TOPSI–Reference Manual

Version 1.0, April 7, 2008

Dr. Mark Könnecke
Labor für Neutronenstreuung
Paul Scherrer Institut
CH–5232 Villigen–PSI
Switzerland

Contents

Chapter 1

SICS Invocation

SICS means SINQ Instrument Control System. SICS is a client server system. This means there are at least two programs necessary to run the experiment. The first is the SICServer which does the actual instrument control work. A user rarely needs to bother about this server program as it is meant to run all the time. See instructions below if things go wrong.

Then there are client programs which interact with the instrument control server. These client programs implement the status displays and a command line application which forwards commands to the SICS server and displays its response. Graphical User Interfaces may be added at a later time. The user has only to deal with these SICS client programs. SICS Clients and the SICServer communicate with each other through the TCP/IP network.

Currently the following SICS clients are available:

- A command line control client for sending commands to the SICS server and displaying its repsonses.
- A status display for the powder diffractometers DMC and HRPT.
- A status display for MORPHEUS and general scans.
- A status display for SANS and SANS2.
- A status display for FOCUS.
- A AMOR control and status program.
- A triple axis control and status program.
- A SICS variable watcher. This application graphically logs the change of a SICS variable over time. Useful for monitoring for instance temperature controllers.
- A graphical client for TRICS.

1.1 Steps necessary for logging in to SICS

The following actions have to be taken in order to interact with the SICS server through a client:

- Start the client application.
- Connect the client to a SICS server.
- In case of command line clients: authorize yourself as privileged SICS user.

1.2 Starting SICS client applications

These programs can be started on a Linux system by issuing the following commands at the command prompt:

sics & for the control client.

powderstatus & for the DMC status display client.

topsistatus & for the MORPHEUS status display.

sansstatus & for the SANS status display.

focustatus for the FOCUS status display.

amor & for the AMOR status display and control application.

tas & for the triple axis status display and control application.

varwatch & for the variable watcher.

trics & for the starting the TRICS graphical client.

Another option to start SICS clients is the Java Webstart mechanism which is available for most platforms. Java webstart requires both Java and Java webstart to be installed on the computer running the client. Then clients can be started directly from a WWW-page. The advantage is that clients are automatically updated in this system as soon as new version have been copied to the WWW-site. Installation instructions for Java webstart and links to start all SICS clients though this mechanism can be found at: the SICS webstart[1] page. This service is only accessible within the PSI network.

1.3 Connecting

After startup any SICS client is not connected to a SICS server and thus not active. A connection is established through the connect menu of the client.

1.4 Authorization

SICS is a multi user instrument control system. In order to prevent malicious manipulations of the instrument SICS supports a hierarchy of user rights. In order to run an experiment you need at least user level privilege. In order to achieve this privilege you have to invoke the Authorize dialog. There you have to enter the apropriate username and password kindly provided by your instrument scientist.

[1] See URL http://lns00.psi.ch/sics/wstart

1.5 Restarting the Server

The SICS server should be running all the time. It is only down if something went wrong. You can check for the presence of the SICS server by loging in to the instrument computer and typing **monit status** at the command prompt. The output will tell you what is happening. If you need to restart the SICS server log in as the instrument user at the instrument computer and invoke the appropriate command to start the server. These are:

DMC Computer = dmc, User = dmc

TOPSI Computer = morpheus, User = morpheus

SANS Computer = sans, User = sans

SANSLI Computer = sans2, User = sans2

TRICS Computer = trics, User = trics

HRPT Computer = hrpt, User = hrpt

FOCUS Computer = focus, User = focus

AMOR Computer = amor, User = amor

TASP Computer = tasp, User = tasp

POLDI Computer = poldi, User = poldi

The SICS server process are controlled through the monit program. Usually the monit daemon is running. If not, for instance after a reboot, it can be started by typing **monit** at the unix prompt logged in as the instrument user. Further monit commands:

monit start target start the monit surveyed process target. For the choice of targets see below.

monit stop target stops the monit surveyed process target. For the choice of targets see below.

monit restart target restart the monit surveyed process target. Possible targets are:

- **sicsserver** The SICServer
- **SerPortServer** The serial port control program
- **sync** The file synchronisation program. This is responsible for coyping data files to the common AFS area.
- **simserver** Only on TASP: a simulation SICS server
- **all** Stop all processes

monit status prints a status listing of everything watched by monit

monit quit Stops monit itself

Stopping everything thus involves two commands:

- monit stop all
- monit quit

Restarting after this involves:

- monit

The older command startsics and killsics are still working and operate on the monit daemon as of now.

If all this does not help look under trouble shooting SICS (cf. Section 7).

Chapter 2

General SICS commands

This section describes general SICS concepts and SICS commands available to all instruments.

2.1 Chapter Overview

- This chapter starts with an overview of some basic (cf. Section 2.2) SICS concepts.
- Than there are system (cf. Section 2.3) commands for closing the server, requesting status information etc.
- The Token command (cf. Section 2.11)is for managing control access to the SICS server.
- A few commands change (cf. Section 2.4) user rights, set output files and the like.
- SICS has a built in macro (cf. Section 2.5) facility which is accessible through a few commands.
- Then there is the famous (cf. Section 2.6) LNS-Rünbuffer system.
- The new batch file processing system.
- Motors and parameters need to be drive (cf. Section 2.7)n.
- SICS has a facility to optimise (cf. Section 2.10) a peak with respect to several parameters.

SICS has various ways (to many!) to log the I/O from and to an instrument control server. This has devolped over time and may need a cleanup.

- There exists a server log where all I/O and internal messages is written to automatically. As this can get quite large the files for this are cyclically reused. This is log aimed at debugging problems with the SICS server.
- Logging of client output to a file can be performed with the LogBook (cf. Section 2.8) command. This is a wrapper around the config file command. This is obsolete.

- Then there is a commandlog (cf. Section 2.9) which writes all I/O coming from clients with user or manager privilege. These log files are kept under /home/INSTRUMENT/log for each day. With this log file all important operations on an instrument can be reconstructed. This gets written automatically without user intervention.

2.2 Basic SICS concepts

2.2.1 General structure

SICS is a client server system. The application the user sees is usually some form of client. A client has two tasks: the first is to collect user input and send it to the SICS server which then executes the command. The clients second task is to listen to the the server messages and display them in a readable format. This aproach has two advantages: clients can reside on machines across the whole network thus enabling remote control from everywhere in the world. The second advantage is that new clients (such as graphical user interface clients) can be written in any feasible language without changes to the server.

2.2.2 SICS Command Syntax

SICS is an object oriented system. This is reflected in the command syntax. SICS objects can be devices such as motors, single counters, histogram memories or other hardware variables such as wavelength or Title and measurement procedures. Communication with these objects happens by sending messages to the target object. This is very simply done by typing something like: object message par1 par2 .. parn. For example, if we have a motor called A1:

```
A1 list
```

will print a parameter listing for the motor A1. In this example no parameters were needed. There exist a number of one-word commands as well. For compatability reasons some commands have a form which resembles a function call such as:

```
drive a1 26.54
```

This will drive motor a1 to 26.54. All commands are ASCII-strings and usually in english. SICS is in general CASE INSENSITIVE. However, this does not hold for parameters you have to specify. On a unix system for instance file names are case sensitive and that had to be preserved. Commands defined in the scripting language are lower case by convention.

Most SICS objects also hold the parameters required for their proper operation. The general syntax for handling such parameters is:

`objectname parametername`

prints the current value of the parameter

`objectname parametername newvalue`

sets the parameter value to newvalue if you are properly authorized.

2.2.3 SICS variables

Most of the parameters SICS uses are hidden in the objects to which they belong. But some are separate objects of their own right and are accessible at top level. For instance things like Title or wavelength. They share a common syntax for changing and requesting their values. This is very simple: The command *objectname* will return the value, the command *objectname newvalue* will change the variable. But only if the authorisation codes match.

2.2.4 Authorisation

A client server system is potentially open to unauthorised hackers who might mess up the instrument and your valuable measurements. A known problem in instrument control is that less knowledgeable user accidentally change instrument parameters which ought to be left fixed. In order to solve these two problems SICS supports authorisation on a very fine level. As a user you have to specify a username and password in order to able to access SICS. Some clients already do this for you automatically. SICS support four levels of access to an instrument:

- **Spy** may look at everything, request any value, but may not actually change anything. No damage potential here.
- **User** is privileged to perform a certain amount of operations necessary to run the instrument.
- **Manager** has the permission to mess with almost everything. A very dangerous person.
- **Internal** is not accessible to the outside world and is used to circumvent protection for internal uses. However some parameters are considered to be so critical that they cannot be changed during the runtime of the SICS-server, not even by Managers.

All this is stated here in order to explain the common error message: You are not authorised to do that and that or something along these lines.

2.3 System Commands

sics_exitus . A single word commands which shuts the server down. Only managers may use this command.

wait *time* waits time seconds before the next command is executed. This does not stop other clients from issuing commands.

resetserver resets the server after an interrupt.

dir a command which lists objects available in the SICS system. Dir without any options prints a list of all objects. The list can be restricted with:

dir var prints all SICS primitive variables

dir mot prints a list of all motors

dir inter driv prints a list of all drivable objects. This is more then motors and includes virtual motors such as environment devices and wavelength as well.

dir inter count Shows everything which can be counted upon.

dir inter env Shows all currently configured environment devices.

dir match wildcard lists all objects which match the wildcard string given in wildcard.

status A single word command which makes SICS print its current status. Possible return values can be: Eager to execute commands, Scanning, Counting, Running, Halted. Note that if a command is executing which takes some time to complete the server will return an ERROR: Busy message when further commands are issued.

status interest initiates automatic printing of any status change in the server. This command is primarily of interest for status display client implementors.

backup [***file***] saves the current values of SICS variables and selected motor and device parameters to the disk file specified as parameter. If no file parameter is given the data is written to the system default status backup file. The format of the file is a list of SICS commands to set all these parameters again. The file is written on the instrument computer relative to the path of the SICS server. This is usually /home/INSTRUMENT/bin.

backup motsave toggles a flag which controls saving of motor positions. If this flag is set, commands for driving motors to the current positions are included in the backup file. This is useful for instruments with slipping motors.

restore [***file***] reads a file produced by the backup command described above and restores SICS to the state it was in when the status was saved with backup. If no file argument is given the system default file gets read.

restore listerr prints the list of lines which caused errors during the last restore.

killfile decrements the data number used for SICS file writing and thus consequently overwrites the last datafile. This is useful when useless data files have been created during tests. As this is critical command in normal user operations, this command requires managers privilege.

sicsidle prints the number of seconds since the last invocation of a counting or driving operation. Used in scripts.

2.4 Configuration Commands

SICS has a command for changing the user rights of the current client server connection, control the amount of output a client receives and to specify additional logfiles where output will be placed. All this is accessed through the following commands:

The SICS server logs all its activities to a logfile, regardless of what the user requested. This logfile is mainly intended to help in server debugging. However, clients may register an interest in certain server events and can have them displayed. This facility is accessed via the **GetLog** command. It needs to be stressed that this log receives messages from **all** active clients. GetLog understands the following messages:

- **GetLog All** achieves that all output to the server logfile is also written to the client which issued this command.
- **GetLog Kill** stops all logging output to the client.

- **GetLog OutCode** request that only certain events will be logged to the client issuing this command. Enables only the level specified. Multiple calls are possible.

Possible values for OutCode in the last option are:

- **Internal** internal errors such as memory errors etc.
- **Command** all commands issued from any client to the server.
- **HWError** all errors generated by instrument hardware. The SICS server tries hard to fix HW errors in order to achieve stable operations and may not generate an error message if it was able to fix the problem. This option may be very helpful when tracking dodgy devices.
- **InError** All input errors found on any clients input.
- **Error** All error messages generated by all clients.
- **Status** some commands send status messages to the client invoking the command in order to monitor the state of a scan.
- **Value** Some commands return requested values to a user. These messages have an output code of Value.

The **config** command configures various aspects of the current client server connection. Basically three things can be manipulated: The connections output class, the user rights associated with it, and output files.

- The command **config OutCode val** sets the output code for the connection. By default all output is sent to the client. But a graphical user interface client might want to restrict message to only those delivering requested values and error messages and suppressing anything else. In order to achieve this, this command is provided. Possible values Values for val are Internal,Command, HWError,InError,Status, Error, Value. This list is hierarchical. For example specifying InError for val lets the client receive all messages tagged InError, Status, Error and Value, but not HWError, Command and Internal messages.
- Each connection between a client and the SICS server has user rights assocociated with it. These user rights can be configured at runtime with the command **config Rights Username Password** . If a matching entry can be found in the servers password database new rights will be set.
- Scientists are not content with having output on the screen. In order to check results a log of all output may be required. The command **config File name** makes all output to the client to be written to the file specified by name as well. The file must be a file accessible to the server, i.e. reside on the same machine as the server. Up to 10 logfiles can be specified. Note, that a directly connected line printer is only a special filename in unix.
- **config close num** closes the log file denoted by num again.
- **config list** lists the currently active values for outcode and user rights.

- **config myname** retruns the name of the connection.
- **config myrights** prints the rights associated with your connection.
- **config listen 0 or 1**switches listening to the commandlog on or off for this conenction. If this on, all output to the commandlog, i.e. all interesting things happening in SICS, is printed to your connection as well.

2.5 Macro Commands

SICS has a built in macro facility. This macro facility is aimed at instrument managers and users alike. Instrument managers may provide customised measurement procedures in this language, users may write batch files in this language. The macro language is John Ousterhout's Tool Command Language (TCL). Tcl has control constructs, variables of its own, loop constructs, associative arrays and procedures. Tcl is well documented by several books and online tutorials, therefore no details on Tcl will be given here. All SICS commands are available in the macro language. Some potentially harmful Tcl commands have been deleted from the standard Tcl interpreter. These are: exec, source, puts, vwait, exit,gets and socket. A macro or batch file can be executed with the command:

exe *name* tries to open the file name and executes the script in this file.

batchrun *name* prepends to name a directory name configured in the variable batchroot and then executes that batchfile. The usage scenerio is that you have a directory where you keep batch files. Then the variable batcroot is set to contain the path to that directory. Batchrun then allows to start scripts in that directory without specifying the full path. Please note that fileeval and batchrun are obsolete and to be replaced by the new batch manager command, exe, as described below.

Then there are some special commands which can be used within macro-sripts:

ClientPut sometext1 ... Usally SICS suppresses any messages from SICS during the processing of batch files. This is in order not to confuse users with the output of intermediate results during the processing of batch files. Error messages and warnings, however, come through always. Clientput now allows to send messages to the user on purpose from within scripts.

SICSType object allows to query the type of the object specified by object. Possible return values are

- **DRIV** if the object is a SICS drivable object such as a motor
- **COUNT** if the object is some form of a counter.
- **COM** if the object is a SICS command.
- **NUM** if the object is a number.
- **TEXT** if object is something meaningless to SICS.

SICSbounds *var newval* checks if the new value newval lies within the limits for variable var(example: SICSbounds D1HL 10). Returns an error or OK depending on the result of the test.

SICSStatus *var* SICS devices such as counters or motor may be started and left running while the program is free to do something else. This command inquires the status of such a running device. Return values are internal SICS integer codes. This command is only of use for SICS programmers.

SetStatus *newval* sets the SICS status to one of: Eager, UserWait, Count, NoBeam, Driving, Running, Scanning, Batch Hatl or Dead. This command is only available in macros.

SetInt *newval, GetInt* sets SICS interrupts from macro scripts. Not recommended! Possible return values or new values are: continue, abortop, abortscan, abortbatch, halt, free, end. This command is only permitted in macros. Should only be used by SICS programmers.

2.6 Rünbuffer Commands

LNS scientists have got used to using Rünbuffers for instrument control. A Rünbuffer is an array of SICS commands which typically represent a measurement. This Rünbuffer can be edited at run time. This is very similar to a macro. In contrast to a macro only SICS commands are allowed in Rünbuffers. When done with editing the Rünbuffer it can be entered in a Rünlist. This is a stack of Rünbuffers which get executed one by one. While this is happening it is possible (from another client) to modify the Rünlist and edit and add additional Rünbuffers on top of the stack. This allows for almost infinite measurement and gives more control than a static batch file. In order to cater for this scheme three commands have been defined:

The **Buf** object is responsible for creating and deleting Rünbuffers. The syntax is:

- **Buf new *name*** creates a new empty Rünbuffer with the name name. name will be installed as a SICS object afterwards.
- **Buf copy *name1 name2*** copies Rünbuffer name1 to buffer name2.
- **Buf del *name*** deletes the Rünbuffer name.

After creation, the Rünbuffer is accessible by his name. It then understands the commands:

- ***NAME* append *what shall we do with a drunken sailor*** will add all text after append as a new line at the end of the Rünbuffer.
- ***NAME* print** will list the contents of the Rünbuffer.
- ***NAME* del *iLine*** will delete line number iLine from the Rünbuffer.
- ***NAME* ins *iLine BimBamBim*** inserts a new line **after** line iLine into the Rünbuffer. The line will consist of everything given after the iLine.
- ***NAME* subst pattern *newval*** replaces all occurences of pattern in the Rünbuffer by the text specified as newval. Currently this feature allows only exact match but may be expanded to Unix style regexp or shell like globbing.
- ***NAME* save *filename*** saves the contents of the Rünbuffer into file filename.

- ***NAME*** **load *filename*** loads the Rünbuffer with the data in file filename.
- ***NAME*** **run** executes the Rünbuffer.

The Rünlist is accessible as object **stack** . Only one Rünlist per server is permitted. The syntax:

- **stack add *NAME*** adds Rünbuffer name to the top of the stack.
- **stack list** lists the current Rünlist.
- **stack del *iLine*** deletes the Rünbuffer iLine from the Rünlist.
- **stack ins *iLine NAME*** inserts Rünbuffer name after Rünbuffer number iLine into the Rünlist.
- **stack run** executes the Rünlist and returns when all Rünbuffers are done.
- **stack batch** executes the Rünlist but does not return when done but waits for further Rünbuffers to be added to the list. This feature allows a sort of background process in the server.

2.7 Drive commands

Many objects in SICS are **drivable** . This means they can run to a new value. Obvious examples are motors. Less obvious examples include composite adjustments such as setting a wavelength or an energy. Such devices are alos called virtual motors. This class of objects can be operated by the **drive, run, Success** family of commands. These commands cater for blocking and non-blocking modes of operation.

run var newval var newval ... can be called with one to n pairs of object new value pairs. This command will set the variables in motion and return to the command prompt without waiting for the requested operations to finish. This feature allows to operate other devices of the instrument while perhaps a slow device is still running into position.

Success waits and blocks the command connection until all pending operations have finished (or an interrupt occured).

drive var newval var newval ... can be called with one to n pairs of object new value pairs. This command will set the variables in motion and wait until the driving has finished. A drive can be seen as a sequence of a run command as stated above immediatly followed by a Success command.

2.8 LogBook command

Some users like to have all the input typed to SICS and responses collected in a file for further review. This is implemented via the LogBook command. LogBook is actually a wrapper around the config file command. LogBook understands the following syntax:

LogBook alone prints the name of the current logfile and the status of event logging.

LogBook file *filename* sets the filename to which output will be printed. Please note that this new filename will only be in effect after restarting logging.

LogBook on This command turns logging on. All commands and all answers will be written to the file defined with the command described above. Please note, that this command will overwrite an existing file with the same name.

LogBook off This command closes the logfile and ends logging.

2.9 The Commandlog

The commandlog is a file where all communication with clients having user or manager privilege is logged. This log allows to retrace each step of an experiment. This log is normally configured in the startup file or can be configured by the instrument manager. There exists a special command, commandlog, which allows to control this log file.

commandlog new *filename* starts a new commandlog writing to filename. Any prior files will be closed. The log file can be found in the directory specified by the ServerOption LogFileDir. Usually this is the log directory.

commandlog displays the status of the commandlog.

commandlog close closes the commandlog file.

commandlog auto Switches automatic log file creation on. This is normally switched on. Log files are written to the log directory of the instrument account. There are time stamps any hour in that file and there is a new file any 24 hours.

commandlog tail [***n***] prints the last n entries made into the command log. n is optional and defaults to 20. Up to 1000 lines are held in an internal buffer for this command.

commandlog intervall Queries and configures the intervall in minutes at which time stamps are written to the commandlog.

It is now possible to have a script executed whenever a new log file is started. In order to make this work a ServerOption with the name logstartfile must exist in the instrument configuration file. The value of this option must be the full path name of the file to execute.

Note: with the command config listen 1 you can have the output to the command log printed into your client, too. With config listen 0 you can switch this off again. This is useful for listening into a running instrument.

2.10 The Peak Optimiser

In instrument control the need may arise to optimise a peak with respect to several variables. Optimising means finding the maximum of the peak with respect to several variables. This is useful during instrument calibration, for example. Four circle diffractometers use this facility on a day to day basis for finding and verifying the exact position of reflections. In order to support both usages a more general module has been implemented. The algorithm is like this:

```
while errors gt precision and cycles lt maxcycles
  for all variables
      do a scan
      Try find the maximum, two halfwidth points and the peak center.
      if failure extend the scan.
      if success shift the variable, remember last shift.
      If shift lt precicison mark this variable as done
   end for
end while
```

Possible outcomes of this procedure are: success, the peak was lost or the maximum number of cycles was reached. This routine requires that the instrument is currently placed somewhere on the peak and not miles away.

The peak optimiser supports another optimisation algorithm which is faster but may not be as accurate. This is hill climbing:

```
while errors gt precision and cycles lt maxcycles
  for all variables
  find the direction into which the intensity rises
  step into this direction until the intensity drops
   end for
end while
```

The Peak Optimiser is implemented as an object with the name opti. It understand the following commands:

opti clear clears the optimiser.

opti addvar name step nStep precision This command adds a variable to optimise to the optimiser. The user has to specify the name of the variable, the step width to use for scanning, the number of steps needed to cover the full peak when scanning and the precision which should be achieved when optimising the peak. The step width and number of steps parameters should cover the whole peak. However, the Optimiser will extend the scan is the specified range is not sufficient.

opti run Starts the optimiser. It will then optimise the peak. This may take some time as it uses a time consuming scan based algorithm.

opti climb Starts the optimiser in hill climbing mode. Hill climbing is faster but may not be as accurate as a scan based optimization.

The behaviour of the optimiser can be configured by modifying some parameters. The synatx is easy: **opti parameter** prints the value of the parameter, **opti parameter newval** sets a new value for the parameter. The following parameters are supported:

maxcycles The maximum number of cycles the optimiser will run when trying to optimise a peak. The default is 7.

threshold When a peak cannot be identified after a scan on a variable, the optimiser will check if there is a peak at all. In order to do that it searches for a count rate higher then the threshold parameter. If such a rate cannot be found the optimiser will abort and complain that he lost the peak.

channel The counter channel to use for scanning. The default is to use the counter. By modifying this parameter, the optimiser can optimise on a monitor instead.

countmode The counting mode to use when scanning. Possible values are **timer** or **monitor**.

preset The preset value to use for counting in the scan. Depending on the status of the countmode parameter this is either a preset time or a preset monitor.

It is the users reponsability to provide meaningful step widths. Usually this is dependent on the instrument resolution and thus fairly constant. Also these optimisation algorithms will fail if the instrument is not positioned at the flank of a peak. Probaly the best will be to do several cycles of hill climbing first, followed by one cycle of scan optimisation for extra accuracy.

2.11 The Token Command

In SICS any client can issue commands to the SICS server. This is a potential source of trouble with users possibly issuing conflicting commands without knowing. In order to deal with this problem a `"token"` mechanism has been developed. In this context the token is a symbol for the control of an instrument. A connection can grab the token and then has full control over the SICS server. Any other connection will not be privileged to do anything useful, except looking at things. A token can be released manully with a special command or is automatically released when the connection dies. Another command exists which allows a SICS manager to force his way into the SICS server. The commands in more detail:

token grab Reserves control over the instrument to the client isssuing this command. Any other client cannot control the instrument now. However, other clients are still able to inspect variables.

token release Releases the control token. Now any other client can control the instrument again. Or grab the control token.

token force password This command forces an existing grab on a token to be released. This command requires manager privilege. Furthermore a special password must be specified as third parameter in order to do this. This command does not grab control though.

Chapter 3

The Scan Command

An important concept in neutron scattering instrument control is a "scan". For a simple scan a range of instrument positions is divided into equidistant steps. The instrument then proceeds to drive to each of these points and collects data at each of them.

The general idea of the scan object is, that you configure the scan by typing commands at the command line. Once, the configuration is finished the requested scan is started. A data file will be written automatically to the default location. The scan command can not only scan over motors but also about some variables which relate to motors. For instance lamda for the wavelength. Scan can scan over more then one variable. The syntax of the scan command in some detail:

scan clear Clears current scan parameters.

scan list lists current scan parameters.

scan var *name start step* Defines a parameter (motor) to be scanned. The name of the parameter, a start value and a step width need to be given. More then one scan variable can be specified.

scan modvar *name start step* Modifies the scan parameters for scan variable name to the new values given.

scan getvars Returns a list of currently active scan variables terminated with the string -END-.

scan np *num* Sets the number of scan points.

scan preset [***val***] Sets the Preset value for the scan. Without a parameter, inquires the current value.

scan mode [***val***] Sets the count mode for the scan. Without a parameter, inquires the current value. Possible values are *timer* or *monitor*.

scan run Executes the scan.

scan cinterest This call enables automatic printing of scan counts to your connection when new values arise. This command is primariliy of interest for status display clients.

scan pinterest This function makes the scan command send a notification (the string ScanVarChange) to you whenever the scan variables get modified. This command is primarily of interest for status display clients.

3.1 Center Scan

Center scan is a convenience command which starts a scan around a specified center value. This mostly used for centering purposes. The syntax is like this:

> **cscan** *var center delta np preset*

All parameters must be specified. The parameters and their meanings:

- **var** is the variable which is to be center scanned. Only one can be specified.
- **center** is the value to use as center of the scan.
- **delta** is the step width to use for the scan.
- **np** is the number of points to scan in each direction.
- **preset** is the preset to use for the counter. As the counter mode, the mode currently configured active in the scan object is used.

> example: **cscan** *d2hr 0.5 0.01 10 10000*

3.2 Simple Scan

Simple scan is a convenience command which starts a scan for one to several variables with a simplified syntax. The syntax is like this:

> sscan *var1 start end var2 start end ... np preset*

All parameters must be specified. The parameters and their meanings:

- **var1 start end** This is how the variables to scan are specified. For each variable scanned the name of the variable, the start value and the end value of the scan must be given. More then one triplet can be given in order to allow for several scan variables.
- **np** is the number of points to scan.
- **preset** is the preset to use for the counter. As the counter mode, the mode currently configured active in the scan object is used.

3.3 Fastscan

At some instruments there is a fastscan facility. This scan starts a motor and the counter. As often as possible or configured, the scan module will calculate the difference to the previous count, normalize it to the monitor difference and print it. This is usually faster then doing a step scan. There are limitations, though:

- Fastscans are restricted to one motor. There is no way that a coordinated movement of several motors can be enforced by fastscan without mechanical coupling.
- Due to the volatile nature of the beam at SINQ, this type of scans come into problems when the beam is off. A warning is printed, but...
- If the motor is driving to fast, there might not be enough neutrons around even in the peak to be registered.

Basically, **if you see a feauture in fastscan, it is there, if not, that does not mean anything!**. Fastscan comes into its own for alignment scans or when searching peaks. The syntax is:

fastscan motor start stop speed Runs a fastscan on motor between start and stop. Speed is the speed with which the motor is run. The motors speed is reset to the original value when fastscan finishes.

diffscan skip val If fastscan produces to much output on to short intervalls, increasing the skip parameter allows to control that. Skip is the number of SICS cyles to skip between measurements. Thus this value is highly dependent on the overall performance of SICS.

3.4 Peak And Center

These two commands are related to the scan command insofar as they act upon the results of the last scan still in memory. The command **peak** prints the position, FWHM and maximum value of the peak in the last scan. The command **center** drives the first scan variable to the peak center of the last scan. Both peak and center use a rather simple but effective method for locating peaks. The prerequisite is that the peak is approximatly gaussian shaped. The algorithm first locates the peak maximum. Then it goes to the left and right of the maximum and tries to find the points of half maximum peak height. The two points are interpolated from the data and the peak position calculated as the middle point between the two halfheight points.

3.5 PSD scans

Some instrument wish to perform scans with position sensitive detectors. In such cases the scan mode can be switched between PSD and normal mode. This is done through the commands:

scan2d Switches PSAD detetcor scanning on.

scan1d Switches back to normal scan mode on a single counter.

Chapter 4

TOPSI Hardware Devices

TOPSI controls a couple of :

- motors (cf. Section 4.1).
- A counter box (cf. Section 4.2) for reading monitors and the single counter.
- various environment.htm (cf. Section 6) control devices.

Each of these devices understands commands. The syntax understood by each of these devices will be discussed.

4.1 SICS motor handling

In SICS each motor is an object with a name. Motors may take commands which basically come in the form

motorname command ; example: D2HR list.

Most of these commands deal with the plethora of parameters which are associated with each motor. The syntax for manipulating variables is, again, simple.

Motorname parametername

will print the current value of the variable.

Motorname parametername newval

will set the parameter to the new value specified. A list of all parameters and their meanings is given below. The general principle behind this is that the actual (hardware) motor is kept as stupid as possible and all the intracacies of motor control are dealt with in software. Besides the parameter commands any motor understands these basic commands:

- ***Motorname list*** gives a listing of all motor parameters.
- ***Motorname* reset** resets the motor parameters to default values. This is software zero to 0.0 and software limits are reset to hardware limits.
- ***Motorname* position** prints the current position of the motor. All zero point and sign corrections are applied.
- ***Motorname* hardposition** prints the current position of the motor. No corrections are applied. Should read the same as the controller box.

- ***Motorname* interest** initiates automatic printing of any position change of the motor. This command is mainly interesting for implementors of status display clients.

Please note that the actual driving of the motor is done via the drive (cf. Section 2.7) command.

4.1.1 The motor parameters

- **HardLowerLim** is the hardware lower limit. This is read from the motor controller and is identical to the limit switch welded to the instrument. Can usually not be changed.
- **HardUpperLim** is the hardware upper limit. This is read from the motor controller and is identical to the limit switch welded to the instrument. Can usually not be changed.
- **SoftLowerLim** is the software lower limit. This can be defined by the user in order to restrict instrument movement in special cases.
- **SoftUpperLim** is the software upper limit. This can be defined by the user in order to restrict instrument movement in special cases.
- **SoftZero** defines a software zero point for the motor. All further movements will be in respect to this zeropoint.
- **Fixed** can be greater then 0 for the motor being fixed and less then or equal to zero for the motor being movable.
- **InterruptMode** defines the interrupt to issue when the motor fails. Some motors are so critical for the operation of the instrument that all operations are to be stopped when there is a problem. Other are less critical. This criticallity is expressed in terms of interrupts, denoted by integers in the range 0 - 4 translating into the interrupts: continue, AbortOperation, AbortScan, AbortBatch and Halt. This parameter can usually only be set by managers.
- **Precision** denotes the precision to expect from the motor in positioning. Can usually only be set by managers.
- **AccessCode** specifies the level of user privilege necessary to operate the motor. Some motors are for adjustment only and can be harmful to move once the adjustment has been done. Others must be moved for the experiment. Values are 0 - 3 for internal, manager, user and spy. This parameter can only be changed by managers.
- **Sign** reverses the operating sense of the motor. For cases where electricians and not physicists have defined the operating sense of the motor. Usually a parameter not to be changed by ordinary users.
- **failafter** This is the number of consecutive failures of positioning operations this motor allows before it thinks that something is really broken and aborts the experiment.

- **maxretry** When a motor finishes driving, SICS checks if the desired position was reached. If the position read back from the motor is not within precision to the desired value, the motor is restarted. This is done at max maxretry times. After maxretry retries, the motor throws an error.
- **ignorefault** If this is bigger then 0, positioning faults from the motor will be ignored.

This list of parameters may be enhanced buy driver specific parameters. motor list will show all parameters.

4.1.2 Motor Error Handling Concepts

As mechanical components motors are prone to errors. SICS knows about two different classes of motor errors:

HWFault This is when there is a problem communicating with the motor, a limit is violated etc. SICS assumes that such errors are so grave that no fix is possible. If such a HWFault is detected a configurable interrupt (see parameter InterruptMode) is set which can be used by upper level code to act upon the problem.

HWPosFault This is a positioning failure, i.e. The motor did not reach the desired position. Such a positioning problem can come from two sources:

- The positioning problem is reported by the motor driver. SICS then assumes that the driver has done something to solve the problem and promotes this problem to a HWFault.
- The motor driver reported no error and SICS figures out by itself, that the desired position has not been reached. SICS thinks that this is the case if the difference between the desired position and the position read from the motor controller is greater then the parameter precision. If SICS detects such a problem it tries to reposition the motor. This is done for the number of times specified through the parameter maxretries. If the position has not been reached after maxretries repositionings, a HWFault is assumed.

In any case lots of warnings and infos are printed.

If SICS tries to drive an axis which is broken hardware damage may occur (and HAS occurred!). Now, SICS has no means to detect if the mispositioning of a motor is due to a concrete block in the path of the instrument or any other reason. What SICS can do though is to count how often a motor mispositions in sequence. This means SICS increments a mispositioning counter if it cannot drive a motor, if the motor is driven succesfully, the mispositioning counter is cleared. If the count of mispositionings becomes higher then the parameter failafter, SICS thinks that there is something really, really wrong and aborts the measurement and prints an error message containing the string: MOTOR ALARM.

There are some common pitfalls with this scheme:

You want upper level code to be signalled when your critical motorfails. Solution: set the parameter interruptmode to something useful and check for the interrupt in upper level code.

SICS falsly reports mispositionings. Solution: increase the precision parameter.

You know that a motor is broken, you cannot fix it, but you want to measure anyway. Solution: increase the precision parameter, if SICS finds the positioning problem, increase maxretries, increase the failafter parameter. In the worst case set the ignorefault parameter to greater 0, this will prevent all motor alarms.

4.2 SICS counter handling

A counter in SICS is a controller which operates single neutron counting tubes and monitors. A counter can operate in one out of two modes: counting until a timer has passed, for example: count for 20 seconds. Counting in this context means that the noutrons coming in during these 20 seconds are summed together. This mode is called timer mode. In the other mode, counting is continued until a specified neutron monitor has reached a certain preset value. This mode is called Monitor mode. The preset values in Monitor mode are usually very large. Therefore the counter has an exponent data variable. Values given as preset are effectively 10 to the power of this exponent. For instance if the preset is 25 and the exponent is 6, then counting will be continued until the monitor has reached 25 million. Note, that this scheme with the exponent is only in operation in Monitor mode. Again, in SICS the counter is an object which understands a set of commands:

- **countername setpreset *val*** sets the counting preset to val.
- **countername getpreset** prints the current preset value.
- **countername preset *val*** With a parameter sets the preset, without inquires the preset value. This is a duplicate of getpreset and setpreset which has been provided for consistency with other commands.
- **countername setexponent *val*** sets the exponent for the counting preset in monitor mode to val.
- **countername getexponent** prints the current exponent used in monitor mode.
- **countername setmode *val*** sets the counting mode to val. Possible values are Timer for timer mode operation and Monitor for waiting for a monitor to reach a certain value.
- **countername getmode** prints the current mode.
- **countername *mode val*** With a parameter sets the mode, without inquires the mode value. This is a duplicate of getmode and setmode which has been provided for consistency with other commands. Possible values for val are either monitor or timer.
- **countername setexponent *val*** sets the exponent for the counting preset in monitor mode to val.
- **countername getcounts** prints the counts gathered in the last run.

- **countername getmonitor *n*** prints the counts gathered in the monitor number n in the last run.
- **countername count *preset*** starts counting in the current mode and the given preset.
- **countername status** prints a message containing the preset and the current monitor or time value. Can be used to monitor the progress of the counting operation.
- **countername gettime** Retrieves the actual time the counter counted for. This excludes time where there was no beam or counting was paused.
- **countername getthreshold *m*** retrieves the value of the threshold set for the monitor number m.
- **countername setthreshold *m val*** sets the threshold for monitor m to val. WARNING: this also makes monitor m the active monitor for evaluating the threshold. Though the EL7373 counterbox does not allow to select the monitor to use as control monitor in monitor mode, it allows to choose the monitor used for pausing the count when the count rate is below the threshold (Who on earth designed this?)
- **countername send *arg1 arg2 arg3 ...*** sends everything behind send to the counter controller and returns the reply of the counter box. The command set to use after send is the command set documented for the counter box elsewhere. Through this feature it is possible to diretclly configure certain variables of the counter controller from within SICS.

Chapter 5

TOPSI Device list

5.1 Motors

Although there are many names and ways to address a motor I recommend to use the systematic alias mnemonic. This 3-letter mnemonic uses the first letter to distinguish between monochromator (M__), sample (S__), diaphragms (D__), and motors used for scan modes of the u-shaped multireflection table (U__). Nevertheless, you may use the other aliases in the following list or TASMAD mnemonic, if this appears to be more natural to you. The second and third letter are used to specify the function or the part to be moved, e.g., a rotation of sample or monochromator for an angle theta (_TH), the rotation of a secondary arm (_TT), both around vertical axes, translations (_TX & _TY) and tiltings of goniometers (_GX & _GY) along perpendicular directions, the blade of a diaphragm (_1L, _2R, _3V, etc.).

5.2 Mnemonic and Function of Motors

See Tables 5.1, 5.2, 5.3

5.3 Counters

There is only one named counter.

5.4 Variables

Instrument holds the name of the Instrument, in SANS case: SANS at SINQ, PSI. This cannot be changed by ordinary members of the public.

Title the title of the experiment. This will be written to the output file. Should be set by the user to something sensible.

User the user of the instrument. This will be written to the output file. Should be set by the user to something sensible.

Table 5.1: TOPSI motors 1

Standard alias mnemonic	TASMAD mnemonic	Motor # (motor # on controller	Location of motor digitizer	Function
MTH	A1	1 (1, 2)	in mono drum	rotation of monochromator crystal around vertical axis
MTT	A2	2 (2, 2)	back of mono drum	rotation of platform around vertical monochromator axis
MTY	MTL	5 (5, 2)	on top of motor 1 in mono drum	horizontal translation of monochromator crystal
MTX	MTU	6 (6, 2)	on top of motor 5 in mono drum	horizontal translation of monochromator crystal perpendicular to MTY
MGX	MGL	9 (9, 2)	uppermost in mono drum	rotation of monochromator crystal around horizontal axis to define horizontal scattering plane
STH, TH	A3	3 (3, 2)	2nd lowermost on platform	rotation of sample around vertical axis
STT, TTH	A4	4 (4, 2)	lowermost on platform	rotation of detector arm around vertical sample axis
STY	STL	7 (7, 2)	on top of motor 3 on platform	horizontal translation of sample (thought to be perpendicular to beam, but depends on STH value)
STX	STU	8 (8, 2)	on top of motor 7 on platform	horizontal translation of sample perpendicular to STY (thought to be along the beam, but depends on STH value)

Table 5.2: TOPSI motors 2

SGY	SGL	11 (11, 2)	2nd uppermost on platform (uppermost when SGX is not mounted)	rotation around a horizontal axis to incline the sample along STY direction
SGX	SGU	12 (12, 2)	uppermost on platform	rotation of sample around an inclined axis (along STX direction if STY=0)
D1R, S1R	MCV	13 (1, 3)	diaphragm 1 (nearest to mono drum)	movement of the right aperture (when looking from the monochromator)
D1L, S1L	S3S	14 (2, 3)	diaphragm 1 (nearest to mono drum)	movement of the left aperture (when looking from the monochromator)
D1V, S1V	ACH	15 (3, 3)	diaphragm 1 (nearest to mono drum)	vertical movement of either the upper or lower aperture (depends on which one is connected)

Comment1 one line of comment to be written to the outputfile. Should be set by the user to something sensible.

Comment2 another line of comment to be written to the outputfile. Should be set by the user to something sensible.

environment a line of text describing special environment conditions to be written to the output file. This is for those cases where the sample environment is not computer controlled and therefore not visible to the program. Users are responsible for filling this in.

Table 5.3: TOPSI motors 3

D2R, S2R	ATL	17 (5, 3)	diaphragm 2	movement of the right aperture (when looking from the monochromator)
D2L, S2L	ATU	18 (6, 3)	diaphragm 2	movement of the left aperture (when looking from the monochromator)
Standard alias mnemonic	TASMAD mnemonic	Motor # (motor # on controller	Location of motor digitizer	Function
D2V, S2V	AGL	19 (7, 3)	diaphragm 2	vertical movement of either the upper or lower aperture (depends on which one is connected)
D3R, S3R	MCH	21 (9, 3)	diaphragm 3 (nearest to detector)	movement of the right aperture (when looking from the monochromator)
D3L, S3L	ACH	22 (10, 3)	diaphragm 3 (nearest to detector)	movement of the left aperture (when looking from the monochromator)
D3V, S3V	CCH	23 (11, 3)	diaphragm 3 (nearest to detector)	vertical movement of either the upper or lower aperture (depends on which one is connected)
UTY	A5	16 (4, 3)	lowermost on multireflection table	horizontal translation of multireflection yoke (perpendicular to detector arm)
UTZ	A6	20 (8, 3)	uppermost on multireflection table	vertical translation of multireflection yoke (perpendicular to detector arm)
UTT	MGU	24 (12, 3)	between support base and detector arm	fine translation of detector arm
AUX		10 (10, 2)	auxiliary	for additional motors, for instance a vertical translation stage on sample cradle

Chapter 6

Sample Environment Devices

6.1 SICS Concepts for Sample Environment Devices

SICS can support any type of sample environment control device if there is a driver for it. This includes temperature controllers, magnetic field controllers etc. The SICS server is meant to be left running continously. Therefore there exists a facility for dynamically configuring and deconfiguring environment devices into the system. This is done via the **EVFactory** command. It is expected that instrument scientists will provide command procedures or specialised Rünbuffers for configuring environment devices and setting reasonable default parameters.

In the SICS model a sample environment device has in principle two modes of operation. The first is the drive mode. The device is monitored in this mode when a new value for it has been requested. The second mode is the monitor mode. This mode is entered when the device has reached its target value. After that, the device must be continously monitored throughout any measurement. This is done through the environment monitor or **emon**. The emon understands a few commands of its own.

Within SICS all sample environement devices share some common behaviour concerning parameters and abilities. Thus any given environment device accepts all of a set of general commands plus some additional commands special to the device.

In the next section the EVFactory, emon and the general commands understood by any sample environment device will be discussed. This reading is mandatory for understanding SICS environment device handling. Then there will be another section discussing the special devices known to the system.

6.2 Sample Environment Error Handling

A sample environment device may fail to stay at its preset value during a measurement. This condition will usually be detected by the emon. The question is how to deal with this problem. The requirements for this kind of error handling are quite different. The SICS model therefore implements several strategies for handling sample environment device failure handling. The strategy to use is selected via a variable which can be set by the user for any sample environment device separately. Additional error handling strategies can be added with a modest amount of programming. The error handling strategies currently implemented are:

Lazy Just print a warning and continue.

Pause Pauses the measurement until the problem has been resolved.

Interrupt Issues a SICS interrupt to the system.

Safe Tries to run the environment device to a value considered safe by the user.

Script Run a user defined script to do any magic things you may want.

6.3 General Sample Environment Commands

6.3.1 EVFactory

EVFactory is responsible for configuring and deconfiguring sample environment devices into SICS. The syntax is simple:

EVFactory new name type par par ... Creates a new sample environment device. It will be known to SICS by the name specified as second parameter. The type parameter decides which driver to use for this device. The type will be followed by additional parameters which will be evaluated by the driver requested.

EVFactory del name Deletes the environment device name from the system.

6.3.2 emon

The environment monitor emon takes for the monitoring of an environment device during measurements. It also initiates error handling when appropriate. The emon understands a couple of commands.

emon list This command lists all environment devices currently registered in the system.

emon register name This is a specialist command which registers the environment device name with the environment monitor. Usually this will automatically be taken care of by EVFactory.

emon unregister name This is a specialist command which unregisters the environment device name with the environment monitor. Usually this will automatically be taken care of by EVFactory Following this call the device will no longer be monitored and out of tolerance errors on that device no longer be handled.

6.3.3 General Commands UnderStood by All Sample Environment Devices

Once the evfactory has been run successfully the controller is installed as an object in SICS. It is accessible as an object then under the name specified in the evfactory command. All environemnt object understand the common commands given below. Please note that each command discussed below MUST be prepended with the name of the environment device as configured in EVFactory!

The general commands understood by any environment controller can be subdivided further into parameter commands and real commands. The parameter commands just print the name of the parameter if given without an extra parameter or set if a parameter is specified. For example:

Temperature Tolerance

prints the value of the variable Tolerance for the environment controller Temperature. This is in the same units as the controller operates, i. e. for a temperature controller Kelvin.

Temperature Tolerance 2.0

sets the parameter Tolerance for Temperature to 2.0. Parameters known to ANY envrironment controller are:

Tolerance Is the deviation from the preset value which can be tolerated before an error is issued.

Access Determines who may change parameters for this controller. Possible values are:

- 0 only internal
- 1 only Managers
- 2 Managers and Users.
- 3 Everybody, including Spy.

LowerLimit The lower limit for the controller.

UpperLimit The upper limit for the controller.

ErrHandler. The error handler to use for this controller. Possible values:

- 0 is Lazy.
- 1 for Pause.
- 2 for Interrupt
- 3 for Safe.
- 4 for Script.

For an explanantion of these values see the section about error (cf. Section 6.2) handling above.

errorscript The user specified script to execute when the controlled value goes out of tolerance. Will be used whne the ErrHandler 4, script, is used.

Interrupt The interrupt to issue when an error is detected and Interrupt error handling is set. Valid values are:

- 0 for Continue.
- 1 for abort operation.

- 2 for abort scan.
- 3 for abort batch processing.
- 4 halt system.
- 5 exit server.

SafeValue The value to drive the controller to when an error has been detected and Safe error handling is set.

MaxWait Maximal time in minutes to wait in a drive temperature command. If maxwait is set to 0: If the temperature is not reached within tolerance, it waits indefinitely.

Settle Wait time [minutes] after reaching temperature. Indicates how long to wait after reaching temperature. If the temperatures goes again out of tolerance during the settling time, the time outside tolerance is not taken into account.

Additionally the following commands are understood:

send par par ... Sends everything after send directly to the controller and return its response. This is a general purpose command for manipulating controllers and controller parameters directly. The protocoll for these commands is documented in the documentation for each controller. Ordinary users should not tamper with this. This facility is meant for setting up the device with calibration tables etc.

list lists all the parameters for this controller.

no command, only name. When only the name of the device is typed it will return its current value.

name val will drive the device to the new value val. Please note that the same can be achieved by using the drive command. and **log frequency** (both below)

6.3.4 Logging

The values of any sample environement device can be logged. There are three features:

- Logging to a file wih a configurable time intervall between log file entries.
- Sums are kept internally which allow the calculation of the mean value and the standard deviation at all times.
- A circular buffer holding 1000 timestamps plus values is automatically updated.

The last two systems are automatically switched on after the first drive or run command on the environment device completed. This system is run through the following commands.

name log clear Resets all sums for the calculation of the mean value and the standard deviation.

name log getmean Calculates the mean value and the standard deviation for all logged values and prints them.

name log frequency val With a parameter sets, without a parameter requests the logging intervall for the log file and the circular buffer. This parameter specifies the time intervall in seconds between log records. The default is 300 seconds.

name log file filename Starts logging of value data to the file filename. Logging will happen any 5 minutes initially. The logging frequency can be changed with the name log frequency command. Each entry in the file is of the form date time value. The name of the file must be specified relative to the SICS server.

name log flush Unix buffers output heavily. With this command an update of the file can be enforced.

name log status Queries if logging to file is currently happening or not.

name log close Stops logging data to the file.

name log tosicsdata dataname copies the content of the circular buffer to a sicsdata buffer. This is used by graphical clients to display the content of the circular buffer.

name log dump Prints the content of the circular log buffer to screen.

name log dumptofile filename Prints the content of the circular log buffer into the file specified as filename. Note, this file is on the computer where the SICS server resides.

6.4 Special Environment Control Devices

This section lists the parameters needed for configuring a special environment device into the system and special parameters and commands only understood by that special device. All of the general commands listed above work as well!

6.4.1 LakeShore Model 340 Temperature Controller

This is *the* temperature controller for cryogenic applications and should replace at least the Oxford & Neocera controllers at SINQ.

The control is handled by a seperate server process TECS (TEmperature Control Server) and is initialized by default on most instruments. If there is already an other device selected, it must be deleted with:

EVFactory del temperature

and TECS must be reinstalled with:

tecs on

(This is just an abbreavation for EVFactory new temperature tecs)

More details can be found on the Sample Environment Home Page[1]

[1]See URL http://sinq.web.psi.ch/sinq/sample.env/tecs.html

6.4.2 ITC-4 and ITC-503 Temperature Controllers

These temperature controller were fairly popular at SINQ. They are manufactured by Oxford Instruments. At the back of this controller is a RS-232 socket which must be connected to a terminal server via a serial cable.

ITC-4 Initialisation

An ITC-4 can be configured into the system by:

EVFactory new Temp ITC4 computer port channel

This creates an ITC-4 controller object named Temp within the system. The ITC-4 is expected to be connected to the serial port channel of the serial port server porgramm at localhost listening at the specified port. For example:

EVFactory new Temp ITC4 localhost 4000 7

connects Temp to the serial port 7, listening at port 4000.

ITC-4 Additional Parameters

The ITC-4 has a few more parameter commands:

timeout Is the timeout for the SerPortServer waiting for responses from the ITC-4. Increase this parameter if error messages contaning ?TMO appear.

sensor Sets the sensor number to be used for reading temperature.

control Sets the control sensor for the ITC-4. This sensor will be used internally for regulating the ITC-4.

divisor The ITC4 does not understand floating point numbers, the ITC-503 does. In order to make ITC4's read and write temperatures correctly floating point values must be multiplied or divided with a magnitude of 10. This parameter determines the appropriate value for the sensor. It is usually 10 for a sensor with one value behind the comma or 100 for a sensor with two values after the comma.

multiplicator The same meaning as the divisor above, but for the control sensor.

Installing an ITC4 step by step

1. Connect the ITC temperature controller to port 7 on the terminal server box. Port 7 is specially configured for dealing with the ideosyncracies of that device. No null modem is needed.

2. Install the ITC4 into SICS with the command:
   evfactory new temperature localhost 4000 7
   You may choose an other name than "temperature", but then it is in general not stored in the data file. Please note, that SICS won't let you use that name if it already exists. For instance if you already had a controller in there. Then the command:
   evfactory del name
   will help.

3. Configure the upper and lowerlimits for your controller appropriatetly.
4. Figure out which sensor you are going to use for reading temperatures. Configure the sensor and the divisor parameter accordingly.
5. Figure out, which sensor will be used for controlling the ITC4. Set the parameters control and multiplicator accordingly. Can be the same as the sensor.
6. Think up an agreeable temperature tolerance for your measurement. This tolerance value will be used 1) to figure out when the ITC4 has reached its target position. 2) when the ITC4 will throw an error if the ITC4 fails to keep within that tolerance. Set the tolerance parameter according to the results of your thinking.
7. Select one of the numerous error handling strategies the control software is able to perform. Configure the device accordingly.
8. Test your setting by trying to read the current temperature.
9. If this goes well try to drive to a temperature not to far from the current one.

ITC-4 Trouble Shooting

If the ITC-4 **does not respond at all**, make sure the serial connection to is working. Use standard RS-232 debugging procedures for doing this. The not responding message may also come up as a failure to connect to the ITC-4 during startup.

If error messages containing the string **?TMO** keep appearing up followed by signs that the command has not been understood, then increase the timeout. The standard timeout of 10 microseconds can be to short sometimes.

You keep on reading **wrong values** from the ITC4. Mostly off by a factor 10. Then set the divisor correctly. Or you may need to choose a decent sensor for that readout.

Error messages when **trying to drive the ITC4**. These are usually the result of a badly set multiplicator parameter for the control sensor.

The ITC4 **never stops driving**. There are at least four possible causes for this problem:

1. The multiplicator for the control sensor was wrong and the ITC4 has now a set value which is different from your wishes. You should have got error messages then as you tried to start the ITC4.
2. The software is reading back incorrect temperature values because the sensor and divisor parameters are badly configured. Try to read the temperature and if it does have nothing to do with reality, set the parameters accordingly.
3. The tolerance parameter is configured so low, that the ITC4 never manages to stay in that range. Can also be caused by inappropriate PID parameters in the ITC4.
4. You are reading on one sensor (may be 3) and controlling on another one (may be 2). Then it may happen that the ITC 4 happily thinks that he has reached the temperature because its control sensor shows the value you entered as set value. But read sensor 3 still thinks he is far off. The solution is to drive to a set value which is low enough to make the read sensor think it is within the tolerance. That is the temperature value you wanted after all.

6.4.3 Haake Waterbath Thermostat

This is sort of a bucket full of water equipped with a temperature control system. The RS-232 interface of this device can only be operated at 4800 baud max. This is why it has to be connected to a specially configured port. The driver for this device has been realised in the Tcl extension language of the SICS server. A prerequisite for the usage of this device is that the file hakle.tcl is sourced in the SICS initialisation file and the command inihaakearray has been published. Installing the Haake into SICS requires two steps: first create an array with initialisation parameters, second install the device with evfactory. A command procedure is supplied for the first step. Thus the initialisation sequence becomes:

inihaakearray name-of-array localhost name port channel
evfactory new temperature tcl name-of-array

An example for the SANS:

inihaakearray eimer localhost 4000 1
evfactory new temperature tcl eimer

Following this, the thermostat can be controlled with the other environment control commands.

The Haake Thermostat understands a single special subcommand: **sensor**. The thermostat may be equipped with an external sensor for controlling and reading. The subcommand sensor allows to switch between the two. The exact syntax is:

temperature sensor val

val can be either intern or extern.

6.4.4 Bruker Magnet Controller B-EC-1

This is the Controller for the large magnet at SANS. The controller is a box the size of a chest of drawers. This controller can be operated in one out of two modes: in **field** mode the current for the magnet is controlled via an external hall sensor at the magnet. In **current** mode, the output current of the device is controlled. This magnet can be configured into SICS with a command syntax like this:

evfactory new name bruker localhost port channel

name is a placeholder for the name of the device within SICS. A good suggestion (which will be used throughout the rest of the text) is magnet. bruker is the keyword for selecting the bruker driver. port is the port number at which the serial port server listens. channel is the RS-232 channel to which the controller has been connected. For example (at SANS):

```
evfactory new magnet bruker localhost 4000 9
```

creates a new command magnet for a Bruker magnet Controller connected to serial port 9. In addition to the standard environment controller commands this magnet controller understands the following special commands:

magnet polarity Prints the current polarity setting of the controller. Possible answers are plus, minus and busy. The latter indicates that the controller is in the process of switching polarity after a command had been given to switch it.

magnet polarity val sets a new polarity for the controller. Possible values for val are **minus** or **plus**. The meaning is self explaining.

magnet mode Prints the current control mode of the controller. Possible answers are **field** for control via hall sensor or **current** for current control.

magnet mode val sets a new control mode for the controller. Possible values for val are **field** or **current**. The meaning is explained above.

magnet field reads the magnets hall sensor independent of the control mode.

magnet current reads the magnets output current independent of the control mode.

Warning: There is a gotcha with this. If you type only magnet a value will be returned. The meaning of this value is dependent on the selected control mode. In current mode it is a current, in field mode it is a magnetic field. This is so in order to support SICS control logic. You can read values at all times explicitly using magnet current or magnet field.

6.4.5 The Eurotherm Temperature Controller

At SANS there is a Eurotherm temperature controller for the sample heater. This and probably other Eurotherm controllers can be configured into SICS with the following command. The eurotherm needs to be connected with a nullmodem adapter.

evfactory new name euro computer port channel

name is a placeholder for the name of the device within SICS. A good suggestion is temperature. euro is the keyword for selecting the Eurotherm driver. port is the port number at which the serial port server listens. channel is the RS-232 channel to which the controller has been connected. **WARNING:** The eurotherm needs a RS-232 port with an unusual configuration: 7bits, even parity, 1 stop bit. Currently only the SANS port 13 is configured like this! Thus, an example for SANS and the name temperature looks like:

```
evfactory new temperature euro localhost 4000 13
```

There are two further gotchas with this thing:

- The eurotherm needs to operate in the EI-bisynch protocoll mode. This has to be configured manually. For details see the manual coming with the machine.
- The weird protocoll spoken by the Eurotherm requires very special control characters. Therefore the send functionality usually supported by a SICS environment controller could not be implemented.

6.4.6 The Risoe A1931 Temperature Controller

This is a temperature controller of unknown origin (probably built at Risoe) which is coming with the Risoe instruments. This temperature controller is connected to the computer systems through a GPIB bus and controller. A A1931 temperature controller is configured into SICS through the command:

evfactory new temperature-name a1931 gpib-controller-name gpibaddress

This creates a new command temperature-name. gpib-controller-name is the name of a GPIB controller within SICS. A GPIB controller is configured into SICS with the command MakeGPIB as described in the SICS managers documentation. gpibaddress is the address of the A1931 on the GPIB bus.

A A1931 temperature device understands a couple of additional commands on top of the standard set:

temperature sensor val The A1931 can switch control to various sensors. This command allows to query the control sensor (command without parameter) or set the control sensoe (command with parameter).

temperature file filename The A1931 can be configured through files containing calibration commands. Sich file can be loaded into the A1931 through the file subcommand. The full path of filename must be given.

6.4.7 The PSI-EL755 Magnet Controller

This is magnet controller developed by the electronics group at PSI. It consists of a controller which interfaces to a couple of power supplies. The magnets are then connected to the power supplies. The magnetic field is not controlled directly but just the power output of the power supply. Also the actual output of the power supply is NOT read back but just the set value after ramping. This is a serious limitation because the computer cannot recognize a faulty power supply or magnet. The EL755 is connected to SICS with the command:

evfactory new name el755 localhost port channel index

with port and channel being the usual data items for describing the location of the EL755-controller at the serial port server. index is special and is the number of the power supply to which the magnet is connected. An example:

```
evfactory new maggi el755 localhost 4000 5 3
```

connects to power supply 3 at the EL755-controller connected to lnsa09 at channel 5. The magnet is then available in the system as maggi. No special commands are supported for the EL755.

6.4.8 PSI-DSP Magnet Controller

The PSI-DSP magnet controller has been developed by the PSI electronics group, most notably by Lukas Tanner, for the SLS. However, these controllers are now being used at SINQ as well. This controller has a binary command protocoll and thus the send command does not work for it. In order to handle this protocoll SICS has to bypass the usual SerPortServer mechanism for communicating with serial devices and to connect to the terminal server directly. This also implies one gotcha: **The PSI-DSP works only at specially configured terminal server ports**.The terminal server port to which the PSI-DSP is connected **MUST** be configured to: 115200 baud, 8 data bits, 1 stop bit, odd parity. In general a system manager is required to do this. The PSI-DSP also requires a null-modem connector between the box and the terminal server. Once these hurdles have been mastered, the PSI-DSP can be configured into SICS with the command:

> evfactory new name psi-dsp terminalservername port

with name being the name of the magnet in SICS, terminalservername the name of the terminal server, for example psts224 and port being the address of the binary port on the terminal server. This is usually the serial port number at the terminal server plus 3000. An example:

> evfactory new maggi psi-dsp psts224 3016

configures a magnet named maggi which is connectd to port 16 at the terminal server psts224. maggi can now be read and driven like any other environment device.

6.4.9 Old Dilution Cryostat (Obsolete)

This is a large ancient device for reaching very low temperatures. This cryostat can be configured into SICS with the command:

```
EVFactory new Temp dillu computer port channel table.file
```

Temp is the name of the dilution controller command in SICS, dillu is the keyword which selects the dilution driver, computer, port and channel are the parameters of the Macintosh-PC running the serial port server program. table.file is the fully qualified name of a file containing a translation table for this cryostat. The readout from the dilution controller is a resistance. This table allows to interpolate the temperature from the resistance measurements and back. Example:

```
evfactory new temperature dillu lnsp19.psi.ch 4000 1 dilu.tem
```

installs a new dilution controller into SICS. This controller is connected to port 1 at the Macintos-PC with the newtwork adress lnsp19.psi.ch. On this macintosh-PC runs a serial port server program listening at TCP/IP port 4000. The name of the translation table file is dilu.tem.

The dilution controller has no special commands, but two caveats: As of current (October 1998) setting temperatures does not work due to problems with the electronics. Second the dilution controller MUST be connected to port 1 as only this port supports the 4800 maximum baud rate this device digests.

6.4.10 Old CryoFurnace Controller (Obsolete)

The CryoFurnace at PSI is equipped with a Neocera LTC-11 temperature controller. This controller can control either an heater or an analag output channel. Futhermore a choice of sensors can be selected for controlling the device. The LTC-11 behaves like a normal SICS environment control device plus a few additional commands. An LTC-11 can be configured into SICS with the following command:

evfactory new name ltc11 computer port channel

name is a placeholder for the name of the device within SICS. A good suggestion is temperature. ltc11 is the keyword for selecting the LTC-11 driver. Computer is the name of the computer running David Maden's SerPortServer program, port is the port number at which the SerPortServer program listens. Channel is the RS-232 channel to which the controller has been connected. For example (at DMC):

```
evfactory new temperature ltc11 localhost 4000 6
```

creates a new command magnet for a LTC-11 temperature Controller connected to serial port 6 at lnsp18.

The additional commands understood by the LTC-11 controller are:

temperature sensor queries the current sensor used for temperature readout.

temperature sensor val selects the sensor val for temperature readout.

temperature controlanalog queries the sensor used for controlling the analog channel.

temperature controlanalog val selects the sensor val for controlling the analog channel.

temperature controlheat queries the sensor used for controlling the heater channel.

temperature controlheat val selects the sensor val for controlling the heater channel.

temperature mode queries if the LTC-11 is in analog or heater control mode.

Further notes: As the CryoFurnace is very slow and the display at the controller becomes unusable when the temperature is read out to often, the LTC-11 driver buffers the last temperature read for 5 seconds. Setting the mode of the LTC-11 is possible by computer, but not yet fully understood and therefore unusable.

Chapter 7

SICS Trouble Shooting

There is no such thing as bug free software. There are always bugs, nasty behaviour etc. This document shall help to solve these problems. The usual symptom will be that a client cannot connect to the server or the server is not responding. Or error messages show up. This section helps to solve such problems.

7.1 Looking at Log Files

The first thing to do, especially when confronted with confusing statements from either users or instrument scientists, is to look at the SICS servers log files. The last 1000 lines of the instrument log are accessible from any SICS client or through the WWW interface. The SICS commands:

commandlog tail shows the last 20 lines of the log.

commandlog tail n shows the last n lines of the log.

will show you the information available. In order to see more, log in to the instrument account. There the following unix commands might help:

- **sicstail** shows the last 20 lines of the current log file and its name
- **sicstail n** shows the last n lines of the current log file.

In order to see some more, cd into the log directory of the instrument account. In there are files with names like:

`auto2001-08-08@00-01-01.log`

This means the log file has been started at August, 8, 2001 at 00:01:01. There is a new log file daily. Load appropriate files into the editor and look what really happened.

Another good ideas is to use the unix command grep on assorted log files. A grep for the strings ERROR or WARNING will more ofteh then not give an indication for the nature of the problem.

The log files show you all commands given and all the responses of the system. Additionaly there are hourly time stamps in the file which allow to narrow in when the problem started. Things to watch out for are:

MOTOR ALARM This message means that the motor failed to reach his position for a couple of times. This is caused by either a concrete shielding element blocking the movement of the instrument, badly adjusted motor parameters, mechanical failures or the air cushions not operating properly.

EL734__BAD_EMERG_STOP Somebody has pushed the emergency stop button. This must be released before the instrument can move again. Moreover the motor controller will not respond to further commands in this mode. Thus restarting SICS on this error message will make SICS fail to initialize the motors affected!

EL*__BAD_PIPE, BAD_RECV, BAD_ILLG, BAD_TMO, BAD_SEND** Network communication problems. Can generaly be solved by restarting SICS.

EL737__BAD_BSY A counting operation was aborted while the beam was off. Unfortunately, the counter box does not respond to commands in this state and ignores the stop command sent to it during the abort operation. This can be safely ignored, SICS fixes this condition.

7.2 Restarting SICS

All of SICS can be restarted through the command:

```
monit restart all
```

7.3 Starting SICS

An essential prerequisite of SICS is that the server is up and running. The system is configured to restart the SICServer whenever it fails. Only after a reboot or when the keepalive processes were killed (see below) the SICServer must be restarted. This is done for all instruments by typing:

```
monit
```

at the command prompt. startsics actually starts two programs: one is the replicator application which is responsible for the automatic copying of data files to the laboratory server. The other is the SICS server. Both programs are started by means of a shell script called **keepalive**. keepalive is basically an endless loop which calls the program again and again and thus ensures that the program will never stop running.

When the SICS server hangs, or you want to enforce an reinitialization of everything the server process must be killed. This can be accomplished either manually or through a shell script.

7.4 Stopping SICS

All SICS processes can be stopped through the commands:

```
monit stop all
monit quit
```

given at the unix command line. You must be the instrument user (for example DMC) on the instrument computer for this to work properly.

7.5 Restart Everything

If nothing seems to work any more, no connections can be obtained etc, then the next guess is to restart everything. This is especially necessary if mechanics or electronics people were closer to the instrument then 400 meters.

1. Reboot the histogram memory. It has a tiny button labelled RST. That' s the one. Can be operated with a hairpin, a ball point pen or the like.
2. Wait 5 minutes.
3. Restart the SICServer. Watch for any messages about things not being connected or configured.
4. Restart and reconnect the client programs.

If this fails (even after a second) time there may be a network problem which can not be resolved by simple means.

7.6 Checking SICS Startup

Sometimes it happens that the SICServer hangs while starting up or hardware components are not properly initialized. In such cases it is useful to look at the SICS servers startup messages. On the instrument account issue the commands:

```
monit stop sicsserver
cd inst_sics
./SICServer inst.tcl | more
```

Replace inst with the name of the appropriate instrument in lower case. For example, from the home directory of the hrpt account on the computer hrpt:

```
cd
monit stop sicsserver
cd hrpt_sics
./SICServer hrpt.tcl | more
```

This allows to page through SICS startup messages and will help to identify the troublesome component. The proceed to check the component and the connections to it.

7.7 HELP debugging!!!!

The SICS server hanging or crashing should not happen. In order to sort such problems out it is very helpful if any available debugging information is saved and presented to the programmers. Information available are the log files as written continously by the SICS server and posssible core files lying around. They have just this name: core.pid, where pid is the process identification number. In order to save them create a new directory (for example dump2077) and copy the stuff in there. This looks like:

```
/home/DMC> mkdir dump2077
/home/DMC> cp log/*.log dump2077
/home/DMC> cp core.2077 dump2077
```

The `/home/DMC>` is just the command prompt. Please note, that core files are only available after crashes of the server. These few commands will help to analyse the cause of the problem and to eventually resolve it.

Chapter 8

TOPSI Command Summary

8.1 Most Used Commands

drive mot1 NewVal mot2 NewVal .... (cf. Section 2.7) Drives motors. Followed by pairs of motor names and new values.

scan clear Clears current scan parameters.

scan list lists current scan parameters.

scan var name start step Defines a variable (motor) to be scanned. The name of the variable, a start value and a stpe width need to be given. More then one scan variable can be specified.

scan np num Sets the number of scan points.

scan preset val Sets the Preset value for the scan. Without a parameter, inquires the current value.

scan mode val Sets the count mode for the scan. Without a parameter, inquires the current value. Possible values are Timer or Monitor.

scan run Executes the scan.

cscan var center delta np preset does a scan center scan for variable var around center. delta is the step width to use, np is the number of points to collect at each side, preset is the preset to use.

sscan var1 start end var2 start end .... np preset simplified scan command. Scans variables var1, var2 .. from the respective start values to the end values. Using np points and a preset of preset.

FileEval filename (cf. Section 2.5) Runs the batch file specified as argument filename.

8.2 Driving

drive mot1 NewVal mot2 NewVal .... (cf. Section 2.7) Drives motors. Followed by pairs of motor names and new values.

run mot1 NewVal mot2 NewVal .... Runs motors.

Known motors are: OmegaA, A1, TwoThetaM, A2, MonoX, MonoY, MonoChi, MonoPhi, CurveM, Table, A3, TwoThetaD, A4.

8.3 Counting

counter SetMode mode Sets the mode of the counter. Possible parameters for mode are Timer and Preset.

counter GetMode Prints the current counting mode of the counter.

counter SetPreset val Sets a new preset for the counter

counter GetPreset Prints the current value of preset.

counter SetExponent val Sets a new expoenent for the preset in monitor mode.

counter GetExponent Prints the current value of exponent.

counter Count preset Starts counting with the preset value specified.

counter GetCounts Inquires the last counts measured.

counter GetMonitor n Gets the last value measured in monitor n.

8.4 Scan (cf. Section 3)

scan clear Clears current scan parameters.

scan list lists current scan parameters.

scan var name start step Defines a variable (motor) to be scanned. The name of the variable, a start value and a stpe width need to be given. More then one scan variable can be specified.

scan NP num Sets the number of scan points.

scan Preset val Sets the Preset value for the scan. Without a parameter, inquires the current value.

scan Mode val Sets the count mode for the scan. Without a parameter, inquires the current value. Possible values are Timer or Monitor.

scan run Executes the scan.

cscan var center delta np preset does a scan center scan for variable var around center. delta is the step width to use, np is the number of points to collect at each side, preset is the preset to use.

sscan var1 start end var2 start end .... np preset simplified scan command. Scans variables var1, var2 .. from the respective start values to the end values. Using np points and a preset of preset.

8.5 Rünbuffer

Buf new name (cf. Section 2.6) New buffer name

Buf copy name1 name2 copies buffers.

Buf del name deletes buffer.

Buffers created with Buf new name are installed as command name and understand:

NAME append bla bla .... Append text to buffer

NAME del iLine Deletes line.

NAME ins iLine bla bla .... Inserts text after line.

NAME print prints contents of buffer to screen.

NAME save file Saves buffer to file.

NAME read file Read buffer contents from file.

NAME run Executes contents of buffer.

There can be a stack of Rünbuffers.

RuLi add buffer Adds an buffer to the stack.

RuLi list Lists the stack.

RuLi del line Deletetes buffer from stack. RuLi ins iLine name Inserts name after iLine.

RuLi run Executes Stack.

RuLi batch Executes stack permanently. New buffers may be added.

8.6 General commands

Success (cf. Section 2.7) wait for the last operation to finish.

wait time (cf. Section 2.3) wait for time to pass....

Dir (cf. Section 2.3) lists all objects in the system.

config Rights username password (cf. Section 2.4) changes authorisation to that of the user identified by username, password.

FileEval filename (cf. Section 2.5) executes batch file filename.

8.7 LogBook

LogBook file name (cf. Section 2.8) sets log file name

LogBook on switches logging on

LogBook off closes LogBook

LogBook lists current logging status

8.8 Variables

Each variable can be inquired by just typing its name. It can be set by typing the name followed by the new value. Currently available variables are:

- Title
- User
- comment1
- comment2
- comment3
- User
- adress
- phone
- email
- sample