r e p o r t

Kavli Institute, MIT NE83-545
Building 300, One Kendall Square
Cambridge MA, 02139–4307
Tel: +1-617-253-7277
Fax: +1-617-253-8084

To: ACIS Science Operations Team

From: Peter Ford, NE83-545 <pgf@space.mit.edu>

Date: June 10th 2019

Subject: The Architecture of acisCtl 2.0

1. Introduction

The uses of the *acisCtl* command are to send commands to the ACIS instrument and to its power supply, to create bias maps and run science observations, to display science results and the status of ACIS hardware and software, and to assist in debugging and patching ACIS flight software. Since many of these functions cannot be performed on the flight instrument, *acisCtl* runs in one of three modes: *flight* mode when it is started with the **–f** flag, when it only displays telemetry, *engineering* mode when started with **–e**, in which it can also send commands to the ACIS DPA, and PSMC mode when started with **–P**, in which it can fully control both the ACIS DPA and its power supply and related mechanisms. Full PSMC mode has been unavailable since pre-launch testing but the hardware and software interfaces no longer exist to support it.

The *acisCtl* script is a small Bourne shell wrapper that initializes some environment variables and launches *acisCtl.tcl*, a script written in the Tcl/Tk language, which loads a set of core *tcl* functions and controls the graphical interface through Tk primitives, but these in turn can start a number of standalone *tcl* and *sh* tasks to control additional display windows. The scripts communicate through a large number of environment variables, many of which are initialized from customizable parameter files which may be updated and saved by the user through the "Parameters..." button in the main menu.

A note on the typographic conventions used in this report. In the text, examples of Tcl/Tk syntax will be written in `Typescript`, environment variables will appear as in a shell script, *e.g.*, `$HOME`, which in Tcl would be accessed as `$env(HOME)`, and external items, *e.g.*, file names and UNIX executables, will be *italicized*. In the code examples, system commands will be colored in `purple`, user-defined commands in `brown`, and numeric constants in `blue`. There is a Glossary of technical terms and abbreviations in Chapter 8.

2. Components

Appendix A lists the Tcl modules that constitute the core task. The *acisCtl* script commands the Tcl/Tk *wish* interpreter to run *acisCtl.tcl*, which embeds the other core modules to add functionality. Tables in Appendix B list the standalone Tcl/Tk scripts that are invoked by the core modules and execute as independent *wish* tasks. The "mode" column indicate which *acisCtl* modes use that module: flight (FL), engineering (EU), PSMC (PS), or all of them. Within *acisCtl* modules, engineering mode is indicated by `$ACISEUFLG=1` and PSMC mode by `$ACISPSMCFLG=1`, both of which are set in the initial *acisCtl* script.

3. Input and Output

All standalone *acisCtl* modules receive ACIS telemetry packets by reading the `channelId` returned from a call to the `openStream` procedure, *e.g.*, when *video11.tcl* wants to read packets, it includes the following code:

```
if {[catch {set fid [openStream \x0c ACISV11TEST} err]} {
    errMsg $err {DEA Housekeeping}
    return
}
```

and reads binary telemetry packets from `fid` in non-blocking mode. The second argument to `openStream` causes it to check whether the value of `$ACISV11TEST` is non-null. If so, `openStream` opens a file of that name and returns its `channelId`. Otherwise, it opens a socket to `$DATAPORT` of `$DATAHOST`. The first argument to `openStream` is a one-byte binary code that is written to the socket, telling *filterServer* which class(es) of packet to send. Refer to Section 5.4 and the *filterServer* manual for details. Since `openStream` performs no error checking, it is executed by the `catch` command to trap any error messages in `$err`.

3.1. Flight Mode

In flight mode, *acisCtl* displays (and optionally records) realtime ACIS telemetry. All components receive data from the flight instrument in the form of 1029-byte minor frames, possibly wrapped in SFDU and/or EHS headers. They can be converted to ACIS packets by a variety of programs: from EHS by *getnrt*, from SFDU by *getPackets*, and from minor frames by *getPackets* or *getp*. To achieve this flexibility, *acisCtl* invokes a shell script to start the input interface. The script is defined by the environment variable `$RCTU_CMD`, and is invoked by the TCL command `[open "| $env(RCTU_CMD)" r]`.

The default value of `$RCTU_CMD` is "*acisTstShim*". Its contents may vary, but after deciding which flags and arguments to use, it should invoke a shell pipe of the following form:

```
(tlmGet -p $COGPORT | $GETPACKETS_CMD | filterServer -p $DATAPORT) 2>&1
```

where *tlmGet* receives binary TCL input in any format acceptable to `$GETPACKETS_CMD`, (*i.e,. getnrt, getp or getPackets*), which converts it to ACIS packets, and *filterServer* parcels them out to the standalone *acisCtl* clients. Once $RCTU_CMD has been started, the core task issues a "*ps clx*" command to determine the process IDs of its sub-tasks, including that of *tlmGet*.

The "Start Raw Input Logging" button sends a "*kill –usr1*" signal to *tlmGet*, which opens and starts to copy the incoming data to the file named in *tlmGet*'s option "**–d** *file*". If this name ends in ".Z" or ".gz", the data will be compressed through "*gzip*". Also, if it contains "%", it will be reformatted by *strftime*() to replace "%" fields with local time and date information, *e.g.*, at τ-time on π-day, "`ACIS-%DT%T-mnf.gz`" would compress the data and write it to "*ACIS-03/14/19T18:28:00-mnf.gz*". For details, consult the manual for *strftime*(3).

Logging ACIS packets is simpler. The "Start Packet Logging" button invokes the `runLogPkts` procedure in *runacis.tcl* which creates and executes the following pipe:

```
filterClient | $1 > ${TLM_LOG_DIR}/`date +"${TLM_LOG_FILE}"`$2
```

where `$1` and `$2` represent the first and second words in `$LOGCOMPRESS`. If the latter is "*gzip .gz*", the output will be compressed, but if `$LOGCOMPRESS` is "*cat*", it won't. `runLogPkts` saves the PID of the *filterClient* task in `$run(pid)` so that it can be killed when the "Stop Packet Logging" button is clicked. The "*+fmt*" of the "*date*" command is used to translate "%" fields into data and time values in a similar way to *strftime*(3).

3.2. Enginerering Mode

The default value of `$RCTU_CMD` is "*acisEUshim*", which starts a pipe:

```
(cserver $CMDPORT | sendCmds | shim lrctu | $GETPACKETS_CMD | filterServer -p $DATAPORT) 2>&1
```

A TCP server, *cserver*, accepts binary command packets as generated by "*bcmd* ", pipes them through "*sendCmds*" to add channel headers, and into "*shim*" which writes them to the engineering unit via the L-RCTU interface. *shim* also writes the output of the engineering unit to *stdout* in the form of ACIS minor frames, which are converted to packets by `$GETPACKETS_CMD`, and sent to *filterServer* to distribute to standalone *acisCtl* tasks via TCP. Earlier versions of *acisCtl* also supported a third I/O function with an alternative version of *shim*, sending commands to the PSMC to control the power supply, open and close doors and valves, etc., and receive PSMC serial digital and 8-bit A/D telemetry. The command function has been removed but the "*psmc.tcl*" module accepts variables `$PSMC_HOST` and `$PSMC_PORT` as alternatives to `$DATAHOST` and `$DATAPORT` when running in flight mode and displays telemetry related to PSMC functions.

4. Variables

4.1. Local

To keep the number of variable names to a minimum, as many as possible have been collected into hashes (associative arrays) whose names reflect that of the module that uses them. In most cases, a hash is restricted to a single module, where it is declared `global` and used to pass values between functions. The following table lists the globals used exclusively within the core task.

Hash	Module	Use
`dea`	*deaif.tcl*	Miscellaneous controls and DEA channel values
`disp`	*textDisp.tcl*	Miscellaneous control values
`hst`	*highspeedtap.tcl*	Miscellaneous control and High-Speed Tap channel values
`image`	*imageLoad.tcl*	Miscellaneous control values
`opt`	*options.tcl*	Miscellaneous control values
`pblk`	*pblocks.tcl*	Miscellaneous control values
`ps`	*acisPrint.tcl*	Miscellaneous control values
`run`	*runacis.tcl*	Miscellaneous control values
`table`	*acisTable.tcl*	Miscellaneous control values

Standalone modules also use hashes in favor of individual variable names to pass values between their internal procedures and, in the case of *debug.tcl*, between it and *debugAux.tcl*.

Hash	Module	Use

`cmd`
`addrBEP addrFEP`
`cmds config`
`fatal cmdres`
`feperr pkts`
`smterm swstat`
	commands.tcl	`cmd` stores miscellaneous control values. `cmds` and `pkts` contain the ASCII names of ACIS commands and telemetry packets. The remainder contain ASCII names of particular packet fields.

`debug`
`addrBEP addrFEP`
`abbrevBEP`
`abbrevFEP`
`fileBEP fileFEP`
`fmtBEP fmtFEP`
`lenBEP lenFep`
`mips[CIJKLNRT]`
`nameBEP yBEP`
`nameFEP yFEP`
	debug.tcl
	debugAux.tcl
		`debug` stores miscellaneous control values. `*BEP` and `*FEP` store global FEP and BEP addresses and names from the load maps, `fileBEP` and `fileFEP`. `mipsC` through `mipsT` contain information used to reverse-assemble MIPS instructions. Note that `addrBep` and `addrFEP` are also global in the *commands* module, but debug runs in a separate *wish* task so they are independent.

`dump`	*dump.tcl*	Miscellaneous control values
`int`	*interface.tcl*	Miscellaneous control values
`man`	*manual.tcl*	Miscellaneous control values
`psmc` `chan`	*psmc.tcl*	`psmc` stores miscellaneous control values; `chan` stores channel IDs

`rctu`
`rctu[CPV]`
	rctu.tcl	`rctu` stores miscellaneous control values; `rctuC` stores channel names; `rctuP` stores flag values; `rctuV` stores channel values.

`show`
`show[CEFMS01]`
	showit.tcl	`show` stores miscellaneous control values; `showC` stores colors; `showE` stores exposure IDs; `showF` stores event record formats; `showM` stores a propeller (see Glossary); `showS` stores CCD names; `show0` stores PHA minima; `show1` stores PHA maxima.

`tlm` `tlmdat`	*showtlm.tcl*	`tlm` stores miscellaneous control values; `tlmdat` stores the structure of each type of telemetry packet.
`v11`	*video11.tcl*	Miscellaneous control and Board 11 channel values
`vtm`	*videotm.tcl*	Miscellaneous control and DEA housekeeping channel values

4.2. Global

Finally, there are a small number of variables and hashes that are genuinely global, mostly used to pass values within modules running in the *acisCtl.tcl* task.

Hash	Module	Use
`acisgeom`	*acisAux.tcl* *acisProcs.tcl*	This is a hash, indexed by top level Tk window name (leading "." included), whose values are the corresponding window geometries. To update an entry, procedures within *acisCtl.tcl* call `putGeometry` in `acisProcs`; standalone procedures call `putGeometry` in `acisAux` with same arguments.
`acispid`	*acisCtl.tcl* *acisProcs.tcl* *acisTable.tcl* *runacis.tcl* *rawpackets.tcl*	This is a hash containing the process IDs of standalone wish procedures started within the *acisCtl.tcl* task, currently `cmd`, `psmc`, and `table`, and used to destroy those tasks when cleaning up within `acisCtl` and `acisProcs`.
`ccd`	*acispower.tcl* *acisProcs.tcl*	A hash containing power off/on bits for each CCD extracted from the most recent configuration table read by *acisCtl*.
`errorInfo`	*commands.tcl* *debug.tcl* *interface.tcl* *manual.tcl* *psmc.tcl* *rctu.tcl* *showit.tcl* *showtlm.tcl* *video11.tcl* *videotm.tcl*	This is not a hash. It is a global variable set by Tcl/Tk when encountering an error. Most error-prone commands within *acisCtl* are executed within a `catch` argument, and errors are handled immediately, but most stand-alone modules are split into two procedures: the first to define the output window(s) when any error will terminate the module, and the second to run under catch, when any error not caught within that procedure will cause its `$errorInfo` text to be written to *stderr* in the terminal window in which *acisCtl* was started.
`fep`	*acispower.tcl* *acisProcs.tcl*	A hash containing power off/on bits for each FEP extracted from the most recent configuration table read by *acisCtl*.
`relay`	*acispower.tcl* *deaif.tcl*	A hash containing status bits for each DEA power relay extracted from the most recent Board 11 housekeeping packet read by *acisCtl*.

4.3. Environment

While most `global` variables are used to restrict their range, the `env` hash, which is initialized by *wish* with the values of the shell's environment variables, and used extensively within *acisCtl* to pass values between modules, *e.g.*, from the *acisCtl.tcl* task to the stand-alone tasks that it starts. Appendix C contains a description of all environment variables used by *acisCtl*.

5. Coding Conventions

5.1. Standalone Modules

The standalone modules are all coded in a similar manner, as in the following example, where "`xxx`" represents some short mnemonic that recalls the name and function of the module:

```
global env xxx errorInfo
InitGlobals.xxx
ShowWindow.xxx
if {[catch ReadPackets.xxx]} {puts stderr "$xxx(title): $errInfo"}
if {$env(ACISxxxTEST) ne {}} {vwait forever}
DestroyWindow.xxx
```

The `global` command gives access to environment variables through `$env()` and system error messages through `$errorInfo`. It also accesses a global hash `$xxx()` for sharing variables such as `$xxx(title)`, the window title between procedures.

`$xxx()` and other variables are initialized by the `InitGlobals.xxx` function and the display window is created by `ShowWindow.xxx`. Any errors up to this point that were not invoked as an argument of a `catch` command will cause the module to crash.

The `ReadPackets.xxx` command does the dynamic work, reading ACIS telemetry and, in engineering mode, sending commands to the instrument. If an uncaught error occurs, `ReadPackets.xxx` will return and print an error message to *stderr*, but the module will keep running.

If environment variable `$ACISxxxTEST` is non-zero, it will be interpreted by `ReadPackets.xxx` as a file supplying telemetry input in place of *filterServer*, so "`vwait forever`" will pause the window display for the user to examine until killing the module with CTRL-C from the terminal. Otherwise, `DestroyWindow.xxx` will clear the window and terminate the module.

5.2. Module Initialization

The first step in the initialization of *acisCtl.tcl* and of each standalone module – *i.e.*, within the `InitGlobals.xxx` procedure in the above example – is to initialize control variables, usually within the $`xxx`( ) hash, and supply default values to the environment variables used in the module. The latter is done in the following consistent manner:

```
global env
foreach ii [list \
    {ACISTOOLSDIR   {~acis/tools}} \
    {ACISxxxTEST {}} \
    {name           {value}} \
    {LibPath        {$env(ACISTOOLSDIR)/lib/acisctl}} \
    {imgdir         {$env(LibPath)/images}} \
] {
    lassign $ii name val
    if {! [info exists env($name)]} {
        eval "set env($name) \"$val\""
    }
}
```

The environment variable names are paired with their default values. If the variable doesn't yet exist, its default is evaluated and assigned, so the defaults can themselves be defined in terms of other environment variables, or even themselves if the programmer is willing to accept the consequences.

5.3. Window Initialization

Most of the standalone modules display a single graphical window. These are also created in a consistent way across all modules, *e.g.*, within `ShowWindow.xxx` in our example, as follows:

```
proc ShowWindow.xxx { }{
    global env xxx
    frame .xxx -background {color}
    set revision "0"
    regexp {: ([^ ]+)} {$Revision$ } xxx revision
    set xxx(name) "ACIS Window Title $revision"
    putGeometry .xxx $xxx(name) +col+row
    ...
}
```

The top-level name of the new window will be known to Tk as "`.xxx`", so its widgets – its buttons, labels, text fields and graphics – will have names beginning "`.xxx.`". Once committed into the CVS document control system, the module will be assigned a numeric revision code and the `regexp` command copies this code into `$revision`, and into the full window name, `$xxx(name)`. Finally, the `putGeometry` procedure inserts the full name into the window title, sources the "*~/.acisctlgeom*" file, and if `acisgeom(.xxx)` doesn't exist, sets it to "`+col+row`", updates "*~/.acisctlgeom",* and positions the top left of the new window at `acisgeom(.xxx)`.

5.4. Reading ACIS Packets

All standalone modules read packets directly from *filterServer*, in processes named `ReadPackets.xxx`, where `xxx` is our fictitious module name. They establish a socket via a call to `openStream`, and described in Section 3, above, and then enter a loop that goes something like this:

```
while {! [catch {set rec [read $xxx(fid) 8]}] && ! [eof $xxx(fid)]} {
    if {$rec eq {} || [fblocked $xxx(fid)]} {
        after 1000
        catch {update}
    } elseif ([binary scan $rec {ii} sync hdr] == 2} {
        set tag [expr ($hdr >> 10) & 63]
        set len [expr 4*($hdr & 1023)-8]
        set rec [read $xxx(fid) $len]
        if {$tag == 62) {set xxx(time) [irigTime $rec]}
        ...
    }
}
```

Since the socket `$xxx(fid)` is non-blocking, we check whether it still exists (or we are reading from a test file named in `$ACISxxxTEST`) by invoking the `eof` command. Otherwise, if we read a null record or if the socket is blocked, we wait for 1000 milliseconds and tell Tcl/Tk to update anything pending. Once we have a 2-word (8 byte) header in `$rec`, we convert it to integers `$sync` and `$hdr`, from which we extract the packet type `$tag` and length `$len`, and read the remainder of the record. Most modules want to display the date and time contained in IRIG-B format in the *pseudoScience* packet with `tag=62`, and in this example we use the `irigTime` procedure from *acisAux.tcl* to convert it to ASCII and save it in `$xxx(time)`. Creating a text field within "`.xxx.`" with the attribute "`-textvariable xxx(time)`" ensures that whenever "`xxx(time)`" is changed updated, the window field will also update.

5.5. Terminating a Module

Most standalone windows terminate because their input sockets are closed when *filterServer* quits and they fall out of the while loop in their `xxxRun` procedure and invoke `DestroyWindow.xxx`:

```
proc DestroyWindow.xxx {} {
    global xxx
    putGeometry .xxx {} {}
    catch {close $xxx(fid)}
    catch {destroy .}
    exit 0
}
```

The call to `putGeometry` with null in the second and third arguments updates the global `acisgeom(.xxx)` with the window coordinates and saves all `acisgeom` values in "*~/.acisctlgeom*", ensuring that the window will appear in that position next time it is created. Since certain error conditions will cause the window to be destroyed when the input socket is closed, this is done *after* saving the window geometry. Finally, any root window "." created by the module is also destroyed.

5.6. Terminating the Telemetry Server

The "Start ACIS Interface" button in the "I/O Server" menu starts the standalone *interface.tcl* module which executes the `$RCTU_CMD` shell command in a new window. This may also start several additional processes, so both *runacis.tcl* in the core task and the *interface.tcl* task itself invoke "*/bin/ps clx*" and store the IDs of their tasks' sub-processes in `run(pids)` and `int(pids)`, respectively. The "Stop" button of *interface.tcl* kills these tasks before destroying its own window and the core task does the same when the "Stop ACIS Interface" or "Quit" buttons are clicked.

While parsing the output of "*/bin/ps clx*", *runacis.tcl* also saves the process ID of any of its subtasks named *tlmGet* in `$run(tlmGet)` so that the *runacis.tcl* buttons can subsequently start or stop logging the input telemetry by sending *tlmGet* a `SIGUSR1` (start) or `SIGUSR2` (stop) signal.

6. External Executables

The following executable programs have been developed for use with *acisCtl*.

Name	Environment	Use
acisEUshim	`$RCTU_CMD`	Shell script to communicate between *acisCtl* and ACIS engineering unit (EU), as described in Section 3. The *cserver* program listens on TCP port `$CMDPORT` for *bcmd* output, interfaces with the EU via "*shim lrctu*", converts the output into ACIS packets and distributes then via TCP with *filterServer*.
acisEUshim500	`$RCTU_CMD`	Identical to *acisEUshim* except that it uses "*shim500 lrctu*" to interface to the EU at 500 baud to emulate flight format 1, *i.e.*, HRC in the focal plane.
acisPmon	`$PMON_CMD`	Shell script to pipe ACIS packets from *filterClient* into the *pmon* program to display science and housekeeping data in a terminal window that is created in *acisCtl* by executing "`$DISP_TERM -e $PMON_CMD`".
acisTables		Perl script to read ACIS configuration and table files from `$ACIS_CFGS` and `$ACIS_PBLKS`, respectively, and write to *stdout*. The syntax is: *acisTables mode item* write binary value of *item* in *mode* table *acisTables –l mode* write ASCII list of packet names in *mode* table *acisTables -l mode item* write ASCII description of *item* in *mode* table where *mode* is "cfg" or"cfgi", *item* is an SIMODE in the configuration file; when it is "data", *item* is the name of a BEP command in the command file. When "cfg" is specified, *acisTables* pauses for the number of seconds requested in the configuration file after writing each command. Otherwise, "cfg" and "cfgi" are identical.
acisTstShim	`$RCTU_CMD`	Shell script to start a TCP server (usually *getTlm*) to wait for a connection from the COG interface, extract ACIS packets (usually via *getPackets* or *getp*), and pass them to the TCP server *filterServer* for distribution to standalone *acisCtl* modules. *acisTstShim* clears existing system semaphores before starting *getPackets*.
dapkts	`$LOAD_RAW_CMD`	An executable program to copy ACIS command packets from *stdin* to *stdout*, pausing after each multi-word packet to keep the overall data rate below 4 kilobaud, *i.e.*, 500 bytes/second, so as not to overload the interfaces to the ACIS engineering unit.

7. Applicable Documents

Chandra Proposers' Observatory Guide, Section 6.22, Revision 21.0, December 2018.

ACIS EGSE Software Manuals, online at "*ftp://acis.mit.edu/pub/acistools.pdf*".

Long-Term ACIS Maintenance and Verification, online at "*ftp://acis.mit.edu/pub/LongTermMaintenance.pdf*".

ACIS EGSE User Commands, MIT, revised January 31, 2019.

ACIS Science Instrument Software User's Guide, MIT 36–54003, NAS8–37716, Revision A (1999).

ACIS IP&CL Structures, MIT, revised July 28 2014.

ACIS IP&CL Structure Definition Notes, MIT 36–53204.0204, Revision N (2003).

8. Glossary

COG	TCP client supplying Chandra realtime telemetry
Configuration	Set of ACIS BEP serial digital commands to execute a science observation
Core Module	A Tcl/Tk file executing in the task initiated by the *acisCtl* script
CVS	Concurrent Versions System – revision control system used to manage *acisCtl*
DEA	ACIS Detector Electronics Assembly – CCD controllers, amplifiers, digitizers
DPA	ACIS Digital Processor Assembly – DEA controller, pixel filter, telemetry source
EGSE	Electronic ground support equipment, including *acisCtl* and auxiliary programs
EHS	Chandra telemetry format – up to 4 SFDU structures per block
EU	The ACIS Engineering Unit – payload simulator using flight spare components
Hash	Tcl associative array variable indexed by string
High-Speed Tap	External interface to an ACIS DEA video board to receive synchronous pixel stream
Housekeeping	Information about ACIS analog and digital systems included in output telemetry
IRIG-B	48-bit time format used in ACIS pre-launch tests and saved in pseudoScience packets
L-RCTU	High-speed serial interface between ACIS DPA unit and UNIX workstation
Image Loader	DMA interface between ACIS pixel switch and UNIX workstation
MIPS	Common architecture of all ACIS processors
Major Frame	Group of 128 consecutive minor frames – 32.8 seconds of Chandra telemetry
Minor Frame	Basic unit of Chandra telemetry – 4 byte sync code + 1025 byte data
Module	In Tcl/Tk, a file to run under the *wish* interpreter, or sourced by another module
PHA	The pulse height amplitude of an ACIS x-ray event
PID	The process ID of a UNIX task
PSMC	The ACIS power and systems management controller – the power supply
Pseudoscience	ACIS packet created by ground s/w to contain timing information
Packet	ACIS multi-word uplink command or downlink telemetry from the ACIS DPA
Propeller	Serial "|/-\" characters, an ASCII indication of the passage of time
RCTU	Remote Command and Telemetry Unit relaying data to Chandra downlink telemetry
SFDU	Standard Formatted Data Unit – Chandra minor frame + ground station header
Shim	Software to interface between UNIX processes and the EU hardware interface
SIMODE	Name of an ACIS configuration – commands to execute a science observation
Standalone Module	A Tcl/Tk file and its secondary imbeds that executes as a separate task
Video	Within ACIS, the name given to the analog processors within the DEA

Appendices

A. Core Modules

The *acisCtl* script passes *acisCtl.tcl* to the *wish* interpreter. This is the 'core' task and it loads the remaining code modules and provides default values for uninitialized environment variables before passing control to *runacis.tcl* to display the startup menu that does the work.

Core Modules			
Name	**Mode**	**Description**	
acisCtl.tcl	all	Loads the core modules listed in column 2, verifies that `$ACISTOOLSDIR`, `$DATAHOST` and `$PRINTER` possess 'reasonable' values, provides default values for the remaining environment variables, and starts *runAcis.tcl*.	
acisPrint.tcl	all	Provide procedures: (a) `printText` to convert ASCII text to PostScript and write it to `$PRINT_CMD`, or (b) `saveText` to write ASCII text to an external file.	
acisProcs.tcl		Define commands to perform a series of common functions:	
	all	`runTcl`	invoke wish to run a stand-alone TCL/TK script
	EU	`doCmd`	send a command to the ACIS instrument
	EU	`doCmdKill`	kill any previous `doCmdLoad` process
	EU	`doCmdLoad`	send raw command/patch load to ACIS
	EU	`doTable`	run `acisTables` to display or execute offline table item
	all	`errMsg`	display error dialog
	all	`infoBlock`	invoke `$EDITOR` to edit a file
	all	`killInterface`	locate and kill all active *acisCtl* telemetry servers
	PS	`killPsmcRctu`	kill any previous `launchPsmcRctu` process
	EU	`killTable`	kill previous `acisTables` process
	all	`launchPmon`	invoke `$DISP_TERM` to display *pmon* with `$PMON_CMD`
	PS	`launchPsmcRctu`	invoke the PSMC telemetry display
	PS	`launchPsmcTlm`	display data channels from `$PSMC_SERVER`
	EU	`loadImage`	run `$LOAD_IMAGE_CMD` to send pixels to image loader
	EU	`loadPblock`	load parameter block to start/stop a science/DEA housekeeping run
	all	`putGeometry`	update and save window geometries
	EU	`selectAB`	toggle the pixel switch between DEA and image loader
	EU	`sendCcdCmd`	set CCD power on or off for individual boards
	PS	`sendDeaPower`	turn DEA power on/off
	EU	`sendDpaBoot`	warm- or cold-boot the active BEP
	PS	`sendDpaPower`	turn DPA power on/off
	EU	`sendFepCmd`	set FEP power on or off for individual boards
	PS	`sendHstCmd`	set high-speed taps on or off
	all	`show_down`	locate and kill all active processes started by *acisCtl*

Core Modules		
Name	**Mode**	**Description**
acispower.tcl	EU	Respond to the "Control FEP/CCD" button on the main menu by creating and displaying the "FEP/CCD Power" dialog which contains separate toggle buttons for each FEP and CCD, with "Send" buttons to command FEP or CCD to power up/down according to the user's selections. The "Refresh" button executes the *bcmd* command "`dump 0 systemconfig`", waits for the instrument response, and sets the toggle buttons accordingly.
acisTable.tcl	all	Display a window containing scrolling text: either the SIMODEs from an ACIS configurations file "*.cfg*" or from an ACIS command packet file "*.dat*". Functions `tableFilter` and `tableSelect` reduce the range of items displayed. Buttons permit the user to display the content of selected configurations or packet(s) converted to ASCII by the *acisTables* script, or (in EU mode only), send the packet(s) to the instrument via `doTable`.
beppower.tcl	EU	Display a dialog of buttons that send *bcmd* commands to the BEP's hardware serial port via `doCmd`, or send FEP or DEA power commands to the BEP via `sendDpaPower` or `sendDeaPower` calls (see *acisProcs.tcl*, above)
deaif.tcl	EU	Display a dialog of buttons and text entry windows to control the BEP-DEA interface. When *acisCtl* starts, all signals are assumed "off" and the coarse focal plane temperature is –120°C. The user is expected to set the buttons and the coarse and fine temperature settings and then click the "Send" button, when a single "`change systemConfig`" command is sent to ACIS through `doCmd`.
highspeedtap.tcl	PS	Display a dialog through which the user can select a video board to transmit pixel output through its high-speed tap interface. Buttons select whether to do this via `sendHstCmd` (see *acisProc.tcl*) or by loading the "hstfly" patch and executing its *switchHst*() method directly at BEP address `0x800e1c20`.
imageLoad.tcl	EU	Open an "Image Loader Control" window to display the names of the image definition files in `$IMAGE_LIB`. These are ASCII files in the format accepted by the *genObjectImage*(1) program. Once a file has been selected, buttons let the user list it in a window (via `textDisp` in *textDisp.tcl*) or copy it to the Image Loader (via `loadImage` in *acisProcs.tcl*).
options.tcl	all	Display a table of environment variable names, their current values, and a short description of each. *acisCtl* maintains a table of all `$env`() names that it uses, but only displays those that are useful in the current mode. All are loaded from *~/.acisctlrc* when *acisCtl* starts up, and all are written back there when the "Save" button is clicked, but within *acisCtl* itself, a value is changed as soon as the user updates the value field. When *acisCtl* is started with the **–D** flag, additional fields will appear below the Options table. The "Exec" button evaluates the TCL command entered in the "Command" field, "Clear" clears it, and "Reload" loads the embedded scripts for *acisCtl.tcl*, which may be necessary if the command loads text that calls a procedure in one of those scripts.

Core Modules		
Name	**Mode**	**Description**
pblocks.tcl	EU	Display a list of *bcmd* files in `$PARAM_BLOCK_LIB` with particular extensions: `cc` continuous clocking parameter blocks `dea` DEA housekeeping parameter blocks `te` timed-exposure parameter blocks `1d` one-dimensional window blocks `2d` two-dimensional window blocks Buttons at the bottom of the window perform the following functions: `Slot` define the BEP slot to hold that parameter block `List` display the contends of the parameter block `Edit` invoke $EDITOR to edit the parameter block `Load` send the parameter block to the active ACIS BEP `Start` start the science run or DEA housekeeping `Stop` stop the science run or DEA housekeeping `Close` close this window In addition, the CC and TE windows have a "`Start Bias`" button to start a bias-only science run with the selected parameter block. All window types use a text entry field to permit the user to filter file names via wild card characters.
rawpackets.tcl	EU	Display a list of *bcmd* and raw packet files (extensions *.bcmd* and *.pkts*) in `$RAW_CMD_LIB` and use buttons to `List` (in a text window), `Edit` (*.bcmd* only), `Execute` (send to ACIS and wait until a *commandEcho* is received in response, `Stop` (the execution), `Cancel` (the selection), or `Close` (the dialog).
runacis.tcl	all	Creates and displays the main *acisCtl* menu using the "`runB` *flag command background title*" procedure and then enters an indefinite wait. The menu buttons either call `runTcl` to execute a stand-alone module or `showWin` to show a window within the current task. In *flight* mode, there are buttons to start and stop the incoming telemetry interface, and buttons to control Raw Input and Packet logging. In *engineering* mode, additional buttons select FEP input between the DEA and the Image Loader. In "psmc" mode, more buttons start/stop additional command and telemetry servers to the PSMC. The ACIS interface is started by starting *interface.tcl* with *wish* as a separate task, which is responsible for raw input logging (see the section below on *tlmGet*). The necessary parameters are passed in the environment. Packet logging is performed by starting *gzip* and saving its PID in `$run(pid)` so that it can be killed later.
textDisp.tcl	all	Execute "`textDisp title ext text`" to display the ASCII string `text` in a window named `title`. with buttons `Save` (to save text into a file named "`$title.$ext`"), `Print` (to `$PRINTER`), or `Close` (the window). The "`#`" character is assumed to begin a comment, which will be colored blue. In lines beginning "`xxx = {`", the "`xxx`" will be bolded.

B. Standalone Modules

These modules are started from core modules as arguments of the Tcl/Tk *wish* interpreter. They all `source` *acisAux.tcl* to supply common procedures and some also `source` *acisPrint.tcl* to provide a `Print` function.

Standalone Modules			
Module	**Embeds**	**Mode**	**Description**
commands.tcl	*acisPrint.tcl*	EU	Read telemetry from `$DATAHOST:$DATAPORT` (or from `$ACISCMDTEST` if defined) and convert and list packet contents in a scrolling text window. Summarize the content of each non-event, non-exposure packet (or group of packets of the same type), coloring to assist interpretation, e.g., red for anomalies, green for commands, blue for other packet names, etc. The bottom of the window contains an entry field for the maximum number of rows to retain in the scrolling window, `All` to list all *addPatch* commands, buttons `Save` to write the retained text to a disk file, `Print` to convert the text to PostScript and send it to `$PRINTER`, `Mark` to add data and time to the text, `Pause` to stop reading the telemetry until the button is checked again, and `Close` to drop the socket and close the window.
	acisAux.tcl	all	`doPrint` Execute `$PRINT_CMD` to print file with title.
		all	`errMsg` *Display* error dialog with title and message. If an error occurs, write the message to *stderr*.
		all	`irigTime` Return IRIG field and `$DATAYEAR` as date/time.
		all	`openStream` Open `$DATAHOST:$DATAPORT` (or test filename if defined in environment), write mode byte to select data packet type (see *filterServer*(1)), set for non-blocking I/O, and return stream descriptor.
		all	`putGeometry` Execute ~/*.acisctlgeom* to initialize `$acisgeom`. If new geometry for this window, save its value. Then update ~/*.acisctlgeom*. with the geometry of all active windows.
debug.tcl	*acisAux.tcl* *acisPrint.tcl*	EU	The "I/O Interface" dialog contains a `Debug` button that displays a scrolling list of `$BEP_MAP` contents. When the user selects an item, a *readBep* or *readFep* command is sent to ACIS to dump all addresses from that item up to, but not including, the address of the next higher item in the load map. The result is converted to ASCII and displayed in the lower scrolling text area. Several conversions are available: hex, data, and two types of disassembly: "`asm1`" which doesn't attempt to recognize global locations, and "`asm2`" which does. There is no "write" function: to update the EU: users must issue "*echo writeBep ...* | *bcmd* | *cclient ...*" commands from the terminal.
	debugAux.tcl	EU	Contains routines to disassemble MIPS code and data segments for *debug.tcl*. Globals are interpreted as positive offsets from the globals in `$BEP_MAP` and `$FEP_MAP`.

Standalone Modules			
Module	**Embeds**	**Mode**	**Description**

dump.tcl	*acisAux.tcl*
	textDisp.tcl
	acisPrint.tcl
		EU	Display a menu of functions to dump or reset data structures within the BEP or one of the FEPs. Each dump action is performed by the `dumpCmd` procedure:
			`dumpCmd tag type lim fmt namelen name`
			`tag` the button's TK sub-tag in `.dump.frame1.`
			`type` *bcmd dump* sub-command
			`lim` maximum number of reply packets expected
			`fmt` expected *formatTag* of reply packets
			`namelen` byte length of ASCII title
			`name` block name for title
			and each reset action is performed by `dumpAsk`:
			`dumpAsk cmd msg`
			`cmd` complete *bcmd* command
			`msg` description of action (for `errMsg`)
			The dump task reads and decodes the headers of all packets received while it is running but only searches for replies while `$dump(state)==1`. Having found the desired packet(s), it reformats them into ASCII and invokes `textDisp` to display the result in a sub-window of the dump task.

interface.tcl	*acisAux.tcl* *acisPrint.tcl*	all	Start the `$RCTU_CMD` script and display its *stdout* in a scrolling window. Lines containing ugly words such as "connection dropped" or "cannot bind" are displayed in red. Comments are in blue. The `DestroyWindow` function is given the job of finding the children of `$RCTU_CMD` and killing them.
manual.tcl	*acisAux.tcl*	all	Create a window in which to display the individual pages of the *acisCtl* manual, which is stored in `$ACISTOOLSDIR`/*lib*/*acisctl*/*images* as a set of GIF files named `$ACISCTLMAN`_*nn.gif*. Buttons at the foot of the window step through the pages and send the current page to `$PRINTER` as a graphic.
psmc.tcl	*acisAux.tcl*	all	Respond to the "Show PSMC Monitor" button in the main menu by displaying a window showing the status of the PSMC in a set of TK graphics. Each PSMC function is shown as enabled, on, off, and/or disabled in state "A" ot "B". In addition, the various under- and over-current status bits are displayed. Active values are shown in color, with red reserved for unexpected or downright dangerous states. The display isn't valuable in engineering mode: only the "Side A/Side B" display will be set from the ACIS bilevels, unless *acisCtl* is invoked with the **–P** option and the PSMC is being accessed via `$PSMC_SERVER` and `$PSMC_CMD`.
rctu.tcl	*acisAux.tcl*	all	Respond to the "Show RCTU Telemetry" button in the main menu by displaying a window showing the values of ACIS-related engineering channels. In *engineering* mode, only the bilevel values are reported unless *acisCtl* is invoked with the **–P** option to show that the PSMC output is being received and formatted by a special interface. In *flight* mode, the engineering channels are displayed in one of three formats, selected at the bottom of the window: engineering units (volts, amps, °C, etc), or in "raw" units (hexadecimal or decimal).

Standalone Modules			
Module	**Embeds**	**Mode**	**Description**
showit.tcl	*acisAux.tcl*	all	Respond to the "Show CCD Events" button in the main menu by displaying a window showing the 10-CCD ACIS focal plane. Read science telemetry packets and show the location of each event as a dot, colored according to its PHA. To the left of the I-array, a table shows the number of events from each CCD, and the minimum and maximum of each PHA. Fields at the bottom of the window let the user select the minimum and maximum PHA to assign to the available colors and the resulting color scale is displayed to the right of the I-array. Buttons select between `Small`, `Large` and `Huge` dots, `Save` the window as a graphic, `Print` it on `$PRINTER`, `Clear` the dots and table, and `Close` the window. The size of the window is governed by `$CCD_SCALE`, which is the number of screen pixels to be used for each row and column of each CCD.
showtlm.tcl	*acisAux.tcl* *acisPrint.tcl*	all	Respond to the "Show Packet Monitor" button in the main menu by displaying a window showing a scrolling text detailing each ACIS telemetry packet received from the I/O Server. The packet names are in boldface, followed in brackets by the number of items they contain; then the packet sequence number followed by the single-valued fields in name=value format with values in blue. Packets and fields whose presence or value implies bad news are colored red. A "Rows:" field beneath the window specifies the number of lines to be retained in the scrolling window. In enginering mode, ticking the "`Eng:`" box displays pseudo-packets in magenta; otherwise they are hidden. Buttons permit the user to `Save` the text as a simple ASCII file, `Print` the text to $PRINTER as colored PostScript, Mark the text with the current date and time, Pause the display while buffering the input, and Close the window.
video11.tcl	*acisAux.tcl*	all	Respond to the "Show Board 11 Telemetry" button in the main menu by displaying a window showing the values of engineering channels reported in DEA housekeeping. In engineering mode, the BEP must run with the *deaeng* patch to access these channels. Also, the positions of the power relays are only displayed when the DEA is powered from the A-side of the PSMC. The analog values from the individual boards are unreliable unless *all* CCDs are powered simultaneously. Analog channel values are displayed in one of three formats, selected by the leftmost the bottom of the window: `Eng` (engineering units, i.e., volts, amps, °C, etc), or `Hex` (hexadecimal) or `Dec` (decimal). Other buttons `Save` the window as a graphic file, `Print` the colored graphic to `$PRINTER`, `Clear` the values, or `Close` the window.
videotm.tcl	*acisAux.tcl*	all	Respond to the "Show All DEA Telemetry" button in the main menu by displaying a window showing the values of engineering channels reported by the BEP-DEA interface board. In engineering mode, the BEP must run with the *deaeng* patch to access these channels. Also, the positions of the power relays are only displayed when the DEA is powered from the A-side of the PSMC. The engineering channels are displayed in one of three formats, selected by the leftmost the bottom of the window: `Eng` (engineering units, i.e., volts, amps, °C, etc), or `Hex` (hexadecimal) or `Dec` (decimal). Other buttons `Save` the window as a graphic file, `Print` the colored graphic to `$PRINTER`, `Clear` the values, or `Close` the window.

C. Appendix – Environment Variables

In the table of environment variables below, the key letters in the "Type" column have the following meaning:

Key	Description
D	used only for debugging and testing of standalone modules
E	useful in engineering mode; can be edited/saved in the "Parameters..." window
F	useful in flight mode; can be edited edited/saved in the "Parameters..." window
H	inherited from the environment
I	cannot be changed by the user; for internal use only
P	useful in (obsolete) PSMC mode; can be edited/saved in the "Parameters..." window
S	used only by interface programs started by *acisCtl*

Many of the variables are initialized by sourcing "`$ACISTOOLSDIR/lib/$ARCH/acisegse.parms`" or, if the "**–p** *file*" option is supplied, by sourcing *file*. The intention is to supply "reasonable" values for sufficient environment variables to get the novice user running.

The personal start-up file "*~/.acisctlrc*" will contain definitions of those variables with "E", "F" or "P" in the "Type" column. Selecting "Parameters . . ." in the root menu displays those parameters that are editable, *i.e.*, useful, in the current mode. Any changes will take effect immediately in the core task, but will not affect standalone modules until these are stopped and restarted. Saving the Parameter table will update all entries in "*~/.acisctlrc*", whether or not they are "editable" in the current mode.

Variable Name	Type	Module	Default	Description
`ACIS_CFGS`	EFP	*acisTables.tcl* *options.tcl*		ACIS configuration table, passed to *acisTables* executable
`ACIS_PBLKS`	EFP	*acisTables.tcl* *options.tcl*		ACIS command table, passed to *acisTables* executable
`ACISCCDTEST`	D	*showit.tcl*	`()`	Read packets from this input file instead of *filterServer*
`ACISCMDTEST`	D	*commands.tcl*	`()`	Read packets from this input file instead of *filterServer*
`ACISCTLMAN`	D	*manual.tcl*	`()`	Read manual pages from `$ACISCTLMAN_%02d.gif`
`ACISCTLTEST`	D	*acisCtl* *options.tcl*	`0`	Set if *acisCtl* started with –D; adds debugging fields, buttons
`ACISCTLWISH`	I	*acisCtl* *acisCtl.tcl* *acisProcs.tcl* *debug.tcl*	`wish`	Overrides "*wish*" as Tcl/Tk interpreter to run *acisCtl* modules
`ACISEUFLG`	I	*acisCtl* *acisCtl.tcl* *commands.tcl* *runacis.tcl*	`0`	Set if *acisCtl* started with **–e** indicating engineering mode; selects type of telemetry interface to run. Value is passed to shell scripts *acisPmon* and *acisTable*, and to standalone modules *psmc* and *rctu*
`ACIS_IN_HRC`	S			Parameters to pass to *getPackets* to determine the location of ACIS data in format 1 telemetry
`ACISPSMCFLG`	P	*acisCtl* *runacis.tcl*	`0`	Set if *acisCtl* started with –P indicating that the PSMC is to be accessed by `$PSMC_SERVER` and `$PSMC_PORT`. It is passed to standalone module *psmc*
`ACISPSMCTEST`	D	*psmc.tcl*	`()`	Read packets from this input file instead of *filterServer*
`ACISRCTUTEST`	D	*rctu.tcl*	`()`	Read packets from this input file instead of *filterServer*

Variable Name	Type	Module	Default	Description
`ACISTLMTEST`	D	*showtlm.tcl*	`()`	Read packets from this input file instead of *filterServer*
`ACISTOOLSDIR`	H	*acisCtl* *acisCtl.tcl*		Location of ACIS EGSE executables and libraries; it is verified at the start of each standalone module and script
`ACISTTMFILE`	EFPS	*psmc.tcl* *rctu.tcl* *options.tcl*		Name of ACIS telemetry format file, exported to the engineering telemetry interface modules
`ACISV11TEST`	D	*video11.tcl*	`()`	Read packets from this input file instead of *filterServer*
`ACISVTMTEST`	D	*videotm.tcl*	`()`	Read packets from this input file instead of *filterServer*
`ARCH`	H	*acisCtl.tcl*		System architecture, *e.g.*. "linux", "solaris", "darwin"
`BEP_MAP`	EFP	*acisCtl.tcl* *commands.tcl* *debug.tcl* *dump.tcl* *options.tcl*		Pathname of BEP load map
`CCD_SCALE`	EFP	*showit.tcl* *options.tcl*	`128`	Number of pixel rows and columns in each CCD
`CMDLOG`	PS	*options.tcl*		Directory to contain PSMC logs used only when *acisCtl* was started with the **–P** option
`CMDPORT`	EP	*acisCtl.tcl* *acisProcs.tcl* *debug.tcl* *dump.tcl* *options.tcl*	`8541`	Command port established by the ACIS interface script in engineering mode to send serial digital commands to the s/w and h/w ports of the ACIS DPA
`COGPORT`	F	*acisCtl* *acisCtl.tcl* *acisProcs.tcl* *runacis.tcl* *options.tcl*	`7543`	TCP listening port established by the ACIS interface script in flight mode
`CTU_SIDE`	PS	*options.tcl*	`actu` `abus`	Used by scripts `$RCTU_CMD`, `$PSMC_CMD` that communicate with ACIS DPA and PSMC via the obsolete *acisServer* or *acisServer-tee* interfaces. It expects that *acisCtl* was started with the **–P** option
`DANGER_CMD`	EP	*acisCtl.tcl* *psmc.tcl* *options.tcl*		Executable program to create a critical ACIS *bcmd* command
`DATAHOST`	EFP	*acisCtl* *acisCtl.tcl* *acispower.tcl* *options.tcl*		Domain name of host running *filterClient*, also exported to all standalone *acisCtl* modules
`DATAPORT`	EFP	*acisCtl* *acisCtl.tcl* *acispower.tcl* *options.tcl*	`7002`	Port on `$DATAHOST` from which to read ACIS telemetry, exported to all *acisCtl* standalone modules
`DATAYEAR`	EFP	*options.tcl*		Calendar year to use when none indicated in telemetry; leave blank to use current year; exported to all *acisCtl* standalone modules

Variable Name	Type	Module	Default	Description
`DISP_TERM`	EFP	*acisCtl.tcl* *acisProcs.tcl* *options.tcl*	`rxvt`	Executable to create a scrolling window within which to run `$EDITOR` (or `$PSMC_SERVER` in PSMC mode.)
`EDITOR`	EP	*acisCtl.tcl* *acisProcs.tcl* *options.tcl*	`gvim`	Shell command to edit an ASCII file
`EXECUTE_CMD`	EP	*options.tcl*		Shell command to format an ACIS command through *bcmd* and send it to the command server
`FEP_MAP`	EFP	*acisCtl.tcl* *commands.tcl* *debug.tcl* *dump.tcl* *options.tcl*		Pathname of FEP load map
`FILTERSERVER_OPTS`	EFPS	*options.tcl*	`-v -n16`	Options to pass to *filterServer*
`FIXED_FONT`	EFP	*textDisp.tcl* *options.tcl*		Fixed-pitch font for all X11 displays; exported to *debug*, *dump*, and *showit* standalone modules
`GETPACKETS_CMD`	EFP	*options.tcl*	`getp`	Shell command to extract ACIS packets from telemetry
`GRAB_WIN1`	I	*acisCtl.tcl* *acisAux.tcl*		Value of `$PRINT_CMD` prior to "**–P**", or the whole string if "**–P**" is missing
`GRAB_WIN2`	I	*acisCtl.tcl* *acisAux.tcl*		Value of `$PRINT_CMD` from "**–P**" to the end, or null if "**–P**" is missing
`GRAB_WINDOW`	EFP	*acisCtl.tcl* *runacis.tcl* *options.tcl*		Command to prompt the user to select an X11 window to be converted to graphic and printer by `$PRINT_CMD`
`HOME`	H	*acisCtl.tcl* *acisAux.tcl* *acisPrint.tcl* *textDisp.tcl*		The user's home directory, mostly used as a default location to which to save window contents
`IMAGE_LIB`	EP	*acisCtl.tcl* *imageLoad.tcl* *options.tcl*		Default pathname of directory containing pixel images to be send to the image loader
`imgdir`	I	*manual.tcl* *psmc.tcl* *video11.tcl* *videotm.tcl*		Pathname of directory containing mostly bitmap images used by Tcl/Tk commands to display window graphics
`LibPath`	I	*acisCtl.tcl*		Pathname of directory containing *acisCtl* modules; this is exported to all standalone modules
`LOAD_IMAGE_CMD`	EP	*acisCtl.tcl* *acisProcs.tcl* *options.tcl*		Shell command to send a pixel image to the image loader
`LOAD_RAW_CMD`	EP	*acisCtl.tcl* *rawpackets.tcl* *options.tcl*		Shell command to send a raw ACIS command file to the command server
`LOGCOMPRESS`	EFP	*acisCtl.tcl* *runacis.tcl* *options.tcl*	`gzip .gz`	Shell command and optional file extension to write input ACIS packets to `$TLM_LOG_FILE`.

Variable Name	Type	Module	Default	Description
`PARAM_BLOCK_LIB`	EP	*acisCtl.tcl* *pblocks.tcl* *options.tcl*		Pathname of directory containing parameter block files, with extensions: *te* (timed-exposure), *cc* (continuous clocking), *1d* (CC windows), *2d* (TE windows), or *dea* (DEA housekeeping)
`PIXEL_AB_DEV`	EP	*acisCtl.tcl* *acisProcs.tcl* *options.tcl*	*/dev/ttya*	Device on `$DATAHOST` that switches FEP pixel input from DEA to Image Loader when sent "1" or "0", respectively
`PMON_CMD`	EFP	*acisCtl.tcl* *acisProcs.tcl* *options.tcl*		Shell command to execute within `$DISP_TERM` to run the *pmon* command to display ACIS science telemetry
`PRINT_CMD`	EFP	*acisCtl.tcl* *acisPrint.tcl* *runacis.tcl* *textDisp.tcl* *options.tcl*		Shell command to print its *stdin* to `$PRINTER`
`PRINTER`	HS	*acisCtl.tcl* *acisPrint.tcl*	`ps`	Printer spool type or name
`PSMC_CMD`	P	*acisCtl.tcl* *psmc.tcl* *options.tcl*		Shell command to send *bcmd* commands to the PSMC. It expects that *acisCtl* was started with the **–P** option and that a suitable command interface is available
`PSMC_PORT`	P	*acisCtl.tcl* *psmc.tcl* *options.tcl*	`7002`	Port on `$PSMC_SERVER` from which to read PSMC telemetry. It expects that *acisCtl* was started with the **–P** option and that a suitable telemetry interface is available
`PSMC_SERVER`	P	*acisCtl.tcl* *acisProcs.tcl* *psmc.tcl* *options.tcl*		Host from which to read PSMC telemetry. In flight mode, this is initialized to `$DATAHOST`. Otherwise, it is initialized from the **–P** option of *acisCtl*.
`PWD`	H	*acisProcs.tcl*		Used by the `infoBlock` command to save and restore the user's working directory when editing a file
`RAW_CMD_LIB`	EP	*acisCtl.tcl* *rawpackets.tcl* *options.tcl*		Pathname of directory containing either *bcmd* commands (extension *.bcmd*) or binary commands (extension *.pkts*)
`RCTU_CMD`	EFP	*acisCtl* *acisCtl.tcl* *interface.tcl* *options.tcl*		Shell command to start *filterServer* to distribute ACIS telemetry and (in engineering mode only) start a command server to send *bcmd* commands to ACIS
`RCTU_DUMP_DIR`	EFP	*acisCtl.tcl* *options.tcl*	`/tmp`	Pathname of directory into which to write raw telemetry logs
`RCTU_DUMP_FILE`	EFP	*acisCtl.tcl* *options.tcl*		File to contain telemetry log; "%" fields will be replaced by *strftime*; files ending "*.gz*" or "*.Z*" will be compressed
`TERMLINES`	EFP	*commands.tcl* *showtlm.tcl*	`200`	Default number of lines to retain in scrolling *acisCtl* windows
`TEXT_FONT`	EFP	*acisCtl.tcl* *options.tcl*		Default text font for all X11 displays; exported to *commands*, *debug*, *dump*, *interface*, and *showtlm* standalone modules
`TCL_LIBRARY`	EFP	*acisCtl* *options.tcl*		TCL function library required by *wish*
`TK_LIBRARY`	EFP	*acisCtl* *options.tcl*		TK function library required by *wish*

Variable Name	Type	Module	Default	Description
`TLM_LOG_DIR`	EFP	*acisCtl.tcl* *runacis.tcl* *options.tcl*	`/tmp`	Pathname of directory into which to write ACIS packet logs
`TLM_LOG_FILE`	EFP	*acisCtl.tcl* *runacis.tcl* *options.tcl*		File to contain packet log; "%" fields will be replaced by *strftime*; files ending "*.gz*" or "*.Z*" will be compressed