SICS–Managers–Manual

Version: October 19, 2005

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

Contents

Chapter 1

Introduction

This document describes the general setup of SICS, the configuration of the SICS server and its clients and the writing of macros in the SICS macro language. This document is aimed at SICS system managers and instrument scientists. This document requires that the SICS user documentation has been read and understood beforehand. For writing macros it is essential to understand John Ousterhouts Tool Command Language, which is described elsewhere.

Chapter 2

SICS programs, Scripts and Prerequisites

2.1 Hardware

The following hardware is usually present for any SICs instrument:

- An instrument computer
- A Lantronix terminal server with 8-16 serial ports for connecting:
- Motor controllers
- Counter boxes
- Temperature controllers.
- Optionally 1-n histogram memory computers are present.

The terminal server software is provided by Lantronix, see the appropriate manuals for the device for a description. The histogram memories are 68000 VME or MEN-A12 onboard computers running the VXworks realtime operating system and some home grown histogramming software documented elsewhere.

2.2 Server programs

For proper operation of an instrument the following software components are required:

SerPortServer This is a TCP/IP server which implements a special protocoll for communicating with serial ports. The actual communication with the serial ports is done through the Lantronix terminal server. Both the serial port protocoll and the SerPortServer are only documented in the source code. The SerPortServer program is on its way out.

TecsServer This is a TCP/IP server which watches over temperature controllers. The only known source of documentation about this software is Markus Zolliker.

FileSync This is a little UDP server which waits for UDP messages from the instrument control program. Then the server starts a script which synchronizes the local data directory with the central data storage on the labarotory server. FileSync is configured through an initialization file usually called fs.ini. See the comments in the initialization file for more information.

SICServer This is the actual instrument control server. The configuration of this program is documented in this manual.

Additionally a client program is needed which connects to the instrument control server and provides a user interface to it.

Chapter 3

General SICS Setup

SICS is a client server system. This implies that there is a server program which implements all the functionlity and client programs which implement the user interface. The client program is the only thing the user is intended to see. This also means that the location of the client programs is quite independent from the computer where the server runs. In the following the layout of a server installation is described as established at SINQ.

For each instrument there is a data aquisition computer. On this computer there exists a user name and consequently a home directory for the instrument. In the following text this instrument root directory will be called sicsroot. This root directory has the following subdirectories:

bin The directory where some executables live.

inst_sics All instrument configuration files and the SICServer live in this directory. Replace inst by the name of the instrument as appropriate.

data The data directory is the central place where all data files collected at the instrument are stored. This directory contains subdirectories for each year. These directories hold the data collected at the instrument in this year plus a file named DataNumber which keeps the current serial number of the data files. This file should never be edited. At the start of each year the instruement manager must create a new directory with an empty DataNumber file. Additionally the path variables both for the data file directory and the DataNumber file have to be set to the new directory in the instrument initialization file.

log The log directory contains the server log files and the automatically generated client log files. Any now and then, and especially when disk space problems loom, the client*.log files should be deleted by the instrument manager.

lib Contains some files needed for running SICS clients locally.

doc This directory holds a copy of the SICS user documentation for the instrument. These are html files which can be viewed with WWW-browsers such as lynx or netscape.

motor This directory holds a script for reading and restoring the motor parameter from the EL734 motor controllers. And the motor parameters stored in parameter files.

It is the instrument scientists responsability to save the motor parameters after changes to the configuration of the instrument.

tmp A directory for temporary and PID files.

help A directory for help files for the help system.

Besides these directories there should be nothing on the instrument account. All evaluated data, personal command files etc. should be held on the normal user account of the instrument user.

3.1 System Control

All commands listed in this section have to issued with the privilege of the instrument user account.

In order to watch over all these servers, the monit[1] software is used. This is a program which watches server processes and restarts them as necessary. Monit can also be used to watch over hardware and the file system. Monit writes a log file and can trigger e-mail alerts on certain problematic conditions.

Monit itself is controlled through a configuration file, .monitrc, which lives in the instrument accounts home directory. Monit also uses another Tcl program runwithpid which is responsible for starting and stopping server programs. In order to configure all this, there is another program: makemonit which creates the monit configuration file and copies and edits the runwithpid program. The syntax of makemonit is:

makemonit instrument terminalserver data-mount-point hm1 hm2 .. instrument is the instrument name in lowercase, terminalserver is the name of the terminal server used for serial port access, data-mount-point is the name of file system onto which the instruments data is written. This is followed by a list of all the histogram memory computers used by the instrument.

monitinit monitinit is an alias which calls makemonit with all required parameters for a given instrument in standard configuration.

Monit itself can be controlled with the following commands:

monit Starts the monit daemon and all configured processes. This is only necessary after a reboot of the system.

monit status This shows a status message listing the status of all configured servers and the checked hardware. The monit status is also available on the WWW from http://lns00.psi.ch/monit/instrument. Replace instrument with the appropriate instrument name.

monit stop target Stops the server process target. The following targest exist:

all All processes

sicsserver The SICS server

[1]See URL http://www.tildeslash.com/monit

SerPortServer The serial port server.

sync The file synchronisation server.

tecs The TecsServer for controlling environment devices.

simserver A simulation server(not on all instruments).

monit start target Starts a server process, targest are the same as described above.

monit restart target Stops and starts the process target. Targets are as listed above.

monit quit Stops monit alltogether. Please note, that servers stay running with this command. In order to sop everything the sequence: monit stop all; monit quit is required.

startsics This script starts all the necessary server programs for driving the instrument through monit.

killsics This script shuts down all instrument control servers properly through monit.

instsync replace inst by the name of the instrument in lower case. This script is invoked by the FileSync server and is responsible for synchronizing the local data store with the one on the labaratory server. This is usally done by calling the unix program rsync with appropriate parameters.

3.2 Moving SICS

3.2.1 Moving the SICS Server to a new Computer

This requires the following steps:

1. Create a new local account on the host computer. There is a prefabricated account with the credentials: instbck/INSTBCKLNS on lnsl15.
2. Create the directory structure.
3. Create and edit a suitable DataNumber file for the instrument and put it into data/YYYY. YYYY is the year.
4. Edit the instrument configuration files and adapt the path names to match the new configuration.
5. Try to start and debug.

3.2.2 Exchanging the Serial Port Server

1. Fetch a new one and make sure that all cables are plugged as they were in the old one.
2. Create a new .monitrc file by running makemonit.
3. Exchange all references to the old terminal server in the instrument configuration files to the new terminal server.
4. Done!

3.2.3 Exchanging the Histogram Memory

1. Get a new histogram memory computer from either Gerd Theidel, the test setup in the electronics barrack.

2. Put into the rack.

3. Configure the HM boot parameters through the console connected to the serial port belonging to the HM. Instructions for this can be found in the histogram memory documentation. Up to date boot configuration parameters should be available under /afs/psi.ch/group/sinqhm/boot/INST where INST is a place holder for the instrument in upper case.

4. Adapt the instrument configuration file to reflect the new name of the HM.

5. Start and debug.

3.3 SICS Trouble Shooting

3.3.1 Check Server Status

One of the first things to do is to check the server status with: monit status.

3.3.2 Inspecting Log Files

Suppose something went wrong over the weekend or during the night and you are not absolutely sure what the problem was. In such a case it is helpful to look at the SICS log files. They live in the log directory of the instrument account. For each day (or after each restart of the SICS server) a new log file is created. They are named according to the following convention:

```
autoYYYY-mm-dd@hh-MM-ss.log
```

with YYYY denoting the year, mm the month, dd the day, hh the hour of creation, MM the minute of creation and ss the seconds of creation. The most recent log file can be looked at with the **sicstail** command. **sicstail num** shows the last num lines of the log file. Within SICS and especially in the SICS command line client, the last 1000 lines of the log are accessible through the **commandlog tail num** command. The command log is also accessible through the WWW at lns00. The log file is equipped with hourly time stamps which allow to find out when exactly a problem began to appear.

There is also another log file, log/monit.log, which logs messages from the monit daemon. This can be used to determine when server processes were restarted or when hardware failed.

Quite often the inspection of the log files will indicate problems which are not software related such as:

- Communication problems (usually network)

- Positioning problems of motors.

- BAD_EMERG_STOP: the motor emergency stop was engaged. It must be released before the motors move again.
- BAD_STP: a motor had been switched off.

3.3.3 Restarting SICS

monit restart sicsserver

3.3.4 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 a nautical mile.

- 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.
- Restart all of SICS with the sequence: monit stop all; monit quit; monit
- Wait for a couple of minutes for the system to come up.

3.3.5 Starting SICS Manually

In order to find out if some hardware is broken or if the SICS server initializes badly it is useful to look at the SICS servers startup messages. The following steps are required:

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

Replace inst by the name of the instrument, as usual. Look at the screen output in order to find out why SICS does not initialize things or where the initialization hangs. Do not forget to kill the SICServer thus started when you are done and to issue the command: **monit start sicsserver** in order to place the SICS server back under monits control again.

3.3.6 Test the SerPortServer Program

Sometimes the SerPortServer program hangs and inhibits the communication with the RS-232 hardware. This can be diagnosed by the following procedure: Find out at which port either a EL734 motor controller or a E737 counter box lives. Then type:**asyncom localhost 4000 portnumber** This yields a new prompt at which you type **ID**. If all is well a string identifying the device will be printed. If not a large stack dump will come up. The asyncom program can be exited by typing **quit**. If there is a problem with the SerPortServer program type: **monit restart SerPortServer** in order to restart it.

3.3.7 Trouble with Environment Devices

The first stop for trouble with temperature or other environment devices is Markus Zolliker. A common problem is that old environment controllers have not be deconfigured from the system and still reserve terminal server ports. Thus take care to deconfigure your old devices when swapping.

3.3.8 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. 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 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 4

The SICS Initialization File

4.1 Overview of SICS Initialization

The command to start the SICServer is: **SICServer inifile** . So, what happens at the initialization of the SICS server? First, internally, a set of standard SICS commands is initialized, than a set of special initialization commands. These are special commands which help to configure the SICS server to match the actual hardware present on the hall floor and to define the commands available later on. Following this, the SICS server will read the initialization file specified on the command line or servo.tcl as a default. Using the data supplied in this file, the server will be configured. At this stage all special initialization commands, all Tcl-mechanisms and all SICS commands already initialized are available. After this is done, the server will delete the initialisation commands from its internal command list (No need to carry them around all the time). Now a status backup file will be read. This file contains normal SICS statements which initialise parameter values to the values they had before the last shutdown of the server. Such a file is automatically written whenever a normal shutdown of the server happens or variables change.

The SICS server configuration file is essentially a SICS macro language file. This implies that all general SICS commands and Tcl mechanisms are available. Additionally the configuration file (and only the configuration file) may use special commands for the installation of:

- SICS server options.
- SICS variables.
- Special commands.
- Hardware

Actually the SICS servers configuration is rarely stored in one file but in several files which are included by the main configuration file. In general the following files are present:

inst.tcl Replace inst with the name of the instrument in lowercase. This is the main initialization file. It should contain all the hardware initialization.

instcom.tcl Again replace inst with name of the instrument in lowercase. This file holds instrument specific commands defined in the Tcl macro language. This file is automatically included by inst.tcl.

scancommand.tcl, tecs.tcl, log.tcl Some macro definitions are used by so many instruments that it was deemed appropriate to hold them in separate files. Such files are included from instcom.tcl.

4.2 SICS Options and Users

The SICS server has an internal options database which holds such values as port number to listen too and the like. Additionally there exists a user database. Currently these values are configured from the initialisation file. The commands to do this are:

- **ServerOption name value** defines the server option name to be value.
- **SicsUser name password accesscode** defines a user with name and password and an access right accesscode. Accesscode is an integer from 1 to 3 for User, Manager and Spy rights. A user with Spy right may look at everything but cannot change anything. A user with user privilege may change most of the SICS parameters, perform measurements etc. A user with manager privilege may do everything to the system.

The Sics server currently uses the following options:

- **ReadTimeOut** The server checks in each cycle of the main loop fro pending commands. Het waits for new commands for a specific period of time. This value is the ReadTimeOut. It should be as short as possible. This value may need to be increased if there are network performance problems. A good value is 100.
- **ReadUserPasswdTimeout** This is the time a client has to send a username password pair after a connection request. If nothing comes in in that time, the connection request is terminated.
- **LogFileBaseName** defines the path and the base name for the server log file.
- **ServerPort** defines the port number the server is listening to. Should be greater than 1024 and less than 64000. The port number should also be different from any other server port number in use on the system. Otherwise evil things may happen.
- **InterruptPort** The SICS server can handle interrupts coming in as UDP messages on a special port. This option defines this UDP port number. The choice of possible port numbers is limited by the constraints given above in the ServerPort section. Espacillay this port number MUST be different from the ServerPort number. The UDP messages accepted are expected to consist of the string SICSINT followed by an interrupt number. For interrupt numbers see file interrupt.h.
- **DefaultTclDirectory** specifies where Tcl defined commands are stored. When this is properly defined Tcl will autoload commands.

- **statusfile** defines the file to which he current state will be saved on close down of the server and restored from at startup time.
- **TelnetPort** The port number where the SICS server will be listening for telnet connections. If this option is missing login via telnet to the SICS server is disabled.
- **TelWord** In order to login to SICS via telnet a magic word is needed. The login word, This option defines this word. If this option is missing telnet login to SICS is disabled.
- **LogFileDir** This server option defines the directory where commandlog log files are kept.
- **RedirectFile** This defines a filename to which all output to stdout and stderr is logged by the SICS server. This is mainly a debugging feature.
- **TecsPort** The port number at which the Tecs temperature control server is listening.

4.3 SICS Variables

In SICS most parameters necessary for instrument control are kept with the SICS object implementing the functionality. However, there are some odd bits of information which need to be known, mostly additional information about and from the user, which should go into data files. Such information is kept in SICS variables, created with the command VarMake detailed below.

Variables are also used in order to control the SINQ automatic file name creation scheme for data files. At SINQ, data files are put into a special directory. The actual name consists of a prefix, a sequential number, the last two digits of the year and an ending. All these components are controlled by variables.

The exact syntax for creating variables looks like this:

- **VarMake name type access** creates a simple variable name. The variable type can be Text, Int or Float. The access parameter defines who may may change the variable. Possible values are: Internal, Manager, User and Spy.

There are quite a few SICS variables which have special usages and should be present. Some of them may need to be set by the user, all of them should be inspectable by the user. Many of these variables are used when writing NeXus data files. These special variables are:

sample The sample name. To be set by the user.

title. The experiment title.

User The name of the user to be specified in a data file.

Instrument. The name of the instrument.

SicsDataPath The full path name of the instruments data directory. Should have a / at the end.

SicsDataPrefix The prefix to use for data file names.

SicsDataPostFix The ending to use for the data file name. Including the '.'.

Adress The users adress.

phone The users phone number.

email The users e-mail adress.

fax The users fax number.

Some of the variables stated above should never be changed by a user. This can be achieved by the variable lock command described in the user documentation.

4.4 SICS Hardware Configuration

Hardware is configured into the SICS system by executing special hardware configuration commands from the server initialisation file. These commands are described here. Much SICS hardware is hooked up to the system via RS-232 interfaces. The SICS server communicates with such devices through a serial port server program running on the instrument computer. All such devices require on initialisation the following parameters:

- **hostname** The name of the instrument computer.
- **port** The port number where the serial port server program is listening for requests. It is usually 4000.
- **channel** The number of the RS-232 interface port on the terminal server.

4.4.1 Bus Access

SICS and its internals cover many common usage cases. However there are always situations where SICS has to be bypassed and commands to be sent to a controller directly. This can happen in order to satisfy a special request or as first step in the integration of a special piece of hardware into SICS. In order to do such things the following facilities are available:

Direct Access to RS232 Controllers or TCP/IP Controllers.

Both controllers listening on a TCP/IP port or RS-232 devices connected to a terminal server can be directly accessed through the RS232 commands. The first step in using this system is always to accounce the controller to SICS using the command:

```
MakeRS232Controller name terminalserver port
```

in the SICS initialization file. For example:

```
MakeRS232Controller hugo psts213 3004
```

name is the SICS name for the controller, terminalserver is the name of the terminal server the device is connected to and port is the port number at which the terminal server publishes the RS232 channel to which the device is connected. This is usally the port number plus 3000.

Now various commands are available for interfacing with the RS232 controller. In the following description the SICS name of the controller is replaced by the symbol rs232name.

rs232name sendterminator prints the current terminator used when sending data to the device as hexadecimal numbers.

rs232name sendterminator h1h2..hn sets the current terminator used when sending data to the device to the characters described by the hexadecimal numbers h1 to hn. The numbers are in the format 0xval, where val is the hex number.

rs232name replyterminator prints the current terminator expected to terminate a response from the device as a hexadecimal number.

rs232name replyterminator h1h2..hn sets the current terminator expected to terminate a response from the device to the characters described by the hexadecimal numbers h1 to hn. The numbers are in the format 0xval, where val is the hex number.

rs232name timeout prints the current timeout when waiting for a reponse from the device.

rs232name timeout val sets the timeout for waiting for responses from the device. The value is in microseconds.

rs232name send data data data sends the remainder of the line to the RS232 device and waits for a response terminated with the proper reply terminator specified. This commands waits at maximum timeout microseconds for a response. If a valid response is obtained it is printed, otherwise an error message occurs.

rs232name write data data data writes the remainder of the line after write to the device without waiting for a response.

rs232 available checks if data is pending to be read from the device.

rs232 read reads data from the device.

Accessing Serial Ports (Old System)

In addition to the system describe above there is another system for accessing serial ports which uses the SerPortServer program. The use of the system is deprecated, new software should use the commands describe above. Nevertheless the older sytem is described here for reference.

Serial port access is implemented as an extension to the Tcl macro language. Essentially this is the same implementation as used in the program psish. This section describes how to use serial port access. Several steps have to be performed:

1. Install the serialport command into the SICS server. This requires two lines to be added to the server startup script:

   - SerialInit
   - TclPublish serialport UserRights

   Where UserRights stands for one of the possible SICS user rights. See documentation for TclPublish above.

2. Each separate serial port will be represented by a name in the SICS server after it has been initialized. This name is also a command. These port names live in the Tcl interpreter and must be made accessible with TclPublish. For example for a port named p1 include this line in the server startup script:

   - TclPublish p1 User

   Replace User with the correct access code you want for a serial port. It is recommended to restrict serial port access to SICS managers only.

3. After starting the SICS server the command serialport is now available.

4. Now a serial port can be initialized with a command like this:

   - serialport name1 SerPortServer.host port channel force
   - Example: serialport p1 localhost 4000 5

   Such a command creates the command name1 and links it with serial port channel channel on the instrument computer (localhost) running the SerPortServer program . Port is the port number on which the SerPortServer is listening for connections (usually 4000). The last flag force is optional. If something is there, the connection to that port is done on a separate socket of its own. This has to do with some feature of the software interface to the SerPortServer serial port server. This software interface tries to direct messages for multiple channels through one socket connection between the host and the Macintosh server. This is perfectly fine as long as none of the commands through this socket takes a long time to execute. However, if a RS-232 device takes a long time to respond, the whole socket is blocked. Fortunately, the serial port server runs a separate thread of execution for each different socket. By forcing a new socket it can be achieved that such a slow device is decoupled from the rest. Exactly this is achieved with the force flag.

5. Once the port has been initialised (for example p1) it is ready to operate. The port object allows to send data to the serial port and receive data from it. Furthermore some configuration is possible. The syntax is like this:

   portname -tmo number Sets the timeout for the serial port. This is the maximum amount of time the serial port server waits for data to arrive from the RS-232 device. Increase this if a lot of `_BAD_TMO` error messages creep up.

portname -sendterm string Sets the terminator which will be automatically added to the string which is sent. Some RS-232 devices require special terminators in order to accept a command. The serial port implementation ensures that such a terminator is sent after each message. This command allows to configure this terminator. Please note, that the terminator string needs to be enclosed in quotes. An example:

- `p1 -sendterm "\r\n"`

This sets the terminator to carriage return - line feed.

portname -replyterm string. The serial port server expects the RS-232 device to send a terminator when it is done with sending answers. It even supports multiple lines to be sent as a reply. This expected reply terminator is set with this command. The string may may be four characters long. An example: `1\r\n` sets the expected terminator to one of `\r\n`. One of them is expected. Thus the first character is the count of terminators to expect, followed by the characters possible as terminators. This string must usually be quoted.

portname blablalakjdl When none of the options -tmo, -replyterm, -sendterm, is found everything after portname is sent to the RS-232 device. The reply from the RS-232 device is printed.

The defaults set for the configuration parameters of the serial port connection are suited for the EL734, EL737 and ITC4 devices usually encountered at SINQ. For other RS-232 devices consult the manuals hopefully delivered with the device. The defaults are: 100 for timeout, `1\r\n` for the reply terminator and `\r\n`for the send terminator.

GPIB Controller Access

GPIB is yet another bus system. Up to 30 devices can share the bus and transfer data on it. SICS likest to speak to GPIB devices through the National Instrument ENET-100 TCP/IP bridge. In order for this to work the National Instruments driver software must have been installed on the computer running SICS. SICS has to be compiled with the define HAVENI defined and the proper paths to the header file and library configured. Then an GPIB controller can be installed into SICS with the command:

```
MakeGPIB name drivertype
```

Name is the name under which the GPIB controller is addressable within SICS afterwards. drivertype is the driver to use for the GPIB device. Supported values are:

sim Simulation

ni National instruments driver, see above.

The GPIB controller supports a couple of commands for communicating with devices on the GPIB bus directly. Use with extra care because it is very easy to lock things up on the GPIB bus. In the following documentation of the command set it is assumed that a GPIB controller has been configured into the system under the name **gpib**. Please note, that managers privilege is required in order to be allowed to wrestle with this controller.

gpib attach controller-no gpib-address gpib-secondary timeout eos eot This attaches the GPIB controller to a certain device at a certain address for later communication. The return value is an integer handle which will be used later on a s a handle devID when referring to the connection. The parameters are:

controller-no The number of the GPIB controller on the computer. There may be more then one GPIB controllerinstalled on a given system. Usually this is 0.

gpib-address The GPIB address of the device on the bus.

gpib-secondary GPIB devices may have a seconadry address. This can be specified with this parameter. Usually this is 0.

timeout The time to wait for answers on the GPIB bus. 13 is 10 seconds and ussually a good value.

eot A parameter determining the termination mode on this connection. Consult NI documentation for this or leave at 0.

eoi A terminator. Set to 1 or understand NI documentation for this parameter.

gpib detach devID Breaks the connection described through devID. devID is the return value from attach.

gpib clear devID Tries to clear the GPIB buffers for the conenction described through devID. Usually in vain.

gpib send devID bal bla bla sends data to the device at devID.

gpib sendwithterm devID string terminator Sends string to the device at devID. The terminator character identified through the integer terminator is automatically appended. Use this to send things which require a terminator. Terminators included in strings sent by send get messed up through Tcl!

gpib read devID Reads data from the device at devID and returns it as a string.

gpib readtillterm devID terminator Read from teh device devID unti the terminator character described through the interger terminator is read. Then return the data read as a string.

4.4.2 Controllers

For the same reason as stated above, it is necessary to represent controllers within SICS. Controllers implement more then only device access but also maintain common state or implement special behaviour.

ECB Controllers

ECB controllers are at the heart of the Risoe data aquisition system. These are essentially Z80 processors wired to the GPIB bus. Functions can be invoked in this processor by sending a function code followed by the contents of 4 8 bit registers. As a result the contents of the registers after the function call are returned. A ECB can be made knwon to SICS through the initialisation command:

```
MakeECB name gpib-controller gbib-controller-number gpib-address
```

The parameters:

name The name used as a token for this controller later on.

gpib-controller the name of the GPIB interface to use. See above.

gbib-controller-no The number of the GPIB board in the system

gpib-address The GPIB address of the ECB on the GPIB bus.

Once installed, the ECB controller understands a few commands:

ecb1 func funcode d e bc Invoke ECB function funcode with the registers d e b c.Returns the contents of the registers d e b c. Function codes and register contents are documented, if at all, in the ECB documentation. In fact, as ECB documentation is not available, the only documentation on ECB is the source code of tascom.

ecb1 clear Tries, usually in vain, to clear the communications interface to the ECB.

ecb1 toint char A helper function which converts the character char to an integer. Tcl does not seem to be able to do that.

Siematic SPS Controllers

Siematic SPS controllers are used at SinQ for handling all the things which fit nowhere else. Such as operating air cushions on some instruments, reading variables from ADC's, reading status of shutters or other parts of the instrument and the like. Those SPS units have an RS-232 connector and understand a simple ASCII command protocoll. The Siematic SPS and its command protocoll are special to PSI and this section is probably of no interest to SICS managers outside. The SPS basiaclly support three operations:

- Push a button (Set a Digital I/O Bit).
- Read a status of instrument status packed into a bit (Read Digital I/O) .
- Read an ADC.

This is so user unfriendly that the usage of the SPS will mostly be packaged into Tcl-macros.

A SPS unit can be configured into the SICS server with the command:
MakeSPS name macintosh port channel
The parameters are: the name of the SPS in SICS, the serial port server computer, the port where the serial port server is listening and the channel number of the SPS unit at the serial port server computer. An example:
MakeSPS sps1 lnsp25.psi.ch 4000 6
configures a SPS unit at lnsp25.psi.ch at channel 5. The serial port server is listening at port number 4000. The SPS unit will be accessible as sps1 in SICS.

After configuartion the following four commands are understood by name, shown with sps1 as example:

sps1 push byte bit Corresponds to pushing the button mapped to the bit bit in the byte byte.

sps1 adc num Reads the value in the ADC num. num can be between 0 to 7 for a maximum of eight ADC's. Please note, that the values read are raw values which usually must be converted in some way to physically meaningful values.

sps1 status bit Reads the status of the bit bit. bit can be in the range 0 - 128.

sps1 stat2 byte bit Reads the status bit bit in status byte byte. Is equivalent to status, but adds some syntatctic sugar.

For all conversion factors, for all mappings of bytes and bits, consult the electronician who coded the SPS.

General Controller Object and Choppers

Chopper systems are handled via a generic controller object. This basicly consists of two components: One object represents the actual controller. This basic object allows to query parameters only. Then there is for each parameter which can be controlled from SICS in this controller an adapter object. These adapter object are virtual motors which can be driven with the normal run or drive commands. Currently two drivers for this scheme exists: one for a simulated device, the other for the Dornier Chopper Controller at FOCUS. The first step when initializing this system is the installation of the general controller object into SICS. This is done with the commands:

```
MakeChopper name sim
MakeChopper name docho mac port channel
```

The first command simply installs a simulated controller. The second command install a controller with a driver for the FOCUS Dornier Chopper system. Mac, port and channel are the usual Macintosh terminal server parameters which describe where the chopper controller is connected to through its RS-232 interface. After both commands the controller is available as command name within SICS.

A drivable parameter at this controller is installed with a command similar to this:

```
ChopperAdapter vname cname pname lower upper
```

vname is the name under which the virtual motor will appear in SICS. cname is the name of the controller object installed into SICS with the commands in the previous paragraph. pname is the name of the drivable parameter in the controller. upper and lower are the upper and lower limits for this parameter. More then one of these commands can be given for each general controller.

After this, the parameter can be modified by a command like:

```
drive vname newvalue
```

4.4.3 Motors

The following commands are available to install motors into the system:

Motor name SIM lowlim uplim err speed This command creates a simulated motor with the lower limits lowlim, the upper limit uplim, an ratio of randomly generated errors err and a driving speed of speed. Use this for testing and instrument simulation. If err is less then 0, the motor will not create failures and thus can be used in a instrument simulation server.

Motor name EL734 host port chan no This command creates a stepper motor named name which is controlled through a El734 motor controller. The parameters host, port, chan have the meanings defined above. no is the number of the motor in the EL734 motor controller.

Motor name EL734DC host port chan no This command creates an analog motor named name which is controlled through a El734DC motor controller. The parameters host, port, chan have the meanings defined above. no is the number of the motor in the EL734DC motor controller.

Motor name el734hp rs232controllername motornum Creates a motor object name using the newer motor drivers which access the motor controller directly, without the serial port server. rs232controllername is the name of a connection to the motor controll which has been set up with MakeRS232, above. Motornum is the number of the motor in the controller.

MakePIMotor name c804 pararray Creates a motor name connected to a C804 motor controller from the manufacturer Physik Instrumente. Pararray is a Tcl array holding the initialization information. The follwoing elements are required in this array:

Computer, port, channel The standard connection parameters.

upperlimit, lowerlimit The limits for this motor.

motor The number of the motor in the motor controller.

Motor name pipiezo pararray Creates a piezo electric positioning device. Again the controller is a Physik Instrumente controller. pararray has the same meaning as for the C804 controller given above.

Motor name ecb ecbcontroller ecb-number lowerlimit upperlimit This creates a motor which is controlled through the Risoe ECB electronic. The parameters:

ecbcontroller The ECB controller to which this motor is connected to. See below for more on ECB controllers.

ecb-number Number of the motor in the ECB system.

lowerlimit The lower hardware limit for this motors operation

upperlimit The upper hardware limit for this motors operation

In contrast to normal motors, the ECB motors have quite a number of hardware parameters which must be configured. The general syntax to configure them is: motorname parametername value. The following parameters are available:

encoder 0 if there is no encoder for this motor, 1-3 for the encoder used for this motor.

control The control bit flag. This falg determines if the motor sets a control bit in the ECB controller. This control bit can be used to drive air cushions and the like. If required set to 1, else leave at 0.

delay Delay time to wait after setting a control bit.

range The speed range for the motor: 0 for slow, 1 for fast

multi The ECB controller supports up to 24 motors. In some instances this is not enough. Then one ECB channel can be multiplexed into several motors. This flag (),1) determines if this is the case.

multchan The multiplexer channel for a multiplexed motor.

port The ECB port a multiplexed motor is using.

acceleration The speed with which the motor accelerates to its final speed.

rotation_dir Rotation direction of the motor.

startspeed Starting speed of the motor.

maxspeed The maximum speed for this motor.

auto Speed in automatic mode

manuell Speed used when driving the motor through the manual control box.

offset When using an encoder: the offset between the motor zero and the encoder zero.

dtolerance hardware tolerance of the motor.

step2dig conversion factor from encoder steps to physical values.

step2deg Conversion factor from motor pseudo encoder steps to physical values.

backlash In order to correct for backlash, Risoe motors always approach a target position from the same direction. In order to do this the motor has to overshoot and drive back when driving in the wrong direction. The parameter backlash determines how much to overshoot.

ECB motors have another quirck: 8 motors in a rack share a power supply! This has the consequence that only one of the 8 motors can run at any given time. In SICS this is directed through the anticollider module described elsewhere.

4.4.4 Counting Devices

MakeCounter name SIM failrate This command creates a simulated single counter accessible as object name. Failrate is the per centage of invocations at which the counter will generate a random failure for testing error treatment code. If failrate is less then 0, there are no failures. This can be used in a instrument simulation server.

MakeCounter name mcstas Creates a counter which interoperates with a McStas (cf. Section 6) simulation. Please note, that the McStas module mccontrol must have been initialized before this initialization can work.

MakeCounter name EL737 host port chan This command creates a single counter name, using an EL737 driver. The counter is at host host, listening at port port and sits at serial port chan.

MakeCounter name EL737hp terminalserver port Creates a counter object name which uses the faster driver which connects directly to the terminal server without the serial port server program. Terminalserver is the name of the instruments terminal server. Port is the port number at which the terminal server publishes the port at which the EL737 controller is connected. Usually this 3000 + port number.

MakeCounter name ecb ecb-controller Installs a counetr on top of the Risoe ECB hardware. The only parameter is the name of the ECB controller to use.

MakeHMControl name counter hm1 hm2 hm3 At some instruments (for instance TRICS) multiple counters or histogram memories are controlled by a master counter which watches over presets and the like. This command installs a virtual counter which does exactly that. The parameters are:

name The name of the virtual counter in SICS

counter The name of the master counter

hm1, hm2, hm3 Up to three slave counting devices.

Histogram Memory

Due to the large amount of parameters, histogram memories are configured differently. A histogram memory object is created using a special creation command. This command is described below. Then a lot of options need to be configured. The commands used for setting these options and their meanings are defined in the user documentation because histogram memories may be reconfigured at runtime. The sequence of configuartion options is ended with the command hmname init. This last command actually initialises the HM. Histogram memory objects can be created using the command:

MakeHM name type The parameter name specifies the name under which the HM will be avialable in the system. type specifies which type of driver to use. Currently these drivers are supported:

SIM for a simulated HM

SINQHM for the SINQ histogram memory

tdc for the Risoe histogram memory.

mcstas for the integration with the McStas (cf. Section 6) simulation.

Please care to note, that the SINQHM requires a counter box for count control. This counter must have been defined before creating the HM object. As an example the configuration of a SINQHM HM with the name banana will be shown:

```
MakeHM banana SINQHM
banana configure HistMode Normal
banana configure OverFlowMode    Ceil
banana configure Rank     1
banana configure dim0   400
banana configure BinWidth 4
banana preset 100.
banana CountMode Timer
banana configure HMComputer psds04.psi.ch
banana configure HMPort 2400
banana configure Counter counter
banana init
```

4.4.5 Velocity Selectors

A velocity selector is configured in a three step process. First a Tcl array is filled with the necessary configuration options for the actual velocity selector driver. In a second step the velocity selector is created with a special command. In a third step the forbidden regions for the velocity selector are defined. Currently two drivers for velocity selctors are known: a SIM driver for a simulated velocity selector and a DORNIER driver for a Dornier velocity selector hooked to a SINQ serial port setup. The last one needs a parameter array containing the fields Host, Port, Channel and Timeout. Host, Port and Channel have the meanings as defined at the very top of this section. Timeout is the maximum time to wait for responses from the velocity selector. A large value is required as the dornier velocity selector is very slow. The second step is performed through the following commands:

VelocitySelector name tilt-motor SIM This command installs a simulated velocity selector with the name name into the system. tilt-motor is used for driving the tilt angle of the selector. tilt-motor must exist before this command can be executed successfully.

VelocitySelector name tilt-motor DORNIER arrayname This command installs a dornier velocity selector into the system. name and tilt-motor have the same meanings as described above. arrayname is the Tcl-array with the driver configuration parameters.

As an example the configuration of a dornier velocity selector named nvs is shown:

```
set dornen(Host) lnsp25.psi.ch
set dornen(Port) 4000
set dornen(Channel) 6
set dornen(Timeout) 5000
VelocitySelector nvs tilt DORNIER dornen
nvs add -20 28800
nvs add 3800 4500
nvs add 5900 6700
nvs add 8100 9600
```

4.5 Initialization of General Commands

This section gives details on the initialization of commands which are common to many different instruments. The command set of SICS can be tailored to cover many specific needs. Moreover this system allows to replace functionality by other implementations suited to another users taste. This is a list of common command initialization commands.

MakeRuenBuffer MakeRuenBuffer makes the RünBuffer system available.

MakeBatchManager [**name**] Installs the new batch buffer management system. If no name is given, the default will be exe.

MakeDrive MakeDrive creates the drive and run command.

Publish name access The SICS server uses Tcl as its internal macro language. However, it was felt that the whole Tcl command set should not be available to all users from the command line without any protection. There are reasons for this: careless use of Tcl may clog up memory, thereby grinding the system to a halt. Invalid Tcl statements may cause the server to hang. Last not least, Tcl contains commands to manipulate files and access the operating system. This is a potential security problem when the server is hacked. However, in order to make macro procedures available the Publish command exists. It makes a Tcl command name available to SICS users with the access code access. Valid values for access are: Internal, Mugger, User and Spy.

TokenInit tokenpassword This command initialises the token control management system with the token command. The single parameter tokenpassword specifies the password for the token force command.

MakeOptimise name countername This command installs the Peak Optimiser into the SICS server. The Peak Optimiser is an object which can locate the maximum of a peak with respect to several variables. The arguments are: name, the name under which the Peak Optimiser can be accessed within SICS and countername, which is the name of an already configured SICS counter box.

MakeO2T nam OM 2TM creates an omega 2Theta virtual motor with name nam for omega 2Theta scans. OM defines an omega motor, 2TM a two theta motor.

MakeDataNumber SicsDataNumber filename This command makes a variable SicsDataNumber available which holds the current sequential data file number. filename is the complete path to the file were this data number is stored. This file should never, ever be edited without good reason, i.e. resetting the sequential number to 0 at the beginning of a year.

MakeXYTable myname Creates a XYTable object with the name myname. This object can store a list of x-y values.

MakeSinq Install the listener module for the accelerator divisions broadcast messages. This creates a command sinq.

MakeMaximize counter Installs a command max into SICS which implements a more efficient algorithm for locating the maximum of a peak then scanning and peak or center.

MakeMaxDetector name Installs name into SICS which implements a command for locating maxima on a two dimensional histogram memory image.

MakeLin2Ang name motor Creates a virtual motor name which translates an input angle into a translation along a tangent to the rotation axis. The distance of the translation table can be configured as variable: name length once this is established.

MakeSWHPMotor realmotor switchscript mot1 mot2 mot3 Creates switched motors mot1, mot2 and mot3 for real motor realmotor. For switching the script switchscript is used. This can be used when several motors are operated through the same motor driver. This implementation is not completely general now.

4.5.1 Monochromators

A monochromator is represented in SICS through a monochromator object which holds all the parameters associated with the monochromator and virtual motors which drive wavelength or energy. The commands to configure such a monochromator are:

MakeMono name M1 M2 M3 M4 This command creates a crystal monochromator object. Such a monochromator object is necessary for the usage of the wavelength or energy variables. The parameter name defines the name of the monochromator object in the system. M1 and M2 are the names of the Theta and two Theta motors respectively. M3 is an optional parameter defining a motor for driving the horizontal curvature. M4 is an optional parameter defining a motor for driving the vertical curvature of the monochromator.

MakeWaveLength nam mono creates a wavelength variable nam. The monochromator mono is used for adjustment.

MakeEnergy nam mono creates a energy variable nam. The monochromator mono is used for adjustment.

MakeOscillator name motor Installs a module name which oscillates motor between the software limits of the motor. This is useful for suppressing preferred orientation effects on powder diffractometers.name then supports the commands: start, stop, status which are self explanatory.

4.5.2 Reoccuring Tasks

Sometimes it may be necessary to execute a SICS command at regular time intervalls. This can be achieved with the sicscron command:

sicscron intervall bla blab blab This command installs a reoccuring task into SICS. The first parameter is the intervall in seconds between calls to the SICS command. Everything behind that is treated as the command to execute.

4.5.3 The SICS Online Help System

SICS has a simple built in help system. Help text is stored in simple ASCII text files which are printed to the client on demand. The help system can search for help files in several directories. Typically one would want one directory with general SICS help files and another one with instrument specific help files. If help is invoked without any options, a default help file is printed. This file is supposed to contain a directory of available help topics together with a brief description. The normal usage is: help topicname . The help system will then search for a file named topicname.txt in its help directories.

A SICS manager will need to configure this help system. A new directory can be added to the list of directories to search with the command:

```
help configure adddir dirname
```

The default help file can be specified with:

```
help configure defaultfile filename
```

Each of these command given without a parameter print the current settings.

4.5.4 Aliases in SICS

SICS knows three different kinds of aliases: object aliases, runtime aliases and command aliases. This is confusing but finds its explanation in the structure of SICS internals.

Object Aliases

An object alias is another name for a first class object installed into the SICS interpreter. For instance a second name for a motor. For instance the motor twotheta is quite often aliased to a4. Such an alias can be used like a normal SICS objects. Even in commands which access internal SICS interfaces like the drive command or others. Object aliases are installed into SICS with the SICSAlias command:

SicsAlias oldname newname This command installs newname as alias for the object oldname.

SicsAlias can only be used within initialization scripts. SicsAlias is considered deprecated and can be replaced with the superior runtime aliases described below.

Runtime Aliases

Runtime aliases are full object aliases which can be configured into the system at run time by a SICS manager. The syntax looks like this:

DefineAlias aliasname SICSobject This defines aliasname to be the alias for the SICS object SICSobject. It is not needed that SICSobject already exists. If SICSobject is already an alias, it is translated before definition. Multiple translation is possible, depending on the order of definition. When an alias is used, and it does not point to an existing object, the behaviour is the same as if an unknown object would have been used.

DefineAlias aliasname This command deletes the alias aliasname.

Command Aliases

Command aliases are shortcuts for lengthy commands. For instance one might want to define A4LL as a shortcut for "a4 softlowerlim". This is just to save typing or adapt SICS to MAD users who appear to have an unlimited memory for 2-4 letter acronyms. It is possible to redefine a SICS object with this for instance to define tt as an alias for temperature. However if one tries to use tt in a drive command it will fail because it is just a text replacement. A command alias can be installed into SICS at any time with manager privilege and the command:

alias shortcut bla bla bla .... This define shortcut as an alias for everything behind it.

A shortcut may take parameters.

4.5.5 The AntiCollision Module

In some cases motors have to be drive in a coordinated way. For instance, at TRICS, the chi motor may need to move first before omega can be driven in order to avoid collisions. Or at the ECB instruments, only one of eight motors in a rack can be driven at any given time. The anti collision module now allows to implement this. Anticollisions pattern of operation: Each collaborating motor is registered with the anti collision module. When trying to start such a motor, the anti collider just takes a note where it shoud go. On the first status check, a program is called which has to arrange the running of the motors into a sequence. This sequence then is executed by the anti collision module. The program which arranges the motors into a sequence is a configurable parameter and usually implemented as a script in SICS own marco language. In the SICS initialization file this requires the commands:

AntiCollisionInstall Creates the anitcollision module.

anticollision register motorname Registers motorname with the anti collision module.

anticollision script scriptname This command configures the script which is responsible for arranging the sequence of operations.

The script configured into anticollision is called with pairs or motor names and targets as parameters, Example:

```
sans2rack mot1 target1 mot2 target2 .....
```

Within the anticollision script, the following command may be used in order to define the sequence.

anticollision clear Clears the sequence list

anticollision add level motor target Add motor with target to level in the sequence.

4.6 The Internal Scan Commands

4.6.1 Scan Concepts

Scans in SICS involve an internal scan module and a lot of scripts which wrap the internal scan module into a syntax liked by the users.

The internal scan module in SICS evolved a little over time. It turned out that scans are a demanding job for a programmer because of the plethora of special scans people wish to perform and the many data file formats which have to be supported. This requires a very high degree of configurability. Under several refactorings the internal scan command has grown to become:

- A controller for the scan process.
- A container to store scanned variables and counts during the process of a scan. This includes commands to store and retrieve such values.
- A container for the configuration of the scan. A scan is configured by specifying functions to be called at the various steps during the scan. These are preconfigured to the standard scan functions. An API is provided to replace some or all of the scan functions by user defined ones.

The internal scan object is augmented by a library of standard scan functions. The transition to the new model is not yet clean in order not to break to much old code.

The standard scan command can be configured into SICS using the command:

MakeScanCommand name countername headfile recoverfil MakeScanCommand initialises the SICS internal scan command. It will be accessible as name in the system. The next parameter is the name of a valid counter object to use for counting. The next parameter is the full pathname of a header description file. This file describes the contents of the header of the data file. The format of this file is described below. The parameter recoverfil is the full pathname of a file to store recover data. The internal scan command writes the state of the scan to a file after each scan point. This allows for restarting of aborted scans.

The scan object (named here xxscan, but may have another name) understands the following commands:

xxscan clear clears the list of scan variables. Must be called before each scan with different parameters.

xxscan add name start step This command adds the variable specified by the argument name to the list of variables scanned in the next scan. The arguments start and step define the starting point and the sptep width for the scan on this variable.

xxscan run NP mode preset Executes a scan. The arguments are: NP the number of scan points, mode the counter mode to use (this can be either timer or monitor) and preset which is the preset value for the counter. Scan data is written to an output file.

xxscan continue NP mode preset Continues an interrupted scan. Used by the recovery feauture.

xxscan silent NP mode preset Executes a scan. The arguments are: NP the number of scan points, mode the counter mode to use (this can be either timer or monitor) and preset which is the preset value for the counter. The difference to run is, that this scan does not produce an output file.

xxscan recover Recovers an aborted scan. The scan object writes a file with all data necessary to continue the scan after each scan point. If for some reason a scan has been aborted due to user intervention or a system failure, this scheme allows to continue the scan when everything is alright again. This works only if the scan has been started with run, not with silent.

xxscan getfile This command returns the name of the current data file.

xxscan setchannel n Sometimes it is required to scan not the counter but a monitor. This command sets the channel to collect data from. The argument n is an integer ID for the channel to use.

xxscan getcounts Retrieves the counts collected during the scan.

xxscan getmonitor i Prints the monitor values collected during the scan for the monitor number i

xxscan gettime Prints the counting times for the scan points in the current scan.

xxscan np Prints the number of points in the current scan.

xxscan getvardata n This command retrieves the values of a scan variable during the scan (the x axis). The argument n is the ID of the scan variable to retrieve data for. ID is 0 for the first scan variable added, 1 for the second etc.

xxscan noscanvar Prints the number of scan variables

xxscan getvarpar i Prints the name , start and step of the scan variable number i

xxscan interest A SICS client can be automatically notified about scan progress. This is switched on with this command. Three types of messages are sent: A string NewScan on start of the scan, a string ScanEnd after the scan has finished and a string scan.Counts = {109292 8377 ...} with the scan values after each finished scan point.

xxscan uuinterest As above but the array of counts is transferred in UU encoded format.

xxscan dyninterest As above but scan points are printed one by one as a list containing: point number first_scan_var_pos counts.

xxscan uninterest Uninterest switches automatic notification about scan progress off.

xxscan integrate Calculates the integrated intensity of the peak and the variance of the intensity for the last scan. Returns either an error, when insufficient scan data is available or a pair of numbers. Peak integration is performed along the method described by Grant and Gabe in J. Appl. Cryst. (1978), 11, 114-120.

xxscan window [**newval**] Peak Integration uses a window in order to determine if it is still in the peak or in background. This command allows to request the size of this window (without argument) or set it (with argument giving the new window size).

xxscan simscan pos FWHM height This is a debugging command. It simulates scan data with a hundred points between an x axis ranging from 10 to 20. A gauss peak is produced from the arguments given: pos denotes the position of the peak maximum, FWHM is the full width at half maximum for the peak and height is its height.

xxscan command tclcommand Sets the tcl command procedure to invoke at each scan point. See below for the description of user defined scans (Old Style). Invoked without argument command returns the name of the current command procedure.

xxscan configure mode Configures the several possible scan modes for the scan object. Currently there are two:

- **standard**, the default mode writing ASCII files.
- **amor**, a special mode the reflectometer AMOR which writes NeXus files.
- **script** Scan functions are overriden by the user.
- **soft** The scan stores and saves software zero point corrected motor positions. The standard is to save the hardware positions as read from the motor controller.
- **user** configures the old style user overridable scans.

xxscan storecounts counts time mon1 mon2 ... This stores an entry of count values into the scan data structure. To be used from user defined scan functions. The scan pointer is incremented by one.

xxscan storecounter Store the counts and monitors in the counter object configured for the scan into the scan data structure. Increments the scan pointer by one.

xxscan appendvarpos i pos Append pos to the array of positions for scan variable i. To be used from user defined scan functions.

xxscan callback scanstart | scanpoint | scanend Triggers callbacks configured on the scan object. May be used by user functions implementing own scan loops.

xxscan function list Lists the available configurable function names. The calling style of these functions is described in the next section about stdscan.

xxscan function functionname Returns the currently configured function for functionname.

xxscan function functionname newfunctionname Sets a new function to be called for the function functionname in the scan.

4.6.2 User Definable Scan Functions

The last commands in the last section allowed to overload the functions implementing various operations during the scan with user defined methods. This section is the reference for these functions. The following operations during a scan be configured:

writeheader Is supposed to write the header of the data file

prepare Prepare does all the necessary operations necessary before a scan starts.

drive Is called to drive to the next scan point

count Is called at each scan point to perform the counting operation

collect Is called for each scan point. This function is supposed to store the scan data into the scan data structure.

writepoint Is called for each scan point and is meant to print information about the scan point to the data ile and to the user.

finish Is called after the scan finishes and may be used to dump a data file or perform other clean up operations after a scan.

userdata This is the name of a user defined object which may be used to store user data for the scan.

The exact invocations of the functions:

- writeheader scanobjectname userobjectname
- prepare scanobjectname userobjectname
- drive scanobjectname userobjectname point
- count scanobjectname userobjectname point mode preset
- collect scanobjectname userobjectname point
- writepoint scanobjectname userobjectname point
- finish scanobjectname userobjname

scanobjectname is the name of the scan object invoking the function. This can be used for querying the scan object. userobjectname is the name of a entity as specified as userdata in the configuration. point is the number of the current scan point.

4.6.3 User Defined Scans(Old Style)

This feauture allows to override only the counting operation during a scan. This feauture is deprecated in favour of the user overridable scan functions described above. As it is still used, however, here is the documentation for reference.

In some cases users wish to control the scan more closely, i.e. do multiple counting operations at the same point etc. This is especially true when magnets are involved. In order to do this a facility has been provided which allows the user to specify a macro routine which is called at each point. This macro routine then performs all necessary operations and then is responsible for storing its data. In order to allow for this commands have been defined which allow to append a line to the scan data file and to store measured data in the scan data structure. The last feature is necessary in order to make scan status displays and scan analysis, such as center detection, work. The following steps are required:

1. Write a suitable macro procedure for the actions required at each scan point. The procedure signature looks like this:

   ```
   proc myScanPoint {point} {
   }
   ```

   And will be called with the number of the current scan point as a parameter. Besides all usual Tcl and SICS commands the following special commands may be used:

   xxxscan line text Appends all the text after line to the scan data file.

   xxxscan storecounts c1 c2 c3 c4 ... Stores the first number given as count data, all the others as monitor values in the internal scan data structure.

2. Test the procedure.

3. Switch the internal scan command command into user scan mode with the command: **xxxscan configure user**

4. Assign your procedure to the internal scan command with the command: **xxxscan command myScanPoint**

5. Use normal scan commands for doing your scan.

6. Switch the internal scan command into normal mode with the command: **xxxscan configure standard**.

In all this replace xxxscan with the name of the internal scan command.

4.6.4 The Scan Command Header Description File

As if all this configurability is not enough, there is another level of configurability. The SICS internal scan command allows to configure the contents of the header of the ASCII scan data file through a template header file. This is only possible when the scan functions are left in their default configuration. If scan functions are overloaded it is the users repsonsability to take care of data file writing. This section describes the contents of the

template file. This header description file consists of normal text mixed with a few special keywords. The normal test will be copied to output verbatim. The keywords indicate that their place will be replaced by values at run time. Currently only one keyword per line is supported. Keywords recognized are:

!!DATE!! Will be replaced with the file creation date.

!!VAR(name)!! Will be replaced with the value of the SICS variable name.

!!DRIV(name)!! Will be replaced with the value drivable variable name. Drivable variables are all motors and all variables which may be used in a drive or run command.

!!ZERO(name)!! Will be replaced with the value of the softzero point for motor name.

!!FILE!! Will be replaced by the creation name of the file.

Please note that text behind such a keyword in the line will not be copied to the output.

4.6.5 Differential Scans

When aligning or when searching peaks a very fast scan is required. This is the differential scan. It starts a motor and collects counts while the motor is running. The counts collected are the monitor normalized difference to the previous reading. This functionality can be configured into SICS with the command:

```
MakeDiffScan
```

in the configuration file. An optional parameter defines another name then diffscan (the default) for this object. Differential scans can only be run against one motor as it cannot be guaranteed that motors involved in a coordinated movement operate at the same speed without mechanical coupling. The procedure to use diffscan is:

- Configure a scan variable into a SICS scan object: xxscan add var start step
- Run diffscan as: diffscan scanobjectname end_position_of_scan This runs the differential scan. Scanobjectname is the name of a SICS internal scan object. It will be used to store the results of the scan. While the scan is running some basic information is printed. The scan will range from the start given in the xxscan add command to the end position given in this call.

The diffscan object has two configurable parameters:

monitor The monitor number to normalize against. For maximum precision this should be a monitor with a lot of intensity on it.

skip The number of SICS main loop cycles to skip between readings. This can be used to control the amount of data generated during a differential scan. This may become a problem if there is fast hardware.

A word of warning: positions found in differential scans may not be totally correct. The differential scan might even miss peaks when the relationship between motor speed and sampling rate is bad.

Diffscan is usally wrapped in a common script command:

fastscan motor start stop speed which does a fast scan for motor from start to stop. Before the scan the motors speed is set to speed. The motor is set to its original speed after termination of the scan.

This script can be copied from one of the older instrument command files.

4.6.6 Peak Analysis

There are several other feautures which can be configured into SICS which interact very closely with the scan module:

MakePeakCenter scancommand MakePeakCenter initialises the peak analysis commands peak and center. The only parameter is the name of the internal scan command.

4.6.7 Common Scan Scripts

There exists a library of script functions around the scan module which are commonly used. They provide an object oriented wrapper around the internal scan command and the **cscan** and **sscan** commands. These commands can be made available by including the scancommand.tcl file into the instruments configuration file.

4.7 Scripting NeXus Files

This section describes the scripting interface to NeXus files, called nxscript. Scripting the generation of NeXus files has the advantage that it can be customised very quickly to special needs. Moreover it might help to reduce the amount of instrument specific code required for an instrument. This scripting interface uses the NeXus dictionary API for the actual writing process. This has the following consequences:

- The interface needs two filenames: the NeXus filename and the dictionary filename when opening files.
- Writing commands have the general form: alias object. This means that object is written to the NeXus file using the specified alias.
- Another property is that some writing commands write several bits of information in one go. In such cases the aliases for the additional bits are derived from the base alias by appending specific strings. Thus dictionary files need to have a special form for this scripting interface to work.
- Nxscript also tries to figure out the dimensions of multidimensional datasets such as histogram memories by itself. In such cases the dimension attribute in the dictionary file must be omitted.
- Nxscript assumes the following policy for data writing: irrespective of errors write everything you can. Thus this interface will complain bitterly and verbosely if something does not work, but never return an error.

4.7.1 Usage

Before this facility can be used nxscript has to be installed into the SICServer from the instrument configuration file. This is done through the command:

MakeNXScript ?name? This creates a NeXus scripting object. If the name is omitted, an object named nxscript is created. If a name is given, this name is used for the scripting object. Having scripting objects with different names is also the only possibility to have more then one NeXus file writing operation going at a given time.

In the following sections it is assumed that an object **nxscript** had been configured into SICS.

File Opening and Closing

nxscript create5 nexusFile dictFile Creates a new NeXus file based on HDF-5 with the name nexusFile. The dictionary file dictFile is used.

nxscript create4 nexusFile dictFile Creates a new NeXus file based on HDF-4 with the name nexusFile. The dictionary file dictFile is used.

nxscript createxml nexusFile dictFile Creates a new NeXus file based on XML with the name nexusFile. The dictionary file dictFile is used.

nxscript reopen nexusFile dictFile Reopens an existing NeXus with the name nexusFile for modification or appending. The dictionary file dictFile is used.

nxscript close Closes the current NeXus file. This command MUST be given at the end of each script in order to make sure that all data is written properly to disk.

Writing Things

nxscript puttext alias bla bla bla .... Writes everything after alias as text data to the alias. The definition string for the alias should not contain a dimension description, this is automatically appended.

nxscript putfloat alias value Writes a single floating point value to alias alias.

nxscript putint alias value Writes a single integer value to alias alias.

nxscript updatedictvar alias value Updates the dictionary value alis to value.

nscript putmot aliasName motorName Writes the position of the motor motorName into the NeXus file as described by aliasName. Theposition is a zero point corrected position. If another alias aliasname_null exists in the dictionary, the zero point of the motor is also written to file.

nxscript putcounter aliasBase counterName Writes counter data to a NeXus file. Data is taken from the single counter counterName. What is written depends on the aliases present in the dictionary file:

aliasBase_preset The preset value.

aliasBase_mode The counter mode

aliasBase_time The actual time counted, without breaks due to insufficient beam.

aliasbase_00 ... aliasBase_09 The monitor readings for monitors 0 to 9. Please note that 00 would denote the normal counting tube at a scanning type of experiment.

nxscript puthm hmAlias hmName ?start? ?length? Writes data from the histogram memory hmName to a NeXus file using the alias hmAlias. Nxscript automatically updates the dim0, dim1, ..., timedim dictionary variables. Thus these can be used to define the dimensions in the dictionary file. If the optional parameters start and end are given, a subset of the data is written. It is the users responsability that the values requested make sense to the histogram memory. In the case of subset writing, the dimensions have to be specified in the definition string belonging to the alias. Nxscript sets a variable timedim in the dictionary though which contains the length of a time binning if appropriate. This is a special feauture for writing extra detectors at SANS and AMOR.

nxscript puttimebinning aliasName hmName Writes the time binning at histogram memory hmName to file using the alias aliasName. The length of the time binning data is automatically appended to the definition string for the alias.

nxscript putarray aliasName arrayName length Writes the Tcl array arrayName to file using the aliasName. The definiton string belonging to aliasName does not need to contain a -dim argument as this is set by this routine. The parameter length is the length of the array. Only rank 1 arrays are supported.

nxsript putglobal attName bla bla bla This writes an global attribute attName. Everything after attName is concatenated to a string which then respresents the value of the attribute.

nxscript makelink targetAlias victimAlias This creates a symbolic link for victimAlias in the group designated by targetAlias.

4.8 Automatic Updating of NeXus Files

Some instruments perform measurements for quite long counting times. In such cases it is advisable to save the data measured so far to file in order to protect against hardware or software failures. To this purpose an automatic file upgrade manager is provided. On installation the automatic update object is connected wth a counting device through the the callback interface. This makes sure that the update manager is automatically notified when counting starts or finishes.

4.8.1 Prerequisites for Using the Automatic Update Manager

In order to use automatic updating, three programs must be provided. Each of these programs can be a script which uses the nxscript facility. It can also be a SICS command.

startScript This program is supposed to write the static part of the file. It is called once when the file is created.

updateScript This program is supposed to create and update the variable data elements in the NeXus file. This is called frequently.

linkScript This program is supposed to create the links within the NeXus file. This is called once after startscript and updateScript have been run.

4.8.2 Installing Automatic Update

An automatic update object is installed into SICS with:

```
updatefactory name countername
```

name is a placeholder for the name under which SICS knows the automatic update object. name is available as a SICS command later on. countername is a placeholder for a counter object (counter or HM) which triggers automatic updating of NeXus files. This object has to support both the countable and callback interfaces of SICS. Suitable SICS objects include counter and histogram memory objects.

4.8.3 Configuring Automatic Update

The SICS command created with updatefactory (see above) supports a few parameters which allow for the configuration of the whole process. Parameters follow the normal SICS syntax. Futhermore there is a subcommand list, which lists all configuration parameters. Supported parameters are:

startScript The program supposed to write the static part of the file.

updateScript The program supposed to create and update the variable data elements in the NeXus file.

linkScript This program supposed to create the links within the NeXus file.

updateintervall The time intervall in seconds between updates. The defualt is 1200, eg. 20 minutes.

4.9 Instrument Specific SICS Initializations

4.9.1 Initialization for Four Circle Diffractometers

This section describes how the modules which are special for a four circle single crystal diffractometer are configured into SICS. The following feautures are available:

MakeHKL theta omega chi phi MakeHKL creates the hkl command for the calculation of settings for a four circle diffractometer. The four parameters are the names of the motors driving the two theta, omega, chi and phi circles of the diffractometer. These motors must already exists before this command may succeed.

MakeHKLMot hkl Creates the drivable H, k, l virtual motors using hkl, an object created by MakeHKL for calculations.

MakeDifrac tth om chi phi cter This command installs the Difrac subsystem into SICS. Difrac is a whole F77 package for controlling a four circle diffractometer. Afterwards Difrac commands are available in SICS with the prefix dif, for example dif ah calls the difrac ah command. Difrac is described in more detail elsewhere. The parameters are the four circle motors two theta, omega, chi and phi and the counter. This is no longer maintained.

MakeMesure name scanobject hklobject omega s2t fileroot datanumberobject MakeMesure installs the single counter four circle diffractometer measurement procedure into SICS. It will be accessible as object name afterwards. MakeMesure takes a lot of parameters:

scanobject The name of the internal scan object.

hklobject The name of the object which does crystallographic calculations.

omega The name of the motor driving omega.

s2t The name of the two theta motor for use in omega two theta scans.

fileroot The full path to the data file directory without final /

datanumberobject The name of the SICS data number object for creating unique file numbers.

MakeUBCalc name hklobject This installs a UB matrix calculation module with the name name into SICS. The second parameter is a hklobject as created with MakeHKL to which any calculated UB's can be transferred.

MakeHklscan scanobject hklobject Installs the hklscan command which allows to scan in reciprocal space. Scanobject is the name of SICS internal scan object, hklobject the name of a reciprocal space calculation object as configured with MakeHKL.

MakeTRICSSupport Installs a command, tricssupport, which helps the TRICS status display.

Commands implemented by tricssupport:

tricssupport oldframe file idet nFrame Loads and sends the frame nFrame of detector idet from file file in UUencoded form.

tricssupport interest Enables this connection to receive notifications whenever a new frame of data had been written

tricssupport newframe Called from scripts. Triggers sending new frames to all registered connections.

There are also a lot of scripted command available for four circle diffractometers. These may be copied from tricscom.tcl. These include:

four print the four all important angles

tricsscan start step np Omega scan with a PSD

psdrefscan file step np mode preset Read reflections from file, drive to them, do a omega scan with tricsscan using the parameters specified.

detscan start step np Do a detector calibration scan.

phscan start step np Do a phi scan

hklscan2d Scanning reciprocal space with the area detector

scan2d Configure SICS for general scanning with the PSD. This is meant to supersede many of the special scans above.

scan1d Configure SICS for general scanning with the single detector.

4.9.2 Triple Axis Spectrometer Specific Commands

One aim for the implementation of the triple axis spectrometer in SICS was to implement as closely as possible the command set of the ILL program MAD. For this, there are two implementations: an older one where most things werde done in C-code. And a newer one which implements a relaxter MAD emulation. The problem with the MAD emulation is that MAD relies on the order of variables and motors in memory. The newer MAD emulation obeys this only for the qh, qk, ql and en variables. This section describes the newer more portable TAS setup. There are some C-modules and a lots of scripts which implement the MAD emulation.

The TAS requires the following initializations in its instrument file:

MakeTasUB tasub Installs the TAS crystallographic calculation module into SICS. It will have the name tasub (recommended).

MakeTasScan iscan tasub Installs the module with the TAS specific scan functions into SICS. The TAS implements its own data format resembling the ILL TAS data format.

Moreover it is required to add the peak center module, drive, exe and a lot of variables: polfile, alf1-4, bet1-4, ETAM ETAS ETAA.

The scripts for the MAD emulation live in the file tasubcom.tcl. This file will need little editing in order to cope with another triple axis machine, just a couple of path variables will need to be changed.

4.9.3 Special Commands for the Reflectometer (AMOR)

There are some special command initializations for the reflectometer AMOR. These commands are most likely not portable to other instruments because they encompass the special geometry at AMOR and the AMOR development has not fully matured. The following initialization commands are available:

MakeAmor2T name da This creates a virtual two theta motor for the reflectometer AMOR. The two parameters are the name of the object in SICS and a Tcl-array with configuration parameters. The needed elements of this array are:

mom The monochromator omega motor.

som The sample omega motor.

coz The height movement of the detector.

cox The movement of the detector along the optical bench.

stz The height movement of the sample connected to the omega circle.

soz The height movement of the sample table.

d4b The motor moving the whole diaphragm 4 up.

d5b The motor moving the whole diaphragm 5 up.

com The omega mevement of the detector.

An example:

```
set a2t(mom) mom
set a2t(som) som
set a2t(coz) coz
set a2t(cox) cox
set a2t(stz) stz
set a2t(soz) soz
set a2t(d4b) d4b
set a2t(d5b) d5b
set a2t(com) com
MakeAmor2T a2t a2t
```

creates a virtual AMOR two theta motor with the name a2t.

MakeStoreAmor hm Creates an object for writing reflectometer data files. The name of the command is storeamor. The parameter hm denotes the histogram memory object.

MakeAmorStatus name scan hm This creates a helper object for the reflectometer status display with name name. This object performs some operations on behalf of the status display for the reflectometer AMOR. The parameter scan denotes the name of the scan object. The parameter hm the name of the histogram memory object.

AMOR Status Display Commands

MakeAmorStatus creates a SICS command which is used by the AMOR status display for displaying proceesed data, especially in TOF-mode. This module provides the following commands:

amorstatus interest This registers this connection for receiving automatic notifications. Automatic notifications send are:

SCANSTART At scan start a message **ScanClear** is sent followed by the uuencoded new x-axis for the plot.

SCANPOINT At each scan point the arrays of counts in both detector are sent in uuencoded form labelled arrow_spinupup and arrow_spinuplo.

COUNTSTART The start of counting on the histogram memory. Send a message **TOFClear** and then the uuencoded time binning labelled arrow_time.

FILELOADED activated each time user defined model data is loaded into the SICS server. This data gets send as arrow_name in uuencoded form. Both x- and y-axis are sent in floating point.

Please note that floating point data is transformed to fixed point by multiplication of 65653 before transfer. The first number in each uuencoded message is an integer describing the length of the data. In case of double data such as fileloaded the y-data follows immediatetly after the x-data. Also the appropriate data is automatically sent after the interest command.

amorstatus collapse sums all counts in all detectors in time and sends the data back as an uuencoded image. The first two numbers in the message define the dimensions of the data.

amorstatus sample name x1 x2 y1 y2 Sums the detector counts on an area defined by the rectangle x1, x2, y1, y2. The length of this is the time binning. The data is sent back in uuencoded form labelled arrow_name.

amorstatus clear Clears all user loaded data.

amorstatus load filename scale loads user defined data for distribution to the status display clients. The y data is scaled according to the scale factor provided.

amorstatus projectytof Returns a UUencoded 2D array of y against TOF.

4.9.4 SANS Special Commands

Some special initializations for SANS Instruments:

MakeMulti name SANS uses a special syntax feature where several motors are grouped into a component group. For example beamstop or detector. MakeMulti creates such a group with the name name. Once such a group has been created, it has to be configured. For this a few configuration commands are available:

name alias motname compname This command makes motor motname available as component motor compname. For example: **bs alias bsx x** makes motor bsx available as x in the beamstop group. Then the bsx motor can be driven by the command **bx x = 12.**.

name pos posname motname value motname value .... The group command supports the notion of named positions. This means that a special combination of angles can be accessed through a name. This commands defines such a named position with the name posname. posname is followed by pairs of motorname value which define the position.

name endconfig Once a group has been completely defined the configuration process must be ended with endconfig.

MakeSANSWave name velo_name ¿ Installs a velocity selector wavelength variable into SICS. The variable will have the name given as first parameter. Usually lambda is a good idea. The second parameter, velo_name, is the name of the velocity selector which is controlled by this wavelength variable. This module contains a hard coded formula which may only be applicable to the SANS at PSI.

MakePSDFrame This installs a command into SICS which allows to retrieve a detector image from the histogram memory, even in TOF-mode.

Many other commands for controlling collimators, attenuators, beamstops and shutters are implemented in Tcl. These commands use non standardizable hardware such as the Siematic SPS.

4.9.5 Special FOCUS Initializations

These initailizations are special to the FOCUS instrument:

InstallFocusmerge datafile Installs the module which is responsible for merging focus data into merged detector banks.

MakeFocusAverager average hmc Installs the average command and the focusraw command into SICS which are used by the FOCUS status client in order to display processed histogram memory data.

Special Internal FOCUS Support Commands

focusraw bankid Dumps in UUencoded form the content of the detector bank bankid. This is required in order to add the TOF-binning to the data and in order to handle the virtual merged detector bank which is built from contributions of the three physical detector banks.

average start stop bankid Sums the detectors between start and stop of detector bankid into a histogram. A standard display in the status client.

focusmerge puttwotheta nxscriptmod bankname alias Writes two theta values for the detector bank bankname into the file described by the nxscript module nxsciptmod to the alias alias. A helper function for data file writing.

focusmerge putmerged nxscriptmod alias Writes the virtual merged detector bank into alias in nxscriptmod.

focusmerge putsum nxscriptmod bankname alias Writes the summed counts for detector bank bankname under alias into nxscriptmod.

focusmerge putelastic nxscriptmod alias theoelastic Calculate the position of the elastic line and write it to alias in nxscriptmod. If no elastic line can be calculated, the theoretical elastic line, theoelastic, is used.

Chapter 5

Programming SICS Macros

The SICS server has a built in macro language. This macro language is basically John Ousterhout's Tool Command Language Tcl. Tcl is described elsewhere. A sound knowledge of Tcl is required for programming SICS macros. The SICS macro language can be used for the following purposes:

- Add hoc measurement procedures.
- Trial measurement procedures.
- Syntax adaptions to one's own favourite syntax.
- Building of more complex commands from the SICS primitives.

The general procedure for defining a macro requires defining the macro in a new file, source this file from the configuration script and the use of the Publish command to make it available. New commands can best be defined as Tcl procedures, but the obTcl object oriented extension to Tcl is known to work as well. The SICS macro language allows to access:

- Most Tcl commands.
- All SICS commands.

In the following sections a few pecularities of the SICS macro system will be discussed.

5.1 Input/Output

It would be quite verbose and confusing for the user if all output from SICS commands called from a macro would appear on the screen during macro execution. Therefore all normal SICS output to a client is suppressed while executing a macro. Except error messages and warnings which will always be written to the client executing the macro. The output of a SICS command is available within the macro script through the normal Tcl mechanism as a return value. This allows for processing of SICS output within a macro. If the output to the client executing the macro is required this can be done with the ClientPut command, detailed in the user documentation.

5.2 Error Handling

Tcl has the feature that it aborts execution of a script when an error occurs. If a macro script needs to handle errors either from Tcl or from SICS commands this can be achieved by using the Tcl catch mechanism.

If things are seriously wrong or the users wishes to interrupt an operation SICS interrupts are used. Scripts implementing measurement procedures may need to test and even modify interrupt values. A script can inquire the current interrupt value of the connection with the command **GetInt**. If a script can handle an error condition it may set the interrupt on the connection object with the **SetInt** command. The textual representations of interrupts for these commands are: continue, abortop, abortscan, abortbatch, halt, free, end.

5.3 Interacting with SICS within a Script

There exist a few commands which allow to inquire or manipulate SICS internals. Most of these commands are only available in macro scripts.

SICSType thing. SICSType lets SICS find out if thing has some meaning within SICS. Possible return values are: DRIV for a drivable variable, COM for a SICS command, NUM for a numeric value and TEXT for anything else.

SICSBounds var newval SICSBounds checks if newval violates the hardware or software limits of the variable var.

SetStatus newval SetStatus sets the SICS status line to a new value. Possible values for newval are: Eager, UserWait, Count, NoBeam, Paused, Driving, Running, Scanning, Batch, Halt, Dead.

SICSStatus var SICSStatus returns a integer value representing the current status of the object var. var must be a drivable or countable object. The integer code returned are defined in the SICS programmers documentation.

5.4 SICS Interfaces in Tcl

Work has begun to implement SICS internal interfaces in Tcl. This opens the port for writing even device drivers in Tcl. Another use is to define virtual motors quickly in Tcl. At the time of writing, July 2005, this is only developed for the object interface and the drivable interface. For the meaning of internal SICS interfaces please consult the SICS programmers documentation. Be warned: with the feautures described in this section, you can mess up SICS badly.

5.4.1 The Object Interface

MakeTclInt name Creates an object name. This object then understands the following commands:

name savescript scriptname Configures a script which will be called when it is time to dump the status of the SICS server. This script will be called with the name of the object as its only parameter.

name backup bla bla bla.... To be used from savescripts. Writes everything behind backup into the status file.

The use of this facility is to place special commands into the status file which may, for instance, request calculations to be made or drive parameters not caught in the standard SICS objects to special values. For example: at SANS2 this is used in order to store attenuator and collimator values. Both are implemented as scripted commands and thus do take part in the standard SICS object saving scheme.

5.4.2 Overriding the Drivable Interface with Tcl

The drivable interface of any given drivable object can be overriden with tcl functions. This includes an object created with MakeTclInt. The syntax is:

TclReplaceDrivable objname key scriptname tclName This replaces the drivable interface function defined by key with the script scriptname in the driveable interface of the SICS object object. tclName is the name of an arbitrary Tcl object which can hold user data. Possible function keys and their function signatures are:

halt haltscript, no parameters

checklimits checklimitsscript targetvalue

setvalue setvaluscript targetvalue

checkstatus checkstatusscript, no parameters

getvalue getvaluescript, no parameters

All procedures, excpet getvaluescript, are supposed to return the approriate SICS return codes (HW*) as integer numbers. Getvaluescript is supposed to return the position of the device.

TclDrivableInvoke objname key A debugging aid: Invokes the scripted function denoted by key of the object objname and prints the results. The function keys are the same as given above.

Chapter 6

The McStas SICS Interface

It is useful to drive a simulation of an instrument with the same interface as is used at the original instruments. One of the better packages for performing simulations of neutron scattering instruments, including samples, is McStas. This section describes the SICS interface to McStas simulations. The interface consists of three parts:

- A McStas controller module which controls the actual simulation.
- A McStas reader which is responsible for reading simulated data into SICS counters and histogram memories.
- Counter and histogram memory drivers which redirect their actions to the McStas controller module.

The general ideas is that all parameters are handled through the normal SICS simulation drivers. The counting operations, however, are linked with the McStas simulation. In order to be portable, many aspects are controlled by scripts. These scripts are configured at the McStas Controller. Several scripts must be defined:

startmcstas This script will be invoked when counting starts and has to collect the necessary settings from SICS, construct a McStas command line and start the simulation. As a return value this script has to return the PID of the started mcstat process.

mcstastest pid Tests if the McStas simulation is still running.

mcstasdump pid Has to send a signal to McStas which causes it to dump its data without terminating. Current versions of McStas do this on receiving the USR2 signal.

mcstasstop pid Stops the McStas simulation.

mcstasread Reads the McStas simulation output and transfers monitor and histogram memory data into the appropriate SICS objects.

6.1 McStas Requirements and SICS Requirements

In order for the McStas SICS interface to work the McStas simulation has to be configured in a certain way:

- All parameters which have to pass between SICS and McStas have to be declared as simulation parameters in the DEFINE INSTRUMENT section of the instrument definition file. Alternatively SICS can write the data to be passed to McStas into a file. But then this file must be read in the INITIALIZE section of the instrument definition and values must be assigned to the appropriate McStas variables.
- In order for the NeXus-XML based reading to work McStas must dump its data into a single file: use the **-f filename** option. The format must be –**format=XML**.
- In order to count on monitor, a modified monitor component, MKMonitor MUST be used in the simulation. This component writes the collected total counts into a file. This file is the read by SICS in order to determine the control monitor. Evaluating the McStas dump \ file each time proved to be to inaccurate. The name of the file containing the monitor must be configured through: mccontrol configure mcmonfile name-of-file.
- The mcstas simulation executable must be declared with allowexec in order to be able to start with the Tcl exec command.

6.2 The McStas Reader

In order to enable transfer from McStas result files into SICS objects a reader object is needed. This module supports XML formatted McStas files, with the output dumped into one file. The McStas options to achieve this are: **-f filename** –**format="XML"** This module supports the following commands:

mcreader open filename Opens a McStas simulation file for reading.

mcreader close Closes a McStas file after use.

mcreader insertmon path object monitornumber scale This transfers a monitor value from a previously opened McStas file into a SICS monitor field. The McStas field read is the values tag belonging to the component. The parameters:

path The path to the correct values field in the simulation file. The format is the same as the path format for NXopenpath in the NeXus-API (which will internally be used). For groups, the name attribute is used a path component.

object The counter object into which the monitor is read. This is a limitation, with the this McStas interface only counters can store monitors.

monitornumber Monitornumber is the monitor\ channel into which the value is to be stored.

scale Scale is an optional scale factor for the monitor. Real monitors have a sensitivity of E-6, McStas monitors have an efficiency of 1.0. This factor allows to correct for this.

mcreader inserthm path hmobject scale Inserts array data stored under path in the histogram memory array of hmobject which must be a valid SICS histogram memory object. The path is the same as given for insertmon, but of course the data part of the detector must be addressed. Scale is again an optional scale factor which allows to scale the McStas counts to real counts.

The mccreader module also supports reading data from any ASCII file into SICS. Mcreader close and open are not required then. For reading histogram memory data, the appropriate data has to be parsed into a SICSdata object first. Then data can be trasnferred using the following commands:

mcreader insertmondirect counter num value Assigns value to the monitor num at the counter object counter. Monitor 0 is the actual counts data.

mcreader inserthmfromdata hm data Inserts the data in the SICSData object data into the histogram memory hm.

6.3 The McStas Controller

The actual control of the "counting" operation of the McStas simulation is done by the McStas controller module in SICS. Internally this module implements the routines for counting virtual neutrons. Towards the SICS interpreter, an interface is exhibited which allows to configure and test the McStas controller. The McStas Controller delegates many tasks to script procedures written in SICS's internal Tcl scripting language. This is done in order to achieve a degree of generality for various types of instruments and in order to allow easier adaption to differing demands. This is the SICS interface implemented:

mccontrol configure mcstart startscriptname Configures the script which starts the McStas simulation. Startscriptname is the name of a Tcl procedure which collects the necessary information from SICS, builds a command line and finally starts the simulation. This script is expected to return either an error or the PID of the started process. Startscriptname will be called with the parameters mode and preset which represent the counting characteristics.

mccontrol configure mcisrunning runtestscriptname Configures the name of a script which tests if the McStas process is still running. Runtestscriptname is called with the PID as a parameter. This returns 0 in the case the McStas simulation was stopped or 1 if it is still running.

mccontrol configure mcdump dumpscriptname Configures the name of the script which causes McStas to dump intermediate results. The script will be called with the PID of the started McStas process as a parameter.

mccontrol configure mckill killscript Configure the name of a procedure which kills the current McStas simulation process. KillScript will be called with the PID of the McStas simulation as a parameter.

mccontrol configure mccopydata copyscript This configures the name of a script which has the task to read the results of a McStas simulation and assign values to SICS monitors and histogram memories.

mccontrol configure update updateintervallinseconds This configures the minimum time between McStas dumps in seconds. The idea is that SICS buffers values during a simulation run and does not interrupt the McStas process to often.

mccontrol configure update monitorscale Configures the scaling factor to use on the monitor in monfile. Normal monitors have a efficiency of 1E-6, the McStas monitor counts every neutron. This can be compensated for by this scaling factor. Note that this scaling factor may be dependent on the wavelength used.

mccontrol configure mcmonfile filename This configures the file which mccontrol is going to read in order to watch the simulation control monitor.

mccontrol list Prints a listing of the configuration parameters.

mccontrol run scriptkey Invokes one of the scripts configure for testing purposes. scripkey can be one of: mcstart, mcisrunning, mcdump, mckill and mccopydata.

mccontrol finish This calls waitpid on the PID of the McStas process. This should be done in the mckill script. Otherwise it may occur that the McStas simulation turns into a Zombie process.

Standard scripts for many of the script routines required are provided for the unix environment in the file mcsupport.tcl. Please note, that all system executables called from scripts must be registrered with SICS using the allowexec command.