# SICS manual for the SANS I instrument

**Dr. Joachim Kohlbrecher**

**Dr. Mark Könnecke**

**Dr. Ronny Vavrin**

## SICS manual for the SANS I instrument

by Dr. Joachim Kohlbrecher, Dr. Mark Könnecke, and Dr. Ronny Vavrin

Published 20.April 2007

**Abstract**

This manual describes how to set up the small angle neutron scattering (SANS) instrument at the SINQ spallation source at PSI and gives a short reminder of important commands. This document can be downloaded (http://kur.web.psi.ch/sans1/SANSDoc/SICS4SANS1.pdf) as a pdf file.

# Table of Contents

# Chapter 1. Quick reference Guide

## 1.1. Instrument control programs

`startsics &`

starts the SICS server.

`killsics &`

kills the SICS server.

`sics &`

starts the SICS command line client.

`sansstatus &`

starts the SANS status display.

`varwatch &`

starts the variable watcher.

## 1.2. The `token` command

`token grab`

reserves control over the instrument to the client issuing this command.

`token release`

releases the token.

`token force <passwd>`

This command forces an exiting grab on a token to be released (manager privileges required).

# 1.3. Executing a macro

`exe BatchPath [<path>]`

defines directory for script files executed by `exe`.

`exe <filename>`

executes script file `<filename>` located in directory defined by `exe BatchPath`. This command allows enhanced batch control through the `SICSBatchEditor`.

`ClientPut <text>`

writes `<text>` to the client in which this command was written.

`BroadCast <text>`

writes `<text>` to all clients.

in a terminal window: `sanscheck <filename>`

checks the batchfile `<filename>` for errors and calculates the total of Monis to be measured.

TCL command language interface

is implemented in the SICS server.

# 1.4. Logging executed commands

`commandlog new <filename>`

starts a new `commandlog` writing to file `<filename>` (file stored in `/home/SANS/log`).

`commandlog`

displays status of `commandlog`.

`commandlog close`

closes the command log.

`commandlog auto`

switches the automatic log on.

`commandlog tail <n>`

prints the last <n> lines of the log file.

## 1.5. Driving a motor

`run <mot1> <val1> [<mot2> <val2> ...]`

starts motion of the motors to the new values without waiting for the requested operation to finish.

`drive <mot1> <val1> [<mot2> <val2> ...]`

drives the motors and waits until movement is finished.

`success`

waits and blocks the command connection until all pending operations have finished.

`<mot> list`

lists all the motor <mot> parameters.

## 1.6. Motor handling

`<motor> list`

lists all motor parameters.

`<motor> reset`

resets motor parameters to default values.

`<motor> position`

prints actual position.

`<motor> interest`

prints position changes.

`<motor> hardlowerlim`

prints hardware lower limit.

`<motor> hardupperlim`

prints hardware upper limit

`<motor> softlowerlim [<val>]`

prints/sets software lower limit.

`<motor> softupperlim [<val>]`

prints/sets software upper limit.

`<motor> softzero [<val>]`

prints/sets software zero.

`<motor> fixed [<val>]`

fixes motor for `<val>` $\geq 0$ or releases it for `<val>` $< 0$.

`<motor> interruptmode [<val>]`

defines interrupt issued if motor movement fails.

`<motor> precision [<val>]`

denotes the positioning precision.

`<motor> accesscode [<val>]`

specify user privileges of `<motor>`.

`<motor> speed`

defunct.

`<motor> sign [<val>]`

reverses operation sense.

## 1.7. Special commands (e.g. `st`, `dt` ...)

`<cop>`

lists current position.

`<cop> back`

drives component to last position.

`<cop> pos <name>`

defines a name for actual position.

`<cop> <name>`

drives component to named position.

`<cop> drop <name>`

deletes named position.

`<cop> drop all`

deletes all named positions.

`<cop> list`

lists all named position.

`<cop> find`

returns name of actual position.

`<cop> <axis> [=] <val> ...`

drive component to new position.

## 1.8. Sample environment devices

`EVFactory new <ED> <type> <par> ...`

creates a new sample environment device called `<ED>`.

`EVFactory del <ED>`

deletes sample environment device `<ED>`.

`emon list`

lists all registered environment devices.

`emon register <ED>`

registers environment device `<ED>`.

`emon unregister <ED>`

unregisters environment device `<ED>`.

`<ED> Tolerance [<val>]`

allowed deviation from preset.

`<ED> Access [<val>]`

changes access to device.

`<ED> Lowerlimit [<val>]`

lower limit for controller.

`<ED> Upperlimit [<val>]`

upper limit for controller.

`<ED> ErrHandler [<val>]`

error handler to use for this controller.

`<ED> Interrupt [<val>]`

interrupt to issue when an error is detected.

`<ED> SafeValue [<val>]`

The value to drive the controller to when an error is detected and safe error handling is set.

`<ED> send <par> [<par> ...]`

sends everything after `send` directly to the controller and returns its response.

`<ED> list`

lists all parameters.

`<ED> [<val>]`

returns current value or will drive device to new value `<val>`.

`<ED> log on/off`

switches logging on/off.

`<ED> log clear`

clears all recorded time stamps and values.

`<ED> log gettime`

retrieves list of all recorded time stamps.

`<ED> log getval`

retrieves list of all recorded values.

`<ED> log getmean`

calculates mean value and standard deviation of logged values.

`<ED> log frequency [<val>]`

specifies the time intervall in seconds between log records.

## 1.9. Beam shutter

`shutter`

returns status of instrument shutter.

`shutter open`

opens beam shutter.

`shutter close`

closes beam shutter.

## 1.10. Neutron Velocity Selector

`lambda`

prints the current wavelength in nm.

`drive lambda <val>`

changes wavelength to value <val> in nm.

`lambda rot <val>`

calculates the rotation speed for the wavelength given by `<val>`.

`lambda wl <val>`

calculates the wavelength for the rotation speed given by `<val>`.

`nvs status`

prints status summary.

`nvs list`

displays rotation speed and tilt angle.

`nvs [rot=<val1>] [tilt=<val2>]`

sets a new tilt angle and/or rotation speed.

`nvs rotinteres`

enables printing of status message about the current state of the selector when it is driven.

`nvs loss`

start a loss current measurement.

`nvswatch [<par>] ...`

is the object for monitoring the velocity selector and understands the commands for sample environment devices.

# 1.11. Positioning an attenuator

`att`

prints the current positioned attenuator.

`att <val>`

positions attenuator `<val>`. Allowed attenuator numbers are: 0, 1, 2, 3, 4 and 5.
0 : square 50 mm x 50 mm slit, attenuation = 1
1 : circular 41 x diameter 0.4 mm slit, attenuation = 1/485
2 : circular 9 x diameter 2 mm slit, attenuation = 1/88
3 : circular 20 mm diameter slit, attenuation = 1/8
4 : circular 30 mm diameter slit, attenuation = 1/3.5
5 : circular 15 mm diameter slit, attenuation = 1/??

# 1.12. Changing collimation

`coll`

prints the current collimatin length.

`coll <val>`

positions the collimation `<val>`. Allowed collimation lengths are: 1, 1.4, 2, 3, 4.5, 6, 8, 11, 15 and 18.

# 1.13. Positioning the detector

`detectorx`

motor to change sample detector distance (`x`).

`detectory`

motor for lateral displacement (`y`).

`detectorrotation`

motor for detector rotation (`phi`).

`dt [x=<val1>] [y=<val2>] [phi=<val3>]`

special SANS command for controlling all three detector axes together.

# 1.14. Positioning the beam stop

`beamstopx`

motor for horizontal movement.

`beamstopy`

motor for vertical movement.

`bs [x=<val1>] [y=<val2>]`

special SANS command for controlling both beam stop axes together.

`bsout`

moves beam stop out of detecion area.

`bsin`

moves beam stop back into position.

`bscfree`

releases the beam stop motors.

`bschange [<val>]`

changes beam stop size, without parameter it returns the beam stop type.
1 for beam stop size of 40 mm x 40 mm
2 for beam stop size of 70 mm x 70 mm
3 for beam stop size of 85 mm x 85 mm
4 for beam stop size of 100 mm x 100 mm

`bscslot`

returns value of the beam stop type.

# 1.15. Sample table

`saz`

motor for vertical translation (`z`).

`say`

motor for horizontal translation parallel to neutron beam (`y`).

`sax`

motor for horizontal translation perpendicular to neutron beam (`x`).

`som`

motor for rotation around vertical axis (`phi`).

`gtheta`

motor for rotation around horizontal axis (`theta`).

`sposi`

horizontal translation for linear translator (`posi`).

`st [<axis n>=<val n>] ...`

is a special SANS command for controlling all seven sample axes together. `<axis n>` can be `x, y, z, omega, phi, theta, posi`.

## 1.16. Bruker electro magnet

`evfactory new magnet bruker`
`lnsa10.psi.ch 4000 9`

initialisation sequence.

`magnet`

is a valid sample environment device if initialised as above.

`magnet polarity`

prints polarity of magnet.

`magnet polarity <val>`

sets polarity, `<val>` can be `plus` or `minus`.

`magnet mode`

prints the mode of the controller.

`magnet mode <val>`

sets mode, `<val>` can be `field` or `current`.

`magnet field`

prints the magnetic field.

`magnet current`

prints the current.

## 1.17. Sample holder for electro magnet

`mz`

motor for vertical movement (`z`).

`mom`

motor for rotation around vertical axis (`omega`).

`msh [z=<val1>] [omega=<val2>]`

is a special SANS command for controlling both axes of the magnet sample holder together.

## 1.18. Haake C25P temperature controller

`start_sea`

starts the sea server from SICS.

`temperature`

reads out the current temperature.

`drive temperature <val>`

changes the set temperature of the thermostat to the value `<val>`.

in a terminal window: `sea`

starts the sea server client from a terminal. Choose Haake as device. The sea server client is the best way to control and monitor the thermostat manually. Please refer to the SEA Wiki page (http://lns00.psi.ch/sinqwiki/Wiki.jsp?page=Sea) for more information.

## 1.19. Eurotherm controller

`evfactory new temperature euro lnsa10.psi.ch 4000 <port>`

initialisation sequence. <port> is the serial port number at sans.psi.ch where the controller is connected (usually port 13).

`temperature`

is a valid sample environment device if initialised as above.

`temperature list`

overview of the temperature controller parameters. Use 'emon unregister temperature' to avoid out of range error messages.

## 1.20. Analogue and digital input and output

`init_adios.tcl`

initialisation sequence.

`AO <channel> <value>`

sends an analogue signal (-10V -> 10V) to channel 0 or 1.

`AI <channel>`

reads the tension at channels 0 to 7. Channels 0 and 1 are reserved for the CJC and the thermocouple readout.

`temp <type>`

returns the temperature as measured by a thermocouple (tc) or a resistor (pt).

`log <time interval> <channel1> <channel2> ...`

stores the values measured at the specified channels into a file in the speficied time interval. Up to 4 channels can be logged simultanuously. The temperature can be logged using channel 1.

`log stop`

stops logging the data.

# 1.21. Counter handling

`counter SetPreset <val>`

sets the preset to `<val>`.

`counter GetPreset`

prints the current preset value.

`counter SetExponent <val>`

sets exponent for `Monitor` mode.

`counter GetExponent`

prints current exponent used in `Monitor` mode.

`counter SetMode <val>`

sets counting mode. Allowed modes are `Timer` and `Monitor`.

`counter GetMode`

prints the current mode.

`counter GetCounts`

prints counts gathered in the last run.

`counter GetMonitor <n>`

prints counts gathered by monitor `<n>` in the last run.

`counter Count <preset>`

starts counting with preset `<preset>`.

`counter Status`

prints the counter status.

`counter GetTime`

retrieves the actual time the counter counted for.

`counter GetThreshold <n>`

retrieves the threshold of monitor `<n>`.

`counter SetThreshold <n> <val>`

sets the threshold for monitor `<n>` to `<val>` and also sets the active monitor for the threshold to `<n>`.

## 1.22. Histogram Memory

`banana configure HistMode <val>`

sets the mode of operation (`Transparent, Normal, TOF, Stroposcopic`).

`banana configure OverFlowMode <val>`

determines how bin overflow is handled (`Ignore, Ceil, Count`).

`banana configure Rank <val>`

defines number of histograms in memory.

`banana configure Length <val>`

gives length of individual histogram.

`banana configure BinWidth <val>`

determines size of single bin in histogram memory in bytes.

`banana timebin`

prints actual time binning aray.

`banana genbin <start> <step> <n>`

determines size of single bin in histogram memory in bytes.

`banana setbin <inum> <val>`

configuring unequally spaced time binnings.

`banana clearbin`

deletes the binning informations.

`banana preset [<val>]`

prints/sets preset `Timer` or `Monitor`.

`banana exponent [<val>]`

prints/sets exponent for `Monitor` preset.

`banana CountMode [<mode>]`

prints/sets count mode (`Monitor`, `Timer`).

`banana init`

transfers configuration from host to the actual histogram memory.

`banana count`

starts counting.

`banana initval <val>`

initialises histogram memory to value `<val>`.

`banana get <i> <istart> <iend>`

retrieves a part of histogram memory.

`banana sum <d0min> <d0max> <d1min> <d1max>`

returns the sum of counts on an area of the detector.

## 1.23. Data acquisition

`StoreData`

writes current instrument status to a NeXus file.

`count [<mode> <preset>]`

start count operation in mode `<mode>` with preset `<preset>`.

`repeat <num> [<mode> <preset>]`

calls `<num>` times `count`.

## 1.24. XY table

`xydata clear`

clears all entries in the x-y table.

`xydata list`

lists the entries in the x-y table on the screen.

`xydata write <filename>`

writes the x-y list to the disk file `<filename>`. This file resides on the machine running the SICS server.

`xydata uuget`

sends the x-y list in an uuen-coded format. This is the command to give to the VarWatch SICS client in order to make it display the x-y list.

`xydata add <xval> <yval>`

creates a new x-y list entry with the values <xval> and <yval>.

## 1.25. Useful commands

`gc <measMonis> <totalCounts>`

GuessCount: executes a measurement of <measMonis> Monis and extrapolates the measurement time and Monis for a final value of <totalCounts> counts. A useful tool to estimate the measurement time for a sample. Example: `gc 5 1E7` forces the instrument to measures for 5 Monis and estimates the measurement time and Monis for 10’000’000 counts.

`qrange [x <detdist>] [wl <lambda>]`

calculates the q range of the current setting. The detector distance x (in mm) and the wavelength wl (in nm) can be varied with the optional parameters. Examples: `qrange` prints the q range of the current setting, `qrange x 6000 wl 0.8` prints the q range of the current setting with forced values for the detector distance (6000mm) and the wavelength (0.8nm).

`sinq`

prints the SINQ status in the SICS client which issued the command.

in a terminal window: `sanscheck <filename>`

checks the batchfile `<filename>` for errors and calculates the total of Monis to be measured.

# Chapter 2. How to start

## 2.1. Log in on the SANS computer

After starting up an X-Terminal click on the XDMCP button and choose the computer *lnsa10.psi.ch*. Login with your `<username>` and `<password>` (case sensitive, this is a Unix machine). Wait for the Common Desktop Environment to start up. Open a terminal window (dtterm). You will be asked for your last name and if this is your first login a subdirectory `/data/lnsg/<lastname>` will be created.

## 2.2. Start the SINQ instrument control software SICS

SICS is a client server system. This means there are at least two programs necessary to run the experiment. The first is the server program, which runs for the SANS instrument on the *lnsa10.psi.ch* workstation. A user rarely needs to bother about this program as it meant to run all the time. Secondly a client program is needed, through which the user can interact with the instrument control server. Its main purpose is to forward commands to the server and to display the answers. For the SANS instrument three SICS clients are available:

1. command line control client (`sics &`)
2. status display (`sansstatus &`)
3. variable watcher (`varwatch`)

Any of the SICS clients first have to be connected to a SICS server before it becomes active.Therefor you first have to establish a connection through the `[Connect]` menu of the client.

### 2.2.1. Command control line client

The command line control client allows you after connecting to the instrument server to read out instrument parameters like motor positions, temperatures, magnetic field, etc. In order to move a motor or to start a data aquisition you need access privileges. The SICS server supports autorisation on different levels to protect the instrument against unauthorized hackers or accidental change of the instrument adjustment of less knowledgable user. Four different levels of access to the instrument are supported.

1. *Spy*: you may read out any instrument value, but may not change anything
2. *User*: you are privileged to perform a certain amount of operations necessary to run the instrument
3. *Manager*: you have the permission to do almost everything
4. *Internal*: is not accessible to the outside world and is used to circumvent protection for internal uses

To change the privileges select `[Set Rights]` from the `[User Parameter]` menu. With an valid username and the corresponding password you will get the privileges to change the instrument set-up. Please do not use this privileges on any other terminal than the one in the SANS cabine. For more information click on `[Help]` button on the top right corner of the command line client. A list of the most needed SICS commands is given in Chapter 3. If you start the sics client the first time you should fill out this items in the menu `[New Experiment]` for documentation of your data.

## 2.2.2. SANS status display

The SANS status display is an application for displaying the status of a SANS measurement at SINQ. The application can be stated by the command `sansstatus &`. First a connection to the SICS server has to be established. You can choose between two data display areas. The first form shows a large colour mapped image of the detector. The second form shows a couple of selected instrument parameters. Switching between the two displays is achieved through the buttons `[Detector]` and `[Parameter]` of the tool bar.

The displayed data window can be updated either manually or automatic. By default manual update is enabled. Manual update happens when hitting the red button at the bottom. Automatic update can be enabled by checking the CheckBox labelled `[Automatic Update]` in the `[Update]` menu. The update intervall can be configured with a dialog which shows up when the `[Set Update Intervall]` menu selection is clicked. Automatic updates impose a high network load. Use with care and only when necessary. For this reason, automatic updates are disabled in the applet version of this program. An updates take approximatly 5 seconds. Consequently update intervalls shorter than that are nonsense. Speeding this up an `[Turbo Update]` button has been introduced. In this modus the client doesn't communicate with the SICS server to recieve the detector information, but it communicates directly with the histograming memory. This speeds the update frequency up to a few frames per second. `[Turbo Update]` will continuously poll the histogram memory for new histogram data. This is only limited by CPU, network and histogram memory performance. Thus `[Turbo Update]` tends to overload your system. It should only be used while adjusting the beamstop, otherwise its use is a waste of system resources which will eventually slow down your data reduction chores. Do NOT forget to switch this off when you are done. The system will automatically switch to `[Auto Update]` mode after 20 minutes.

The detector display has a lot of options. Below the colour picture there is a line which displays the current detector coordinate and the data value at the current mouse position. By choosing the `[Open Old File]` item in the `[File]` menu you can load an old data set, i.e. you can use the SANS status display as a viewer for old SANS data files. A double clicking on the colour display brings up a dialog which allows to configure the colour mapping of the detector display. By holding down the mouse button and dragging the mouse a rectangular area of the detector display can be selected. When releasing the mouse button, the selected region is summed and displayed as a X-Y graph in a separate window. In the title of this plot the total counts in the selected rectangular area on the detector is printed. Dragging the mouse downwards in this new window allows to zoom in into details of the histogram. Dragging the mouse upwards zooms out again. At the bottom of the window is a dismiss button which removes the histogram window. The histogram in the histogram window will NOT be automatically updated when new data arrives.

### 2.2.3. Variable watcher `varwatch`

This little SICS client allows to plot the value of a variable in a SICS server against time. This will probably be mostly used for watching the temperature stability of a temperature controller. But it is not restricted to temperature variables, all numeric SICS variables can be monitored that way.

The configuration of the variable watcher involves two steps:

1. Select your favourite SICS server from the choices in the `[Connect]` menu. `[Custom]` allows to specify a SICS server directly by host name and port number.
2. Configure the variable to watch using the `[Configure]` item in the `[Plot]` menu. Three parameters are relevant:
   a. The watch frequency, which is the time in minutes between plot updates.
   b. The backlog, which is the number of values which will be plotted.
   c. The SICS variable to watch. This text must be a valid SICS command which returns a reply in the form `blabla = <number>` .

Further user interface elements are two buttons below the plot. These allow to start and stop the recording of the variables value. The yellow text field besides these buttons displays the current operation of the variable watcher which can be one of Inactive or Recording. Below is a line showing the current status of the SICS server. Please note, that the most recent value is always at the right hand side of the plot.

The plot can be zoomed into by dragging downwards with the left mouse button. Zooming out is achieved by dragging outwards with the mouse or with the `[Reset]` or `[Rescale Y]` options in the `[Plot]` menu. The plot can be printed using the `[Print]` option in the `[Plot]` menu. Next to plotting a variable in a SICS server against time the variable watcher can be used to display a `XYTable` which is a class maintaining a list of x-y values. If a `XYTable` object is available in the system under the name `<xydata>` the command `<xydata> uuget` has to be given to the `varwatch` SICS client in order to make it display the x-y list. More information about handling a `XYTable` can be found in Section 3.5.3.

## 2.3. Instrument preparation before the first measurement

Before you can start with your first data aquisition be aware of a few things. For running a measurement you need at least to run two programs: the SANS status display (`sansstatus &`) and the command line client (`sics &`). After connecting both clients to the SANS server and authorizing the command line client at the server with user privileges you are ready to change the instrument set-up. The initial step at the beginning of your measurement are the following:

1. Check if beam is closed, if not than do it now.

2. Set the experiment informations via the dialog box [New Experiment].
3. Define the directory, where you intend to store your batch files with exe batchpath /data/lnsg/<username>[/<batchdir>].
4. Choose the needed instrument settings like sample detector distance, collimation, wavelength, etc.
5. Adjust the beam stop by performing the following steps:
   - Move in an attenuator with an attenuation factor larger than 100.
   - Put a strong forward scatterer into the beam, e.g. teflon
   - Center the beam stop roughly by the SICS command bs x 0 y 0.
   - Open the beam.
   - Use the command banana preset 1000 and banana count to start an aquisition without storing the result.
   - Adjust the beam stop with the bs-command.
   - After each beam stop movement you can reset the histograming memory of the detector by banana initval 0.
   - Remove the attenuator.
   - Perform a fine adjustment of the beam stop with an unattenuated beam if neccessary.
   - Stop the data aquisition by pressing the [Interrupt] button in the lower left corner of the command line client.

# Chapter 3. Instrument control commands

## 3.1. Some basic SICS commands and concepts

### 3.1.1. The `token` command

In SICS any client can issue commands to the SICS server. This is a potential cause for trouble with users issuing conflicting commands without knowing. In order to deal with this problem a token mechanism has been developed. A connection can grab a token and then has full control over the SICS server. Any other connection will not be privileged to do anything useful, except for looking at things. A token can be released manually 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 issuing 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.

### 3.1.2. How to execute Macros

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 (http://cseng.awl.com/authordetail.qry?AuthorID=69) Tool Command Language (TCL) (http://www.tcltk.com). A set of important Tcl commands are described in section Section 3.2. To execute batch files, the following commands are available:

`exe BatchPath [<pathname>]`

By this command the directory name, in which SICS is searching for a batch file, is stored in the SICS variable `exe BatchPath`. Calling `exe BatchPath` without parameters will return the actual contents of the variable.

`exe <filename>`

This command tries to open the file `filename` located in the directory defined by `exe BatchPath` and executes the script in this file. This command allows enhanced batch control through the `SICSBatchEditor`.

If you want to print information from a macro script to a client (about the progress of the batch job for example), special commands are available:

`ClientPut some text ...`

This command writes everything after `ClientPut` to the client which started the script.

`BroadCast some text ...`

This command writes everything after `BroadCast` to all clients.

To check a batchfile for errors and to estimate the total Monis to be measured, a special program can be executed in a terminal window. Navigate to the folder containing the batchfile and execute the following command

in a terminal window: `sanscheck <filename>`

checks the batchfile `<filename>` for errors and calculates the total of Monis to be measured.

### 3.1.3. Logging the executed commands

Some users wish to have all communication of the SICS server with all the clients having user or manager privileges being collected in a file for further review. This is implemented via the `commandlog` command. This log allows to retrace each step of an experiment. It is usually switched off and must be configured by the instrument manager. `commandlog` understands the following syntax:

`commandlog new <filename>`

starts a new `commandlog` writing to `<filename>`. The log file can be found for the SANS server in the directory `/home/SANS/log`.

`commandlog`

without further parameters displays the status of the `commandlog`.

`commandlog close`

closes the command log file.

If the user wishes a transscript of the SICS session of the command line client he is just working with, he can use the `[Open Logfile]` option in the `[File]` menu. It will allow you to specify a file on your local disk area, where all input/output is logged to. You will get everything which appears in the client's I/O window. The input is prepended with ???. In contrast to the `commandlog`, which log communication of all clients connected to the server, the `[Open Logfile]` option of the client only allows you to log the communication of its own to a local file.

### 3.1.4. Some variables for documentation

SICS supports a couple of variables for documentation of a measurement. Currently available are:

```
user, adress, phone, fax, email, title, subtitle, environment, sample,
comment
```

Each variable can be inquired by just typing its name. It can be set by typing the name followed by the new name, e.g. `title nanocrystalline Fe`. These variables can also be set by the `[Set Experiment]` dialog box, which can be started via the `[New Parameter]` option of the `[User Parameter]` menu item in the command line client. The user is required to fill this variables with utmost care, because if you want us to retrieve your data in ten years time you will be happy that the information will be there.

### 3.1.5. 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 a temperature. This class of ob jects 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 do things to the instrument while for example a slow device is 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 ob ject 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 immediatetly followed by a Success command.

## 3.1.6. 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>`. 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.

`<motorname> position`

prints the current position of the motor.

`<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` or `run` command described in the last Section 3.1.5

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 0 for the motor being fixed and less than 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 shall be stopped when there is a problem. Others 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.

`Speed`

defunct.

`Sign`

allows to reverse 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.

## 3.1.7. Special SANS commands

This section describes some commands special to SANS. One feature of SANS is that components are used as a whole rather than refering to single motors. For SANS the beamstop, the detector and the sample table is managed like this. Within a component each axis can be adressed specifically by using an `axis` = `value` pair. Axis defined for each component will be listed below. Additionally these components support common commands as well. These mainly deal with named positions. A named position is a set of values which can be driven to by just specifying its name. For instance: `bs PositionA` drives the BeamStop `bs` to the position defined as `PositionA`. All component drive commands do not block, i.e. you can type further commands. If it is needed to wait for a component to arrive, use the `Success` command right after your command.

Commands supported by all components (in the following, the name of the component will be represented by `<COP>`):

`<COP>`

The name of the component alone will yield a listing of the current position of the component. This is also a way how to find out which axis the component supports.

`<COP> back`

drives the component back to the position before the last command. Please note, that back does not operate on itself, i.e. two times `<COP> back` will not do a trick.

`<COP> pos <name>`

This command promotes the current position of the component to a named position `<name>`. Afterwards this position can be reached by typing: `<COP> <name>` .

`<COP> drop <name>`

deletes the named position specified as second parameter.

`<COP> find`

returns the name `<name>` of the actual position of the component if the position was named before by `<COP> pos <name>`.

`<COP> <axis 1> [=] <val 1> [<axis 2> [=] <val 2> ...]`

drives the component to the new position specified by 1-n sets of `<axis n> = <val n>` pairs. The equal sign is not mandatory and can be left out. The axis refers to the internal axis of the component which is seen in a listing as created by typing `<COP>`. A relative movement of an axis can be performed if an `++` or `--` precedes the value for `<val n>`. A value `++10`, for example would mean an increase of the actual axis position by 10.

The components which follow the component syntax described above are `bs` (beam stop), `dt` (detector), `st` (sample table), and `msh` (magnet sample holder). They are described in more detail in later sections.

## 3.1.8. Sample Environment Devices

### 3.1.8.1. 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 continuously. 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 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 modus. The device is monitored in this modus when a new value for it has been requested. The second modus is the monitor modus. This modus 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` command understands a few commands of its own.

Within SICS all sample environment 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 paragraphs the `EVFactory`, `emon` and the general commands understood by any sample environment device will be discussed. Reading this is mandatory for understanding SICS environment device handling. Then there will be another section later on in this manual discussing the special devices known to the system.

### 3.1.8.2. 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 differentiated. 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 separatetly. 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.

### 3.1.8.3. General Sample Environment Commands

*`EVFactory` command:*

`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 `<name>`. 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.

*`emon` command:*

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. Following this call the device will no longer be monitored and out of tolerance errors on that device no longer be handled.

## 3.1.9. General Commands UnderStood by All Sample Environment Devices

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`.
`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
For an explanantion of these values see the section about error handling above.

`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 for abort scan
3 for abort batch processing
4 halt system
5 exit server
For an explanantion of these values see the section about error handling above.

`SafeValue`

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

Additionally the following commands are understood:

`<name> send <par> [<par> ...]`

Sends everything after send directly to the controller and return its response. This is a general purpose command which allows to manipulate 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.

`<name> list`

lists all the parameters for this controller.

`<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.

`<name> log on`

Switches logging on. If logging is on, at each cycle in the `<emon>` the current value of the environment variable will be recorded together with a time stamp. Be careful about this, for each log point a bit of memory is allocated. At some time the memory is exhausted! `<name> log clear` frees it again and `log frequency` (both below) allows to set the logging time intervall.

`<name> log of`

Switches logging off.

`<name> log clear`

Clears all recorded time stamps and values.

`<name> log gettime`

This command retrieves a list of all recorded time stamps.

`<name> log getval`

This command retrieves all recorded values.

`<name> log getmean`

Calculates the mean value and the standard deviation for all logged values and prints it.

`<name> log frequency [<val>]`

With a parameter `<val>` sets, without a parameter requests the logging intervall for the log created. This parameter specifies the time intervall in seconds between log records. The default is 5 minutes. A value of 0 means a record for each cycle of the SICServer.

# 3.2. TCL command language interface

The macro language implemented in the SICS server is John Ousterhout's (http://cseng.awl.com/authordetail.qry?AuthorID=69) Tool Command Language (TCL) (http://www.tcltk.com). Tcl has control constructs, variables of its own, loop constructs, associative arrays and procedures. Tcl is well documented by several books (http://www.cica.indiana.edu/cica/faq/tcl/tcl.html), online tutorials and manuals (http://www.scriptics.com/man/tcl8.0/contents.htm). For getting further informations on Tcl have a look on the TCL WWW Info (http://www.sco.com/Technology/tcl/Tcl.html) of Tcl Web server (http://www.tcltk.com). 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`. Below only a small subset of the most important Tcl commands like assigning variables, evaluating expressions, control and loop constructs are described. For complete description of Tcl command have a look on the manual pages (http://www.scriptics.com/man/tcl8.0/contents.htm) or on one of the many books about Tcl/Tk.

## 3.2.1. `set` - Read and Write variables

**Synopsis.** `set varName ?value?`

**Description.** Returns the value of variable `varName`. If `value` is specified, then set the value of `varName` to `value`, creating a new variable if one doesn't already exist, and return its value. If `varName` contains an open parenthesis and ends with a close parenthesis, then it refers to an array element: the characters before the first open parenthesis are the name of the array, and the characters between the parentheses are the index within the array. Otherwise `varName` refers to a scalar variable.

## 3.2.2. `expr` - Evaluate an expression

**Synopsis.** `expr arg ?arg arg ...?`

**Description.** Concatenates `arg`'s (adding separator spaces between them), evaluates the result as a Tcl expression, and returns the value. The operators permitted in Tcl expressions are a subset of the operators permitted in C expressions, and they have the same meaning and precedence as the corresponding C operators. Expressions almost always yield numeric results (integer or floating-point values). For example, the expression

```
expr 8.2 + 6
```

evaluates to 14.2. Tcl expressions differ from C expressions in the way that operands are specified. Also, Tcl expressions support non-numeric operands and string comparisons. For some examples of simple expressions, suppose the variable `a` has the value 3 and the variable `b` has the value 6. Then the command on the left side of each of the lines below will produce the value on the right side of the line:

```
expr 3.1 + $a                     6.1
 expr 2 + "$a.$b"                 5.6
 expr 4*[llength "6 2"]           8
 expr {{word one} < "word $a"}    0
```

**Math functions.** Tcl supports the following mathematical functions in expressions:

| | | | |
|---|---|---|---|
| `acos` | `cos` | `hypot` | `sinh` |
| `asin` | `cosh` | `log` | `sqrt` |
| `atan` | `exp` | `log10` | `tan` |
| `atan2` | `floor` | `pow` | `tanh` |
| `ceil` | `fmod` | `sin` | |

Each of these functions invokes the math library function of the same name; see the manual entries for the library functions for details on what they do. Tcl also implements the following functions for conversion between integers and floating-point numbers and the generation of random numbers:

`abs(arg), double(arg), int(arg), rand(), round(arg), srand(arg).`

## 3.2.3. `if` - Execute scripts conditionally

**Synopsis.** `if expr1 ?then? body1 elseif expr2 ?then? body2 elseif ... ?else? ?bodyN?`

**Description.** The `if` command evaluates `expr1` as an expression (in the same way that `expr` evaluates its argument). The value of the expression must be a boolean (a numeric value, where 0 is false and anything is true, or a string value such as "true" or "yes" for true and "false" or "no" for false); if it is true then `body1` is executed by passing it to the Tcl interpreter. Otherwise `expr2` is evaluated as an expression and if it is true then `body2` is executed, and so on. If none of the expressions evaluates to true then `bodyN` is executed. The `then` and `else` arguments are optional "noise words" to make the command easier to read. There may be any number of `elseif` clauses, including zero. `BodyN` may also be omitted as long as `else` is omitted too. The return value from the command is the result of the body script that was executed, or an empty string if none of the expressions was non-zero and there was no `bodyN`.

## 3.2.4. `for` - "For" loop

**Synopsis.** `for start test next body`

**Description.** `For` is a looping command, similar in structure to the C `for` statement. The `start, next,` and `body` arguments must be Tcl command strings, and `test` is an expression string. If a `continue` command is invoked within `body` then any remaining commands in the current execution of `body` are skipped; processing continues by invoking the Tcl interpreter on `next`, then evaluating `test`, and so on. If a `break` command is invoked within `body` or `next`, then the `for` command will return immediately. The operation of `break` and `continue` are similar to the corresponding statements in C. `For` returns an empty string.

```
for {set x 0} {$x<10} {incr x} {
     puts "x is $x"
}
```

### 3.2.5. `while` - Execute script repeatedly as long as a condition is met

**Synopsis.** `while test body`

**Description.** The `while` command evaluates `test` as an expression (in the same way that `expr` evaluates its argument). The value of the expression must be a proper boolean value; if it is a true value then `body` is executed by passing it to the Tcl interpreter. Once `body` has been executed then `test` is evaluated again, and the process repeats until eventually `test` evaluates to a false boolean value. `Continue` commands may be executed inside `body` to terminate the current iteration of the loop, and `break` commands may be executed inside `body` to cause immediate termination of the `while` command. The `while` command always returns an empty string.

```
set x 0
while {$x<10} {
     puts "x is $x"
     incr x
}
```

## 3.3. Instrument settings

### 3.3.1. Beam shutter

Every instrument area in the SINQ neutron guide hall is controlled by the Local Beam Controlsystem (LBC). It uses fixed installed barriers to prevent entry to the area around the active beam during experimental work. If the user carries out his work in accordance with the operating instructions, he will be protected from direct beam radiation. A danger zone which is subject to the LBC interlocking system has two beam shutters

1. the neutron guide main shutter which only can be opended by the radiation safety officer
2. and a secondary shutter to close the beam at the instrument separately from the others if more then one instrument is build up at the same neutron guide.

The secondary shutter for the instrument is the one the user can handle. Normally the secondary shutter can only be operated if the experimental area (yellow fence) of the instrument is locked. The shutter can then be opened and closed by a button on the key box at the entrance door of the instrument area and also by the SICS command `shutter` which has the following syntax:

`shutter`

without parameter yields the actual status of the shutter which can be `shutter is open`, `shutter is closed`, or `Enclosure is broken`. The last status message is returned if the LBC system wouldn't allow to open the beam. If this message is returned the shutter is closed.

`shutter close`

closes the secondary beam shutter.

`shutter open`

opens the secondary beam shutter.

## 3.3.2. Neutron velocity selector

The neutron velocity selector is a high-speed rotor. Blades inserted in the rotor are only transparent for neutrons which manage pass the rotor in a time intervall defined by the rotation speed of the selector. Thus neutrons in a certain speed range (wavelength range) are selected. The wavelength distribution of neutrons is also dependent of the tilt angle between the rotation axis and the neutron beam. Extensive time-of-flight measurements have been done to determine the wavelength $\lambda$ and resolution $\Delta\lambda/\lambda$ as a function of selector speed and tilting angle. The dependency of the wavelength $\lambda$ [nm] on the rotation speed $\nu$ [RPM] can well be described by

$$\lambda(\nu,\xi)= A(\xi)/\nu+B(\xi)$$

where $A(\xi)$ ans $B(\xi)$ are parameters depending on the tilting angle $\xi$. The experimentally determined relationships $A(\xi)$ and $B(\xi)$ are

$$B(\xi)= 0.0122+3.61x10^{-4}\xi+3.14\ x10^{-4}\xi^{2} +3.05x10^{-5}\xi^{3} +9.32x10^{-7}\xi^{4}$$

The wavelength resolution $\Delta\lambda/\lambda$ of the selector should be independent of the rotation speed and only be dependent of the tilting angle $\xi$. However, this is only true for long collimations. For short collimation lengths a slight dependency of the resolution on the rotaion speed could be measured. Also the shape of the resolution function is than not necessarily triangular. If the wavelength resolution of the selector is not so important for the refinement of your data analysis you can use the values given in the following table for the dependency of the wavelength resolution $\Delta\lambda/\lambda$ on the tilting angle $\xi$.

| $\xi$ | $\Delta\lambda/\lambda$ | **A($\xi$)** | **B($\xi$)** |
|---|---|---|---|
| -15 | 0.12 | 19812 | 0.0218 |

| $\xi$ | $\Delta\lambda/\lambda$ | **A($\xi$)** | **B($\xi$)** |
|---|---|---|---|
| -10 | 0.115 | 17774 | 0.0182 |
| -5 | 0.095 | 15493 | 0.0163 |
| 0 | 0.1 | 12716 | 0.01097 |
| 5 | 0.155 | 9342 | 0.0269 |
| 10 | 0.3 | 5293 | 0.0869 |

A high speed-device like a velocity selector has to account for gyroscopic forces when moving the device. In praxis this means that the selector must be stopped before the tilt angle can be changed. Furthermore there are forbidden areas of rotation speeds. In these areas the velocity selector is in destructive resonance with itself. For controlling the neutron velocity selector three command are available: `nvs` , `nvswatch`, and `lambda` which are described below.

`lambda`

The name of the variable alone prints the current wavelength in nm.

`lambda rot <value>`

calculates the rotation speed for the wavelength given by `<value>`.

`lambda wl <value>`

calculates the wavelength for the rotation speed given as parameter `<value>`.

`drive lambda <newval>, run lambda <newval>`

The `lambda` variable can be driven using the normal `drive` and `run` commands.

`nvs status`

Prints a status summary of the velocity selector.

`nvs list`

Displays rotation speed and tilt angle of the velocity selctor.

`nvs [rot=<newval>] [tilt=<newval>]`

This command sets a new tilt angle and/or rotation speed for the velocity selector. Either one or both of the keywords `tilt` or `rot` may be given, followed by a number.

`nvs rotinterest`

Enables printing of status messages about the current state of the selector when it is driven.

`nvs loss`

Starts a loss current measurement on the velocity selector and prints the result.

The commands described so far cover the actual handling of the velocity selector. During a measurement users might want to use further functions such as:

- Monitor the rotation speed of the velocity selector.
- Log the rotation speeds of the velocity selector.
- Initiate error handling when the velocity selector fails to stay within a predefined tolerance of rotation speeds.

Now, these are tasks usually connected with sample environment devices. Now, the SICS programmers have been lazy. Moreover they wanted to avoid duplicating code (and bugs). Consequently, they tricked the velocity selector to be a sample environment device as well.

This means besides the actual velocity selector object (in this case called `nvs`) there exists another object for monitoring the velocity selector. The name of this device is the name of the velocity selector object with the string watch appended. For example if the velocity selector has the SICS name `nvs`, the monitor object will be `nvswatch`. The commands understood by the watch object are fully decribed in the section about sample environment devices. Please note, that all driving commands for the `watch` object have been disabled. Driving can only be achieved through the velocity selector object or the `lambda` command.

## 3.3.3. Positioning an attenuator

`att`

prints the current positioned attenuator.

`att <val>`

positions attenuator `<val>`. Allowed attenuator numbers are 0, 1, 2, 3, 4 and 5.
0 : square 50 mm x 50 mm slit, attenuation = 1
1 : circular 41 x diameter 0.4 mm slit, attenuation = 1/485
2 : circular 9 x diameter 2 mm slit, attenuation = 1/88
3 : circular 20 mm diameter slit, attenuation = 1/8
4 : circular 30 mm diameter slit, attenuation = 1/3.5
5 : circular 15 mm diameter slit, attenuation = 1/??

## 3.3.4. Changing the collimation

`coll`

prints the current collimation length.

`coll <val>`

sets the collimation `<val>`. Allowed collimation lengths are 1, 1.4, 2, 3, 4.5, 6, 8, 11, 15 and 18.

### 3.3.5. Positioning the detector

The detector can be moved via three motors named `detectorx` , `detectory`, and `detectorrotation`. These motors can be driven by the `run` or `drive` commands described in Section 3.1.5. The commands how to change motor parameters like `Precision`, `SoftZero` etc. are described in Section 3.1.6. The axes of the detector motors are defined as:

`detectorx`

An increasing value moves the detector away and a decreasing value towards the sample position.

`detectory`

moves the detector laterally by a maximum of 480 mm in order to increase the accessible q-range at any detector position.

`detectorrotation`

rotates the detector around its vertical axis to reduce parallaxes effects.

> **Warning**
>
> The BerSANS software package for the primary data reduction does not handle the detector rotation.

Instead of driving the motors individually one can refer to all motors as a whole by the `dt` command. The command `dt` without other parameters will yield a listing of the current position of the detector:

```
dt
Status listing for dt
dt.x = 18800.048828
dt.y = 0.004000
dt.phi = 0.414000
```

To move the detector you can call `dt` with parameters defining the new position of the motors, e.g.

```
dt x = 800 y = ++100 phi 0
```

The `=` sign is not mandatory and can be left out. The command above drives the motor `detectorx` to the position `800`, the motor `detectory` to a position 100 mm further into `y`-direction, the motor `detectorrotation` to position `0`. All the three movements are done parallel. Relative movements can be performed by preceding an `--` or `++` to the motor position. The whole set of parameters valid for the `dt` command is described in the Section 3.1.7 about special SANS commands.

### 3.3.6. Positioning the beam stop

The beam stop can be adjusted by two motors named `beamstopx` and `beamstopy`. They can be driven by the `run` or `drive` commands which are described in Section 3.1.5. The commands how to change motor parameters like `Precision`, `SoftZero` etc. are described in Section 3.1.6. The axes of the beam stop motors are defined as:

`beamstopx`

moves the beam stop horizontally.

`beamstopy`

moves the beam stop vertically.

Instead of driving the motors individually one can refer to both motors as a whole by the `bs` command. The command `bs` without other parameters will yield a listing of the current position of the beam stop:

```
bs
Status listing for bs
bs.x = 0.300000
bs.y = 2.500000
```

To move the beam stop you can call `bs` with parameters defining the new position of the motors, e.g.

```
bs x = 2 y ++10
```

The = sign is not mandatory and can be left out. The command above drives the motor `beamstopx` to the position `2` and the motor `beamstopy` to a position 10 mm further into y-direction. Both movements are done parallely. Relative movements can be performed by preceding an `--` or `++` to the motor position. The whole set of parameters valid for the `bs` command is described in the Section 3.1.7 about special SANS commands.

Additionally to the commands for the movement of the two beam stop axes a few other commands have been established:

`bsout`

moves the beam stop out of the detection area, so that there is nowhere a shadow of the beam stop on the detector. This position is outside the software limits of the motors in the area of the beam stop magazines. In this area an uncontrolled movement could lead to a collisioni with the magazines. Therefore one can not move the beam stop anymore with the `bs` command after calling `bsout`, because the motors are fixed automatically.

`bsin`

releases the beam stop motors and moves them back to the previous position. After calling `bsout` you have to call first `bsin` to continue with the movement of the beam stop with the `bs` command.

`bsfree`

is a manager command, which releases the beam stop motors, if they are still in the `bsout`-position. This command should only be used when something went wrong with the SICS server during the time the beam stop was in `bsout`-position.

`bschange [<val>]`

allows the user to change the size of the beam stop. Four different sizes are available and can be selected by the parameter `<val>`. Valid values for `<val>` are:
1 for beam stop size of 40 mm x 40 mm
2 for beam stop size of 70 mm x 70 mm
3 for beam stop size of 85 mm x 85 mm
4 for beam stop size of 100 mm x 100 mm

The `bschange` command automatically recognizes the actually used beam stop size, puts it into the empty magazine and picks up the new beam stop. `bschange` automatically closes the instrument beam shutter, if it was open, but it doesn't reopen it again afterwards. Calling `bschange` without a parameter returns the number of the actually used beam stop size.

### 3.3.7. Calculating the q range

The q range of the current setting can comfortably be calculated with the following command:

`qrange [x <detdist>] [wl <lambda>]`

calculates the q range of the current setting. The detector distance x (in mm) and the wavelength wl (in nm) can be varied with the optional parameters. Examples: `qrange` prints the q range of the current setting, `qrange x 6000 wl 0.8` prints the q range of the current setting with forced values for the detector distance (6000mm) and the wavelength (0.8nm).

### 3.3.8. Estimating the measurement time

The measurement time of a sample depends on many factors and needs to be estimated for each sample individually. This procedure is simplified with the command GuessCount.

`gc <measMonis> <totalCounts>`

GuessCount: executes a measurement of <measMonis> Monis and extrapolates the measurement time and Monis for a final value of <totalCounts> counts. Example: `gc 5 1E7` forces the instrument to measures for 5 Monis and estimates the measurement time and Monis for 10'000'000 counts.

# 3.4. Sample environments

## 3.4.1. Sample table

One standard sample set-up is the sample table with a vertical translator, a xy-table and a rotation table. Optionally, another linear translator stage or a double goniometer can be mounted on the rotation table. The available motor axes for the sample table are defined as follows:

`saz`

vertical translation of the sample table

`say`

horizontal translation parallel to the neutron beam direction

`sax`

horizontal translation perpendicular to the neutron beam

`som`

rotation around the vertical axis $\omega$

`gphi`

rotation around the horizontal axis $\Phi$

`gtheta`

rotation around the horizontal axis $\Theta$, $\Theta \perp \Phi \perp \omega$

`sposi`

horizontal translation. Linear translation stage can be mounted on the rotation table and is used for the movement of the temperature controlled (Haake temperature controler, Section 3.4.4) sample holder.

The motors can be driven by the `drive` or `run` commands described in Section 3.1.6. Instead of driving the motors individually one can refer to all motors as a whole by the `st` command. The command `st` without parameters will yield a listing of the current positions of the sample table motors:

```
st
Status listing for st
```

```
st.omega = 0.504000
st.x = 12.965000
st.y = -18.992001
st.z = 106.121002
st.posi = 173.875000
```

The axes are named `x`, `y`, `z`, `posi`, `omega`, `phi` and `theta` which move the motors `sax`, `say`, `saz`, `spos`, `som`, `gphi` and `gtheta`, respectively. In the above example the optional linear translator stage was mounted on the rotation table so that the position of the motor `spos` is listed but not those of the motors `gphi` and `gtheta`. The whole set of parameters valid for the `st` command are described in Section 3.1.7 about special SANS commands. A frequently used parameter for `st` is the `pos` parameter. The command

```
st pos P1
```

reads out the actual positions of all the motors of the sample table and defines for it the name `P1`. Afterwards this position can be reached by typing simply `st P1`. Instead of remembering the positions of all the motors one only has to remember the name `P1` to bring the sample in position.

### 3.4.2. Bruker electromagnet

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 <Mac-PC> <Mac-port> <Mac-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. `<Mac-PC>` is the name of the Macintosh PC to which the controller has been connected, `<Mac-Port>` is the port number at which the Macintosh-PC's serial port server listens. `<Mac-channel>` is the RS-232 channel to which the controller has been connected. For example (at SANS):

```
evfactory new magnet bruker lnsa10.psi.ch 4000 9
```

creates a new command magnet for a Bruker magnet Controller connected to serial port 9 at lnsa10.

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`.

## 3.4.3. Sample holder for electro magnet

Another standard sample seup is a vacuum chamber, which is directly connected to the collimator and detector tubes, so that the SANS can be operated at about $10^{-2}$ mbar in a single vacuum without windows or with thin aluminium or sapphire windows to work at ambient pressure or at vacuum conditions down to $10^{-6}$ mbar. The chamber is large enough to carry an electromagnet. For this setup a sample changer with an optional heated sample position is available. This sample changer can be moved vertically by 245 mm and can also be rotated by $\pm 10$ degree. The two available motors are defined as follows:

`mz`

moves the electromagnet sample holder in vertical direction.

`mom`

rotates the sample around the vertical axis $\omega$ by $\pm 10$ degree.

The motors can be driven by the `run` or `drive` command described in Section 3.1.6. Instead of driving the motors individually one can refer to both motors as a whole by the `msh` command. The command `msh` without parameters will yield a listing of the current position of the electromagnet sample holder:

```
msh
Status listing for msh
msh.z = 0.000000
msh.omega = 0.000000
```

The axes of the sample holder `mz` and `mom` are named in the `msh` command `z` and `omega`. The whole set of parameters valid for the `msh` command are described in Section 3.1.7 about special SANS commands.

## 3.4.4. Haake C25P temperature controller

This is sort of a bucket full of water equipped with a temperature control system. It is connected with a blue ethernet cable to the port nr. 8 of the SANS terminal. The SEA software written by Markus Zolliker is the best way to control and monitor the thermostat manually; it is started from a terminal window with `sea`. Please refer to the SEA Wiki page (http://lns00.psi.ch/sinqwiki/Wiki.jsp?page=Sea) for more information.

To add an external Pt100 sensor, plug in the sensor and restart the thermostat and the sea client (choose 'Haake' as device and check the box 'has sample sensor'). If the external sensor should be used as reference to control the thermostat, check the box 'control on sample'.

`start_sea`

starts the sea server from SICS.

`temperature`

reads out the current temperature.

`drive temperature <val>`

changes the set temperature of the thermostat to the value `<val>`.

in a terminal window: `sea`

starts the sea server client from a terminal. Choose Haake as device.

## 3.4.5. Eurotherm temperature controller

At SANS there is a Eurotherm temperature controller for the sample heater available. 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. `computer` is the name of the Macintosh PC to which the controller has been connected, `<port>` is the port number at which the Macintosh-PC's 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 Macintosh port 13 (the last in the upper serial port connection box) is configured like this! Thus, an example for SANS and the name temperature looks like:
>
> ```
> EVFactory new temperature euro lnsa10.psi.ch 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 controller.
- The weird protocol spoken by the Eurotherm requires very special control characters. Therefore the send functionality usually supported by a SICS environment controller could not be implemented.

# 3.5. Data handling and acquisition

## 3.5.1. File naming conventions and storing data

Data files are stored by the SICS server on the *lnsa10.psi.ch* workstation in the directory defined by the SICS variable `SICSDataPath`. By default this directory is `/home/SANS/data/`. The file name of a data file is composed of four parts:

1. the prefix stored in the variable `SICSDataPrefix`, by default `sans`
2. the run number stored in the variable `SICSDataNumber`, which is incremented before each storing process and has 5 digits (leading 0)
3. the actual year (4 digits)
4. and the post-fix stored in the variable `SICSDataPostfix`, by default `.hdf`

A typical data file name would be `/home/SANS/data/sans123452006.hdf`. All data files are written in NeXus format.

## 3.5.2. Data acquisition

**SICS counter handling.** The SICS counter concept may include several monitors per counter. At the SANS instrument two monitors are installed: one between beam shutter and neutron velocity selector, which is used for normalizing the SANS measurement on the incident neutron flux, and a second one after the selector just in front of the attenuator. For the SANS instrument only one counter is handled which is named `counter`. A few words have to be lost about the SICS handling of preset values for counters. Two modes of operation have to be distinguished: counting until a timer has passed, for example counting for 20 seconds. This mode is called `Timer` mode. In the other mode, counting is continued until a control monitor has reached a certain preset value. This mode is called `Monitor` mode. At the SANS instrument the first monitor between beam shutter and neutron selector is used in this 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. The commands understood are:

`counter SetPreset <val>`

sets the counting preset to `<val>`.

`counter GetPreset`

prints the current preset value.

`counter SetExponent <val>`

sets the exponent for the counting preset in monitor mode to `<val>`.

`counter GetExponent`

prints the current exponent used in monitor mode.

`counter 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.

`counter GetMode`

prints the current mode.

`counter GetCounts`

prints the counts gathered in the last run.

`counter GetMonitor <n>`

prints the counts gathered in the monitor number `<n>` in the last run.

`counter Count <preset>`

starts counting in the current mode and the preset `<preset>`.

`counter 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..

`counter GetTime`

retrieves the actual time the counter counted for. This excludes time where there was no beam or counting was paused.

**Histogram memory.** The histogram memory is used in order to control the large area sensitive detector. It takes care of putting counts detected in the detector into the proper bin in memory. Next to a conventional mode of a SANS measurement where all detected neutrons are accumulated for a given time or monitor count, also a time of flight mode and a stroboscopic mode are available, where there is for each detector pixel a row of memory locations mapping the time bins. As usual in SICS the syntax is the name of the histogram memory followed by qualifiers and parameters. For the SANS the name of the histogram memory is `banana`.

The histogram memory has a plethora of configuration options coming with it which define memory layout, modes of operation, handling of bin overflow and the like. Additionally there are histogram memory model specific parameters which are needed internally in order to communicate with the histogram memory. In most cases the histogram memory will already have been configured at SICS server startup time. However, there are occasion where these configuration options need to be enquired or modified at run time. The command to enquire the current value of a configuration option is: `banana configure <option>`, the command to set it is: `banana configure <option> <newvalue>`. A list of common configuration options and their meaning is given below:

`HistMode`

describes the modes of operation of the histogram memory. Possible values are:

`Transparent`

counter data will be written as is to memory. For debugging purposes only.

`Normal`

neutrons detected at a given detector will be added to the apropriate memory bin.

`TOF`

time of flight mode, neutrons found in a given detector will be put added to a memory location determined by the detector and the time stamp.

`Stroboscopic`

This mode serves to analyse changes in a sample due to an varying external force, such as a magnetic field, mechanical stress or the like. Neutrons will be stored in memory according to detector position and phase of the external force.

`OverFlowMode`

This parameter determines how bin overflow is handled. This happens when more neutrons get detected for a particular memory location then are allowed for the number type of the histogram memory bin. Possible values are:

`Ignore`

overflow will be ignored, the memory location will wrap around and start at 0 again.

`Ceil`

the memory location will be kept at the highest posssible value for its number type.

`Count`

as `Ceil`, but a list of overflowed bins will be maintained.

`Rank`

defines the number of histograms in memory.

`Length`

gives the length of an individual histogram.

`BinWidth`

determines the size of a single bin in histogram memory in bytes.

For time of flight mode the time binnings can be retrieved and modified with the following commands. Note that these commands do not follow the configure syntax given above. Please note, that the usage of the commands for modifying time bins is restricted to instrument managers.

`banana timebin`

prints the currently active time binning array.

`banana genbin <start> <step> <n>`

generates a new equally spaced time binning array. Number `<n>` time bins will be generated starting from `<start>` with a stepwidth of `<step>`.

`banana setbin <inum> <value>`

Sometimes unequally spaced time binnings are needed. These can be configured with this command. The time bin `<iNum>` is set to the value `<value>`.

`banana clearbin`

Deletes the currently active time binning information.

`banana preset [<val>]`

with a new value `<val>` sets the preset time or monitor for counting. Without `<val>` prints the current value.

`banana exponent [<val>]`

with a new value `<val>` sets the exponent to use for the preset time in Monitor mode. Without `<val>` prints the current value.

`banana CountMode [<mode>]`

with a new value `<mode>` sets the count mode. Possible values are `Timer` for a fixed counting time and `Monitor` for a fixed monitor count which has to be reached before counting finishes. Without a value for `<mode>` the command prints the currently active value.

`banana init`

after giving the configuration commands this needs to be called in order to transfer the configuration from the host computer to the actual histogram memory.

`banana count`

starts counting using the currently active values for `CountMode` and `preset`. This command does not block, i.e. in order to inhibit further commands from the console, you have to give `Success` afterwards.

`banana Initval <val>`

initialises the whole histogram memory to the value `<val>`. Usually 0 in order to clear the histogram memory.

`banana get <i> <iStart> <iEnd>`

retrieves the histogram number `<i>`. A value of -1 for `<i>` denotes retrieval of the whole histogram memory. `<iStart>` and `<iEnd>` are optional and allow to retrieve a subset of a histogram between `<iStart>` and `<iEnd>`.

`banana sum <d0min> <d0max> <d1min> <d1max> ...<dnmin> <dnmax>`

calculates the sum of an area on the detector. For each dimension a minimum and maximum boundary for summing must be given.

**Storing Data and starting a SANS measurement.** Instead of initializing and starting a measurement by the `banana` command a few other commands have been introduced to take care for those things:

`StoreData`

This command does what it says. It writes the current state of the instrument including counts to a NeXus data file.

`count [<mode> <preset>]`

starts a count operation in mode `<mode>` with a preset `<preset>`. `<mode>` can have the values `Timer` or `Monitor`. The parameters are optional. If they are not given, the count operation will be started with the current setting in the histogram memory object `banana`. Before the count operation is started, the `count` command waits until all other commands executed earlier are finished. During the count operation no other commands can be executed. After the count, `StoreData` will be automatically called.

`repeat <num> [<mode> <preset>]`

calls `num` times `count`. `num` is a required parameter. The other two are optional and are handled as described above for `count`.

## 3.5.3. XY table

XYTable is a class which maintains a list of X-Y values. These can be plotted in the VarWatch SICS client by configuring it with a special command. Before you may use an XYTable object it had to be installed into the system by the SICS administrator. This can be done by the command `MakeXYTable <xydata>` in the SANS initialization file. For this documentation it is assumed that this has happened already and a XYTable object is available in the system under the name `<xydata>`.

Interaction with the XYTable object happens through the following commands:

`<xydata> clear`

clears all entries in the x-y table.

`<xydata> list`

lists the entries in the x-y table on the screen.

`<xydata> write <filename>`

writes the x-y list to the disk file filename. This file resides on the machine running the SICS server.

`<xydata> uuget`

sends the x-y list in an uuencoded format. This is the command to give to the VarWatch SICS client in order to make it display the x-y list.

`<xydata> add <xval> <yval>`

creates a new x-y list entry with the values `<xval>` and `<yval>`.

## 3.5.4. Status of the actual acquisition process

A webpage (http://lns00.psi.ch/sicszone/sansstatus?instrument=sans) with the actual instrument status is available (only inside the PSI network).

`sinq`

prints the SINQ status in the SICS client which issued the command.

# Chapter 4. Other programs

## 4.1. Grasp

Grasp stands for 'Graphical Reduction and Analysis SANS Program for Matlab' which was developed by Charles Dewhurst at the Institut Laue-Langevin (ILL) in Grenoble, France. Please refer to the Grasp webpage (http://www.ill.fr/lss/grasp/grasp_main.html).

## 4.2. BerSANS software package

The BerSANS data reduction software has been developed by Uwe Keiderling at the Hahn-Meitner-Institut (HMI) in Berlin, Germany. The Manual of the BerSANS Software Package can be downloaded (http://kur.web.psi.ch/sans1/BerSANS/SANS-Manual.pdf) as pdf file.

## 4.3. SASfit program

SASfit for analyzing and plotting small angle scattering data has been written by Joachim Kohlbrecher at the Paul Scherrer Institute (PSI) in Villigen, Switzerland. Please refer to the SASfit webpage (http://kur.web.psi.ch/sans1/SANSSoft/sasfit.html).