CAMP Programmer's Guide

Order Number: TRI-CD-96-??

April 1999

This document describes how to write Instrument drivers and client applications for the CAMP (Control and Monitoring of Peripherals) system.

Revision/Update Information:	This manual has been revised for v1.3
Software Version:	CAMP v1.3
Author:	Ted Whidden DAS Group, TRIUMF

TRIUMF, Vancouver, B.C.

Contents

EXAMPLES

TABLES

Preface

Intended Audience

This manual is intended for programmers who will be writing instrument drivers or client applications for CAMP.

Associated Documents

- The *Operation Manual for MuSR at TRIUMF* describes how to use the CAMP online user interface.

Conventions

The following conventions are used in this manual:

Convention	Meaning
Ctrl-x	A sequence such as Ctrl-x indicates that you must hold down the Ctrl key while you press another key or pointing device button.
[ ]	Brackets indicate that whatever is enclosed is optional; you can select none, one, or all of the choices.
{}	Braces surround a required choice of options; you must choose one of the options listed.
|	The OR symbol separates alternatives within braces or brackets.
<>	Angled brackets indicate that you should substitute a word or value. Text within the brackets describes the appropriate substitution.
bold text	Boldface text represents the introduction of new terms.

1 Definitions

2 Client Applications

2.1 Introduction

Each CAMP client keeps a local copy of the CAMP data structures that have been requested. This includes a *system section* that contains general information about the CAMP System, and a *variable list* which is a linked-list of the data structures for every variable of every Instrument. The CAMP API maintains these lists, that is, the allocating and updating of the data is done through a high-level interface.

2.2 Routines

The routines available to client applications are organized into groups as described in Table 2–1.

Table 2–1 Summary of client routines

Name	Description
camp_clnt*	routines to maintain the client connection, etc.
camp_ins*	routines for accessing the local Instrument data
camp_path*	routines for maintaining CAMP paths
camp_var*	routines for accessing the local Variable data
campSrv_*	routines that make RPC requests to the CAMP Server

3 Command Line Interface

The CAMP Command Line Interface (*CLI*) is used to issue requests to the CAMP Server. It is intended for debugging, for online user modifications for and simple CAMP applications. Only a subset of the information stored in the CAMP Server can be retrieved using the Command Line Interface.

3.1 Invoking

The CAMP Command Line Interface is executed with the command:

```
$ camp_cmd [-node nodename] "command"
```

Where *nodename* is the optional Internet nodename of the CAMP Server and *command* is a valid CAMP CLI command. See Appendix B for the format of CAMP CLI commands. Note that *command* must be enclosed within quotation marks, because CAMP commands are case-sensitive.

4 Instrument Drivers

CAMP Instrument Drivers are written as interpreted scripts in the Tcl language. Script drivers are particularly useful because they necessitate no rebuilding of the CAMP Server and have been implemented so as to incur a minimal cost in speed. Script drivers also allow for rapid development and modification.

4.1 Interpreted Script Drivers

The CAMP Server has an embedded interpreter, *Tcl* (pronounced 'tickle'), which provides for the capability of script Instrument Drivers. The same interpreter is used for the parsing of all CAMP commands, making script drivers highly integrable with the CAMP Server. Tcl is a full interpreted language that provides features such as: variables, dynamic arrays (called *lists*), procedures, control flow, error messages, string manipulation, and file I/O. Information on the Tcl language is available in books and on the World Wide Web; start at *http://www.sco.com/Technology/tcl/Tcl.html*.

Script drivers consist of one file per driver, saved in (text) files named *camp:[drv]camp_ins_*.tcl* or */camp/drv/camp_ins_*.tcl* which must contain all of the information to define the variables and the the driver functions.

The arrangement or hierarchy of the device commands and parameters is represented in the form */~/variable* or */~/structure/variable*. Since the unique identifier of the Instrument is not known in the Definition file, the Instrument identifier is always replaced with the ~ character. This character will be expanded properly when the file is parsed. The arrangement is specified by the CAMP commands *CAMP_INSTRUMENT* and *CAMP_STRUCT*, and the driver must begin with a *CAMP_INSTRUMENT* command. These and all CAMP Driver commands are defined in Appendix C.

Other Variable definition commands will follow the Instrument Variable definition. If a Structure Variable is used, ensure that the Variables that will be placed within the Structure are defined after the Structure Variable. Other variables are defined with commands like *CAMP_FLOAT* (numeric) and *CAMP_SELECT* (selection menu). These are all described in Appendix C.

Besides CAMP Variables, a driver will often contain Tcl procedure definitions, which are invoked by the CAMP Variables' readProc, writeProc, etc. These procedures should be named so as to not conflict with procedures in other instrument definitions! Therefore "*proc lake330_set_curve*" rather than "*proc set_curve*". Additionally, there is usually an *insSet* command to set the default instrument interface parameters.

A sample script driver file is shown in Example 4–1. In this example, an instrument with four Variables is defined. The Instrument Variable has minimal procedures *onlineProc* and *offlineProc*. The Variables have procedures defined according to their read and set attributes. Note that a procedure, *lake622_read*, is defined after the Variables which is called by all of the *readProc* procedures. The Tcl interpreter stores this procedure permanently when the file is first read (i.e., when the Instrument is added). The procedure is not deleted when the Instrument is deleted in case any other instances of the Instrument type exist, which would be using the same copy of the procedure. Be careful that multiple instances of the same Instrument type do not have Tcl variables. This can be avoided by making the Tcl variable/procedure names unique to the instance of the Instrument. These unique Tcl variables should then be explicitly deleted when the Instrument is deleted to prevent memory leaks. Note also in the example the use of a call to *insSet* to initialize the setting of the interface.

Tcl drivers have been found to be robust, fast, simple, quick to write and sophisticated. They have become the favoured method of writing drivers and have thus replaced compiled drivers. The embedded aspect of Tcl allows callable commands to be added very easily in the future. If, however, this is not sufficient, compiled drivers may be written. The system provides all of the necessary functionality for, and has implemented, drivers written in DCL, but this has been found to be highly inefficient and is not recommended.

4.2 Definition Files

The purpose of the Instrument Definition file is to define the Variable tree of the Instrument if it has a compiled driver. Since Tcl script drivers are used for everything, Definition Files are no longer used.

The file consists of lines of commands that are similar in format to other CAMP commands. The format of the commands is given in Appendix C. The filename must be of the form *camp_ins_*.def*. This file is installed in the directory *camp:*.

The first command of the Definition file must be the definition of the Instrument Variable. The Variables that follow will be below the Instrument Variable in the path. Since the unique identifier of the Instrument is not known in the Definition file, the Instrument identifier is always replaced with the ~ character. This character will be expanded properly when the file is parsed. Variable definition commands will follow the Instrument Variable definition. If a Structure Variable is used, ensure that the Variables that will be placed within the Structure are defined after the Structure Variable. A sample Definition file is shown in Example 4–2.

Example 4–1 Sample Tcl Script Instrument Driver

```
CAMP_INSTRUMENT /~ -D -T "LakeShore 622 MPS" \
    -H "LakeShore 622 Magnet Power Supply" -d on \
    -onlineProc { insIfOn /~ } -offlineProc { insIfOff /~ }
    CAMP_FLOAT /~/i_out -D -R -P -L -A \
        -T "Output current" \
        -d on -r on -p_int 30 \
        -readProc {
            lake622_read /~/i_out "IOUT?" "%f"
        }
    CAMP_FLOAT /~/ramp_trgt -D -S -R -L -A \
        -T "Ramp target current" \
        -d on -s on -r on \
        -readProc {
            lake622_read /~/ramp_trgt "RAMP?" "RAMP1,%*f,%f,%*f"
        } -writeProc {
            set target [format "%+.4f" $camp_target]
            set rate [format "%+.4f" [varGetVal /~/ramp_rate]]
            insIfWrite /~ "RAMP1,0,$target,$rate"
            varRead /~/ramp_trgt
         }
    CAMP_FLOAT /~/ramp_rate -D -S -R -L -A \
        -T "Ramp rate (A/s)" \
        -d on -s on -r on \
        -readProc {
            lake622_read /~/ramp_rate "RAMP?" "RAMP1,%*f,%*f,%f"
        } -writeProc {
            set rate [format "%+.4f" $camp_target]
            set target [format "%+.4f" [varGetVal /~/ramp_trgt]]
            insIfWrite /~ "RAMP1,0,$target,$rate"
            varRead /~/ramp_rate
        }
    CAMP_SELECT /~/ramp_stat -D -S -R -P -L -A \
        -T "Ramp status" -selections HOLDING RAMPING \
        -d on -s on -r on -p_int 30 \
        -readProc {
            lake622_read /~/ramp_stat "RMP?" "%d"
        } -writeProc {
            insIfWrite /~ "RMP$camp_target"
            varRead /~/ramp_stat
        }
proc lake622_read { path cmd fmt } {
    set buf [insIfRead $path $cmd 32]
    set status [scan $buf $fmt val]
    if { $status != 1 } { return -code error
                          "failed parsing \"$buf\"" }
    varDoSet $path -v $val
    return -code ok $buf
}
insSet /~ -if_set rs232_tt 0.5 "undefined" 9600 7 odd 1 \
                           CRLF CRLF 2 0
```

Example 4–2 Sample Instrument Driver Definition File

```
CAMP_INSTRUMENT /~ -D -T "LakeShore 622 MPS" \
    -H "LakeShore 622 Magnet Power Supply" -d on
    CAMP_FLOAT /~/i_set -D -S -R -L -A \
        -T "Output current setting" \
        -d on -s on -r on -p_int 30
    CAMP_FLOAT /~/i_out -D -R -P -L -A \
        -T "Output current" -d on -r on -p_int 30
    CAMP_STRUCT /~/heat -D -d on
        CAMP_INT /~/heat/heat_set -D -S -R -L -A \
            -T "Heater current setting (mA)" \
            -d on -s on -r on
        CAMP_SELECT /~/heat/heat_stat -D -S -R -P -L -A \
            -T "Heater status" \
            -d on -s on -r on -p_int 30 -selections OFF ON
```

A Client Routines

All function names and associated arguments beginning with the "f_" prefix are intended for FORTRAN compatibility.

The argument type *boolean* is equivalent to *longword (unsigned)* under VAX/VMS.

The *campSrv_** routines each make one RPC call to the CAMP Server. The return values that these calls hold in common are listed in Table A–1.

Table A–1 Return values for *campSrv_ routines**

Name	Description	Cause
CAMP_FAIL_RPC	RPC call failed	Server has fallen off, or timeout due to Server being slow or stuck.

camp_clntEnd()—end CAMP connection

Finish being a CAMP client. See also camp_clntInit.

FORMAT

camp_clntEnd*()*
f_camp_clntEnd*()*

RETURNS

None.

ARGUMENTS

None.

DESCRIPTION

The *camp_clntEnd()* function ends the CAMP client connection. The function will free all memory allocated for data received from the CAMP Server and the RPC client and authorization structures. A call to this routine is necessary for each call to *camp_clntInit()*.

camp_clntInit()—initialize CAMP connection

Initialize as a CAMP client. See also camp_clntEnd

FORMAT

status = **camp_clntInit**(*serverName, clientTimeout*)
status = **f_camp_clntInit**(*f_serverName,*
f_clientTimeout)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

serverName

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The name of the host to connect to.

f_serverName

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

clientTimeout

type:	**longword**
access:	**read only**
mechanism:	**by value**

The timeout value for all calls to the server in seconds.

f_clientTimeout

type:	**longword**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *camp_clntInit()* function initializes the CAMP client connection. The function will check for the existance of the hostname, then check for the Server program on the host, and then get a copy of the CAMP Server System Section. Be sure to call *camp_clntEnd()* to end the client connection.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_FAILURE	Failed to initialize as client.

Any condition values returned by *campSrv_sysGet( )*.

camp_clntUpdate()—update CAMP data

Update the local copy of the CAMP data list.

FORMAT

status = **camp_clntUpdate***()*
status = **f_camp_clntUpdate***()*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

None.

DESCRIPTION

The *camp_clntUpdate()* function updates the local copy of the CAMP data. The routine first calls the Server for a small status structure. If the Server's system section has changed since the last update, the client will request an update. Then, if any Instruments have been added or deleted since the last call to *camp_clntUpdate()*, the complete variable list will be updated. If the application never intends to receive a copy of the complete variable list, or indeed the system section, then *camp_clntUpdate()* should not be used. Instead, more specific calls to *campSrv_varGet()* should be made.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.

Any condition values returned by *campSrv_sysGetDyna()*, *campSrv_sysGet()* or *campSrv_varGet()*.

camp_insGetFile()—get Instrument initialization file name

Get the local value of an Instrument initialization file name.

FORMAT

status = **camp_insGetFile***(path, filename)*
status = **f_camp_insGetFile***(f_path, f_filename)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

filename

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full file path of the Instrument's initialization file.

f_filename

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_insGetFile()* routine returns the local value of the Instrument initialization file name.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The Instrument does not exist locally.

camp_insGetIfTypeIdent()—get Instrument interface type ident

Get the local value of an Instrument interface type identifier.

FORMAT

status = **camp_insGetIfTypeIdent**(*path, typeIdent*)
status = **f_camp_insGetIfTypeIdent**(*f_path, f_typeIdent*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

typeIdent

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The location to return the Instrument interface's type identifier.

f_typeIdent

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_insGetTypeIdent()* routine returns the local value of the Instrument interface type identifier.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The Instrument does not exist locally.
CAMP_INVAL_IF	Instrument interface is undefined.

camp_insGetLine()—get Instrument line status

Get the local value of an Instrument line status.

FORMAT

status **= camp_insGetLine***(path, pFlag)*
status **= f_camp_insGetLine***(f_path, pFlag)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pFlag

type:	**boolean**
access:	**write only**
mechanism:	**by reference**

The location to put the status flag.

DESCRIPTION

The *camp_insGetLine()* routine returns the local value of the Instrument line status.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The Instrument does not exist locally.

camp_insGetLock()—get Instrument lock status

Get the local value of an Instrument lock status.

FORMAT

status **= camp_insGetLock***(path, pFlag)*
status **= f_camp_insGetLock***(f_path, pFlag)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pFlag

type:	**boolean**
access:	**write only**
mechanism:	**by reference**

The location to put the status flag.

DESCRIPTION

The *camp_insGetLock()* routine returns the local value of the Instrument lock status.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The Instrument does not exist locally.

camp_insGetTypeIdent()—get Instrument type ident

Get the local value of an Instrument type identifier.

FORMAT

status = **camp_insGetTypeIdent**(*path, typeIdent*)
status = **f_camp_insGetTypeIdent**(*f_path, f_typeIdent*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

typeIdent

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The location to return the Instrument's type identifier.

f_typeIdent

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_insGetTypeIdent()* routine returns the local value of the Instrument type identifier.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The Instrument does not exist locally.

camp_pathAtTop()—test if path is at root

Test whether the specified path is at the root.

FORMAT

status = **camp_pathAtTop**(*path*)
status = **f_camp_pathAtTop**(*f_path*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathAtTop()* routine simply tests whether the given path is at the root of the tree (i.e., is equal to "/").

RETURN VALUES

0	False.
1	True.

camp_pathCompare()—compare two paths for equality

Compare two CAMP paths for equality.

FORMAT

status **= camp_pathCompare***(path1, path2)*
status **= f_camp_pathCompare***(f_path1, f_path2)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathCompare()* routine tests whether the given paths are equal. It does this by expanding the paths (i.e., in case the default character "~" has been used) and does a string comarison on the expanded strings.

RETURN VALUES

0	False.
1	True.

camp_pathDown()—navigate down a level

Navigate down a level of a path.

FORMAT

path **= camp_pathDown**(*path, ident*)
f_camp_pathDown(*f_path, f_ident*)

RETURNS

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**modify**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**modify**
mechanism:	**by descriptor**

ident

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The identifier to append to *path*.

f_ident

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathDown()* routine appends an identifier to *path*.

RETURN VALUES

The pointer to *path* is returned.

camp_pathDownNext()—navigate down to the next level

Navigate down to the next level of a path.

FORMAT

path_curr = **camp_pathDownNext***(path, path_curr)*
f_camp_pathDownNext*(f_path, f_path_curr)*

RETURNS

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

path_curr

type:	**ASCIZ string**
access:	**modify**
mechanism:	**by reference**

A subpath of *path*. This path will be appended with the next identifier from *path*.

f_path_curr

type:	**character string**
access:	**modify**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathDownNext()* routine appends the next identifier in *path* to the subpath *path_curr*.

RETURN VALUES

The pointer to *path_curr* is returned.

camp_pathGetFirst()—get first path ident

Return the first identifier in the path.

FORMAT

ident = **camp_pathGetFirst**(*path, ident*)
f_camp_pathGetFirst(*f_path, f_ident*)

RETURNS

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

ident

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

Location to return the first identifier in the path.

f_ident

type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathGetfirst()* routine returns the first identifier in the given path.

RETURN VALUES

The pointer to *ident* is returned.

camp_pathGetLast()—get last path ident

Return the last identifier in the path.

FORMAT

ident = **camp_pathGetLast**(*path, ident*)
f_camp_pathGetLast(*f_path, f_ident*)

RETURNS

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

ident

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

Location to return the last identifier in the path.

f_ident

type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathGetLast()* routine returns the last identifier in the given path.

RETURN VALUES

The pointer to *ident* is returned.

camp_pathGetNext()—get next path ident

Return the next identifier in the given path.

FORMAT

ident = **camp_pathGetNext**(*path, path_sub, ident*)
f_camp_pathGetNext(*f_path, f_path_sub, f_ident*)

RETURNS

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

path_sub

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A subpath of *path*. *ident* will be the next identifier in *path* after the subpath *path_sub*.

f_path_sub

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

ident

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

Location to return the next identifier in the path.

f_ident

type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathGetNext()* routine returns the next identifier in *path* which follows the substring *path_sub*.

RETURN VALUES

The pointer to *ident* is returned.

camp_pathInit()—initialize a CAMP path

Initialize a CAMP path string.

FORMAT

camp_pathInit*(path)*
f_camp_pathInit*(f_path)*

RETURNS

None.

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The destination string.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathInit()* routine initializes a CAMP path string (namely to "/").

camp_pathUp()—navigate up a level

Navigate up a level of a path.

FORMAT

ident = **camp_pathUp**(*path*)
f_camp_pathUp(*f_path*)

RETURNS

type:	**ASCIZ string**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**modify**
mechanism:	**by reference**

A CAMP path.

f_path

type:	**character string**
access:	**modify**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_pathUp()* routine navigates *path* up one level.

RETURN VALUES

The pointer to *path* is returned.

camp_varCheckNumTol()—check the tolerance of Numeric

Check the tolerance constraints for a Numeric Variable’s target value.

FORMAT

status = **camp_varCheckNumTol***(path, target)*
status = **f_camp_varCheckNumTol***(f_path, f_target)*

RETURNS

type:	**boolean**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

target

type:	**D floating**
access:	**read only**
mechanism:	**by value**

A given target value to be used to compare with the Variable’s current value.

f_target

type:	**D floating**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *camp_varCheckNumTol()* routine compares *target* with the value of the Numeric Variable. If either the comparison under the constraint of percentage tolerance or minimum/maximum fails, then *FALSE* will be returned.

RETURN VALUES

The result of the comparison with *target* under the tolerance constraints is returned as a boolean.

camp_varExists()—test Variable existance

Test a Variable path for existance.

FORMAT

status = **camp_varExists***(path)*
status = **f_camp_varExists***(f_path)*

RETURNS

type:	**boolean**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *camp_varExists()* routine tests the Variable for existance in the local Variable list.

RETURN VALUES

The result of the test for existance is returned as a boolean.

camp_varGetAlarm()—get Variable alarm parameters

Get the local value of a Variable's alarm parameters.

FORMAT

status = **camp_varGetAlarm***(path, pFlag, action)*
status = **f_camp_varGetAlarm***(f_path, pFlag, f_action)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pFlag

type:	**boolean**
access:	**write only**
mechanism:	**by reference**

Location to return the alarm flag.

action

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The returned value of the alarm action.

f_action

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetLnk()—get Link Variable value

Get the local value of a Link Variable’s target value.

FORMAT

status **= camp_varGetLnk***(path, val)*
status **= f_camp_varGetLnk***(f_path, f_val)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

Location to return the target value.

f_val

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetLog()—get Variable logging parameters

Get the local value of a Variable's logging parameters.

FORMAT

status = **camp_varGetLog**(*path, pFlag, action*)
status = **f_camp_varGetLog**(*f_path, pFlag, f_action*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pFlag

type:	**boolean**
access:	**write only**
mechanism:	**by reference**

Location to return the logging flag.

action

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The returned value of the logging action.

f_action

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetNum()—get Numeric Variable value

Get the local value of a Numeric Variable's target value.

FORMAT

status **= camp_varGetNum***(path, pVal)*
status **= f_camp_varGetNum***(f_path, pVal)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pVal

type:	**D floating**
access:	**write only**
mechanism:	**by reference**

Location to return the target value.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetp()—get Variable pointer

Get the pointer to a Variable’s data structure.

FORMAT

pVar **= camp_varGetp***(path)*

RETURNS

type:	**CAMP_VAR structure**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

DESCRIPTION

The *camp_varGetp()* routine returns the local value of the pointer to a Variable’s data structure. This routine differs from camp_varGetTrueP( ) in that it will attempt to return the effective Variable of Link Variable’s and not the pointer to the Link Variable itself.

RETURN VALUES

The pointer to the Variable’s data structure is returned. This value will be *NULL* if *path* doesn’t exist locally. See Chapter 1 for definition of the structure.

camp_varGetpIns()—get Instrument pointer

Get the pointer to a Variable’s Instrument.

FORMAT

pIns **= camp_varGetpIns***(pVar)*

RETURNS

type:	**CAMP_VAR structure**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

pVar

type:	**CAMP_VAR structure**
access:	**read only**
mechanism:	**by reference**

The pointer to the Variable for which the Instrument pointer is desired.

DESCRIPTION

The *camp_varGetpIns()* routine returns the local value of the pointer to a Variable’s Instrument Variable.

RETURN VALUES

The pointer to the Instrument Variable’s data structure is returned. This value will be *NULL* if *path* doesn’t exist locally. See Chapter 1 for definition of the structure.

camp_varGetPoll()—get Variable polling parameters

Get the local value of a Variable’s polling parameters.

FORMAT

status = **camp_varGetPoll**(*path, pFlag, pInterval*)
status = **f_camp_varGetPoll**(*f_path, pFlag, pInterval*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pFlag

type:	**boolean**
access:	**write only**
mechanism:	**by reference**

Location to return the polling flag.

pInterval

type:	**F floating**
access:	**write only**
mechanism:	**by reference**

Location to return the polling interval.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetSelID()—get Selection Variable index value

Get the local value of a Selection Variable's target value as an index.

FORMAT

status = **camp_varGetSelID**(*path, pVal*)
status = **f_camp_varGetSelID**(*f_path, pVal*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pVal

type:	**byte (unsigned)**
access:	**read only**
mechanism:	**by reference**

Location to return the target value index.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetSelIDLabel()—get Selection Variable label

Get a Selection Variable's label for a given index.

FORMAT

status = **camp_varGetSelIDLabel**(*path, val, label*)
status = **f_camp_varGetSelIDLabel**(*f_path, f_val, f_label*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**byte (unsigned)**
access:	**read only**
mechanism:	**by value**

The index for which the label is desired.

f_val

type:	**byte (unsigned)**
access:	**read only**
mechanism:	**by reference**

label

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The location to return the label.

f_label

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetSelLabel()—get Selection Variable label value

Get the local value of a Selection Variable’s target value as a label.

FORMAT

status = **camp_varGetSelLabel**(*path, pVal*)
status = **f_camp_varGetSelLabel**(*f_path, pVal*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

Location to return the target value label.

f_val

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetSelLabelID()—get Selection Variable index

Get a Selection Variable’s index for a given label.

FORMAT

status **= camp_varGetSelLabelID***(path, label, pVal)*
status **= f_camp_varGetSelLabelID***(f_path, f_label, pVal)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

label

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The label for which the index is desired.

f_label

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pVal

type:	**byte (unsigned)**
access:	**read only**
mechanism:	**by reference**

The location to return the index.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetStats()—get Variable statistics

Get the local value of a Variable's statistics.

FORMAT

status = **camp_varGetAlarm**(*path, pN, pLow, pHi, pOffset, pSum, pSumSq, pSumCub*)

status = **f_camp_varGetAlarm**(*f_path, pN, pLow, pHi, pOffset, pSum, pSumSq, pSumCub*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pN

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

Location to return the number of readings.

pLow

type:	**D floating**
access:	**write only**
mechanism:	**by reference**

Location to return the low value.

pHi

type:	**D floating**
access:	**write only**
mechanism:	**by reference**

Location to return the high value.

pOffset

type: **D floating**
access: **write only**
mechanism: **by reference**
Location to return the offset value. This offset is used in calculating the sum, sum of squares, and sum of cubes, and must be considered when using these values.

pSum

type: **D floating**
access: **write only**
mechanism: **by reference**
Location to return the sum.

pSumSq

type: **D floating**
access: **write only**
mechanism: **by reference**
Location to return the sum of squares.

pSumCub

type: **D floating**
access: **write only**
mechanism: **by reference**
Location to return the sum of cubes.

DESCRIPTION

The *camp_varGetStats()* routine returns the statistics for a Numeric Variable.

Note: The values for sum, sum of squares, and sum of cubes have an incorporated offset. Use *camp_varNumCalcStats()* **to calculate useful statistics properly.**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetStatus()—get Variable status

Get the local value of a Variable's status.

FORMAT

status **= camp_varGetStatus***(path, pStatus)*
status **= f_camp_varGetStatus***(f_path, pStatus)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pStatus

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

Location to return the status value.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetStr()—get String Variable value

Get the local value of a String Variable’s target value.

FORMAT

status = **camp_varGetStr***(path, val)*
status = **f_camp_varGetStr***(f_path, f_val)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

Location to return the target value.

f_val

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varGetTrueP()—get true Variable pointer

Get the pointer to a Variable’s true data structure.

FORMAT

pVar = **camp_varGetTrueP***(path)*

RETURNS

type:	**CAMP_VAR structure**
access:	**write only**
mechanism:	**by reference**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

DESCRIPTION

The *camp_varGetTrueP()* routine returns the local value of the pointer to a Variable’s data structure. This routine differs from camp_varGetp( ) in that it return the pointer to Link Variables, and not the pointer to the effective Variable.

RETURN VALUES

The pointer to the Variable’s data structure is returned. This value will be *NULL* if *path* doesn’t exist locally. See Chapter 1 for definition of the structure.

camp_varGetValStr()—get Variable value as a string

Get the local value of a Variable’s target value as a string.

FORMAT

status = **camp_varGetValStr**(*path, val*)
status = **f_camp_varGetValStr**(*f_path, f_val*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

Location to return the target value. The string will be formatted accordingly for integer and floating-point types, and for Selection types the integer index is converted to a string. For String types no conversion is made.

f_val

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

camp_varNumCalcStats()—calculate Variable statistics

Calculate the statistics for a Numeric Variable.

FORMAT

status **= camp_varNumCalcStats***(path, pMean, pStdDev, pSkew)*

status **= f_camp_varNumCalcStats***(f_path, pMean, pStdDev, pSkew)*

RETURNS

type:	**boolean**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

pMean

type:	**D floating**
access:	**read only**
mechanism:	**by reference**

Location to return the mean.

pStdDev

type:	**D floating**
access:	**read only**
mechanism:	**by reference**

Location to return the standard deviation.

pSkew

type:	**D floating**
access:	**read only**
mechanism:	**by reference**

Location to return the skewness.

DESCRIPTION

The *camp_varNumCalcStats( )* routine calculates the statistics for a Numeric Variable. The sum, sum of squares, and sum of cubes are kept in the Variable's data structure, but this routine is necessary to convert to more useful numbers.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

campSrv_cmd()—send CAMP command string

Sends a command string to the CAMP interpreter.

FORMAT

status = **campSrv_cmd**(*cmd*)
status = **f_campSrv_cmd**(*f_cmd*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

cmd

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_cmd

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_cmd()* routine sends a command string to be parsed by the CAMP interpreter. This is equivalent to using the CAMP Command-line Interface (see *CAMP User Manual*).

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_FAILURE	An error occurred while parsing the command.

Any return values shown in Table A–1.

Other possibilities may depend upon the command string.

campSrv_insAdd()—add an Instrument

Issues a request to the CAMP Server to add an Instrument.

FORMAT

status = **campSrv_insAdd**(*typeIdent, ident*)
status = **f_campSrv_insAdd**(*f_typeIdent, f_ident*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

typeIdent

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The identifier of a valid CAMP Instrument type.

f_typeIdent

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

ident

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A unique identifier for the Instrument. Can be a maximum of 15 characters.

f_ident

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_insAdd()* routine issues a request to the CAMP Server to add a new Instrument. The operation will fail if the Instrument type is invalid, or if the unique identifier already exists.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS_TYPE	Instrument type not known to CAMP.

CAMP_INVAL_INS	Invalid Instrument identifier. Either there are invalid characters or it already exists.

Any return values shown in Table A–1.

campSrv_insDel()—delete an Instrument

Issues a request to the CAMP Server to delete an Instrument.

FORMAT

status = **campSrv_insDel**(*path*)
status = **f_campSrv_insDel**(*f_path*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_insDel()* routine issues a request to the CAMP Server to delete an Instrument. The operation will fail if the Instrument path does not exist, or if the Instrument is locked by a different process.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.
CAMP_INS_LOCKED	Could not delete the Instrument because it is locked by another process.

Any return values shown in Table A–1.

campSrv_insIf()—set an Instrument interface

Issues a request to the CAMP Server to set an Instrument interface.

FORMAT

status = **campSrv_insIf***(path, typeIdent, pSpec)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

typeIdent

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full path of the file to save.

pSpec

type:	**CAMP_IF_SPEC structure**
access:	**read only**
mechanism:	**by reference**

The structure defining the new interface setting. See Chapter 1 for the definition.

DESCRIPTION

The *campSrv_insIf()* routine issues a request to the CAMP Server to set an Instrument interface.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_CANT_SET_IF	Can't set the interface while the Instrument is online.
CAMP_INVAL_IF_TYPE	The requested interface type is undefined.

Any return values shown in Table A–1.

campSrv_insIfRead()—read from the physical instrument

Instructs the CAMP Server to read data from the physical instrument.

FORMAT

status **= campSrv_insIfRead***(path, cmd, cmd_len, buf_len)*

status **= f_campSrv_insIfRead***(f_path, f_cmd, f_cmd_len, f_buf_len)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

cmd

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A string to send to the physical instrument to prompt for the readback. Enclose this string within quotes if it contains any whitespaces.

f_cmd

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

cmd_len

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

The length (in bytes) of *cmd*.

f_cmd_len

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

buf_len

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

The length (in bytes) of the buffer to be allocated for the data that will be returned by the physical instrument. This value does not have to be the exact length of the returned data, but must be large enough to hold the data. It is recommended that one overestimate this value.

f_buf_len

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_insIfRead()* routine instructs the CAMP Server to send a command prompt to the physical instrument and then wait for data to be sent from the instrument.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.
CAMP_INVAL_INS_TYPE	Instrument type not known to CAMP.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_IF	Instrument interface is undefined.
CAMP_INVAL_IF_TYPE	The Instrument interface type is undefined.
CAMP_NOT_CONN	Instrument interface is not online.

Any return values shown in Table A–1.

Other possibilities will depend upon the interface type.

campSrv_insIfWrite()—write to the physical instrument

Instructs the CAMP Server to write data to the physical instrument.

FORMAT

status = **campSrv_insIfWrite**(*path, cmd, cmd_len*)
status = **f_campSrv_insIfWrite**(*f_path, f_cmd, f_cmd_len*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

cmd

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The string to be sent to the physical instrument. Enclose this string within quotes if it contains any whitespaces.

f_cmd

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

cmd_len

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

The length (in bytes) of *cmd*.

f_cmd_len

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_insIfWrite()* routine instructs the CAMP Server to send a command string to the physical instrument.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_IF	Instrument interface is undefined.
CAMP_INVAL_IF_TYPE	The Instrument interface type is undefined.
CAMP_NOT_CONN	Instrument interface is not online.

Any return values shown in Table A–1.

Other possibilities will depend upon the interface type.

campSrv_insLine()—set the Instrument line

Issues a request to the CAMP Server to set an Instrument online or offline.

FORMAT

status = **campSrv_insLine**(*path, flag*)
status = **f_campSrv_insLine**(*f_path, f_flag*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**boolean**
access:	**read only**
mechanism:	**by value**

Set *flag* to *FALSE* for offline or *TRUE* for online.

f_flag

type:	**boolean**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_insLine()* routine issues a request to the CAMP Server to set an Instrument online or offline. The operation will fail if the Instrument is already locked by a different process or if the interface is undefined.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.

CAMP_INVAL_IF	The interface is undefined.

Any return values shown in Table A–1.

campSrv_insLoad()—load an Instrument initialization

Issues a request to the CAMP Server to load an Instrument initialization.

FORMAT

status = **campSrv_insLoad***(path, filename, flag)*
status = **f_campSrv_insLoad***(f_path, f_filename, f_flag)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

filename

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full path of the file to load.

f_filename

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

Not used.

f_flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_insLoad()* routine issues a request to the CAMP Server to load an Instrument initialization file. See *CAMP User Manual* for information on the contents of these files.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_FILE	Invalid filename.
CAMP_FAILURE	There was an error while parsing the file.

Any return values shown in Table A–1.

campSrv_insLock()—(un)lock an Instrument

Issues a request to the CAMP Server to (un)lock an Instrument.

FORMAT

status = **campSrv_insLock**(*path, flag*)
status = **f_campSrv_insLock**(*f_path, f_flag*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**boolean**
access:	**read only**
mechanism:	**by value**

Set *flag* to *FALSE* for unlock or *TRUE* for lock.

f_flag

type:	**boolean**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_insLock()* routine issues a request to the CAMP Server to lock or unlock an Instrument. The operation will fail if the Instrument is already locked by a different process or if the request is made from a foreign host.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.

CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_LOCKER	The request was illegally made from a foreign host.

Any return values shown in Table A–1.

campSrv_insSave()—save an Instrument initialization

Issues a request to the CAMP Server to save an Instrument initialization.

FORMAT

status **= campSrv_insSave***(path, filename, flag)*
status **= f_campSrv_insSave***(f_path, f_filename, f_flag)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Instrument.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

filename

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full path of the file to save.

f_filename

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

Not used.

f_flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_insLoad()* routine issues a request to the CAMP Server to save an Instrument initialization file. See *CAMP User Manual* for information on the contents of these files.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_INS	Invalid Instrument identifier. The specified Instrument does not exist.
CAMP_INVAL_FILE	Invalid filename.
CAMP_FAILURE	There was an error while writing the file.

Any return values shown in Table A–1.

campSrv_sysDir()—do remote dir

Requests the CAMP Server to get a directory listing on the Server host.

FORMAT

status = **campSrv_sysDir**(*filespec*)
status = **f_campSrv_sysDir**(*f_filespec*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

filespec

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The file specification of the requested directory listing.

f_filespec

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_sysDir()* routine requests the CAMP Server to perform a directory listing using the *filespec* on the Server host and to then return this listing. The listing is returned in the *pDir* pointer in the system section (see Chapter 1).

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.

Any return values shown in Table A–1.

campSrv_sysGet()—get a copy of the system section

Requests the CAMP Server for a copy of the system section.

FORMAT

status **= campSrv_sysGet***()*
status **= f_campSrv_sysGet***()*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

None.

DESCRIPTION

The *campSrv_sysGet()* routine requests the CAMP Server for a complete copy of the system section.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.

Any return values shown in Table A–1.

campSrv_sysGetDyna()—get a copy of the status part of the system section

Requests the CAMP Server for a copy of the small status part of the system section.

FORMAT

status = **campSrv_sysGetDyna***()*
status = **f_campSrv_sysGetDyna***()*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

None.

DESCRIPTION

The *campSrv_sysGetDyna()* routine requests the CAMP Server for a copy of the small status structure of the system section. This structure is used to determine if an update of the local system section is necessary, and when the last Instrument was added or deleted.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.

Any return values shown in Table A–1.

campSrv_sysLoad()—load a configuration

Issues a request to the CAMP Server to load a new configuration.

FORMAT

status = **campSrv_sysLoad**(*filename, flag*)
status = **f_campSrv_sysLoad**(*f_filename, f_flag*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

filename

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full pathname of the file to load.

f_filename

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

If *flag* is 0 (zero), the configuration before loading a new configuration is retained. This is the default. If *flag* is 1 (one), all current instruments are deleted before loading the file. If any instruments are locked, the latter will fail.

f_flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_sysLoad()* routine issues a request to the CAMP Server to load a configuration file. As indicated in the parameters section, this may either add to or replace the current configuration. See *CAMP User Manual* for information on the contents of these files.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INS_LOCKED	Could not delete the current Instruments because there is a locked Instrument.
CAMP_INVAL_FILE	Could not find the specified file.
CAMP_FAILED	Failure during the parsing of the file.

Any return values shown in Table A–1.

campSrv_sysRundown()—rundown CAMP Server

Issues a request to the CAMP Server to rundown.

FORMAT

status = **campSrv_sysRundown***()*
status = **f_campSrv_sysRundown***()*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

None.

DESCRIPTION

The *campSrv_sysRundown()* routine issues a request to the CAMP Server to rundown. The Server will first check to see that there are no locked instruments. If there are none, the Server will exit cleanly.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INS_LOCKED	Could not rundown because there is a locked Instrument.

Any return values shown in Table A–1.

campSrv_sysSave()—save a configuration

Issues a request to the CAMP Server to save the configuration.

FORMAT

status = **campSrv_sysSave**(*filename, flag*)
status = **f_campSrv_sysSave**(*f_filename, f_flag*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

filename

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full pathname of the file to save.

f_filename

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

Not used.

f_flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_sysSave()* routine issues a request to the CAMP Server to save the configuration to a file. See *CAMP User Manual* for information on the contents of these files.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INS_LOCKED	Could not delete the current Instruments because there is a locked Instrument.
CAMP_INVAL_FILE	Could not find the specified file.

CAMP_FAILED	Failure during the parsing of the file.

Any return values shown in Table A–1.

campSrv_sysUpdate()—update the CAMP Server

Issues a request to the CAMP Server to update its internal parameters.

FORMAT

status = **campSrv_sysUpdate***()*
status = **f_campSrv_sysUpdate***()*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

None.

DESCRIPTION

The *campSrv_sysUpdate()* routine issues a request to the CAMP Server to update its internal parameters which are initialized at startup. These include the available:

- instrument types
- interface types
- alarm actions
- log actions

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.

Any return values shown in Table A–1.

campSrv_varAlarm()—set Variable alarm parameters

Requests the CAMP Server to set the Variable alarm parameters.

FORMAT

status **= campSrv_varAlarm***(path, flag, action)*
status **= f_campSrv_varAlarm***(f_path, f_flag, f_action)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**boolean**
access:	**read only**
mechanism:	**by value**

Set *flag* to *FALSE* for off or *TRUE* for on.

f_flag

type:	**boolean**
access:	**read only**
mechanism:	**by reference**

action

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A valid CAMP alarm action.

f_action

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_varAlarm()* routine requests the CAMP Server to set the alarm parameters of a Variable.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid Variable path. The specified Variable does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_DATA_ATTR	The Variable is not alarmable (does not have the ALARM attribute).

Any return values shown in Table A–1.

campSrv_varGet()—get CAMP Variable data

Get a copy of one or more CAMP Variables from the CAMP Server.

FORMAT

status = **campSrv_varGet***(path, flag)*
status = **f_campSrv_varGet***(f_path, f_flag)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The CAMP path of the request. This might be the path below which Variables are to be returned, or the specific Variable to be returned, depending upon the value of *flag*.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

flag can take on a value which is an ORed combination of the bits shown in Table A–2. These flags are defined in *camp_defs.h*.

Table A–2 Flags used by campSrv_varGet( )

Flag	Description
CAMP_ XDR_ UPDATE	Just return the value of the parameters that may change. Ensure that this is **NEVER** used the first time a Variable(s) is requested. The first request must be for a complete copy of the Variable. From then on, it is recommended that this flag be set to reduce the overhead of the transfer.
CAMP_ XDR_NO_ NEXT	Return only one Variable at the top of the specified path. If not set, the Variables at the level of the specified path, which "come after" the specified Variable, are sent. That is, the Variables in the linked-list that follow, at the same level, the one specified. This flag is useful when only one Variable (or one Variable and everything below it) is desired.

Table A–2 (Cont.) Flags used by campSrv_varGet()

Flag	Description
CAMP_XDR_NO_CHILD	Do not return the Variables below the specified path. This flag is useful when only one Variable is desired.
CAMP_XDR_CHILD_LEVEL	Return only those Variables which are one level below the specified path.

f_flag

type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_varGet()* function is used to get copies of Variable data from the CAMP Server. One Variable, all the Variables at the same level of the path, or all the Variables below a specified path may be requested. Furthermore, once a complete copy of a Variable(s) has been received, further requests for the same Variable may specify that only parameters that change are to be sent, for efficiency.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid path.

Any return values shown in Table A–1.

campSrv_varLog()—set Variable logging parameters

Requests the CAMP Server to set the Variable logging parameters.

FORMAT

status **= campSrv_varLog***(path, flag, action)*
status **= f_campSrv_varLog***(f_path, f_flag, f_action)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**boolean**
access:	**read only**
mechanism:	**by value**

Set *flag* to *FALSE* for off or *TRUE* for on.

f_flag

type:	**boolean**
access:	**read only**
mechanism:	**by reference**

action

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

A valid CAMP log action.

f_action

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_varAlarm()* routine requests the CAMP Server to set the logging parameters of a Variable.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid Variable path. The specified Variable does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_DATA_ATTR	The Variable is not logable (does not have the LOG attribute).

Any return values shown in Table A–1.

campSrv_varPoll()—set Variable polling parameters

Requests the CAMP Server to set the Variable polling parameters.

FORMAT

status **= campSrv_varPoll**(*path, flag, interval*)
status **= f_campSrv_varPoll**(*f_path, f_flag, f_interval*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

flag

type:	**boolean**
access:	**read only**
mechanism:	**by value**

Set *flag* to *FALSE* for off or *TRUE* for on.

f_flag

type:	**boolean**
access:	**read only**
mechanism:	**by reference**

interval

type:	**F floating**
access:	**read only**
mechanism:	**by value**

The polling interval in seconds.

f_interval

type:	**F floating**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_varPoll()* routine requests the CAMP Server to set the polling parameters of a Variable.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid Variable path. The specified Variable does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_DATA_ATTR	The Variable is not pollable (does not have the POLL attribute).

Any return values shown in Table A–1.

campSrv_varRead()—read a Variable

Requests the CAMP Server to read a Variable.

FORMAT

status = **campSrv_varRead**(*path*)
status = **f_campSrv_varRead**(*f_path*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_varRead()* routine requests the CAMP Server to get a new reading for a Variable.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid Variable path. The specified Variable does not exist.
CAMP_INVAL_IF	Instrument interface is undefined.
CAMP_INVAL_IF_TYPE	The Instrument interface type is undefined.
CAMP_NOT_CONN	Instrument interface is not online.

Any return values shown in Table A–1.

Other possibilities will depend upon the interface type and the Instrument type.

campSrv_varSetLnk()—set a Link Variable

Requests the CAMP Server to set a Link Variable.

FORMAT

status = **campSrv_varSetLnk**(*path, val*)
status = **f_campSrv_varSetLnk**(*f_path, f_val*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The target value of the Variable.

f_val

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_varSetLnk()* routine requests the CAMP Server to set the target value (i.e., the equivalent path) of a Link Variable.

RETURN VALUES

See *campSrv_varSetNum( )*.

campSrv_varSetNum()—set a Numeric Variable

Requests the CAMP Server to set a Numeric Variable.

FORMAT

status = **campSrv_varSetNum**(*path, val*)
status = **f_campSrv_varSetNum**(*f_path, f_val*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**D floating**
access:	**read only**
mechanism:	**by value**

The target value of the Variable.

f_val

type:	**D floating**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_varSetNum()* routine requests the CAMP Server to set the target value of a Numeric Variable.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid Variable path. The specified Variable does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.
CAMP_INVAL_IF	Instrument interface is undefined.

CAMP_INVAL_IF_TYPE	The Instrument interface type is undefined.
CAMP_NOT_CONN	Instrument interface is not online.

Any return values shown in Table A–1.

Other possibilities will depend upon the interface type and the Instrument type.

campSrv_varSetNumTol()—set the tolerance of a Numeric Variable

Requests the CAMP Server to set the tolerance of a Numeric Variable.

FORMAT

status = **campSrv_varSetNumTol***(path, tol, min, max)*
status = **f_campSrv_varSetNumTol***(f_path, f_tol, f_min, f_max)*

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

tol

type:	**F floating**
access:	**read only**
mechanism:	**by value**

The percentage alarm tolerance.

f_tol

type:	**F floating**
access:	**read only**
mechanism:	**by reference**

min

type:	**D floating**
access:	**read only**
mechanism:	**by value**

The alarm minimum.

f_min

type:	**D floating**
access:	**read only**
mechanism:	**by reference**

max

type:	**D floating**
access:	**read only**
mechanism:	**by value**

The alarm maximum.

f_max

type:	**D floating**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_varSetNumTol()* routine requests the CAMP Server to set the alarm tolerance values of a Numeric Variable. After setting the new tolerances, the Server will also do the equivalent of a call to *campSrv_varSetNum()* to ensure that the new tolerances are seen immediately by the Instrument driver.

RETURN VALUES

See *campSrv_varSetNum()*.

campSrv_varSetSel()—set a Selection Variable

Requests the CAMP Server to set a Selection Variable.

FORMAT

status = **campSrv_varSetSel**(*path, val*)
status = **f_campSrv_varSetSel**(*f_path, f_val*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**byte (unsigned)**
access:	**read only**
mechanism:	**by value**

The target value of the Variable.

f_val

type:	**byte (unsigned)**
access:	**read only**
mechanism:	**by reference**

DESCRIPTION

The *campSrv_varSetSel()* routine requests the CAMP Server to set the target value of a Selection Variable.

RETURN VALUES

See *campSrv_varSetNum()*.

campSrv_varSetStr()—set a String Variable

Requests the CAMP Server to set a String Variable.

FORMAT

status = **campSrv_varSetStr**(*path, val*)
status = **f_campSrv_varSetStr**(*f_path, f_val*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

val

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The target value of the Variable.

f_val

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_varSetStr()* routine requests the CAMP Server to set the target value of a String Variable.

RETURN VALUES

See *campSrv_varSetNum()*.

campSrv_varZero()—zero Variable statistics

Requests the CAMP Server to zero Variable statistics.

FORMAT

status = **campSrv_varZero**(*path*)
status = **f_campSrv_varZero**(*f_path*)

RETURNS

type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

path

type:	**ASCIZ string**
access:	**read only**
mechanism:	**by reference**

The full CAMP path of the Variable.

f_path

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

DESCRIPTION

The *campSrv_varZero()* routine requests the CAMP Server to zero statistics of all Numeric Variables at or below the specified path.

RETURN VALUES

CAMP_SUCCESS	Normal successful completion.
CAMP_INVAL_DATA	Invalid Variable path. The specified Variable does not exist.
CAMP_INS_LOCKED	The Instrument is locked by another process.

Any return values shown in Table A–1.

B CLI Command Format

This section describes the format of all(?) CAMP Command Line Interface commands. These commands are issued by the user to the CAMP Server using the CAMP Command Line Interface (see Chapter 3).

For all commands except *sysGet* . . . , *insGet* . . . , *lnkGet* . . . and *varGet* . . . a status integer is returned. When issued from the Command Line Interface this integer follows the VMS convention for status integers (odd for success and even for failure). When issued from a Tcl script, the status is either 0 (zero) for success and 1 (one) for failure. For the commands *sysGet* . . . , *insGet* . . . , *lnkGet* . . . and *varGet* . . . , a string is returned. When issued from the Command Line Interface this string is printed to standard output and set to the global symbol "CAMP_RESULT". When issued from a Tcl script, the string is returned as with all Tcl procedures.

Note: All command verbs, parameters, qualifiers and other strings in the commands must be separated by whitespace and are case-sensitive.

insAdd—add an Instrument

Issues a request to the CAMP Server to add an Instrument.

FORMAT

insAdd *<typeIdent> <ident>*

PARAMETERS

typeIdent

The identifier of a valid CAMP Instrument type.

ident

A unique identifier for the Instrument. Can be a maximum of 15 characters.

DESCRIPTION

The *insAdd* command issues a request to the CAMP Server to add a new Instrument. The operation will fail if the Instrument type is invalid, or if the unique identifier already exists.

insDel—delete an Instrument

Issues a request to the CAMP Server to delete an Instrument.

FORMAT **insDel** *<path>*

PARAMETERS

path
The unique CAMP path of the Instrument.

DESCRIPTION

The *insDel* command issues a request to the CAMP Server to delete an Instrument. The operation will fail if the Instrument path does not exist, or if the Instrument is locked by a different process.

insGet . . . —get Instrument information

Requests the CAMP Server for an Instrument parameter.

FORMAT

insGetLockHost *<path>*
insGetLockPid *<path>*
insGetDefFile *<path>*
insGetIniFile *<path>*
insGetDataitemsLen *<path>*
insGetTypeIdent *<path>*
insGetInstance *<path>*
insGetClass *<path>*
insGetIfStatus *<path>*
insGetIfTypeIdent *<path>*
insGetIfDelay *<path>*
insGetIf *<path>*

PARAMETERS

path
The unique CAMP path of the Instrument.

DESCRIPTION

The *insGet . . .* commands request the Server to return information about an Instrument. See also insGetIfCamac . . . , insGetIfGpib . . . and insGetIfRs232 . . . for commands specific to interface types. The *insGetIf* command returns a concatenation of all the interface parametetrs.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

insGetIfCamac . . . —get Camac Instrument interface information

Requests the CAMP Server for Instrument interface information specific to the *camac* interface type.

FORMAT

insGetIfCamacB *<path>*
insGetIfCamacC *<path>*
insGetIfCamacN *<path>*

PARAMETERS

path
The unique CAMP path of the Instrument.

DESCRIPTION

The *insGetIfCamac . . .* commands request the Server to return information (branch, crate, or number) about an Instrument interface of type *Camac*.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

insGetIfGpib . . . —get gpib Instrument interface information

Requests the CAMP Server for Instrument interface information specific to the *gpib* interface type.

FORMAT

insGetIfGpibAddr *<path>*
insGetIfGpibReadTerm *<path>*
insGetIfGpibWriteTerm *<path>*

PARAMETERS

path
The unique CAMP path of the Instrument.

DESCRIPTION

The *insGetIfGpib . . .* commands request the Server to return information about an Instrument interface of type *gpib*.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

insGetIfRs232 . . . —get rs232 Instrument interface information

Requests the CAMP Server for Instrument interface information specific to the *rs232* interface type.

FORMAT

insGetIfRs232Name *<path>*
insGetIfRs232Baud *<path>*
insGetIfRs232Data *<path>*
insGetIfRs232Parity *<path>*
insGetIfRs232Stop *<path>*
insGetIfRs232ReadTerm *<path>*
insGetIfRs232WriteTerm *<path>*
insGetIfRs232Timeout *<path>*

PARAMETERS

path
The unique CAMP path of the Instrument.

DESCRIPTION

The *insGetIfRs232 . . .* commands request the Server to return information about an Instrument interface of type *rs232*.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

insIf . . . —interact directly with an instrument

Requests the CAMP Server read from or write to an instrument or force it on/off line. For details, see the corresponding Tcl driver commands in Appendix C.

FORMAT

insIfOff *<path>*
insIfOn *<path>*
insIfRead *<path> <r_cmd> <buf_len>*
insIfReadVerify *<path> <r_cmd> <buf_len> <var> <fmt> <n_tries>*
insIfWrite *<path> <w_cmd>*
insIfWriteVerify *<path> <w_cmd> <r_cmd> <buf_len> <var> <fmt> <n_tries> <val> <tol>*

PARAMETERS

path
The unique CAMP path of the Instrument.

r_cmd
A string that demands a response from an Instrument.

buf_len
The size of the buffer needed to hold the response.

var
Path of variable which stores the result.

fmt
Tcl format specifier for parsing response.

n_tries
Maximum number of retries.

w_command
String to write to Instrument.

val
Value to compare with response.

tol
Tolerance for accepting response as correct val.

DESCRIPTION

The *insIf* . . . commands request the Server to give direct access to the Instrument. For more detailed information on each of these commands, see Appendix C.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

insLoad—load an Instrument initialization

Issues a request to the CAMP Server to load an Instrument initialization.

FORMAT

insLoad *<path> <filename>*

PARAMETERS

path
The unique CAMP path of the Instrument.

filename
The full path of the file to load.

DESCRIPTION

The *insLoad* command issues a request to the CAMP Server to load an Instrument initialization file.

insSave—save an Instrument initialization

Issues a request to the CAMP Server to save an Instrument initialization.

FORMAT **insSave** *<path> <filename>*

PARAMETERS

path
The unique CAMP path of the Instrument.

filename
The full path of the file to load.

DESCRIPTION The *insSave* command issues a request to the CAMP Server to save an Instrument initialization to a file.

insSet—set an Instrument

Issues a request to the CAMP Server to set Instrument-specific parameters.

FORMAT

insSet *<path>*
[-lock {on | off}]
[-line {on | off}]
[-if <type> <delay> . . .]
[-if_mod <value>]

PARAMETERS

path
The unique CAMP path of the Instrument.

OPTIONS

-lock {on | off}
Requests that the calling process lock the Instrument for exclusive control.

-line {on | off}
Sets the Instrument's interface to online or offline in software.

-if <type> <delay> . . .
Defines the Intrument interface. Types are none, rs232, gpib, camac. For a complete description of this command see insSet -if camac, insSet -if rs232, and insSet -if gpib.

-if_mod <val>
Modifies the Instrument interface definition. For interfaces of type rs232, *val* is the name of the port. For interfaces of type gpib, *val* is the GPIB address of the port.

DESCRIPTION

The *insSet* command issues a request to the CAMP Server to set Instrument-specific parameters. This includes locking, setting the interface on/offline, and setting the interface definition.

insSet -if gpib

Issues a request to the CAMP Server to define an Instrument interface as GPIB.

FORMAT

insSet *<path> -if gpib <delay> <addr> <term>*

PARAMETERS

path
The unique CAMP path of the Instrument.

delay
The access delay time in seconds. The CAMP Server ensures that at least this amount of time has passed between consecutive accesses to the physical device. Floating-point.

addr
The GPIB address. Integer.

term
A character to expect as end of input. Valid strings are: *none*, *LF* and *CR*.

insSet -if rs232

Issues a request to the CAMP Server to define an Instrument interface as RS-232C.

FORMAT **insSet** *<path> -if rs232 <delay> <name> <baud> <data bits> <parity> <stop bits> <read term> <write term> <read timeout> <retries>*

PARAMETERS

path
The unique CAMP path of the Instrument.

delay
The access delay time in seconds. The CAMP Server ensures that at least this amount of time has passed between consecutive accesses to the physical device. Floating-point.

name
The name of the port.

baud
The baud rate. Valid strings are: *300*, *600*, *1200*, *2400*, *4800*, *9600* and *19200*.

data bits
The number of data bits. Integer. Usually 7 or 8.

parity
The interface parity. Valid strings are: *none*, *odd* and *even*.

stop bits
The number of stop bits. Integer. Usually 1 or 2.

read terminator
Characters expected to terminate a read operation. Valid strings are: *none*, *LF*, *CR* and *CRLF*.

write terminator
Characters to append to each write operation. Valid strings are: *none*, *LF*, *CR* and *CRLF*.

read timeout
The timeout value for read operations. Integer in seconds.

retries
The number of times to retry a read operation after a failure. Integer.

insSet -if camac

Issues a request to the CAMP Server to define an Instrument interface as Camac.

FORMAT

insSet *<path> -if camac <delay> <branch> <crate> <number>*

PARAMETERS

path
The unique CAMP path of the Instrument.

delay
The access delay time in seconds. The CAMP Server ensures that at least this amount of time has passed between consecutive accesses to the physical device. Floating-point.

branch
The Camac branch number. Integer, usually 0.

crate
The Camac crate number. Integer, usually 0.

number
The slot number in the Camac crate.

lnkGet . . . —get Link Variable information

Requests the CAMP Server for a Link Variable parameter.

FORMAT

lnkGetVarType *<path>*
lnkGetPath *<path>*

PARAMETERS

path
The unique CAMP path of the Instrument.

DESCRIPTION

The *lnkGet* . . . commands request the Server to return information about a Link Variable.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

lnkSet—set a Link Variable

Requests the CAMP Server to set the value of a Link Variable.

FORMAT **lnkSet** *<path> <val>*

PARAMETERS

path
The unique CAMP path of the Variable.

val
The path setting of the Link Variable. This is a effective path of the Link.

DESCRIPTION

The *lnkSet* command issues a request to set the value of Link Variable. After setting this Variable, all *varSet* and *varRead* commands referring to the Link Variable's path will effectively refer to the Variable pointed to by the Link Variables's value. This mechanism provides the means of accessing the same Variable via multiple paths, much like a symbolic link in the UNIX file system.

sysAdd . . . —add to the system configuration

Requests the CAMP Server to add something to the system definition.

FORMAT

sysAddAlarmAct *<ident> <proc>*
sysAddLogAct *<ident> <proc>*
sysAddIfType *<ident> <conf> <defaultDefn>*
sysAddInsType *<ident>*
sysAddInsAvail *<type> <ident>*

PARAMETERS

ident
identifier of an action, interface, or instrument type

proc
procedure name, used to carry out an action

conf
configuration parameters

defaultDefn
default interface parameters

type
instrument type

DESCRIPTION

The *sysAdd . . .* commands request the Server to add to the system's internal database: An alarm action, log action, interface type, instrument type, or instrument (of an existing type) available.

When issued from the CAMP Command Line Interface, the result is printed to standard output and is set to the global symbol "CAMP_ RESULT" (under VMS). When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

sysDir—perform directory listing

Issues a request to the CAMP Server to perform a directory listing and return the listing.

FORMAT

sysDir *<filespec>*

PARAMETERS

filespec
The file specification of the listing.

DESCRIPTION

The *sysDir* command issues a request to the CAMP Server to perform a directory listing and return the listing. Note that the first item in the list returned is the filespec that was requested. Be careful of the format of the filespec - the command will be interpreted by the Tcl interpreter in the CAMP Server. The character *[* which is used in VMS directory specifications, is special to Tcl. You must precede this character with a backslash (\) in directory specifications.

sysGet . . . —get system information

Requests the CAMP Server for system information.

FORMAT

sysGetAlarmActs
sysGetDir
sysGetIfConf *<ifType>*
sysGetIfTypes
sysGetInsTypes
sysGetInsNames
sysGetLogActs
sysGetLoggedVars
sysGetAlertVars

DESCRIPTION

The *sysGet* . . . commands request the Server to return information about the system. Most commands return lists of data with each item delimited by a space.

When issued from the CAMP Command Line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT" (under VMS). When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

sysLoad—load a configuration

Issues a request to the CAMP Server to load a new configuration.

FORMAT

sysLoad *<filename> [<flag>]*

PARAMETERS

filename
The full pathname of the file to load.

flag
If *flag* is 0 (zero), the configuration before loading a new configuration is retained. This is the default. If *flag* is 1 (one), all current instruments are deleted before loading the file. If any instruments are locked, the latter will fail.

DESCRIPTION

The *sysLoad* command issues a request to the CAMP Server to load a configuration file. As indicated in the parameters section, this may either add to or replace the current configuration.

sysReboot—reboot CAMP Server

Issues a request to the CAMP Server to restart.

FORMAT **sysReboot**

PARAMETERS *None.*

DESCRIPTION The *sysReboot* command issues a request to the CAMP Server to reboot. The Server will first check to see that there are no locked instruments. If there are none, the Server will exit cleanly and then restart. In the current implementation, the VxWorks version of CAMP does not restart.

sysSave—save a configuration

Issues a request to the CAMP Server to save the configuration.

FORMAT

sysSave *<filename>*

PARAMETERS

filename
The full pathname of the file to save.

DESCRIPTION

The *sysSave* command issues a request to the CAMP Server to save the configuration to a file.

sysShutdown—shutdown CAMP Server

Issues a request to the CAMP Server to shutdown.

FORMAT **sysShutdown**

PARAMETERS *None.*

DESCRIPTION The *sysShutdown* command issues a request to the CAMP Server to shutdown. The Server will first check to see that there are no locked instruments. If there are none, the Server will exit cleanly.

sysUpdate—update the CAMP Server

Issues a request to the CAMP Server to update its internal parameters.

FORMAT

sysUpdate

PARAMETERS

None.

DESCRIPTION

The *sysUpdate* command issues a request to the CAMP Server to update its internal parameters which are initialized at startup. These include the available:

- instrument types
- interface types
- alarm actions
- log actions

varGet . . . —get Variable information

Requests the CAMP Server for a Variable parameter.

FORMAT

varGetIdent *<path>*
varGetPath *<path>*
varGetVarType *<path>*
varGetAttributes *<path>*
varGetTitle *<path>*
varGetHelp *<path>*
varGetStatus *<path>*
varGetStatusMsg *<path>*
varGetTimeLastSet *<path>*
varGetPollInterval *<path>*
varGetLogInterval *<path>*
varGetLogAction *<path>*
varGetAlarmAction *<path>*
varGetVal *<path>*

PARAMETERS

path
The unique CAMP path of the Variable.

DESCRIPTION

The *varGet* . . . commands request the Server to return information about a Variable. See also varNumGet . . . for commands specific to Numeric Variables.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

varNumGet . . . —get Numeric Variable information

Requests the CAMP Server for information specific to Numeric Variables.

FORMAT

varNumGetTimeStarted *<path>*
varNumGetNum *<path>*
varNumGetLow *<path>*
varNumGetHi *<path>*
varNumGetSum *<path>*
varNumGetSumSquares *<path>*
varNumGetSumCubes *<path>*
varNumGetSumOffset *<path>*
varNumGetTol *<path>*
varNumGetTolType *<path>*
varNumGetUnits *<path>*

PARAMETERS

path
The unique CAMP path of the Variable.

DESCRIPTION

The *varNumGet . . .* commands request the Server to return information specific to Numeric Variables, particularly their logging. Note that *varNumGetNum* returns the number of times the Variable has been read; use *varGetVal* to access the current value.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

varRead—read a Variable

Requests the CAMP Server to read a Variable.

FORMAT

varRead *<path>*

PARAMETERS

path
The unique CAMP path of the Variable.

DESCRIPTION

The *varRead* command issues a request to get a new reading for a Variable by executing the *readProc* for that Variable.

Note that *varRead* does not return the new value of the variable; use a subsequent *varGetVal* for that.

varSelGetValLabel—get Slection Variable value string

Requests the CAMP Server for the selected label for a Selection Variable.

FORMAT **varSelGetValLabel** *<path>*

PARAMETERS

path
The unique CAMP path of the Variable.

DESCRIPTION

The *varSelGetValLabel* command request the Server to return the value of a Selection Variable as a string. Note that *varGetVal* returns the selection value as an integer index.

When issued from the CAMP Command-line Interface, the result is printed to standard output and is set to the global symbol "CAMP_RESULT". When issued from a Tcl script the result is returned as a string in the same way as all Tcl procedures.

varSet—set Variable parameters

Requests the CAMP Server to set a Variable.

FORMAT

varSet *<path>*
[-a {on|off}]
[-a_act <alarm_act>]
[-l {on|off}]
[-l_act <log_act>]
[-p {on|off}]
[-p_int <poll_int>]
[-tol <tol>]
[-toltype <tolType>]
[-units <units>]
[-z]
[-v <val>]

PARAMETERS

path
The unique CAMP path of the Variable.

OPTIONS

-a {on|off}
Set the Variable alarm status.

-a_act <alarm_act>
Set the Variable alarm action. *alarm_act* is a valid CAMP alarm action.

-l {on|off}
Set the Variable logging status.

-l_act <log_act>
Set the Variable logging action. *log_act* is a valid CAMP log action.

-p {on|off}
Set the Variable polling status.

-p_int <poll_int>
Set the Variable polling interval. *poll_int* is the interval value as a floating-point number in seconds.

-tol <tol>
Numeric Variables only. Set the alarm tolerance of a Variable.

-tolType <tolType>

Numeric Variables only. Set the method of tolerance checking. Possible values for *<tolType>* are *0* for *Plus/minus* and *1* for *Percent*.

-units <units>

Set the units for a Variable. The specified *units* should be a short string.

-v <val>

Set the value of a Variable. The data type of *val* should correspond to the type of the Variable, i.e. integer or floating-point for Numeric type, integer for Selection type, and string for String type.

-z

Zero the statistics of a Variable.

DESCRIPTION

The *varSet* command issues a request to set one or more of the parameters for a CAMP Variable. If the value is to be set (-*v*) then the given value is applied by executing the Instrument driver's *writeProc* for the specified variable. For other parameters, *varSet* is equivalent to *varDoSet*.

varTest . . . —test variable against a value

Tests the current value of a Variable against a given value.

FORMAT

varTestAlert *<path> <set_point>*
varTestTol *<path> <set_point>*

PARAMETERS

path
The unique CAMP path of the Variable.

set_point
The value to test against. This value must be of the same data type as the variable.

DESCRIPTION

These commands issue a request to test the value of a variable to see if it is within its tolerance limits of the specified set point. The return value is 0 for out of tolerance and 1 for within tolerance. This is all *varTestTol* does.

varTestAlert also tests the tolerance but, in addition, sets the alert status of the variable and executes any "alarm action" associated with the variable. CAMP variables with the *-a* (alarm) property should have *varTestAlert* in their *readProc*.

C Driver Commands

All of the CLI Commands (see *CAMP User Manual*) are relevant in the context of a driver. Similarly, some of the driver commands are available to the CLI interface. These driver commands should be listed in Appendix B, but there are probably many omissions. Go ahead and try these commands on the CLI if you need to. The declarative commands like CAMP_INSTRUMENT can only be used in the driver (or in the defunct definition files.)

Note: **All command verbs, parameters, qualifiers and other strings in the commands must be separated by whitespace and are case-sensitive.**

The continuation character \ must be used if commands span more than one line.

CAMP_FLOAT—define a Numeric (Floating) Variable

Defines a Numeric (floating-point) Variable in a CAMP Instrument Driver.

FORMAT

CAMP_FLOAT *<path>*
[-D] [-S] [-R] [-P] [-L] [-A]
[-T <title>]
[-H <help>]
[-readProc <read_proc>]
[-writeProc <write_proc>]
[-tol <tol>]
[-min <min>]
[-max <max>]
[-d {on|off}]
[-s {on|off}]
[-r {on|off}]
[-p {on|off}]
[-p_int <poll_int>]
[-l {on|off}]
[-l_act <log_act>]
[-a {on|off}]
[-a_act <alarm_act>]
[-alert {on|off}]
[-z]
[-m <msg>]
[-units <units>]
[-v <val>]

PARAMETERS

path
The unique CAMP path of the Instrument. Note that in Definition files the first identifier in the path is always "~". The parser will replace "~" with the identifier of the current instance of the Instrument type.

OPTIONS

-D
Set the Variable display

-S
Set the Variable setability attribute on.

-R
Set the Variable readability attribute on.

-P
Set the Variable polling attribute on. (A variable can be polled even if it is not declared as readable.)

-L
Set the Variable logging attribute on.

-A
Set the Variable alarm attribute on.

-T <title>
Set the Variable title to *title*.

-H <help>
Set the Variable help to *help*.

-d {on|off}
Set the Variable display status.

-s {on|off}
Set the Variable setability status.

-r {on|off}
Set the Variable readability status.

-p {on|off}
Set the Variable polling status.

-p_int <poll_int>
Set the Variable polling interval. *poll_int* is the interval value as a floating-point number in seconds.

-l {on|off}
Set the Variable logging status.

-l_act <log_act>
Set the Variable logging action. *log_act* is a valid CAMP log action.

-a {on|off}
Set the Variable alarm status.

-a_act <alarm_act>
Set the Variable alarm action. *alarm_act* is a valid CAMP alarm action.

-alert {on|off}
Set the Variable alert status. The command is used to alert the user that a Variable is out of tolerance.

-z
Zero the statistics of a Variable.

-m <msg>
Set the message associated with a Variable.

-units <units>
Set the units of a Variable.

-v <val>
Set the value of a Variable. The data type of *val* should correspond to the type of the Variable, i.e. integer or floating-point for Numeric type, integer for Selection type, and string for String type.

-readProc <read_proc>
For Tcl script drivers only. Set the read script to *read_proc*. This procedure is executed upon each request to read the Variable. The procedure normally consists of a call to *insIfRead* to read data from the physical instrument followed by a call to *varDoSet* to set the value of the Variable. See Section 4.1 for more information about Tcl.

-writeProc <write_proc>
For Tcl script drivers only. Set the write script to *write_proc*. This procedure is executed upon each request to read the Variable. The procedure normally consists of a call to *insIfWrite* to write data to the physical instrument followed by a call to *varDoSet* to set the value of the Variable. It is often possible to read the value back from the physical instrument, before using *varDoSet*, to more accurately reflect the state of the physical instrument. See Section 4.1 for more information about Tcl.

-tol <tol>
Set the alarm tolerance of a Variable.

-min <min>
Set the alarm minimum of a Variable.

-max <max>
Set the alarm maximum of a Variable.

DESCRIPTION

The *CAMP_FLOAT* command defines a Numeric Variable in a CAMP Instrument Driver that is to be interpreted as a floating-point number.

CAMP_INSTRUMENT—define an Instrument Variable

Defines an Instrument Variable in a CAMP Instrument Driver.

FORMAT **CAMP_INSTRUMENT** *<path>*
[-D] [-S] [-R] [-P] [-L] [-A]
[-T <title>]
[-H <help>]
[-insType <ins_type>]
[-initProc <init_proc>]
[-deleteProc <del_proc>]
[-onlineProc <on_proc>]
[-offlineProc <off_proc>]
[-d {on|off}]
[-s {on|off}]
[-r {on|off}]
[-p {on|off}]
[-p_int <poll_int>]
[-l {on|off}]
[-l_act <log_act>]
[-a {on|off}]
[-a_act <alarm_act>]
[-alert {on|off}]
[-z]
[-m <msg>]
[-v <val>]

PARAMETERS ***path***

The unique CAMP path of the Instrument. For the *CAMP_INSTRUMENT* command this path is always set to "/~". The parser will replace "~" with the identifier of the current instance of the Instrument type (i.e., the same Instrument Definition file is used for multiple instances of the same Instrument type).

See CAMP_FLOAT for most options. The following are specific to the Instrument Variable type.

OPTIONS

-insType <ins_type>

Set the Variable Instrument type to *ins_type*. This is only necessary if the Instrument type name in the driver filename (i.e., the string following *camp_ins_*) is not desired as the Instrument type name.

-initProc <init_proc>

For Tcl script drivers only. Set the initialization script to *init_proc*. This procedure is executed when the Instrument is added. There are no requisite tasks for this procedure. See Section 4.1 for more information about Tcl.

-deleteProc <del_proc>

For Tcl script drivers only. Set the deletion script to *del_proc*. This procedure is executed when the Instrument is deleted. There are no requisite tasks for this procedure. See Section 4.1 for more information about Tcl.

-onlineProc <on_proc>

For Tcl script drivers only. Set the online script to *on_proc*. This procedure is executed when the Instrument is set online. This procedure is required and must make a call to *insIfOn* to set the Instrument interface status to *ONLINE*. Otherwise, access to the physical instrument will not be allowed. See Section 4.1 for more information about Tcl.

-offlineProc <off_proc>

For Tcl script drivers only. Set the offline script to *off_proc*. This procedure is executed when the Instrument is set offline. This procedure is required and must make a call to *insIfOff* to set the Instrument interface status to *OFFLINE*. Otherwise, access to the physical instrument will not be disabled when this is requested. See Section 4.1 for more information about Tcl.

DESCRIPTION

The *CAMP_INSTRUMENT* command defines an Instrument Variable in a CAMP Instrument Driver. This must always be the first command in an Instrument Definition file and can only be present once.

CAMP_LINK—define a Link Variable

Defines a Link Variable in a CAMP Instrument Driver.

FORMAT

CAMP_LINK *<path>*
[-D] [-S] [-R] [-P] [-L] [-A]
[-T <title>]
[-H <help>]
[-readProc <read_proc>]
[-writeProc <write_proc>]
[-d {on | off}]
[-s {on | off}]
[-r {on | off}]
[-p {on | off}]
[-p_int <poll_int>]
[-l {on | off}]
[-l_act <log_act>]
[-a {on | off}]
[-a_act <alarm_act>]
[-alert {on | off}]
[-z]
[-m <msg>]
[-v <val>]
[-varType <var_type>]

PARAMETERS

path
The unique CAMP path of the Instrument. Note that in Definition files the first identifier in the path is always "~". The parser will replace "~" with the identifier of the current instance of the Instrument type.

See CAMP_FLOAT for most options. The following are specific to the Selection Variable type.

OPTIONS

-varType <var_type>
Set the Variable type of the Variable to be linked. *var_type* may be any one of: *CAMP_FLOAT*, *CAMP_INSTRUMENT*, *CAMP_INT*, *CAMP_SELECT*, *CAMP_STRING*, *CAMP_STRUCT*.

DESCRIPTION

The *CAMP_LINK* command defines a Link Variable in a CAMP Instrument Driver.

CAMP_INT—define a Numeric (Integer) Variable

Defines a Numeric (integer) Variable in a CAMP Instrument Driver.

FORMAT

CAMP_INT *<path>*
[-D] [-S] [-R] [-P] [-L] [-A]
[-T <title>]
[-H <help>]
[-readProc <read_proc>]
[-writeProc <write_proc>]
[-tol <tol>]
[-min <min>]
[-max <max>]
[-d {on|off}]
[-s {on|off}]
[-r {on|off}]
[-p {on|off}]
[-p_int <poll_int>]
[-l {on|off}]
[-l_act <log_act>]
[-a {on|off}]
[-a_act <alarm_act>]
[-alert {on|off}]
[-z]
[-m <msg>]
[-units <units>]
[-v <val>]

PARAMETERS

path
The unique CAMP path of the Instrument. Note that in Definition files the first identifier in the path is always "~". The parser will replace "~" with the identifier of the current instance of the Instrument type.

OPTIONS

See CAMP_FLOAT.

DESCRIPTION

The *CAMP_INT* command defines a Numeric Variable in a CAMP Instrument Driver that is to be interpreted as an integer.

CAMP_SELECT—define a Selection Variable

Defines a Selection Variable in a CAMP Instrument Driver.

FORMAT

CAMP_SELECT *<path>*
[-D] [-S] [-R] [-P] [-L] [-A]
[-T <title>]
[-H <help>]
[-readProc <read_proc>]
[-writeProc <write_proc>]
[-selections <sel 0> [<sel 1> . . . <sel n-1>]]
[-d {on|off}]
[-s {on|off}]
[-r {on|off}]
[-p {on|off}]
[-p_int <poll_int>]
[-l {on|off}]
[-l_act <log_act>]
[-a {on|off}]
[-a_act <alarm_act>]
[-alert {on|off}]
[-z]
[-m <msg>]
[-units <units>]
[-v <val>]

PARAMETERS

path
The unique CAMP path of the Instrument. Note that in Definition files the first identifier in the path is always "~". The parser will replace "~" with the identifier of the current instance of the Instrument type.

See CAMP_FLOAT for most options. The following are specific to the Selection Variable type.

OPTIONS

-selections <sel 0> [<sel 1> . . . <sel n-1>]
Set the text labels of the Selection Variable. *sel 0* through *sel n-1* are the text labels. Enclose a text label within quotation marks if it contains a whitespace character.

DESCRIPTION

The *CAMP_SELECT* command defines a Selection Variable in a CAMP Instrument Driver.

CAMP_STRING—define a String Variable

Defines a String Variable in a CAMP Instrument Driver.

FORMAT

CAMP_STRING *<path>*
[-D] [-S] [-R] [-P] [-L] [-A]
[-T <title>]
[-H <help>]
[-readProc <read_proc>]
[-writeProc <write_proc>]
[-d {on | off}]
[-s {on | off}]
[-r {on | off}]
[-p {on | off}]
[-p_int <poll_int>]
[-l {on | off}]
[-l_act <log_act>]
[-a {on | off}]
[-a_act <alarm_act>]
[-alert {on | off}]
[-z]
[-m <msg>]
[-units <units>]
[-v <val>]

PARAMETERS

path
The unique CAMP path of the Instrument. Note that in Definition files the first identifier in the path is always "~". The parser will replace "~" with the identifier of the current instance of the Instrument type.

OPTIONS

See CAMP_FLOAT.

DESCRIPTION

The *CAMP_STRING* command defines a String Variable in a CAMP Instrument Driver.

CAMP_STRUCT—define a Structure Variable

Defines a Structure Variable in a CAMP Instrument Driver.

FORMAT

CAMP_STRUCT *<path>*
[-D]
[-T <title>]
[-H <help>]
[-d {on | off}]
[-z]
[-m <msg>]

PARAMETERS

path
The unique CAMP path of the Instrument. Note that in Definition files the first identifier in the path is always "~". The parser will replace "~" with the identifier of the current instance of the Instrument type.

OPTIONS

See CAMP_FLOAT.

DESCRIPTION

The *CAMP_STRUCT* command defines a Structure Variable in a CAMP Instrument Driver.

insIfOff—set an interface offline

Instructs the CAMP Server to set an Instrument interface offline.

FORMAT **insIfOff** *<path>*

PARAMETERS ***path***
The unique CAMP path of the Instrument.

DESCRIPTION This command instructs the CAMP Server to actually set an Instrument interface offline.

insIfOn—set an interface online

Instructs the CAMP Server to set an Instrument interface online.

FORMAT **insIfOn** *<path>*

PARAMETERS

path
The unique CAMP path of the Instrument.

DESCRIPTION

This command instructs the CAMP Server to actually set an Instrument interface online.

insIfRead—read from the physical instrument

Instructs the CAMP Server to write a command to the physical instrument, and read the data returned by the instrument.

FORMAT

insIfRead *<path> <cmd> <buf_len>*

PARAMETERS

path
The unique CAMP path of the Instrument.

cmd
A string to send to the physical instrument to prompt for the readback. Enclose this string within quotes if it contains any whitespaces.

buf_len
The length of the buffer to be allocated for the data that will be returned by the physical instrument. This value does not have to be the exact length of the returned data, but must be large enough to hold the data. It is recommended that one overestimate this value.

RETURNS

reading
The string returned by the instrument.

DESCRIPTION

This command instructs the CAMP Server to send a command prompt to the physical instrument and then read the response sent from the instrument.

insIfReadVerify—read from the physical instrument and set a camp variable according to the result

Instructs the CAMP Server to write a command to the physical instrument, and read the data returned by the instrument. This data is scanned by the Tcl *scan* command, and that result is used to set a CAMP Variable.

FORMAT

insIfReadVerify *<path> <cmd> <buf_len> <var> <fmt> <tries>*

PARAMETERS

path
The unique CAMP path of the Instrument.

cmd
A string to send to the physical instrument to prompt for the readback. Enclose this string within quotes if it contains any whitespaces.

buf_len
The length of the buffer to be allocated for the data that will be returned by the physical instrument. This value does not have to be the exact length of the returned data, but must be large enough to hold the data. It is recommended that one overestimate this value.

var
The unique CAMP path of the Variable which is to receive the result.

fmt
A string used as the format specification for the Tcl *scan* command to extract the meaningful parameter from the instrument's response string. Enclose this string within quotes if it contains any whitespaces.

tries
If the instrument reading fails, or the scanning of the string fails, then the read-and-scan are repeated as many as <tries> times.

RETURNS

reading
The raw string returned by the instrument.

var
The specified Variable is set to a value extracted from the string.

DESCRIPTION

This command instructs the CAMP Server to send a command prompt to the physical instrument and then read the response sent from the instrument. This string is scanned by the Tcl *scan* command using the format string <fmt>. This format should extract a portion of the response string which is used to set the CAMP variable <var>. If the scan operation fails, the CAMP server retries, up to <tries> times, before quitting.

insIfWrite—write to the physical instrument

Instructs the CAMP Server to write data to the physical instrument.

FORMAT

insIfWrite *<path> <cmd>*

PARAMETERS

path

The unique CAMP path of the Instrument.

cmd

The string to be sent to the physical instrument. Enclose this string within quotes if it contains any whitespaces.

DESCRIPTION

This command instructs the CAMP Server to send a command string to the physical instrument.

insIfWriteVerify—write to the physical instrument and check the readback

Instructs the CAMP Server to write data to the physical instrument.

FORMAT **insIfWriteVerify** *<path> <w_cmd> <r_cmd> <buf_len> <var> <fmt> <tries> <val> [<tol>]*

PARAMETERS

path
The unique CAMP path of the Instrument.

w_cmd
The string written to the physical instrument, performing the 'write' operation.

r_cmd
The string written to the physical instrument to prompt for a readback.

buf_len
The length of the buffer to be allocated for the readback string. It should be at least as large as the longest possible response string.

var
The unique CAMP path of the Variable which is to receive the readback result.

fmt
A string used as the format specification for the Tcl *scan* command to extract the meaningful parameter from the instrument's response string.

tries
If something fails in the write-prompt-read-scan sequence, the whole cycle is repeated up to <tries> times.

val
The expected value for the readback parameter.

tol
The tolerance for comparing the readback with the specified <val>. If omitted, it defaults to zero.

RETURNS

reading
The raw string returned by the instrument.

var
The specified Variable is set to a value extracted from the string, if it matches the target value.

DESCRIPTION

This command instructs the CAMP Server to send a command string to the physical instrument, then send another string to prompt for a readback, and then read the response sent from the instrument. This response string is scanned by the Tcl *scan* command using the format string <fmt>. This format should extract a portion of the response string to produce a result. If this result matches a specified target value, <val> plus or minus <tol>, it is used to set the CAMP variable <var>. If anything fails in this sequence, the CAMP server retries, up to <tries> times, before quitting. If the optional <tol> is omitted, it is treated as zero, and if <var> is a non-numeric Variable, the tolerance is ignored.

sleep—wait some time

Puts this procedure to sleep for a period of time.

FORMAT

sleep *<time>*
[lock]

PARAMETERS

time
The number of seconds to sleep; floating point.

lock
Forces all concurrent threads to sleep.

DESCRIPTION

This command issues a request for the current procedure to be suspended for <time> seconds. By default, the other threads (if multithreaded) are allowed to proceed. Using the *lock* flag, however, turns the global lock on, so that no other threads may execute while the current thread is sleeping.

CAUTION: turning the global lock on for long periods (i.e., several seconds) causes unpredictable problems in both the VAX/VMS and VxWorks camp servers.

varDoSet—set Variable parameters

Instructs the CAMP Server to set a Variable.

FORMAT

varDoSet *<path>*
[-d {on|off}]
[-s {on|off}]
[-r {on|off}]
[-p {on|off}]
[-p_int <poll_int>]
[-l {on|off}]
[-l_act <log_act>]
[-a {on|off}]
[-a_act <alarm_act>]
[-alert {on|off}]
[-z]
[-m <msg>]
[-units <units>]
[-v <val>]

PARAMETERS

path
The unique CAMP path of the Instrument.

OPTIONS

-d {on|off}
Set the Variable display status.

-s {on|off}
Set the Variable setability status.

-r {on|off}
Set the Variable readability status.

-p {on|off}
Set the Variable polling status.

-p_int <poll_int>
Set the Variable polling interval. poll_int is the interval value as a floating-point number in seconds.

-l {on|off}
Set the Variable logging status.

-l_act <log_act>
Set the Variable logging action. log_act is a valid CAMP log action.

-a {on|off}
Set the Variable alarm status.

-a_act <alarm_act>
Set the Variable alarm action. log_act is a valid CAMP alarm action.

-alert {on|off}
Set the Variable alert status. The command is used to alert the user that a Variable is out of tolerance.

-z
Zero the statistics of a Variable.

-m <msg>
Set the message associated with a Variable.

-units <units>
Set the units of a Variable.

-v <val>
Set the value of a Variable. The data type of val should correspond to the type of the Variable, i.e. integer or floating-point for Numeric type, integer for Selection type, and string for String type.

DESCRIPTION

This command issues a request to set one or more of the various CAMP Variable parameters.