KineticSystems Company, LLC
AD3Z-2PA1
2915 Device Driver and API Library
for Windows 2000
User's Manual

March 14, 2003

KineticSystems Company, LLC

2915 Device Driver and API Library for Windows 2000

900 N. State Street, Lockport, Illinois 60441 (815) 838-0005 (815) 838-4424

Device Driver & API Library for Windows 2000

2915 PCI Interface

Document Revision: Friday, March 14, 2003

Software Version: 1.0

Operating System: Microsoft Windows 2000

Contents

1 HARDWARE AND SOFTWARE INSTALLATION 1

1.1 Directory Structure 1
1.2 Header Files 2

2 CAMAC COMMAND LINE UTILITIES 3

2.1 Command Summary 3
2.2 CACTRL CAMAC Utility 3
2.3 CAM CAMAC Utility 4
2.4 CCSTAT CAMAC Utility 5

3 CAMAC APPLICATION PROGRAMMING INTERFACE (API) 6

3.1 CAMAC Library Call Summary 6
3.2 Initialization Calls 7
3.3 Single-Action Data Transfer Calls 7
3.4 Block Transfer Calls 7
3.5 Block Transfer Mode 7
3.5.1 Q-Mode Transfers 7
3.5.2 Data Word Size 8
3.6 Status Array 9
3.7 Asynchronous Event Handling (LAMS) 10

4 CAMAC APPLICATION PROGRAMMING INTERFACE (API) 12

4.1 CAMAC Operational Routines 12
4.1.1 cab16 12
4.1.2 cab24 15
4.1.3 caclos 18
4.1.4 cactrl 20
4.1.5 calam 22
4.1.6 cam16 28
4.1.7 cam24 30
4.1.8 camsg 33
4.1.9 caopen 35
4.1.10 ccstat 37
4.1.11 cxlam 39
4.1.12 camlookupmsg 44
4.2 CAMAC List Building Routines 46
4.2.1 cablk 47
4.2.2 caexec 51
4.2.3 caexew 55
4.2.4 cahalt 58
4.2.5 cainaf 61
4.2.6 cainit 64
4.2.7 canaf 68

5 DEMANDS 72

5.1 The Demand Process 72
5.2 Demand Configuration File 72
5.3 Application Registration for Demands 73
5.4 Demand Processing 73
5.5 User Application Program 74
5.6 Demand Process Dataflow 75
5.7 Demand Utilities 76
5.7.1 Program DMDSTS 76

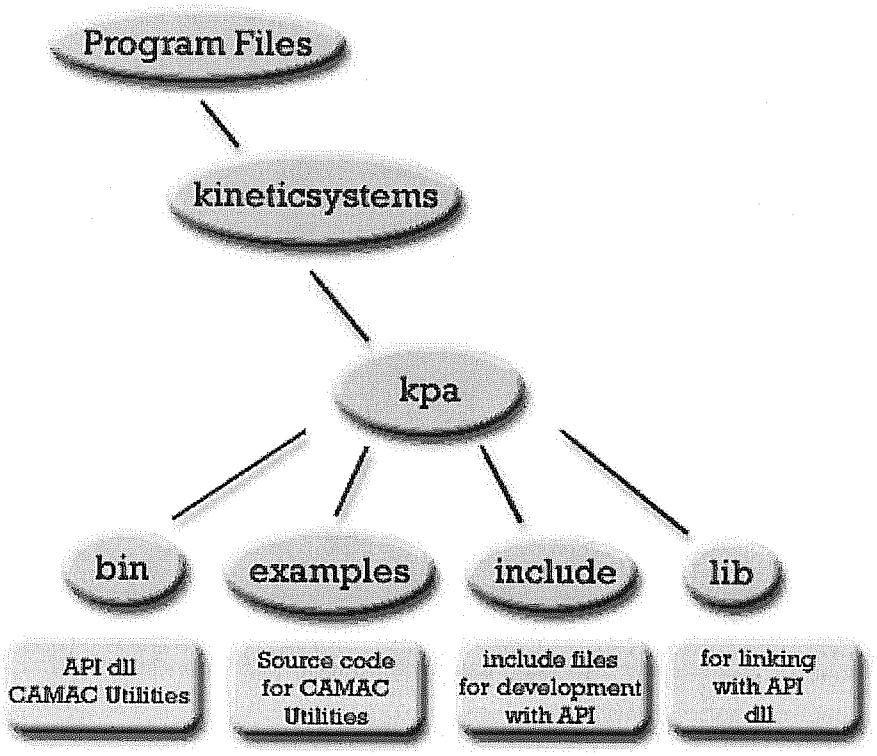
6 APPENDIX A - CAMAC ERROR CODES 77

1 Hardware And Software Installation

The Windows Hardware Manager installs the 2915 driver, API library and associated files. Power down your PC, and insert the 2915 into an empty PCI slot. When the PC is powered back up, Windows will detect the newly installed hardware and invoke the Hardware Manager. At this point, you should insert the provided software disk into your computer, and follow the Hardware Manager dialogs through the installation procedure. After the Hardware Manager indicates the new hardware has been successfully installed, remove the disk and store it in a safe place.

1.1 Directory Structure

The installation procedure will have installed the 2915 software driver and the API (Application Programming Interface) package. The API can be found on your computer's root drive, under "Program Files\kineticsystems\kpa". The package is organized into several folders:



- **bin** This folder contains the API dll (kpaapi.dll), the Demand process (dmdproc.exe), the Demand statistics package (dmdsts.exe), an example demand configuration file (demand.cfg), and 3 simple CAMAC utilities (cam.exe, cactrl.exe, and ccstat.exe). The Demand and utilities software is described in different portions of this manual.

  Software developed for the 2915 will use kpaapi.dll. When you run your own applications, the system must find this file; you may wish to copy it into a system location (such as the WINNT directory on your root drive) or update your environment's PATH variable to include its location. The system will also find it if it is in the same directory as your application.

- **examples** This folder contains the source code for the CAMAC utilities. These simple programs can be used as templates on which to build your own applications.

- **include** This folder contains the headers you will need to compile your applications. More information is provided in the next section.

- **lib** this folder contains the kpaapi.lib file that you will need to link to the kpaapi.dll. Consult your development environment's compiler documentation for instructions on how to include this file in your project.

1.2 Header Files

Various include files are provided as part of the API package. Some files will need to be included in every project, some only in projects that make use of certain types of functionality (i.e., demand processing). Some files are required only by the headers themselves, and do not need to be directly included in your project.

- **ksc_api.h** This is the primary include file for the API, and will need to be included in every 2915 application. It should be the first of the kpa include files included in your code.
- **ksc_handle.h** This file contains definitions for managing the API handle obtains from routines such as caopen(). Every call to the API will require a handle, so this file should be included in every project.
- **kscuser.h** This file contains various definitions and function templates for the API.
- **camerr.h** this file contains error codes specific to CAMAC returned by the api.
- **kerrors_msg.h** This file contains error codes general to the API.
- **ksc_demands.h** This file is used for demand processing, and contains definitions required for interfacing with the demand mechanism.
- **ksc_genlist.h** This file is used for the various list building routines (i.e., cablk(), cainaf(), etc.).

2 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. 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 allows 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.

2.1 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.

2.2 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 7). 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 dataway 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
```

2.3 CAM CAMAC Utility

The CAM utility allows the user to do simple CAMAC operations. This utility should be used with caution as it performs 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 7). 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 to 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 an "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

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

2.4 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 7). 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

3 CAMAC Application Programming Interface (API)

This section contains a detailed description of the various functions associated with the 2915 Application Programmer Interface (API). This set of functions allows an application program to access the full functionality of the 2915 PCI hardware interface for performing CAMAC operations. A basic set of functions is provided for executing both single transfer and block transfer operations. Additionally, a set of CAMAC Command List Building functions is provided to execute a Command List. A CAMAC Command List is a user-constructed sequence of CAMAC commands that is passed to the 2915 driver with one function call. The goal of the Command List is to improve data throughput by eliminating system calls for each command. Instead, one system call is made and many commands can be executed. The 2915 Command List is implemented at the driver level.

3.1 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 five groups:

Initialization calls

caopen (chan, device, StatusArray)
caclos (chan, StatusArray)

Single-Action Data Transfer Calls

cam16 (chan, C, N, A, F, data, StatusArray)
cam24 (chan, C, N, A, F, data, StatusArray)

Block Transfer Calls

cab16 (chan, C, N, A, F, mode, DataArray, TransCount, StatusArray)
cab24 (chan, C, N, A, F, mode, 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)

3.2 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 (2915) to be accessed by the program and should

not be called again until the channel has been closed. The returned pointer from the *caopen* function points to a KSC API handle, which is allocated when the 2915 is opened.

3.3 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.

3.4 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 2915 PCI Interface. Use these routines for reading or writing blocks of data between computer 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.

The maximum size of a block transfer operation is 32 Kbytes. If block transfers of larger that 32 Kbytes need to be performed, multiple CAMAC operations must be performed.

3.5 Block Transfer Mode

3.5.1 Q-Mode Transfers

The 2915 hardware supports four Transfer Mode that aid in transferring data to or from a CAMAC module. These Transfer Mode rely on the CAMAC Q-response for determining data validity and whether subsequent operations are to be performed.

The four modes supported by the 2915 include Q-Ignore, Q-Stop, Q-Repeat and Q-Scan. The following describes each transfer mode.

The Q-Stop Block Transfer Mode continues to transfer a block of CAMAC write or read data as long as valid Q-responses (Q=1) are received and the transfer count has not been exhausted. If a No-Q condition occurs during a CAMAC block write operation, the transfer terminates immediately. If t No-Q condition occurs during a CAMAC block read operation, the transfer terminates as soon as the last valid (Q=1) data word is transferred to PCI memory.

The Q-Ignore Block Transfer Mode continues to transfer data to or from a CAMAC module as long as the transfer count is not exhausted. The 2915 ignores the CAMAC Q-response during the transfer of data.

The Q-Repeat Block Transfer Mode continues to transfer data until the transfer count is exhausted or a Q-Repeat timeout occurs. During a Q-repeat operation, the dataway cycle for the current data word is repeated until a CAMAC Q-response of one is returned.

For CAMAC write operations, a CAMAC dataway cycle is executed for each data word to be written until a CAMAC Q-response of one is received. After a Q=1 is received, the next CAMAC write data word is retrieved and the process repeats.

For CAMAC read operations, a CAMAC dataway cycle is repeated until a CAMAC Q-response of one is received. After receiving the Q=1 response, the PCI interface stores the data in memory and then executes additional dataway cycles to obtain subsequent data. This process continues for each data word until the transfer count is exhausted.

The Q-Scan Block Transfer Mode continues to transfer data until the transfer count is exhausted or a Station Number Greater Than 23 (N>23) occurs. When the Q-Scan operation is started, the CAMAC command specified by the Station Number, Subaddress and Function code is executed. During the Q-Scan operation, the Function code (F) remains unchanged, but the CAMAC Station Number (N) and Subaddress (A) are altered during the scan. The N and A are altered as follows:

After a CAMAC command is executed, if the CAMAC Q-response is 0, the Station Number (N) is incremented and the Subaddress (A) is reset to zero for the subsequent operation. If the CAMAC Q-response is 1, the Subaddress is incremented, or, if the Subaddress was 15, it is reset to zero and the Station Number is incremented.

The Q-Scan operation continues in this mode until the transfer count is exhausted, or the Q-Scan operation is forced to scan past slot 23 in the chassis.

The Q-Scan operation provides an efficient mechanism to transfer data to/from channel oriented modules that allow their data to be accessed at incrementing subaddresses. These channel oriented accessible modules should be grouped together to facilitate better performance of the block transfer operation.

To specify the type of block transfer operation to perform, the user can use the following #defines.

#define	Description
QSTP	Selects the Q-Stop Block Transfer Mode
QIGN	Selects the Q-Ignore Block Transfer Mode
QRPT	Selects the Q-Repeat Block Transfer Mode
QSCN	Selects the Q-Scan Block Transfer Mode

Q-mode Block Transfer Selection

3.5.2 Data Word Size

The 2915 supports two data word sizes, 16 bit CAMAC data words and 24-bit CAMAC data words. When allocating memory locations for the write and read buffers, the user must ensure that the buffers are longword (32-bits) aligned. This restriction is in place due to the fundamental Direct Memory Access (DMA) circuitry on the 2915. When a block transfer operation requires the movement of data to/from host computer memory, it always transfers 32-bits at a time. This transfer scheme allows an increase in performance over transferring 16-bits of data at a time.

For 16-bit block transfer operations, the 32-bit host computer memory word 'holds' two 16-bit CAMAC data words. The lower shortword of the 32-bit memory location, bits 15 through 0, contains the first 16-bit CAMAC data word and the upper shortword of the 32-bit memory location, bits 31 through 16, contains the second 16-bit CAMAC data word. This organization continues until the last block transfer word is reached. If an odd number of 16-bit data words is required using a block transfer write to CAMAC, the last shortword is ignored. During CAMAC read operations that require an odd number of 16-bit CAMAC data

words, a shortword set to zero is appended to the last DMA operation to memory to pad the unused upper shortword of the last longword.

The following #defines can be used to specify the CAMAC data word size during block transfer operations that are downloaded into a list processing sequence.

#define	Description
WTS16	Selects 16-bit CAMAC Data Word Size
WTS24	Selects 24-bit CAMAC Data Word Size

CAMAC Data Word Size Selection

3.6 Status Array

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 the first element of the status array 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 in the following table. The error codes follow the Windows operating system standard for error codes as well. The odd codes were selected as successful status to facilitate users migrating from older operating systems to prevent the need for modifications if their software tested for odd status.. Note that the function ***camsg*** can be used to decode the returned error number into ASCII text. In the table below the symbolic name for the status array element is shown along with its decimal array index, starting with 0.

STATUS ARRAY

0 ERR	**Error Status**: Contains the returned error code. An odd return status indicates a successful transfer. Any other value indicates an error or warning.
1 StaCSR	**Control and Status Register**: Contains the state of the 2915 Control and Status register. This is copied from the I/O status block. See the I/O status block for the 2915 device driver and the 2915 hardware manual for a more complete description.
2 StaERS	**Error Status Register**: Contains the state of the 2915 Error Status Register. This is copied from the I/O status block. See the I/O status block for the 2915 device driver and the 2915 hardware manual for a more complete description.
3 StaLCS	**List Status Register**: Field is zero and is reserved for future KSC use.
4 StaSum	A variable using bit 1 to indicate the sum of CAMAC NO-X responses and bit 0 to indicate the sum of CAMAC NO-Q responses for all the CAMAC operations in the Command List. If there were any CAMAC NO-Qs, bit zero of StaSum would be set and if there were any CAMAC NO-Xs, bit one of StaSum would be set.
5 StaCnt	A variable returning the number of words not transferred for the last Block Transfer operation. A zero will be returned if the last Block Transfer operation was successful or if there were no Block Transfers in the Command List.
6 StaLis	A variable returning the Fortran index into the CCL of the last command in the Command List that was executed by the driver.
7 StaDat	A variable returning the Fortran index into the Data Buffer of the last Data word read or written by the driver.
8 StaWC	A variable returning the total number of Word Count Buffer errors that occurred. This number can be greater than the number of Word Count Buffer records.
9 StaQXE	A variable returning the total number of QXE Buffer errors that occurred. This number can be greater than the number of QXE Buffer records.

3.7 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 1 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 that provide a single control command to clear LAM and disable LAM.*

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 that 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.

4 CAMAC Application Programming Interface (API)

4.1 CAMAC Operational Routines

4.1.1 cab16

Syntax

```
int cab16(      void  **hdlptr,
                short int  *c,
                short int  *n,
                short int  *a,
                short int  *f,
                short int  *mode,
                short int  *data,
                long  int  *dataln,
                int  errarr[]);
```

Purpose

The *cab16* function is used to execute a 16-bit block transfer write or read operation.

Description

The *cab16* function performs block transfer operations to or from a CAMAC module(s) utilizing 16-bit data words. For the 16-bit data transfers, only the lower 16-bits of the 24 bit CAMAC data word is used during the transfer. The array used to move data to or from the module must be longword aligned. Since the PCI bus is organized as 32-bit data words, the array for data must be aligned on a longword boundary for facilitating Direct Memory Access (DMA). If an odd number of 16-bit data words is to be transferred, the application software must allocate an additional 16-bit data entry in host memory to accommodate the re-alignment of data onto a longword boundary. When the 2915 executes a CAMAC block transfer operation with a transfer count specification that is odd, an additional 16-bit data word is sent to memory to force the longword alignment.

The *cab16* function supports all four types of block transfer operations. These four modes consist of Q-Ignore, Q-Stop, Q-Repeat and Q-Scan. Please refer to the *Transfer Mode* section of this manual for details on each operating mode.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function

Parameter Name	Direction	Description
c	Input	Address of the chassis to be accessed
n	Input	Slot number of the module to be accessed
a	Input	Subaddress within the module to be accessed
f	Input	Function code to be performed
mode	Input	Type of CAMAC block transfer to perform. Please refer to *Transfer Mode* section of this manual for additional information.
data	Input/Output	CAMAC Write (Input) or Read (Output) data
dataln	Input	Requested number of CAMAC data 16-bit data words
errarr	Output	Returned 10-element status array. Please refer to *Status Array* section of this manual for additional information.

The *mode* parameter in the *cab16* function is used to specify the CAMAC block transfer Q-mode and a specification as to the termination technique when a No-X condition occurs. The following table shows the available selections as *#defines* in the *kscuser.h* include file. Note that only one defined Q-mode can be specified for each block transfer.

#define	Description
QSTP	Selects the Q-Stop Block Transfer Mode
QIGN	Selects the Q-Ignore Block Transfer Mode
QRPT	Selects the Q-Repeat Block Transfer Mode
QSCN	Selects the Q-Scan Block Transfer Mode

Q-mode Block Transfer Selection

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR141 Data buffer not long word aligned.

ERR701 An invalid CAMAC sub-address (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.

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.

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.

ERR709 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 $24 \leq F \leq 31$).

ERR714 Illegal CAMAC crate number.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"

main()
{
  int     status;                       // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                         // Handle for operations
  int     errstat[STAMAX];              // array with list of errors
  short n;                              // slot
  short a;                              // sub address
  short f;                              // function
  short c;                              // crate
  short qmode;                          // q mode for transfer
  short ShortWriteBuffer[8192];         // short write data buffer
  unsigned long TransferCount;          // transfer count for block

//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
exit(status);

  }
//
// Setup the parameters for the block transfer
//
  c=1;
  n=1;
  f=16;
  a=0;
  qmode = QSTP ;
  TransferCount = 100;
  status = cab16(&hdl, &c, &n, &a, &f, &qmode, ShortWriteBuffer,
&TransferCount,errstat);
  if (status & 1) != 1)
```

```
  {
        printf("****ERROR**** cab16\n");
        camsg(errstat);
        exit(status);
  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

  }
}
```

4.1.2 cab24

Syntax

```
int cab24(      void  **hdlptr,
                short int  *c,
                short int  *n,
                short int  *a,
                short int  *f,
                short int  *mode,
                short int  *data,
                int  *dataln,
                int  errarr[]);
```

Purpose

The *cab24* function is used to execute a 24-bit block transfer write or read operation.

Description

The *cab24* function performs block transfer operations to or from a CAMAC module(s) utilizing 16-bit data words. For these 24-bit data transfers, the entire 24-bits of the 24-bit CAMAC data word are used during the transfer. The array used to move data to or from the module must be longword aligned. Since the PCI bus is organized as 32-bit data words, the array for data must be aligned on a longword boundary for facilitating Direct Memory Access (DMA). Due to the architecture of the 2915, each 24-bit CAMAC data word is contained in a single 32-bit PCI memory word. The additional 8-bits of the PCI memory data word are ignored. When CAMAC read operations are performed, the upper 8 bits of the 32-bit PCI memory word are padded with zeros.

The *cab24* function supports all four types of block transfer operations. These four modes consist of Q-Ignore, Q-Stop, Q-Repeat and Q-Scan. Please refer to the *Transfer Mode* section of this manual for details on each operating mode.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
c	Input	Address of the chassis to be accessed
n	Input	Slot number of the module to be accessed
a	Input	Subaddress within the module to be accessed
f	Input	Function code to be performed
mode	Input	Type of CAMAC block transfer to perform. Please refer to *Transfer Mode* section of this manual for additional information.
data	Input/Output	CAMAC Write (Input) or Read (Output) data
dataln	Input	Requested number of CAMAC data 16-bit data words
errarr	Output	Returned 10-element status array. Please refer to *Status Array* section of this manual for additional information.

The *mode* parameter in the *cab24* function is used to specify the CAMAC block transfer Q-mode and the termination technique when a No-X condition occurs. The following table shows the available selections as *#defines* in the *kscuser.h* include file. Note that only one defined Q-mode can be specified for each block transfer.

#define	Description
QSTP	Selects the Q-Stop Block Transfer Mode
QIGN	Selects the Q-Ignore Block Transfer Mode
QRPT	Selects the Q-Repeat Block Transfer Mode
QSCN	Selects the Q-Scan Block Transfer Mode

Q-mode Block Transfer Selection

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR141 Data buffer not long word aligned.

ERR701 An invalid CAMAC sub-address (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.

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.

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.

ERR709	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 $24 \leq F \leq 31$).
ERR714	Illegal CAMAC crate number.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;                       // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                         // Handle for operations
  int     errstat[STAMAX];              // array with list of errors
  short n;                              // slot
  short a;                              // sub address
  short f;                              // function
  short c;                              // crate
  short qmode;                          // q mode for transfer
  int LongWriteBuffer[8192];            // long write data buffer
  unsigned long TransferCount;          // transfer count for block

//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
exit(status);

  }
//
// Setup the parameters for the block transfer
//
```

```
  c=1;
  n=1;
  f=16;
  a=0;
  qmode = QSTP;
  TransferCount = 100;
  status = cab24(&hdl, &c, &n, &a, &f, &qmode, LongWriteBuffer,
 &TransferCount,errstat);
  if (status & 1) != 1)
  {
        printf("****ERROR**** cab24\n");
        camsg(errstat);
        exit(status);
  }
//
// Close the device
//
   status = caclos (&hdl, errstat);
   if ((status & 1) != 1)
   {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

   }
}
```

4.1.3 caclos

Syntax

```
int caclos(     void  **hdlptr,
                int *error
```

Purpose

The *caclos* function is used to close the current CAMAC session with the 2915.

Description

The *caclos* function is used to unassign a channel from the CAMAC 2915 device and deallocate the per-process space for the controller. This routine is the opposite of the *caopen* routine that opens a device for communication. Once the *caclos* is executed, the device session is closed.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
error		Error Code

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR603 The *caclos* error is unknown.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"

main()
{
  int     status;                    // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                      // Handle for operations
  int     errstat[STAMAX];           // array with list of errors
  int     lwdata;                    // long write data
  short n;                           // slot
  short a;                           // sub address
  short f;                           // function
  short c;                           // crate
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
exit(status);
}
```

```
//
// Setup the parameters for the single transfer
//
  c=1;
  n=1;
  f=16;
  a=0;
  lwdata = 0x112233;
  status = cam24(&hdl, &c, &n, &a, &f, &lwdata, errstat);
  if (status & 1) != 1)
  {
        printf("****ERROR**** cam24\n");
        camsg(errstat);
        exit(status);
  }
//
// Close the device
//
   status = caclos (&hdl, errstat);
   if ((status & 1) != 1)
   {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

   }
}
```

4.1.4 cactrl

Syntax

```
int cactrl(     void  **hdlptr,
                short int  *c,
                short int *func
                int  errarr[]);
```

Purpose

The *cactrl* function is used to execute various CAMAC Crate Controller functions.

Description

The *cactrl* function generates CAMAC crate-wide operations. These operations include the CAMAC Initialize (Z) cycle, the CAMAC Clear (C) cycle, and setting/clearing the CAMAC Inhibit (I) signal. These operations are addressed to the specified CAMAC Crate Controller by the 2915 with the station number set

to 30 (N=30). All CAMAC Crate Controllers have an internal register accessible at N=30 for generating crate-wide operations.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
c	Input	Address of the chassis to be accessed
func	Input	Requested crate-wide operation. (see Note 1)
errarr	Output	Returned 10-element status array. Please refer to *Status Array* section of this manual for additional information.

Note 1: There are 4 valid values that can be used with this command. These *#defines* are found in the *kscuser.h* include file and are listed below.

#define	Description
INIT	Execute a CAMAC Initialize (Z) cycle
CLEAR	Execute a CAMAC Clear(C) cycle
SETINH	Set the CAMAC Inhibit (I) signal
CLRINH	Clear the CAMAC Inhibit (I) signal

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR224 Illegal CAMAC crate number.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;                 // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                   // Handle for operations
  int     errstat[STAMAX];        // array with list of errors
  short ctrlval;                  // cactrl value
  short c;                        //  crate

//
```

```
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
//
// Setup the parameters for the control call
//
  c=1;
  ctrlval=INIT;
  status = cactrl (&hdl, &c, &ctrlval, errstat);
  if (status & 1) != 1)
  {
        printf("****ERROR**** cactrl\n");
        camsg(errstat);
        exit(status);
  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

  }
}
```

4.1.5 calam

Syntax

```
int calam(
        void **hdlptr,
        short int *c,
        short int *lam_id,
        short int *lam_type,
        short int *priority,
        void (*apc_addr)(),
        void *parm,
```

```
short *clrN,
short *clrA,
short *clrF,
short *dsbN,
short *dsbA,
short *dsbF,
int *error);
```

Purpose

The *calam* function is used to register a LAM for subsequent asynchronous notification of an application program.

Description

The *calam* function requests the Demand Process to service LAMs for the LAM specified in the call to the function. The process of enabling a LAM to be serviced by the Demand Process is called *booking* a LAM. When the LAM pipe message is received, an Asynchronous Procedure Call (APC) is made to the Demand Process which disables the LAM and calls the user specified APC routine. Prior to executing the user APC routine, the Demand Process APC executes the CAMAC commands passed into the *calam* routine at the time is was booked. The Demand Process APC will execute either the Clear or Disable CAMAC command passed into the routine based on the setting of the *lam_type*. By setting the *lam_type* to a 0 (Type 0), the Demand Process will unbook the LAM and issue the disable CAMAC command to the specified module. When the *lam_type* is set to a 1 (Type 1), the Demand Process will leave the LAM booked and issue the clear CAMAC command to the specified module.

In general, it is up to the user application program to actually enable a module to generate a LAM. The LAM Mask Registers in the crate controller and other associated enables for the LAM are taken care of by the *calam* functions. The command to enable the LAM generation within a module should be placed in the application program after the *calam* function is used. The Demand Process will enable LAMs for a crate if they are not already enabled from a previous request to enable another LAM in the same crate.

If the LAM for a module in not enabled by the user application, a LAM that it generates will never be serviced. If the user application enables the modules LAM prior to calling the *calam* routine, the LAM could be generated by the module prior to it being booked. This situation MUST be avoided as the results of this sequence may be erroneous.

Once the Demand Process has completed processing its portion of the LAM service, it passes control onto the user specified APC. The user's APC is called with 4 arguments that define specific information regarding the source of the LAM. This information contains the Station Number (N) of the device generating the LAM, the handle returned from the *caopen* routine of the parent program that called the *calam*, the chassis that generated the LAM, and the user specified parameter passed into the *calam* routine when it was booked. Please refer to the *Demands* section of this manual for additional information regarding the APC calling conventions.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
c	Input	Address of the chassis to be accessed
lam_id	Input	Specifies the Station Number (N) of the LAM to be booked
lam_type	Input	Specifies the type of LAM booking. Type 0 indicates an unbook and disable and a Type 1 indicates a remain booked and clear LAM
priority	Input	Not Supported. This parameter is for legacy parameter placeholding.
apc_addr	Input	This parameter specifies the address of the APC to be called once the LAM is generated.
parm	Input	This value is passed onto the APC once the LAM is serviced.
clrN	Input	Specifies the Station Number (N) to be accessed when a Type 1 LAM is serviced
clrA	Input	Specifies the Subaddress (A) to be accessed when a Type 1 LAM is serviced
clrF	Input	Specifies the Function (F) to be performed when a Type 1 LAM is serviced
dsbN	Input	Specifies the Station Number (N) to be accessed when a Type 0 LAM is serviced
dsbA	Input	Specifies the Subaddress (A) to be accessed when a Type 0 LAM is serviced
dsbF	Input	Specifies the Function (F) to be performed when a Type 0 LAM is serviced
error		Error code

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR714 Illegal CAMAC crate number.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"
//
// Global variable definitions....shared between main and APC routine
//
int  *hdl;                                  // Handle for operations
```

```
HANDLE hEvent;                                  // So APC can wake up mainline

void ApcRoutine (int *dmd_id,
                 struct KSC_handle *handle,
                 int chassis,
                 void *UserArg);


main()
{
 int  status;                                   // return status from the functions
 int  iStatus;
 int lpcnt=0;
 int UserParm;

 char    devname[] = "kpa00";
 int  errstat[STAMAX];                          // array with list of errors

 short n;                                       // slot
 short a;                                       // subaddress
 short f;                                       // function
 short c;                                       // crate
 short n_clr,n_dis;
 short a_clr,a_dis;
 short f_clr,f_dis;
 short   c_3291 = 1;                            // crate address for 3291
 short n_3291 = 1;                              // slot number for 3291
 short swdata;                                  // short write data
 short srdata;                                  // short read data
 short nLamType;
 short nLamPriority;
 short fix;
 fix = n_3291 - 1;

//
// Open the device
//

 status = caopen(&hdl, devname, errstat);
 if ((status & 1) == 0)
 {
  printf("CAOPEN, error opening device = %s\n", devname);
  camsg(errstat);
  exit(status);
 }
//
// Crate event flag & check status
//
 hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

 if (NULL == hEvent)
 {
  iStatus = GetLastError();
  printf("Error creating event object: 0x%x\n", iStatus);
  exit(iStatus);
 }
```

```
//
// Set up the APC for the LAM (Book the LAM)
//
  n_clr = n_3291;
  f_clr = 10;
  a_clr = 0;
  n_dis = n_3291;                          // filled in but not used for type 1
  f_dis = 24;
  a_dis = 0;

  nLamType = 1;                            // clear but remain booked
  nLamPriority = -1;                               // not used
  iStatus = calam(&hdl,                            // handle from caopen
            &c_3291,                               // crate with our 3291
            &n_3291,                               // slot containing 3291
            &nLamType,                             // forever let us process it
            &nLamPriority,                         // not used, but must be here
            &ApcRoutine,                           // APC to call
             &UserParm,                            // user parameter
             &n_clr,                               // station number for clear
             &a_clr,                               // subaddress for clear
             &f_clr,                               // function for clear
             &n_dis,                               // station number for disable
             &a_dis,                               // subaddress for disable
             &f_dis,                               // function code for disable
            errstat);                              // CAMAC status

 if ((iStatus & 1) == 0)
 {
  printf("Failure to book the 3291 LAM\n");
  camsg(errstat);
  exit(iStatus);
 }
while (TRUE)
 {
  iStatus = ResetEvent(hEvent);

  if (iStatus == FALSE)
  {
   iStatus = GetLastError();
   printf("Error Resetting Event: 0x%x\n", iStatus);
   exit(iStatus);
  }

//
// Generate a LAM on the 3291
//
  c=1;
  n = n_3291;
  f=14;
  a=0;
  iStatus = cam16(&hdl,                            // handle from caopen
                  &c,                              // crate for 3291
                  &n,                              // slot number for 3291
                  &a,                              // subaddress
```

```
                    &f,                         // function code
                    &srdata,                    // data space
                    errstat);                   // error array
//
// Waiting for a LAM
//
   printf("Waiting for a LAM\n");
   iStatus = WaitForSingleObject(hEvent, INFINITE);

   if (WAIT_OBJECT_0 != iStatus)
   {
    printf("Error in WaitForSingleObject: 0x%x\n", iStatus);
                        cwait();

    exit(iStatus);
   }
  printf("Object Received \n");
 }
}                   // end of main
// APC procedure.
//
// This routine is called when a LAM is received from the demand process
// as a result of the 3291 getting a LAM.
//
void ApcRoutine(int *dmd_id,
          struct KSC_handle *handle,
          int chassis,
          void *user_arg)
{
//
// Set the event so mainline will continue
//
 printf("APC triggered\n");
 SetEvent(hEvent);

}
```

4.1.6 cam16

Syntax

```
int cam16(      void  **hdlptr,
                short int  *c,
                short int  *n,
                short int  *a,
                short int  *f,
                short int  *data,
                int  errarr[]);
```

Purpose

The *cam16* function is used to execute a 16-bit single transfer write, read or control operation.

Description

The *cam16* function performs a single transfer operation to the CAMAC crate. This command accommodates write, read and control operations. All data words moved using this command are 16-bits in width. Therefore, only the lower 16-bits of the 24-bit CAMAC data word can be accessed using this function.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
c	Input	Address of the chassis to be accessed
n	Input	Slot number of the module to be accessed
a	Input	Subaddress within the module to be accessed
f	Input	Function code to be performed
data	Input/Output	This parameter either specifies CAMAC write data for write operations to the CAMAC crate or read data returned from executing a CAMAC read operation.
errarr	Output	Returned 10-element status array. Please refer to *Status Array* section of this manual for additional information.

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

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.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;                    // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                      // Handle for operations
  int     errstat[STAMAX];           // array with list of errors
  short n;                           //  slot
  short a;                           //  sub address
  short f;                           //  function
  short c;                           //  crate
  short swdata;                      // short write data
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
//
// Setup the parameters for the single transfer
//
  c=1;
```

```
  n=1;
  f=16;
  a=0;
  swdata = 0x1122;
  status = cam16(&hdl, &c, &n, &a, &f, swdata,errstat);
  if (status & 1) != 1)
  {
        printf("****ERROR**** cam16\n");
        camsg(errstat);
        exit(status);
  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

  }
}
```

4.1.7 cam24

Syntax

```
int cam24(      void  **hdlptr,
                short int  *c,
                short int  *n,
                short int  *a,
                short int  *f,
                int  *data,
                int  errarr[]);
```

Purpose

The *cam24* function is used to execute a 24-bit single transfer write, read or control operation.

Description

The *cam24* function performs a single transfer operation to the CAMAC crate. This command accommodates write, read and control operations. All data words moved using this command are 24-bits in width.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
c	Input	Address of the chassis to be accessed
n	Input	Slot number of the module to be accessed
a	Input	Subaddress within the module to be accessed
f	Input	Function code to be performed
data	Input/Output	This parameter either specifies CAMAC write data for write operations to the CAMAC crate or read data returned from executing a CAMAC read operation.
errarr	Output	Returned 10-element status array. Please refer to *Status Array* section of this manual for additional information.

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR701	An invalid CAMAC sub-address (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.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int    status;                    // return status from the functions
  char   devname[] = "kpa00";
```

```
    int     *hdl;                   // Handle for operations
    int     errstat[STAMAX];        // array with list of errors
    short n;                        //  slot
    short a;                        //  sub address
    short f;                        //  function
    short c;                        //  crate
    int lwdata;                     // long write data
  //
  // Open the device
  //
    status = caopen(&hdl, devname, errstat);

  //
  // Check if device opened properly
  //
    if ((status & 1) == 0)
    {
            printf("CAOPEN, error opening device = %s\n", devname);
            camsg(errstat);
 exit(status);

    }
  //
  // Setup the parameters for the single transfer
  //
    c=1;
    n=1;
    f=16;
    a=0;
    lwdata = 0x112233;
    status = cam24(&hdl, &c, &n, &a, &f, &lwdata, errstat);
    if (status & 1) != 1)
    {
            printf("****ERROR**** cam24\n");
            camsg(errstat);
            exit(status);
    }
  //
  // Close the device
  //
    status = caclos (&hdl, errstat);
    if ((status & 1) != 1)
    {
            printf("****ERROR**** caclos\n");
            camsg(errstat);
            exit(status);

    }
}
```

4.1.8 camsg

Syntax

int camsg(int *error);

Purpose

The *camsg* function is used to translate error codes received from various CAMAC API functions and print a message to the standard output device.

Description

The *camsg* function can be called whenever an error is detected as a result of executing a CAMAC API function. An error code returned from the API functions can be printed to the standard output device. The printed error may be as a result of an error from the device driver, the API, or from the operating system.

Parameters

Parameter Name	Direction	Description
error	Input	Completion status or error code returned from a previous function call to a CAMAC API routine.

Return Values

For a comprehensive list, please refer to the *Error Codes* section of this manual.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
```

```
{
  int     status;             // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;               // Handle for operations
  int     errstat[STAMAX];    // array with list of errors

//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

  }
}
```

4.1.9 caopen

Syntax

```
int caopen(     void **hdlptr,
                char *device,
                int  *error
```

Purpose

The *caopen* function opens a session with the 2915.

Description

The *caopen* function assigns a channel to a device and initializes the CAMAC library so that subsequent CAMAC operations may be executed. This function must be called at the start of a program before attempting any CAMAC operations. Once the channel has been open to the device, it should not be re-opened until the channel is unassigned by a call to the *caclos* function.

The *caopen* function initializes the handle parameter. The handle is a pointer to a process and controller specific region that has been allocated for the user application. The *caopen* should be called as part of the process's initialization. The handle obtained as a result of the *caopen* function should be passed to any other CAMAC API function requiring use of the handle. The *caclos* function is used to close the channel and release this per process handle and controller space.

The second parameter in this function call is the name of the CAMAC driver associated with the 2915. In order to open a valid connection to the driver, the device name of *kpa00* must be used.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
device	Input	Character string containing the name of the device to be opened. The device name for the 2915 is *kpa00*.
error	Output	Returned value

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

KSC_BAD_ARG	One or more of the arguments is not readable or writeable.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;                       // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                         // Handle for operations
  int     errstat[STAMAX];              // array with list of errors

//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

  }
}
```

4.1.10 ccstat

Syntax

```
int ccstat(     void **hdlptr,
                short int *c,
                int data[]
                int errarr[]);
```

Purpose

The *ccstat* function is used to retrieve the current status of the 3922 Crate Controller.

Description

The *ccstat* function performs a read operation to the 3922 Crate Controller to determine its current status. The status information returned includes the state of the CAMAC Inhibit (I) line, the state of the Service Request Enable (LAM Enable) signal, the current LAM pattern. And the entire 3922 Control/Status Register contents. As a result of the *ccstat* function, the 2915 executes various CAMAC commands directed at Station Number (N) 30 of the target crate. The N=30 commands are internal operations directed at the 3922. Therefore, no CAMAC dataway cycles occur as a result of this command.

The third argument in the *ccstat* function returns a pointer to four 32-bit words. The contents of these words are described below.

Word 1

Bit 31 ----------Bit 1	Bit 0
0	INH

Word 2

Bit 31 ----------Bit 1	Bit 0
0	SRR ENA

Word 3

Bit 31----------Bit 24	Bit 23----------Bit 0
0	LAM Status 24 through 1

Word 4

Bit 15	Bit 14	Bit 13	Bit 12-10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5-3	Bit 2	Bit 1-0
SLP	RSVD	OFF LINE	0	L24	SRR ENA	RSVD	RD INH	0	INH	0

INH – Read as a 1 if the 3922 is asserting the CAMAC Inhibit (I) signal.
SRR ENA – Read back as a 1 if the Service Request on the 3922 is enabled.
LAM Status 24 – 1 are the LAM Status bits reflecting the current state of the CAMAC LAM signals.

RD INH – Read back as a 1 if the CAMAC Inhibit (I) signal is asserted by any module.
RSVD are reserved bits.
L24 – Read back as a one when Internal LAM 24 is set.
OFF LINE – Read back as a 1 when the 3922 front panel switch is in the Off-Line position.
SLP – Read back as a one is a Selected LAM is present in the crate.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
c	Input	Address of the chassis to be accessed
data	Output	Array of four 32-bit words reflecting the current state of the 3922 Crate Controller.
errarr	Output	Returned 10-element status array. Please refer to *Status Array* section of this manual for additional information.

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR141	Data buffer not long word aligned.
ERR601	An invalid channel number was specified. The passed handle is invalid.
ERR701	An invalid CAMAC sub-address (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.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"
```

```
main()
{
  int     status;                          // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                            // Handle for operations
  int     errstat[STAMAX];                 // array with list of errors
  int CrateStatus[4];                      // ccstat returns
  short c;                                 //  crate

//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
//
// Get the current status of the 3922 crate controller
//
  status = ccstat (&hdl, &c, CrateStatus, errstat);
  if (status & 1) != 1)
  {
        printf("****ERROR**** ccstat\n");
        camsg(errstat);
        exit(status);
  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
        printf("****ERROR**** caclos\n");
        camsg(errstat);
        exit(status);

  }
}
```

4.1.11 cxlam

Syntax

```
int cxlam(
        void **hdlptr,
```

```
short int *c,
short int *lam_id,
short int *lam_type,
short int *priority,
void (*apc_addr)(int *, struct KSC_handle *, int, void *),
int *error);
```

Purpose

The *cxlam* function is used to register a LAM for subsequent asynchronous notification of an application program. This function call is similar in nature to the *calam* function call, except that it performs a more sophisticated LAM processing algorithm in order to service an asynchronous notification. Please refer to the *Demand* section of this manual for additional information.

Description

The *calam* function requests the Demand Process to service LAMs for the LAM specified in the call to the function. The process of enabling a LAM to be serviced by the Demand Process is called *booking* a LAM. When the LAM pipe message is received, an Asynchronous Procedure Call (APC) is made to the Demand Process which disables the LAM and calls the user specified APC routine. Prior to executing the user's APC routine, the Demand Process APC executes a specific sequence of CAMAC commands in order to clear the pending LAM.

Once the Demand Process receives notification of a LAM, it verifies the crate address that generated the LAM. Once this is determined, the Demand Process reads the LAM Status Register of the 3922 generating that generated the LAM. The LAM Status Register is a 24-bit register located on the 3922 that contains a bit that corresponds to each station number within the crate. With this information, the Demand Process can determine which module in the crate is requesting service.

The Demand Process will try to clear the source of the LAM within a module by first using the selective clear operation to the module generating the LAM. The selective clear operation performed is an F(23)A(12) command using the LAM pattern as the data for the selective clear. If this does not clear the LAM, then the Demand Process executes an F(11)A(12) command to clear the LAM. If this is not successful, an F(10)A(0) is then tried. As a last resort, an F(10)A(*i*) command is executed where *i* corresponds to the bit positions set in the LAM Status Register.

In general, it is up to the user application program to actually enable a module to generate a LAM. The LAM Mask Registers and other associated enables for the LAM are taken care of by the *cxlam* functions. The command to enable the LAM generation within a module should be placed in the application program after the *cxlam* function is used. The Demand Process will enable LAMs for a crate if they are not already enabled from a previous request to enable another LAM in the same crate.

If the LAM for a module in not enabled by the user application, a LAM that it generates will never be serviced. If the user application enables the modules LAM prior to calling the *cxlam* routine, the LAM could be generated by the module prior to it being booked. This situation MUST be avoided as the results of this sequence may be erroneous.

Parameters

Parameter Name	Direction	Description
hdlptr	Input	Handle returned by caopen function
c	Input	Address of the chassis to be accessed
lam_id	Input	Specifies the Station Number (N) of the LAM to be booked
lam_type	Input	Specifies the type of LAM booking. Type 0 indicates an unbook and disable and a Type 1 indicates a remain booked and clear LAM
priority	Input	Not Supported. This parameter is for legacy support
apc_addr	Input	This parameter specifies the address of the APC to be called once the LAM is generated.
error	Output	Error Return

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR714 Illegal CAMAC crate number.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"
//
// Global variable definitions....shared between main and APC routine
//
int  *hdl;                                  // Handle for operations
HANDLE hEvent;                                  // So APC can wake up mainline

void ApcRoutine (int *dmd_id,
                 struct KSC_handle *handle,
                 int chassis,
                 void *UserArg);
main()
{
 int  status;                                   // return status from the functions
 int  iStatus;
 char    devname[] = "kpa00";
 int  errstat[STAMAX];                          // array with list of errors

 short n;                                // slot
```

```
  short a;                          // sub address
  short f;                          // function
  short c;                          // crate
  short c_3296 = 1;                         // crate address for 3296
  short n_3296 = 6;                         // slot number for 3296
  short swdata;                             // short write data
  short srdata;                             // short read data
  short nLamType;
  short nLamPriority;

 //
 // Open the device
 //
  status = caopen(&hdl, devname, errstat);
  if ((status & 1) == 0)
  {
   printf("CAOPEN, error opening device = %s\n", devname);
   camsg(errstat);
   exit(status);
  }
 //
 // Create event flag for notification of main process
 //
  hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
 //
 // Check status of event creation
 //
  if (NULL == hEvent)
  {
   iStatus = GetLastError();
   printf("Error creating event object: 0x%x\n", iStatus);
   exit(iStatus);
  }
 //
 // Set up the APC for the LAM (Book the LAM)
 //
  nLamType = 3;
  nLamPriority = -1;
  iStatus = cxlam(&hdl,                             // handle from caopen
            &c_3296,                                // crate with our 3296
            &n_3296,                                // slot containing 3296
            &nLamType,                              // forever let us process it
            &nLamPriority,                          // not used, but must be here
            &ApcRoutine,                    // APC to call
            errstat);                               // CAMAC status

  if ((iStatus & 1) == 0)
  {
   printf("Failure to book the 3296 LAM\n");
   camsg(errstat);
  exit(iStatus);
  }
 //
 // Enable the LS switch on the 3296 to generate a LAM
 //
   c=1;
```

```
    n = n_3296;
    f=26;
    a=0;

   iStatus = cam16(&hdl,                                    // handle from caopen
                &c,
                &n,
                &a,
                &f,
                &srdata,
                errstat);


   while (TRUE)
   {
    iStatus = ResetEvent(hEvent);

    if (iStatus == FALSE)
    {
     iStatus = GetLastError();
     printf("Error Resetting Event: 0x%x\n", iStatus);
                                cwait();
     exit(iStatus);
    }
//
// Waiting for a LAM
//
    printf("Waiting for a LAM\n");
    iStatus = WaitForSingleObject(hEvent, INFINITE);

    if (WAIT_OBJECT_0 != iStatus)
    {
     printf("Error in WaitForSingleObject: 0x%x\n", iStatus);
     exit(iStatus);
    }
   printf("Object Received  #%d\n",++lpcnt);
  }
}                    // end of main

// APC procedure.
//
// This routine is called when a LAM is received from the demand process
// as a result of the 3296 getting a LAM through its front panel switch.
//
void ApcRoutine(int *dmd_id,
            struct KSC_handle *handle,
            int chassis,
            void *user_arg)
{
//
// Set the event so mainline will continue
//
 printf("APC triggered\n");
 SetEvent(hEvent);

}
```

4.1.12 camlookupmsg

Syntax

```
void camlookupmsg (int *returnCode,
                   char *szSeverityBuffer,
                   int sizeSeverityBuffer,
                   char *szNameBuffer,
                   int sizeNameBuffer,
                   char *szDescBuffer,
                   int sizeDescBuffer);
```

Purpose

The camlookupmsg function is used to lookup descriptive strings associated with an error code.

Description

The camlookupmsg function takes an error code returned from other API functions and populates 3 user-supplied buffers with strings that describe the severity of the return code, the name of the return code, and a description of the return code. The user also specifies the size of each of the supplied buffers; this call will truncate any message larger than the specified size.

This function is similar to function *camsg*, that passes the same information to standard output.

Parameters

Parameter Name	Direction	Description
returnCode	Input	Return Code from previous API call
szSeverityBuffer	Input/Output	User Buffer to hold the Severity Description
sizeSeverityBuffer	Input	Size of the Severity Buffer
szNameBuffer	Input/Output	User Buffer to hold the Name Description
sizeNameBuffer	Input	Size of the Name Buffer
szDescBuffer	Input/Output	User Buffer to hold the Return Code Description
sizeDescBuffer	Input	Size of the Return Code Description Buffer

For each of the three return strings (severity, name, and description), the caller supplies a buffer to be filled in, and the size of the buffer. Return code strings that exceed the specified size of the buffer will be truncated. It is valid to pass a NULL for any of the 3 buffers; any buffer specified as NULL will be ignored, and no string will be returned for that category.

Return Values

None

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;                       // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                         // Handle for operations
  int     errstat[STAMAX];              // array with list of errors
  char severity[256];
  char name[256];
  char description[256];
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        camlookupmsg (&status,
                      severity,
                      sizeof(severity),
                      name,
                      sizeof(name),
                      description,
                      sizeof(description));
        printf("CAOPEN failure:\n");
        printf("\tName %s\n",name);
        printf("\tSeverity %s\n",severity);
        printf("\tDescription %s\n",description);
        exit(status);
  }
```

4.2 CAMAC List Building Routines

This chapter describes the routines provided that will allow the user to build CAMAC command lists (CCL). The CAMAC command list provides an efficient mechanism to predetermine a sequence of CAMAC operations to be performed and executed with a single function call. This functionality results in an increase in performance since the application software does not have to make as many calls to the CAMAC driver or the operating system.

The CAMAC command list generated by the application software must be unidirectional. This means that all data transfers must be either transfer data to the CAMAC crate or from the CAMAC crate, but not both. There is one exception to this rule, and that is the use of the Single Inline Write (*cainaf*) list instruction. This instruction can be embedded in either a write list or a read list. The reason that this instruction is allowed to be embedded in a list is that the associated write data for the command is contained in the list itself. Therefore, a transfer of data from the write or read buffer is not required in order to obtain the write data.

The 2915 requires that block transfers return multiples of thirty-two bit long words. Therefore, if a 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.

The following table summarizes the available list building and support functions.

List Building Function	Function Category	Description
cainit	Initialization	This function is used to initialize a CAMAC command list prior to adding the instruction to be performed once the list is executed.
cablk	Command	This function adds a CAMAC block transfer operation to the CAMAC command list.
cainaf	Command	This function adds a single inline write CAMAC transfer operation to the CAMAC command list. This command does not transfer data from the data buffer, but embeds the write data in the list.
canaf	Command	This function adds a single CAMAC transfer operation to the CAMAC command list. This command does transfer one 16 or 32-bit data word from the specified buffer.
cahalt	Command	This function places the end-of-list marker in the CAMAC command list. The CAMAC command list processor executes the elements contained in a list until this special halt command is encountered.
caexew	Execute	This function executes a preloaded CAMAC command list and waits until the requested operation is complete.
caexec	Execute	This function executes a preloaded CAMAC command list and does not wait until the operation is complete. Instead, this function requires the use of an event flag to communicate completion information to the calling process.

4.2.1 cablk

Syntax

```
long int cablk(   struct s_header *header,
                  short *c,
                  short *n,
                  short *a,
                  short *f,
                  short *mode,
                  int *datcnt,
                  int *datind,
                  int *error
               );
```

Purpose

The *cablk* function adds a command to the CAMAC command list which when executes results in a CAMAC block transfer operation.

Description

The *cablk* function performs a block transfer operation to or from a CAMAC module(s) utilizing either 16-bit or 24-bit data words. A portion of the *mode* parameter for this function is used to indicate the CAMAC data word size. For the 16-bit data transfers, only the lower 16-bits of the 24 bit CAMAC data word are used during the transfer.

The *cablk* function supports all four types of block transfer operations. These four modes consist of Q-Ignore, Q-Stop, Q-Repeat and Q-Scan. Please refer to the *Transfer Mode* section of this manual for details on each operating mode. A portion of the *mode* parameter for this function is used to indicate the block transfer-operating mode.

Parameters

Parameter Name	Direction	Description
header	Input	Header array that is built by the *cainit* function and contains pointers to the CAMAC command list and data buffer. This is updated as additional list elements are added.
c	Input	Address of the chassis to be accessed
n	Input	Slot number of the module to be accessed
a	Input	Subaddress within the module to be accessed
f	Input	Function code to be performed
mode	Input	Type of CAMAC block transfer to perform. Please refer to

Parameter Name	Direction	Description
		Transfer Mode section of this manual for additional information. (See below)
datcnt	Input	Number of CAMAC operations to perform
datind	Output	This parameter is returned with the index in to the data buffer marking the starting location for the block of data used for the operation.
error	Output	Returned error code

The *mode* parameter in the *cablk* function is used to specify the CAMAC block transfer Q-mode, the CAMAC data word size, and a specification as to the termination technique when a No-X condition occurs. The following table shows the available selections as *#defines* in the *kscuser.h* include file. Note that only one defined Q-mode or word size can be specified for each block transfer.

#define	Description
QSTP	Selects the Q-Stop Block Transfer Mode
QIGN	Selects the Q-Ignore Block Transfer Mode
QRPT	Selects the Q-Repeat Block Transfer Mode
QSCN	Selects the Q-Scan Block Transfer Mode

Q-mode Block Transfer Selection

#define	Description
WTS16	Selects 16-bit CAMAC Data Word Size
WTS24	Selects 24-bit CAMAC Data Word Size

CAMAC Data Word Size Selection

#define	Description
AD	Inclusion of the *AD* in the mode description causes the CAMAC command processor to ignore No-X conditions during processing.

CAMAC Data Word Size Selection

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR702 Invalid Mode specification

ERR701 An invalid CAMAC sub-address (A) was found. The CAMAC subaddress was either less than 0 or greater than 15.

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.

ERR712 The CAMAC command list is not large enough to hold all the commands.

ERR715	Direction error, the CAMAC command list should be unidirectional.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;           // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                                   // Handle for operations
  int     errstat;                                // array with list of errors
  short n;                                        // slot
  short a;                                        // sub address
  short f;                                        // function
  short c;                                        // crate
  short qmode;                                    // q mode for transfer
  int LongReadBuffer[8192];                       // short write data buffer
  unsigned long TransferCount;                    // transfer count for block
  int camac_list[8192];                           // list for camac processing
  int DataIndex;                                  // data index
  int listmax;                                    // max size of list
  int datamax;                                    // max size of data
  int     indicator;
  int     zero=0;                                 // zero value
  struct s_header header;
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
  listmax = 1024;
  datamax = 1024;
```

```
    n=1;
    f=16;
    a=0;
    qmode = QIGN | WTS24 ;
    status = cainit(&header, camac_list, &listmax, LongReadBuffer, &datamax,
                    &zero, &zero, &zero, &zero, errarr);
    if ((status & 1) != 1)
    {
      printf("****ERROR**** cainit\n");
      camsg(errstat);
    }
    c=1;
    n=1;
    f=0;
    a=0;
    qmode = QIGN | WTS24 ;
    TransferCount = 100;
    status = cablk (&header, &c, &n, &a, &f, &qmode, &TransferCount, &DataIndex, errstat);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** cablk\n");
     camsg(errstat);
    }
    status = cahalt(&header, errarr);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** cahalt\n");
     camsg(errstat);
    }
    status = caexew(&header, &hdl, errarr);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** caexew\n");
     camsg(errstat);
     exit(status);
    }
//
// Close the device
//
    status = caclos (&hdl, errstat);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** caclos\n");
     camsg(errstat);
     exit(status);
    }
}
```

4.2.2 caexec

Syntax

```
int caexec(    struct s_header *header,
               void **hdl,
               int *error,
               HANDLE event
           );
```

Purpose

The *caexec* function loads and executes a CAMAC command list without waiting for the routine to complete. An event flag must be used with this function in order to provide a notification mechanism to the main application on completion.

Description

The *caexec* function executes a CAMAC command list built using the CAMAC list building routines. Control is returned to the user process after the operation is queued to the driver. The user application must then check the *event* flag to determine when the requested operation is complete.

The execution of a CAMAC command list is beneficial when an application program needs to perform computations or other activity while the command list is executed.

Parameters

Parameter Name	Direction	Description
header	Input	Header array that is built by the *cainit* function and contains pointers to the CAMAC command list and data buffer. This is updated as additional list elements are added.
hdlptr	Input	Handle returned by caopen function
error	Output	Error Code
event	Input	Event to be signaled when the operation is complete.

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR143	The CAMAC header is not initialized.
ERR144	The CAMAC header is initialized, but not correctly.
ERR601	An invalid channel number is specified.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"

HANDLE hEvent;                                  // event for caexe (w/o wait)
main()
{
  int     status;           // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                                 // Handle for operations
  int     errstat[STAMAX];                      // array with list of errors
  short n;                                      //  slot
  short a;                                      //  sub address
  short f;                                      //  function
  short c;                                      //  crate
  short qmode;                                  //  q mode for transfer
  int LongReadBuffer[8192];                     // short write data buffer
  unsigned long TransferCount;                  // transfer count for block
  int camac_list[8192];                         // list for camac processing
  int DataIndex;                                // data index
  int listmax;                                  // max size of list
  int datamax;                                  // max size of data
  int     indicator;
  int     zero=0;                               // zero value
  struct s_header header;
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);
```

```
 }
 hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);        // create event for done indication
 status = ResetEvent(hEvent);

 if (status == FALSE)
 {
  status = GetLastError();
  printf("Error Resetting Event: 0x%x\n", status);
}
if (NULL == hEvent)
{
 status = GetLastError();
 printf("Error creating event object: 0x%x\n", status);
}

 listmax = 1024;
 datamax = 1024;

 n=1;
 f=16;
 a=0;
 qmode = QIGN | WTS24 ;
 status = cainit(&header, camac_list, &listmax, LongReadBuffer, &datamax, errarr,
                 &zero, &zero, &zero, &zero, errarr);
 if ((status & 1) != 1)
 {
   printf("****ERROR**** cainit\n");
   camsg(errstat);
 }
 c=1;
 n=1;
 f=0;
 a=0;
 qmode = QIGN | WTS24 ;
 TransferCount = 100 ;
 status = cablk (&header, &c, &n, &a, &f, &qmode, &TransferCount, &DataIndex, errarr);
 if ((status & 1) != 1)
 {
  printf("****ERROR**** cablk\n");
  camsg(errstat);
 }
 status = cahalt(&header, errarr);
 if ((status & 1) != 1)
 {
  printf("****ERROR**** cahalt\n");
  camsg(errstat);
 }
 status = caexec(&header, &hdl, errarr, hEvent);
 if ((status & 1) != 1)
 {
  printf("****ERROR**** caexec\n");
  camsg(errstat);
  exit(status);
 }
 status = WaitForSingleObject(hEvent, INFINITE);
```

```
  if (WAIT_OBJECT_0 != status)
  {
    printf("Error in WaitForSingleObject: 0x%x\n", status);
  }



//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** caclos\n");
   camsg(errstat);
   exit(status);
  }
}
```

4.2.3 caexew

Syntax

```
int caexew(    struct s_header *header,
               void  **hdl,
               int *error,
          );
```

Purpose

The *caexew* function loads and executes a CAMAC command list and waits for the routine to complete. If processing needs to continue while the CAMAC command list is being processed, use the *caexec* function that incorporates use of event notification for completion indication.

Description

The *caexew* function executes a CAMAC command list built using the CAMAC list building routines. Control is not returned to the user process until the operation is complete.

Parameters

Parameter Name	Direction	Description
header	Input	Header array that is built by the *cainit* function and contains pointers to the CAMAC command list and data buffer. This is updated as additional list elements are added.
hdlptr	Input	Handle returned by caopen function
error	Output	Error Code

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR143 The CAMAC header is not initialized.

ERR144 The CAMAC header is initialized, but not correctly.

ERR601 An invalid channel number is specified.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;             // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                                   // Handle for operations
  int     errstat[STAMAX];                        // array with list of errors
  short n;                                        // slot
  short a;                                        // sub address
  short f;                                        // function
  short c;                                        // crate
  short qmode;                                    // q mode for transfer
  int LongReadBuffer[8192];                       // short write data buffer
  unsigned long TransferCount;                    // transfer count for block
  int camac_list[8192];                           // list for camac processing
  int DataIndex;                                  // data index
  int listmax;                                    // max size of list
  int datamax;                                    // max size of data
  int     indicator;
  int     zero=0;                                 // zero value
  struct s_header header;
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
  listmax = 1024;
  datamax = 1024;

  n=1;
  f=16;
  a=0;
  qmode = QIGN | WTS24 ;
  status = cainit(&header, camac_list, &listmax, LongReadBuffer, &datamax, errarr,
```

```
                &zero, &zero, &zero, &zero, errarr);
  if ((status & 1) != 1)
  {
    printf("****ERROR**** cainit\n");
    camsg(errstat);
  }
  c=1;
  n=1;
  f=0;
  a=0;
  qmode = QIGN | WTS24 ;
  TransferCount = 100 ;
  status = cablk (&header, &c, &n, &a, &f, &qmode, &TransferCount, &DataIndex, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** cablk\n");
   camsg(errstat);
  }
  status = cahalt(&header, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** cahalt\n");
   camsg(errstat);
  }
  status = caexew(&header, &hdl, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** caexew\n");
   camsg(errstat);
   exit(status);
  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** caclos\n");
   camsg(errstat);
   exit(status);
  }
}
```

4.2.4 cahalt

Syntax

```
int cahalt(   struct s_header *header,
              int   *error
          );
```

Purpose

The *cahalt* function adds a command to the CAMAC command list that marks the end of the CAMAC list. This function must always be called in order to provide a terminating entry in the list.

Description

The *cahalt* function adds a command to the CAMAC command list. This command is used to indicate the termination of a CAMAC command list. The CAMAC command processor executes list instructions until this halt is encountered.

Parameters

Parameter Name	Direction	Description
header	Input	Header array that is built by the *cainit* function and contains pointers to the CAMAC command list and data buffer. This is updated as additional list elements are added.
error	Output	Error Code

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR143 The CAMAC header is not initialized.

ERR144 The CAMAC header is initialized, but not correctly.

Example

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;             // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                                   // Handle for operations
  int     errstat[STAMAX];                        // array with list of errors
  short n;                                        // slot
  short a;                                        // sub address
  short f;                                        // function
  short c;                                        // crate
  short qmode;                                    // q mode for transfer
  int LongReadBuffer[8192];                       // short write data buffer
  unsigned long TransferCount;                    // transfer count for block
  int camac_list[8192];                           // list for camac processing
  int DataIndex;                                  // data index
  int listmax;                                    // max size of list
  int datamax;                                    // max size of data
  int     indicator;
  int     zero=0;                                 // zero value
  struct s_header header;
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
  listmax = 1024;
  datamax = 1024;

  n=1;
  f=16;
  a=0;
  qmode = QIGN | WTS24 ;
  status = cainit(&header, camac_list, &listmax, LongReadBuffer, &datamax, errarr,
                  &zero, &zero, &zero, &zero, errarr);
  if ((status & 1) != 1)
  {
    printf("****ERROR**** cainit\n");
```

```
    camsg(errstat);
   }
   c=1;
   n=1;
   f=0;
   a=0;
   qmode = QIGN | WTS24 ;
   TransferCount = 100 ;
   status = cablk (&header, &c, &n, &a, &f, &qmode, &TransferCount, &DataIndex, errarr);
   if ((status & 1) != 1)
   {
    printf("****ERROR**** cablk\n");
    camsg(errstat);
   }
   status = cahalt(&header, errarr);
   if ((status & 1) != 1)
   {
    printf("****ERROR**** cahalt\n");
    camsg(errstat);
   }
   status = caexew(&header, &hdl, errarr);
   if ((status & 1) != 1)
   {
    printf("****ERROR**** caexew\n");
    camsg(errstat);
    exit(status);
   }
//
// Close the device
//
   status = caclos (&hdl, errstat);
   if ((status & 1) != 1)
   {
    printf("****ERROR**** caclos\n");
    camsg(errstat);
    exit(status);
  }
}
```

4.2.5 cainaf

Syntax

```
int cainaf(   struct s_header *header,
              short           *c,
              short           *n,
              short           *a,
              short           *f,
              short           *mode,
              int             *data,
              int             *error
          );
```

Purpose

The *cainaf* function adds a command to the CAMAC command list which when executed results in single CAMAC write operation. The data for this operation is included in the CAMAC command list and not extracted from the data buffer.

Description

The *cainaf* function adds a single inline write operation to the CAMAC command list. The write data for the *cainaf* instruction is contained in the CAMAC command list. Therefore, this instruction does not require data transfer from the data buffer for the list. This instruction is beneficial for setting up CAMAC module parameters that do not vary from list execution to list execution. Since this command does not transfer any data from the list data buffer, one can embed this instruction in either a write CAMAC command list or a read CAMAC command list.

Parameters

Parameter Name	Direction	Description
header	Input	Header array that is built by the *cainit* function and contains pointers to the CAMAC command list and data buffer. This is updated as additional list elements are added.
c	Input	Address of the chassis to be accessed
n	Input	Slot number of the module to be accessed
a	Input	Subaddress within the module to be accessed
f	Input	Function code to be performed
mode	Input	Type of CAMAC operation to perform. Please refer to *Transfer Mode* section of this manual for additional information. (See below)
data	Input	32-bit word reserved for specification of the CAMAC write data embedded in the list.
error	Output	Error Code

The *mode* parameter in the *cainaf* function is used to specify the CAMAC transfer Q-mode, the CAMAC data word size, and the termination technique used when a No-X condition occurs. Even though the *cainaf* only executes a single write operation, the CAMAC Q and X returns are evaluated and continued list processing is based on their values as it relates to the selected Q-mode. The following table shows the available selections as *#defines* in the *kscuser.h* include file. Note that only one defined Q-mode or word size can be specified for each transfer.

#define	Description
QSTP	Selects the Q-Stop Block Transfer Mode
QIGN	Selects the Q-Ignore Block Transfer Mode
QRPT	Selects the Q-Repeat Block Transfer Mode
QSCN	Selects the Q-Scan Block Transfer Mode

Q-mode Transfer Selection

#define	Description
WTS16	Selects 16-bit CAMAC Data Word Size
WTS24	Selects 24-bit CAMAC Data Word Size

CAMAC Data Word Size Selection

#define	Description
AD	Inclusion of the *AD* in the mode description causes the CAMAC command processor to ignore No-X conditions during processing.

CAMAC Data Word Size Selection

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR143 The CAMAC header is not initialized.

ERR144 The CAMAC header is initialized, but not correctly.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"
```

```
main()
{
  int     status;              // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                                   // Handle for operations
  int     errstat[STAMAX];                        // array with list of errors
  short n;                                        // slot
  short a;                                        // sub address
  short f;                                        // function
  short c;                                        // crate
  short qmode;                                    // q mode for transfer
  int lwdata;                                     // write data
  int camac_list[8192];                           // list for camac processing
  int DataIndex;                                  // data index
  int listmax;                                    // max size of list
  int datamax;                                    // max size of data
  int     indicator;
  int     zero=0;                                 // zero value
  struct s_header header;
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
  listmax = 1024;
  datamax = 1024;

status = cainit(&header, camac_list, &listmax, LongReadBuffer, &datamax, errarr,
                &zero, &zero, &zero, &zero, errarr);
  if ((status & 1) != 1)
  {
    printf("****ERROR**** cainit\n");
    camsg(errstat);
  }
  c=1;
  n=1;
  f=16;
  a=0;
  qmode = QIGN | WTS24 ;
  lwdata = 0x112233;
  status = cainaf (&header, &c, &n, &a, &f, &qmode, &lwdata, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** cainaf\n");
   camsg(errstat);
```

```
  }
  status = cahalt(&header, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** cahalt\n");
   camsg(errstat);
  }
  status = caexew(&header, &hdl, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** caexew\n");
   camsg(errstat);
   exit(status);
  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** caclos\n");
   camsg(errstat);
   exit(status);
  }
}
```

4.2.6 cainit

Syntax

```
int cainit(   struct s_header *header,
              void    *control_list,
              int     *control_list_size,
              short   *data_buffer,
              int     *data_buffer_size,
              int     *status_buffer,
              int     *WC_buffer,
              int     *WC_buffer_size,
              int     *QXE_buffer,
              int     *QXE_buffer_size,
              int     *error);
```

Purpose

The *cainit* function initializes the CAMAC list building header. This function must be called prior to building a CAMAC command list.

Description

The *cainit* function is used to initialize the header and other data structures for the CAMAC command lists. This function should be called whenever a new CAMAC command list is built. The header holds the sizes, lengths, and pointers to other data structures used during processing time. The header is a parameter for all subsequent calls to the list building functions.

The *cainit* function requires array parameters that should be declared sufficiently large enough to contain all the list processing elements and data buffers. The data buffer used for the CAMAC command list must be large enough to "hold" all the data for a CAMAC list operation. All data transferred throughout the list operation is moved through this buffer. When generating a list of operations, a tally must be made to ensure that the total number of data words transferred for each individual list element is accounted for when determining the total data buffer size.

Parameters

Parameter Name	Direction	Description
header	Input	The header information contains pointer to other data structures, lengths of structures, and other vital information regarding the operation of the CAMAC command processor.
control_list	Input	This array is used to hold the CAMAC command list. The control list should be declared as a longword array with a size of *control_list_size*.
control_list_size	Input	This parameter specifies the number of elements available in the CAMAC command list.
data_buffer	Input/Output	This array holds all the data for the associated data transfers contained in the CAMAC command list. The shortword array should be declared with a size of *data_buffer-size*. The address of this array must be longword aligned and is initialized when the *cainit* function is used.
data_buffer_size	Input	This parameter specifies the size of the *data_buffer*. The buffer must be declared by the user sufficiently large so that the array can hold all requests from the CAMAC command list.
status_buffer	Input	This parameter is not used but here for legacy support. It may be set to a pointer to a value of zero.
WC_buffer	Input	This parameter is not used but here for legacy support. It may be set to a pointer to a value of zero.
WC_buffer_size	Input	This parameter is not used but here for legacy support. It may be set to a pointer to a value of zero.
QXE_buffer	Input	This parameter is not used but here for legacy support. It may be set to a pointer to a value of zero.
QXE_buffer_size	Input	This parameter is not used but here for legacy support. It may be set to a pointer to a value of zero.
error	Output	Error Code

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR103 Th header size does not match the header size of the current version.

ERR141	Data buffer not longword aligned.
ERR142	Control list buffer not longword aligned.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"


main()
{
  int     status;           // return status from the functions
  char    devname[] = "kpa00";
  int     *hdl;                                   // Handle for operations
  int     errstat[STAMAX];                        // array with list of errors
  short n;                                        // slot
  short a;                                        // sub address
  short f;                                        // function
  short c;                                        // crate
  short qmode;                                    // q mode for transfer
  int LongReadBuffer[8192];                       // short write data buffer
  unsigned long TransferCount;                    // transfer count for block
  int camac_list[8192];                           // list for camac processing
  int DataIndex;                                  // data index
  int listmax;                                    // max size of list
  int datamax;                                    // max size of data
  int     indicator;
  int     zero=0;                                 // zero value
  struct s_header header;
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
  listmax = 1024;
```

```
    datamax = 1024;

    n=1;
    f=16;
    a=0;
    qmode = QIGN | WTS24 ;
    status = cainit(&header, camac_list, &listmax, LongReadBuffer, &datamax, errarr,
                    &zero, &zero, &zero, &zero, errarr);
    if ((status & 1) != 1)
    {
      printf("****ERROR**** cainit\n");
      camsg(errstat);
    }
    c=1;
    n=1;
    f=0;
    a=0;
    qmode = QIGN | WTS24 ;
    TransferCount = 100 ;
    status = cablk (&header, &c, &n, &a, &f, &qmode, &TransferCount, &DataIndex, errarr);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** cablk\n");
     camsg(errstat);
    }
    status = cahalt(&header, errarr);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** cahalt\n");
     camsg(errstat);
    }
    status = caexew(&header, &hdl, errarr);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** caexew\n");
     camsg(errstat);
     exit(status);
    }
//
// Close the device
//
    status = caclos (&hdl, errstat);
    if ((status & 1) != 1)
    {
     printf("****ERROR**** caclos\n");
     camsg(errstat);
     exit(status);
    }
}
```

4.2.7 canaf

Syntax

```
int canaf(    struct s_header *header,
              short           *c,
              short           *n,
              short           *a,
              short           *f,
              short           *mode,
              int             *DatInd,
              int             *error
          );
```

Purpose

The *canaf* function adds a command to the CAMAC command list which when executed results in a single CAMAC transfer.

Description

The *canaf* function is used to add a single CAMAC transfer operation to the CAMAC command list. This function will only execute a single transfer. For block transfer operations, the *cablk* list instruction should be used.

This command will allocate one element in the CAMAC command list. If the CAMAC operation is a read or write operation, then space in the data buffer will also be allocated for the command entry. The parameter *Datind* will be returned with a value corresponding to the index into the data buffer where the commands' data is located. Note that the data buffer used to transfer data for these single operations is specified in the *cainit* function.

The *canaf* function supports all four types of transfer operations. These four modes consist of Q-Ignore, Q-Stop, Q-Repeat and Q-Scan. Please refer to the *Transfer Mode* section of this manual for details on each operating mode. A portion of the *mode* parameter for this function is used to indicate the block transfer-operating mode.

Parameters

Parameter Name	Direction	Description
header	Input	Header array that is built by the *cainit* function and contains pointers to the CAMAC command list and data buffer. This is updated as additional list elements are added.
C	Input	Address of the chassis to be accessed
N	Input	Slot number of the module to be accessed
A	Input	Subaddress within the module to be accessed

Parameter Name	Direction	Description
F	Input	Function code to be performed
mode	Input	Type of CAMAC block transfer to perform. Please refer to *Transfer Mode* section of this manual for additional information. (See below)
DatInd	Output	This parameter is returned with the index in to the data buffer marking the starting location for the word of data used for the operation.
error	Output	Error Code

The *mode* parameter in the *canaf* function is used to specify the CAMAC transfer Q-mode, the CAMAC data word size, and the termination technique used when a No-X condition occurs. Even though the *canaf* only executes a single write operation, the CAMAC Q and X returns are evaluated and continued list processing is based on their values as it relates to the selected Q-mode. The following table shows the available selections as *#defines* in the *kscuser.h* include file. Note that only one defined Q-mode or word size can be specified for each transfer.

#define	Description
QSTP	Selects the Q-Stop Block Transfer Mode
QIGN	Selects the Q-Ignore Block Transfer Mode
QRPT	Selects the Q-Repeat Block Transfer Mode
QSCN	Selects the Q-Scan Block Transfer Mode

Q-mode Transfer Selection

#define	Description
WTS16	Selects 16-bit CAMAC Data Word Size
WTS24	Selects 24-bit CAMAC Data Word Size

CAMAC Data Word Size Selection

#define	Description
AD	Inclusion of the *AD* in the mode description causes the CAMAC command processor to ignore No-X conditions during processing.

CAMAC Data Word Size Selection

Return Values

The most common error codes are listed here. For a comprehensive list, please refer to the *Error Codes* section of this manual.

ERR143 The CAMAC header is not initialized.

ERR144 The CAMAC header is initialized, but not correctly.

ERR202 A CAMAC in-line read operation was specified. Only CAMAC write and control functions can be specified as in-line operations.

Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ksc_api.h"
#include "kscuser.h"
#include "camerr.h"
#include "strfunc.h"
#include "cmdlist.h"

main()
{
  int   status;              // return status from the functions
  char  devname[] = "kpa00";
  int   *hdl;                                   // Handle for operations
  int   errstat[STAMAX];                        // array with list of errors
  short n;                                      // slot
  short a;                                      // sub address
  short f;                                      // function
  short c;                                      // crate
  short qmode;                                  // q mode for transfer
  int LongReadBuffer[8192];                     // short write data buffer
  int camac_list[8192];                         // list for camac processing
  int DataIndex;                                // data index
  int listmax;                                  // max size of list
  int datamax;                                  // max size of data
  int   indicator;
  int   zero=0;                                 // zero value
  struct s_header header;
//
// Open the device
//
  status = caopen(&hdl, devname, errstat);

//
// Check if device opened properly
//
  if ((status & 1) == 0)
  {
        printf("CAOPEN, error opening device = %s\n", devname);
        camsg(errstat);
        exit(status);

  }
  listmax = 1024;
  datamax = 1024;

status = cainit(&header, camac_list, &listmax, LongReadBuffer, &datamax, errarr,
                &zero, &zero, &zero, &zero, errarr);
  if ((status & 1) != 1)
  {
    printf("****ERROR**** cainit\n");
```

```
    camsg(errstat);
  }
  c=1;
  n=1;
  f=0;
  a=0;
  qmode = QIGN | WTS24 ;
  TransferCount = 100 ;
  status = canaf (&header, &c, &n, &a, &f, &qmode,  &DataIndex, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** canaf\n");
   camsg(errstat);
  }
  status = cahalt(&header, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** cahalt\n");
   camsg(errstat);
  }
  status = caexew(&header, &hdl, errarr);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** caexew\n");
   camsg(errstat);
   exit(status);
  }
//
// Close the device
//
  status = caclos (&hdl, errstat);
  if ((status & 1) != 1)
  {
   printf("****ERROR**** caclos\n");
   camsg(errstat);
   exit(status);
  }
}
```

5 Demands

5.1 The Demand Process

The Demand Process acts as a server to application processes for handling demands from the KSC2915 device driver. Application processes send registration requests to the Demand Process for all demands received from the CAMAC system they wish to service. Demands are enabled by the Demand Process for each CAMAC crate if demands are not currently enabled on the crate. 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).

5.2 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:

Crate Number	Demand ID	Crate Type	Demand Queue Length	Description
0 to 7	0 to 255	2	10	text comment

The syntax of the demand configuration file is:

crate, id, type, qlength, desc

Where:

crate	Decimal Crate number on the K-bus Parallel Bus.
Id	Demand Id generated by the Crate. See the 2915 and 3922 CAMAC controllers for description.
type	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.
```

!Crate !	Demand Id	Type	Queue Length	Description
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

5.3 Application Registration for Demands

The Demand Process establishes a single system-wide named pipe "\\.\pipe\dmdproc". The Demand Process reads registration requests from user processes (see KSC_enable_demand). 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.

5.4 Demand Processing

The 2915 driver returns demands to the Demand Process in a 9-element long word array. The first element of the array contains the contents of the 2915 Service Request Register. Bits 0 through 7 represent crates 0 through 7 with the respective bit set to 1 if a LAM is enabled in any of the crates. The 2nd through 9th elements contain the LAM status register for each crate (a maximum of 8 crates).

The Demand Process looks at each of the crate LAM status registers which is a 24-bit register that indicates the present state of all LAM requests in the CAMAC crate. Each bit corresponds to the appropriate Station in the crate. The Demand Process will attempt to clear the module LAM by first reading the crate LAM Status Register with a F(1) A(12). If the read is successful an attempt is made to clear the LAMs using F(23) A(12) using the LAM status as the clear pattern. If this didn't work then F(11) A(12) is tried to clear all LAMs within the module. If this doesn't work then LAMs are cleared using F(10) A(0). Finally F(10) A(i) where i corresponds to the bit positions set in the LAM Status Register is tried.

For each LAM asserted the crate 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 increment the unknown-demand counter. 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
Crate Number of Demand
Demand ID in Crate
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 crate, the Demand Process will disable demand recognition for that crate.

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, 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 attempt to clear the LAM. This attempt to clear competes with all other requests to the device driver and will effect the response of the demand delivery to the registered process.

5.5 User Application Program

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

All Application programs must contain the following elements (see program \KPA001\EXAMPLES\TEST_DMD.C):

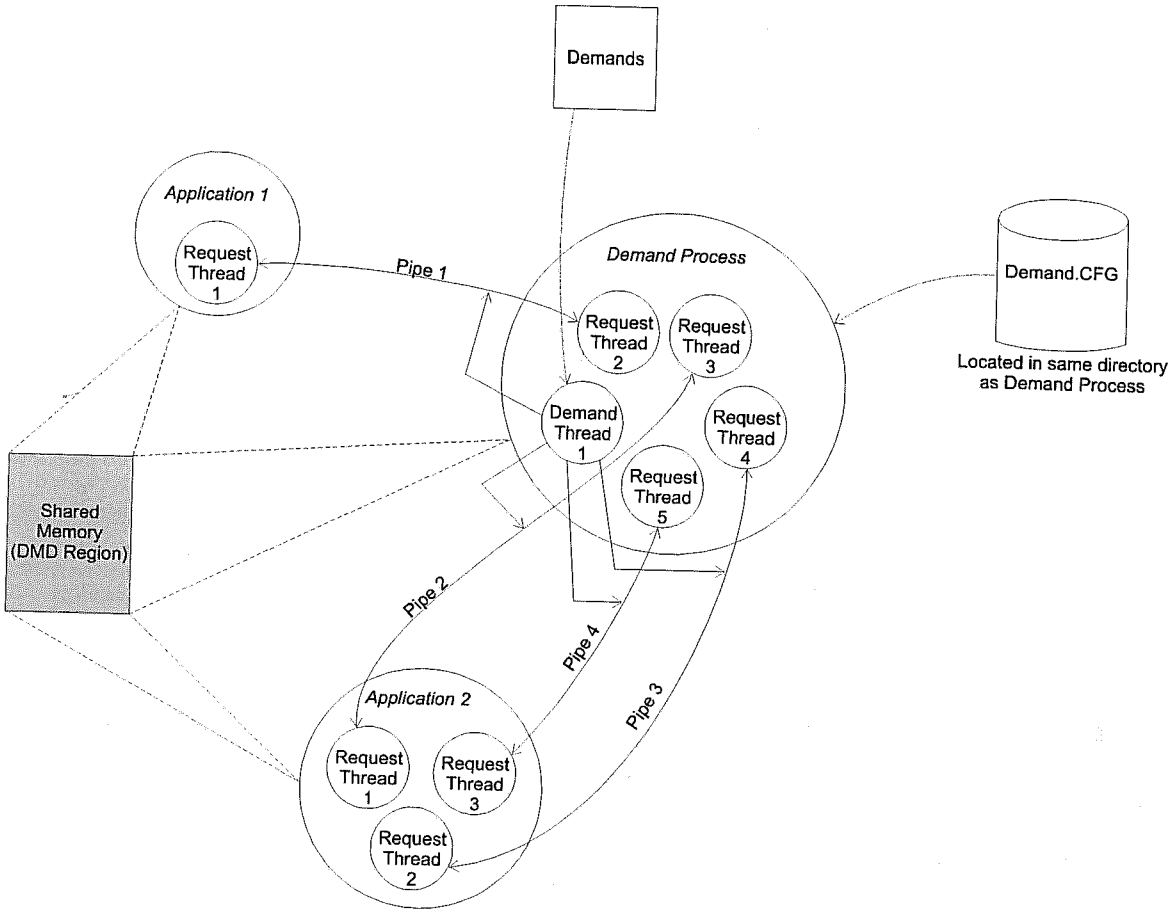
- Call KSC_init to create the structure KSC_handle required by all other KSC and CAM modules.
- Call KSC_enable_demand for each demand to be received. The application maps the demand region to ensure the Demand Process is running, and another process is not currently capturing the demand. This module 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 that 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, showing a process registering for LAMS (Demands) using the CAMAC library.

5.6 Demand Process Dataflow

The Win2K (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 KSC2915. 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

5.7 Demand Utilities

5.7.1 Program DMDSTS

The DMDSTS program is a diagnostic that reads 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

Dump Mode

Usage:	DMDSTS /CRATE=x	Region header plus crate "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. Not valid with /CONT switch.

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
/CRATE=x	Lowest, overrides no other switches

For each switch, the information displayed is:

Switch	Region Header	Crate 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
/CRATE=x	X	Only crate "x", even if not config or not enabled	All for the crate

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. The /OUT switch will be ignored if used the /CONT switch.

6 APPENDIX A - 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. Some error codes listed are from legacy device drivers, but are included here for completeness.

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.

103. The Header size does not match the Header size of the current version.

104. The length of the CAMAC Control List is greater than the specified size of the CAMAC Control List.

105. The Status Buffer size does not match the Status Buffer size of the current version.

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) .

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 longword.

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.

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.

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 longword.

122. The length of the CAMAC Control List is over 32K-1 words. The largest CAMAC Control List allowed is 32K-1 words.

123. The CAMAC Control List has a size of zero but must have a size of at least one.

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.

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.

129. The size of the QXE Buffer is over 32K-1 words. The largest QXE Buffer allowed is 32K-1 words.

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.

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.

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.

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.

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.

203. An illegal LAM type was specified, the command types are zero through seven.

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.

205. The remainder of the Data Buffer is too small to hold the data for the CAMAC block transfer.

206. An illegal CAMAC word size for the CAMAC device was encountered.

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.

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. 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.

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.

708. Invalid priority.

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).

711. The Data Buffer is not big enough to hold all the data for the CAMAC Control List.

712. The CAMAC Control List is not big enough to hold all the commands.

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.