the CAMAC Control List (*advanced Fortran routines*).

105. The Status Buffer size does not match the Status Buffer size of the current version (*advanced Fortran routines*).

106. The process does not have either read or write access to the Data Buffer. Check that the Data Buffer has been properly declared.

107. The System does not have enough contiguous Real Time Page Table Entries to double map the Data Buffer. The number of Real Time Page Table Entries can be changed by modifying the Sysgen parameter REALTIME_SPTS.

108. The process does not have a big enough Working Set to lock down the Data Buffer. The Working Set size can be changed by modifying the Authorize parameter WSquo.

109. Unknown VMS error while trying to lock the CAMAC Control List into memory.

110. Unknown VMS error while trying to lock the Data Buffer into memory.

111. Unknown VMS error while trying to lock the Status Buffer into memory.

112. The CAMAC Control List does not have enough space at the end for the CAMAC driver to insert a number of halt instructions. The length of the CAMAC Control List must be four long words less than the size of the CAMAC Control List so four Halt instructions can be added.

113. The Data Buffer has a length of zero but must have a length of at least one. A dummy word must be entered into the Data Buffer (Header(DatLen)=1) (*advanced Fortran routines*).

114. The driver does not have read access to the Header. Check that the Header has been properly declared.

115. The size of the Header is over 64K words. Check that the size of the Header has been declared as a long word (IINTEGER*4 variable) (*advanced Fortran routines*).

116. The process does not have either read or write access to the CAMAC Control List. Check that the CAMAC Control List has been properly declared.

117. The System does not have enough contiguous Real Time Page Table Entries to double map the CAMAC Control List. The number of Real Time Page Table Entries can be changed by modifying the Sysgen parameter REALTIME_SPTS.

118. The process does not have a big enough Working Set to lock down the CAMAC Control List. The Working Set size can be changed by modifying the Authorize parameter WSquo.

119. The length of the CAMAC Control List is over 64K words. Check that the variable specifying the length of the CAMAC Control List has been declared as a long word (INTEGER*4 variable) (*advanced Fortran routines*).

120. The CAMAC Control List does not fit in one segment. The CAMAC Control List plus the CAMAC Control List offset cannot fit within one segment (IBM PC only).

121. The size of the CAMAC Control List is over 64K words. Check that the variable specifying the size of the CAMAC Control List has been declared as a long word (INTEGER*4 variable) (*advanced Fortran routines*).

122. The length of the CAMAC Control List is over 32K-1 words. The largest CAMAC Control List allowed is 32K-1 words (*advanced Fortran routines*).

123. The CAMAC Control List has a size of zero but must have a size of at least one (*advanced Fortran routines*).

124. The process does not have either read or write access to the QXE Buffer. Check the address and the size of the QXE Buffer in the Header.

125. The System does not have enough contiguous Real Time Page Table Entries to double map the QXE Buffer. The number of Real Time Page Table Entries can be changed by modifying the Sysgen parameter REALTIME_SPTS.

126. The process does not have a big enough Working Set to lock down the QXE Buffer. The Working Set size can be changed by modifying the Authorize parameter WSquo.

127. The QXE Buffer does not fit in one segment. The QXE Buffer plus the QXE Buffer Offset cannot fit within one segment (IBM PC only).

128. The size of the QXE Buffer is over 64K words. Check that the variable specifying the size of the QXE Buffer has been declared as a long word (INTEGER*4 variable) (*advanced Fortran routines*).

129. The size of the QXE Buffer is over 32K-1 words. The largest QXE Buffer allowed is 32K-1 words (*advanced Fortran routines*).

130. The process does not have either read or write access to the Status Buffer. Check that the Status Buffer has been properly declared.

131. The System does not have enough contiguous Real Time Page Table Entries to double map the Status Buffer' The number of Real Time Page Table Entries can he changed by modifying the Sysgen parameter REALTIME-SPTS.

132. The process does not have a big enough Working Set to lock down the Status Buffer. The Working Set size can be changed by modifying the Authorize parameter WSquo.

133. The size of the Status Buffer is over 64K words. Check that the variable specifying the size of the Status Buffer has been declared a long word (INTEGER*4 variable) (*advanced Fotran routines*).

134. The process does not have either read or write access to the Word Count Buffer. Check the address and the of the Word Count Buffer in the Header.

135. The System does not have enough contiguous Real Time Page Table Entries to double map the Word Count Buffer. The number of Real Time Page Table Entries can be changed by modifying the Sysgen parameter REALTIME_SPTS.

136. The process does not have a big enough Working Set to lock down the Word Count Buffer. The Working Set size can be changed by modifying the Authorize parameter WSquo.

137. The WC Buffer does not fit in one segment. The WC Buffer plus the WC Buffer Offset cannot fit within one segment (*IBM PC only*).

138. The size of the WC Buffer is over 64K words. Check that the variable specifying the size of the WC Buffer has been declared as a long word (INTEGER*4 variable).

139. The size of the WC Buffer is over 32K-1 words. The largest WC Buffer allowed is 32K-1 words.

140. Unknown VMS error while trying to lock the Word Count Buffer into memory.

141. Unknown VMS error while trying to lock the (M Buffer into memory.

201. An illegal command was found in the CAMAC Control List (*advanced Fortran routines*).

202. An In-Line CAMAC read was specified. Only CAMAC write and control functions can be specified in an In-Line CAMAC Control List command (*advanced Fortran routines*).

203. An illegal LAM type was specified, the command types are zero through seven (*advanced Fortran routines*).

204. A block transfer CAMAC control function was specified. Only CAMAC read and write functions can be specified for block transfer CAMAC Control List commands (*advanced Fortran routines*).

205. The remainder of the Data Buffer is too small to hold the data for the CAMAC block transfer (*advanced Fortran routines*).

206. An illegal CAMAC word size for the CAMAC device was encountered (*advanced Fortran routines*).

207. Block transfer timeout. The CAMAC software driver has timeout because the CAMAC hardware has not responded.

208. Block transfer timeout. The CAMAC software driver has timeout because the CAMAC hardware has not responded.

209. Bad interrupt mode (*advanced Fortran routines*).

210. The QIO request was in some way canceled.

211. Out of data error. The Data Buffer was not big enough to hold or accept the data for the single naf.

212. Error in purging the data-path.

213. Single transfer timeout. The CAMAC software driver has timeout because the CAMAC hardware has not responded.

214. Single transfer timeout. The CAMAC software driver has timeout because the CAMAC hardware has not responded.

215. Error in allocating a data-path

216. Error in allocating mapping registers.

217. Error in purging the data-path.

218. Error in purging the data-path.

219. No PHYIO privileges, PHYIO privileges are needed for the operation.

220. Error in purging the data-path.

221. Power failure error.

222. The CAMAC Control List could not hold the enter LAM command.

223. The CAMAC driver could not allocate enough system memory to book the LAM request.

224. Illegal CAMAC crate. The CAMAC crate is probably off-line.

301. Invalid crate number during a CAMAC block transfer operation. The specified crate is not online.

302. An N greater than 23 error has occurred during a CAMAC block transfer operation.

303. A CAMAC NO-Q error has occurred during a CAMAC block transfer operation.

304. CAMAC no-sync error during a CAMAC block transfer operation.

305. A CAMAC NO-X error has occurred during a CAMAC block transfer operation.

306. A CAMAC non-existent memory error has occurred during a block transfer operation.

307. A CAMAC STE-error has occurred during a CAMAC block transfer operation.

308. A CAMAC timeout error has occurred during a CAMAC block transfer operation.

309. An undefined CAMAC error has occurred during a CAMAC block transfer operation.

310. Invalid crate number during a CAMAC single transfer operation. The specified crate in not online.

311. An N greater than 23 error has occurred during a CAMAC NAF operation.

312. A CAMAC NO-Q error has occurred during a CAMAC NAF operation.

313. A CAMAC STE - error during a CAMAC single transfer operation.

314. A CAMAC NO-X error has occurred during a CAMAC NAF operation.

315. A CAMAC non-existent memory error has occurred during a single transfer operation.

316. A CAMAC STE-error has occurred during a CAMAC single transfer operation.

317. A CAMAC timeout error has occurred during a CAMAC NAF operation.

318. An undefined CAMAC error has occurred during a CAMAC NAF operation.

401. Access violation, either the I/O status block cannot be written by the caller, or the parameters for device-dependent function codes are incorrectly specified.

402. The specified device is offline and not currently available for use.

403. Insufficient system dynamic memory is available to complete the service. There are probably no free IRPS, use SHOW MEMORY to see the number of free IRPS.

404. An invalid channel number was specified.

405. The specified channel does not exist, was assigned from a more privileged access mode, or the process does not have the necessary privileges to perform the specified functions on the device.

406. The QIO error is unknown to the CAMAC software.

501. Access violation, the device string cannot be read by the caller, or the channel number cannot be written by the caller.

502. The CAMAC device is allocated to another process.

503. Illegal device name. No device name was specified, the logical name translation failed, or the device string contains invalid characters.

504. The device name string has a length of 0 or has more than 63 characters.

505. No I/O channel is available for assignment.

506. The specified CAMAC device does not exist. Check the device string for misspellings or a missing colon and check that the device driver has been loaded.

507. The process tried to assign a CAMAC device on a remote node. CAMAC operations cannot be performed over a network.

508. The CAOPEN error is unknown to the CAMAC software.

601. An invalid channel number was specified.

602. The specified channel is not assigned or was assigned from a more privileged mode.

603. The CACLOS error is unknown to the CAMAC software.

701. An invalid CAMAC subaddress (A) was found. The CAMAC subaddress was either less than 0 or greater than 15 ($A < 0$ or $A > 15$).

702. Invalid mode byte. The mode byte for the Advanced Fortran routines is invalid (advanced Fortran routines).

703. A invalid CAMAC block transfer type was found. The legal block transfer types are QSTP, QIGN, QRPT, and QSCN with corresponding values of 0, 8, 16, and 24, respectively.

704. An invalid CAMAC function code (F) was found. The CAMAC Function code was either less than 0 or greater than 31 ($F < 0$ or $F > 31$).

705. An invalid CAMAC crate controller function was found. The valid CAMAC crate controller functions are INIT, CLEAR, SETINH, CLRINH, and ONLINE with corresponding values of 0, 1, 2, 3, and 4, respectively.

706. An invalid CAMAC slot number (N) was found. The slot number was either less than 1 or greater than 30 ($N < 1$ or $N > 30$).

707. Invalid LAM type (*advanced Fortran routines*).

708. Invalid priority (*advanced Fortran routines*).

709. A CAMAC block transfer control operation was specified which is invalid. Only CAMAC Read or Write block transfers are allowed. The function code (F) for the block transfer was either between 8 and 15 inclusive or between 24 and 31 inclusive ($8 \leq F < 15$ or $24 \leq F \leq 31$).

710. An in-line CAMAC read was specified. Only in-line CAMAC control and write operations are legal (F8 through F31) (*advanced Fortran routines*).

711. The Data Buffer is not big enough to hold all the data for the CAMAC Control List (*advanced Fortran routines*).

712. The CAMAC Control List is not big enough to hold all the commands (*advanced Fortran routines*).

713. A CAMAC block transfer with a block size of zero was found. A CAMAC block transfer must have a size of at least one word.

714. Illegal CAMAC crate number.

10. Installation

The device driver is installed using the InstallShield product. The user only needs to determine in what directory the device driver this product should be placed into. Normally, this software is placed in: C:\KCAxxx] (xxx= product release and version number). Depending if the kit was acquired via an FTP site or was distributed on floppies, use the normal NT installation procedure.

10.1 Directory Structure

The following documents the contents of each of the sub-directories.

\DRIVER	Contains Driver Image
\API	Contains API Shareable image (KSCAPI.LIB)
\CAMAC_UTIL	CAMAC utilities
\TEST	Simple test programs
\EXAMPLES	Example source code
\DEMAND	Demand Process and configuration file
\INCLUDE	Include files
\DOCUMENTATION	Contains this document and release notes

10.1.1 Include Files

The following is a description of the include files provided in the include directory.

C header files

- CAMAC.H- Contains CAMAC specific parameters
- CAMERR.H- Defines all of the CAMAC library error codes
- CMDLIST.H- Contains the macros and definitions used for building load and go command lists. The user must define the symbol: "KSCADP_SH" to select the generation of the correct command list generation.
- KERRORS_MSG.H- Defines all of the KSC API error codes
- KSC_API.H - Contains prototypes for the KSC API library
- KSC_GENLIST.H- Contains prototypes and definitions for the KSC list building routines
- KSC_HANDLE.H- Contains the definition of the KSC API handle
- KSCIOCTL.H- Defines all of the IOCTL codes for the 2115 NT device driver
- KSCUSER.H- Contains all of the CAMAC library prototypes and list building header

FORTRAN include files

- CAUSER.INC- Contains FORTRAN function prototype (FORTRAN 77), header definition, parameters, and CAMAC error codes.
- KERRORS_MSG.INC- Contains FORTRAN error codes for the KSC API.
- KSC_API.INC- Contains FORTRAN definition for the KSC API.
- KSC_GENLIST_2115.INC- Contains prototype and definitions for the KSC API list building routines.

10.2 Post Installation

The 2115 is automatically configured by the POST (Power Up and Self Test) of the PCI based processor upon system bootstrap. This configuration information is then used by the driver to determine IRQ levels and bus address space requirements. The user has a choice of requesting that the device driver be loaded upon bootstrap by placing an entry into the Windows NT startup window. The driver may also be automatically loaded by running REGEDT32.EXE to modify the NT registry. Modify the value of *Start* from 3 to 1 under the following registry tree.

HKEY_LOCAL_MACHINE
└ SYSTEM
└ CurrentControlSet
└ Services
└ KSC2115

The driver may be manually started by entering NET START KSC2115 at the command prompt. If the driver loads successfully the message "The KSC2115 service was started successfully." is displayed. The driver may be manually stopped by entering NET STOP KSC2115 at the command prompt. If the driver unloads successfully the message "The KSC2115 service was stopped successfully." is displayed.

If the driver did not load successfully event messages are logged to the Windows NT Event Viewer. The Event Viewer may be found under "Start Menu", "Programs", "Administrative Tools", "Event Viewer". Messages relevant to the KSC2115 are identified under the "Source" column. Hopefully this event logging will be helpful in determining the cause of the problem.

Event logging may be turned off by modifying the value of *EventLogLevel* from 2 to 0 under the following registry tree.

HKEY_LOCAL_MACHINE
└ SYSTEM
└ CurrentControlSet
└ Services
└ KSC2115
└Parameters

10.3 Program Groups

The Window Program Group "KSC CAMAC2115" is created in the program manager. The following sub-groups are also created off the main program group.

- **CAMAC UTILITIES**
  - **CAMAC_COMMANDS**- Runs the CAM utility program
  - **CAMAC CONTROL**- Runs the CCTRL utility program
  - **CRATE STATUS**- Runs the CCSTS utility program
- **DEMAND PROGRAM**
  - **Demand Process**- Starts the Demand Process
  - **Demand Status**- Start the Demand Status process
  - **Edit Configuration File**- Calls up note pad to allow user to edit the configuration file
  - **Start Driver and Demand Process**- This will load the 2115 device driver and request the execution of the Demand process
  - **Stop Driver**- Unloads the 2115 device driver
- **TEST PROGRAMS**
  - **LAM3473**- Example LAM program that requires a 3473 change of state card
  - **TEST_API**- Example test program that tests the KSC API library
  - **TEST_CAMAC**- Example test program that tests the CAMAC library calls
  - **TEST_DEMAND**- Test program that tests the demand functioning
- **README**- Calls up note pad to read the read me file.

Index

A

autoexec.bat · 195

C

CAB16 · 23
CAB16E · 27
CAB24 · 13, 30
CAB24E · 33
caBLK · 64, 66
CACLOS · 13, 36, 54
CACTR · 38
caEBLK · 68, 70
caEXEC · 71
caEXEW · 73
caHALT · 75
caINAF · 76, 77
caINIT · 63, 64, 68, 71, 73, 75, 76, 79, 82, 83, 85
CALAM · 21, 40, 44
CAM CAMAC Utility · 123
CAM16 · 13, 17, 46
CAM24 · 49
CAMAC command lists · 63
CAMAC.H · 194
CAMERR.H · 194
CAMSG · 52
caNAF · 83, 85
CAOPEN · 13, 15, 36, 53, 54, 71, 73
CAUSER.INC · 15, 194
CCL · 63
CCSTAT · 55
CMD_LIST_TYPES · 93, 94
CMDLIST.H · 194
CXLAM · 58

D

DEMAND process · 21, 44
Demand Process Pipe · 174
demands · 9
DMDREGION · 172
DMDSTS · 175, 178

E

END_EXEC_LIST · 95
END_READ_LIST · 95
END_WRITE_LIST · 95

G

GetLastError · 184

I

Installation · 194

K

KCAPI.OLB · 22
KCAUSER.H · 15
KERRORS_MSG.H · 194
KERRORS_MSG.INC · 195
KSC_ALLOC · 148
ksc_api.h · 126
KSC_API.INC · 195
KSC_bdcast_trigger · 98
KSC_case · 99, 106, 107, 119
KSC_demand_read · 128
KSC_display_partitions · 131
KSC_dump_list · 86, 101
KSC_else · 103, 104, 106, 108, 112
KSC_enable_demand · 176
KSC_ENABLE_EVENT · 174
KSC_end_list · 105
KSC_end_sublist · 106
KSC_endcase · 99, 106, 107, 119
KSC_endif · 104, 106, 108, 112
KSC_exec_clocked_list) · 157
KSC_exec_list · 145
KSC_exec_rlist · 140
KSC_exec_wlist · 143
KSC_finish · 86, 109, 113, 114
KSC_gen_demand · 110
KSC_GENLIST.H · 194
KSC_GENLIST_2115.INC · 195
KSC_get_failure · 145
KSC_handle · 176
KSC_HANDLE.H · 194
KSC_if · 104, 108, 111
KSC_Init · 126, 127, 148, 176
KSC_init_list · 86, 98, 99, 103, 105, 106, 107, 108, 109, 110, 111, 113, 115, 117, 118, 119, 121
KSC_load_cmdlist · 153, 161
KSC_load_test_val · 115
KSC_loadgo · 127, 151
KSC_mark_list · 117
KSC_mbuf_done · 156, 164, 165
KSC_OPENERROR · 148
KSC_print_symbolic · 126, 127, 171
KSC_read_cmdlist · 159
KSC_read_multibuf · 138, 156, 157, 170

KSC_set_partitions · 161, 166
KSC_set_timeouts · 168
KSC_store_flag · 118
KSC_SUCCESS · 149
KSC_switch · 119
KSC_time_stamp · 121
kscapi.h · 86
KSCIOCTL.H · 194
kscuser.h · 63, 194

L

LAM · 44
libgiapi.a · 86
Linker Requirements · 22
List Building Routines · 63

M

Microsoft Visual C++ · 15

R

ReadFile · 184
RESETDRIVER · 185

S

START_LIST · 94
status array · 11, 18, 19
Status buffer · 184

T

TEST_API · 127

W

WriteFile · 184