SPEC Short Manual

Randolf Butzbach

as of

December 4, 2003

Abstract

This document is intended to provide users with a brief overview of the most commonly used commands in **spec**. **spec** provides users with a powerful language to control their experiment. To program in **spec** is easy and fun. First, the most common macros for positioning, counting, scanning and plotting are briefly described on one page. After a brief introduction in command line editing, a list of commonly used macros follows in a more detailed description. Nevertheless, this list is far from being complete. Each beamline has further standard macros for the control of a particular experiment, which are described in a special section. Then a few notes are given to modify the standard macros, to write own macros and 'batches', and how to run them. Even though, it has nothing to do with (but is assumed to be due to) **spec**, a list of frequently observed problems of beamline operation and their solutions is given then. Finally, a list of further help resources is given.

Contents

1 The Most Important Macros **4**

2 How to use the Command Line? **5**

3 Positioning Macros **6**

4 Counting Macros **8**

5 Scan Macros **9**

6 Plot Macros **11**

7 Monochromator Macros **12**
7.1 Traditional Style 12
7.2 Pseudo-Motor Style 12

8 Multichannel Analyzer Macros **13**
8.1 Tutorial 13
8.2 List of macros 14

9 UNIX/File Macros **20**

10 Miscellaneous Macros **21**

11 Beamline-Related Macros **23**
11.1 Absorption Beamline 23
11.2 Fluo-Topo Beamline 23

12 Writing and Modifying Macros **24**
12.1 Macro Commands 24
12.2 How to write my own macros? 26
12.3 How to modify standard macros? 26
12.4 How to write batch files? 27

13 Where to find further macros? **28**

14 Trouble Shooting **29**
14.1 **spec** has crashed 29
14.1.1 My motor does not move. 29
14.1.2 When I press the Enter key, I just get blank lines. 29
14.1.3 I cannot abort a movement or scan 30
14.1.4 `RST Gamma error: Can't setup connection.` 30
14.2 I cannot start spec anymore 31

14.2.1 You have already a valid spec session running 31
14.2.2 You simply closed your **spec** window 31
14.2.3 You have lost remote control of spec 32
14.3 Further problems when starting **spec** 32
14.3.1 `bind(): address already in use` 32
14.4 Workarounds to known bugs 33

15 Further Help-Resources 34

1 The Most Important Macros

All movements and countings can be aborted by typing CTRL-C / STRG-C

`shopen` *[shutter]*

Opens shutter

`shclose` *[shutter]*

Closes shutter

`umv` *motor dest*

Moves motor *motor* to the absolute *dest* (update move).

`umvr` *motor delta*

Moves motor *motor* relative to current position. (update move relative).

`wm` *motor [motor2 motor3 . . .]*

Prints user and dial position of one or several motors (where motor)

`wa`

Prints user and dial position of all motors (where all).

`set` *motor value*

Sets the current user coordinates position of *motor* to *value*

`ct` *[time]*

Start counting on all counters for *time* seconds. (count)

`ascan` *motor start finish interv time*

Scans in absolute coordinates.

`dscan` *motor start finish interv time*

Scans relative to current position.

`newfile` *filename*

Sets a new data file

`cpsetup`

Menu for plot parameters

`plotselect` *[detector . . .]*

Selects detector(s) to be plotted.

`pplot` *[scan_nr]*

Print plot

2 How to use the Command Line?

At ANKA, **spec** is linked with the powerful GNU Readline library, which makes it easy to work with the command line. Here are just a couple of goodies:

- All inputs are savend in a history. You can repeat and edit commands from the history. To see the history, type **`history`** or **`hi`** from the **spec** prompt.
- CURSORUP/CURSORDOWN scrolls in the history list.
- PGUP/PGDN + *string* Scrolls in the history, but filters for *string*. *E.g.* when you typer `ascan` and then press repeatedly the PGUP-key, you will see all your **`ascan`**s from the past. Clearly, when you make an **`ascan`**, the change some parameters, issue **`ct`**, **`wm`**, *etc* , and then you want to repeat the last **`ascan`**, simply type `as`+PGUP and you are back to your last **`ascan`**. You want to have the last but one **`ascan`**? Press one more time PGUP.
- The TABULATOR-key expands your abbreviated input to the full length. Works for filenames and for the arguments of many (internal) **spec** commands.
- CTRL-R *substring* CTRL-A: After pressing CTRL-R Readline searches online for your *substring* in the whole history. If you want to edit this line before relaunching this command, press CTRL-A first.
- The exclamation mark is substitued by
  - **`!!`** – the previous command
  - **`!`***string* – The most recent command starting with *string*.
  - **`!`***linenr* – line number *linenr* in the history.
  - **`!?`***string* – the most recent command containing string
  - **`!$`** – the last word in the previous command
  - **`!^`** – the first word in the previous command

There are many more features in readline. Check out the **spec** manual with `h readline` from the **spec** prompt.

Note: You can use all of these tricks on your Linux command line, too!

3 Positioning Macros

`mv` *motor dest*

Moves motor *motor* (absolute) to the *dest*

`mvr` *motor delta*

Moves motor *motor* by *delta* mm or degrees

`umv` *motor dest*

Like **`mv`**, but the current position is live shown. Useful when moving should be aborted by CTRL-C.

`umvr` *motor delta*

Like **`mvr`**, but the current position is live shown. Useful when moving should be aborted by CTRL-C

`wm` *motor [motor2 motor3 . . .]*

Prints user and dial position of one or several motors incl. upper and lower limit

`wa`

Prints user and dial position of all motors (where all).

`wu`

Same as **`wa`**, but prints only user positions of all motors.

`wsl1` *[`all`]*

Where slit 1. Macro to display the positions of slit (pseudo) motors in a clear style. Similar macros are available for any other slit (**`wsl2`**, **`wsl3`**, etc). If the keyword `all` is given, also limits and dial values are shown. Actually, any argument will do the trick.

`tw` *mot [mot2 ...] delta [delta2 ...] [count_time]*

tweaks motors around the current position by *delta* by typing ENTER.

`twct` *motor stepwith [count_time]*

tweaks the motor around the current position by *stepwidth* by typing ENTER while showing the current count rate accumulated in *count_time*. When count time is omitted, the default time is assumed.

joy *motor stepsize*

moves the motor joystick-like using the K and L key.

lm *[motor motor2 . . .]*

Shows the limits of *motor*. If *motors* is omitted, the limits of all motors are shown.

set_lm *motor low_limit high_limit . . .*

The **set_lm** macro is used to establish the low limit and high limit for one motor in units of the user positions.

set *motor value*

Sets the current user coordinates position of *motor* to *value*

set_dial *motor value*

Overwrites the dial (motor controller) position. After a **set_dial**, the absolute position of the axis is usually lost, incl limits! What you want, is probably a **set**, which sets only the user coordinates of your axis, keeping all hard limits in mind.

home *motor [***+**|**-***] [home_pos]*

Executes homing on motor *motor*. You can see the current position (according to the controller) by **umw**

Example:

```
home sl1l; uwm sl1l
```

Executes homing on slit 1 left blade and shows updated controller register

CEN

Variable, which contains the center of the FWHM of the last scan; can be used in combination with **mv**:

Example:

```
mv th CEN
```

Moves th to the center between the upper and the lower position of the half maximum value of the last scan.

COM

Same as **CEN**, but contains the center of mass of the last scan.

usersave

Saves the current motor positions in a file called recover.mac and defines a macro **recover** to recover them later.

4 Counting Macros

`ct` *[time]*

Start counting on all counters for *time* seconds. If *time* is omitted, counting is started for one second If *time* is negative, counting to $|time|$ monitor counts is enabled. *I.e.* the counters will count until the monitor has seen $|time|$ counts.

`uct` *[time]*

As ct, but the elapsed time and accumulated counts are displayed life.[1]

[1] Not possible at beamlines where the counter is controlled via Gamma.

5 Scan Macros

There are many different scan commands arround for almost all purposes. Here, only the very standard scan commands are listed. All scans can be aborted by typing CTRL-C. When time is negative, the counts are acquired until the detector declared as `monitor` has acquired as many counts as given by $|time|$.

`ascan` *motor start finish interv time*

> scans motor *motor* from *start* deg/mm to *finish* deg/mm (absolute coordinates) in *interv* intervals. Each point is counted for *time* seconds.

`dscan` *motor start finish interv time*

> scans motor *motor* from *start* deg/mm to *finish* deg/mm relative to the current position in *interv* intervals. Each point is counted for *time* seconds.

`a2scan` *motor1 start1 finish1 motor2 start2 finish2 interv time*

> a2scan scans two motors, as specified by *motor1* and *motor2*. Each motor moves the same number of intervals with starting and ending positions given by *start1* and *end1*, *start2* and *end2*, respectively. The step size for each motor is $(start-end)/intervals$. The number of data points collected will be $intervals+1$. Count time is given by time, which if positive, specifies seconds and if negative, specifies monitor counts.

`d2scan` *motor1 start1 finish1 motor2 start2 finish2 interv time*

> d2scan scans two motors, as specified by *motor1* and *motor2*. Each motor moves the same number of intervals. If each motor is at a position X before the scan begins, it will be scanned from $X+start$ to $X+end$. The step size for each motor is $(start-end)/intervals$. The number of data points collected will be $intervals+1$. Count time is given by time, which if positive, specifies seconds and if negative, specifies monitor counts.
>
> Upon termination, the motors are returned to their starting positions.

`a3scan, a4scan a5scan`

> Analogue **`a2scan`**, but scans three, four, five axes, respectively.

`d3scan, d4scan`

> Analogue **`d2scan`**, but scans three and four axes, respectively.

`mesh` *motor1 start1 end1 intervals1 motor2 start2 end2 intervals2 time*

The mesh scan traces out a grid using *motor1* and *motor2*. The first motor scans from *start1* to *end1* using the specified number of intervals. The second motor similarly scans from *start2* to *end2*. Each point is counted for for *time* seconds (or monitor counts).

The scan of *motor1* is done at each point scanned by *motor2*. That is, the first motor scan is nested within the second motor scan.

A mesh scan creates only one scan entry in the spec data file with a total of (intervals1+1)*(intervals2+1) points.

`timescan` *[counting_time [sleep_time]]*

starts the counters in *sleep_time* intervals for *counting_time* seconds. If *counting_time* is ommitted, the default time of 1 second is assumed.

`rscan` *motor start1 end1 interv1 [start2 end2 interv2 ...] time*

allows users to define various measurement density regions among the scanned area; each region is assigned its particular size and intervals number.

`lup` *motor start finish interv time*

Same as **`dscan`**, but executes a **`mv`** *motor* **`CEN`** after the scan instead of moving to the start position (line up).

6 Plot Macros

`newfile` *filename*

Sets a new data file

`newsample` *description*

Defines a new sample description

`setplot`

Prompts user for plot parameters like on-line update, lin-/log-plot, error bars, etc *for the live plot*.

`plotselect` *[detector . . .]*

Selects detector(s) to be plotted. If *detector* is omitted, the user is promped for her choice.

Example:

```
plotselect ion1 ion2 ion3
```

Plots first second and third ionization chamber.

`cpsetup`

Menu to set up the parameters (like titles, data files..) for C-Plot, used by **`cplot`** and for printing using **`pplot`**.

`cplot` *[scan_nr]*

Plot a graph from a datafile on the screen. If *scan_nr* is omitted, the last scan is used.

`pplot` *[scan_nr]*

Plot a graph from a datafile on the printer. If *scan_nr* is omitted, the last scan is used.

`plot3d, contour, pplot3d, pcontour`

Analogue to **`cplot`** and **`pplot`**, respectively, but makes a 3D or contour plot from a data file.

7 Monochromator Macros

7.1 Traditional Style

Traditionally, the monochromator is driven by a special set of macros:

`getE`

Returns currently set photon energy.

`setE` *energy*

Defines current photon energy in keV.

`moveE` *energy*

Moves monochromator to *energy* in keV.

`Escan` *start end interv time*

Executes an energy scan from *start* to *end* energy in *interv* intervals.

`setmono` *[d-spacing] [offset]*

sets the *d*-spacing and the vertical beam offset of the DCM. If invoked without arguments, the user is prompted for the values. If the *d*-spacing is given as zero, the user is asked for the crystal type and the Miller-indices. The settings can be saved by using the **`sav_mono`**. Might be disabled to normal users.

7.2 Pseudo-Motor Style

Energy and wavelength can be used like a physical motor, *i.e.* using the ususal **`mv`**, **`mvr`**, **`umv`**, **`set`**, **`lm`**, **`ascan`**, ... macros.

Currently, only the pseudo motor **`E`** is available, which represents a fixed exit setting of the monochromator in photon energy. The unit is keV.

Example:

```
mv E 12.4
```

Moves the monochromator to 12.4 keV (1 Å).

8 Multichannel Analyzer Macros

8.1 Tutorial

1. If not yet running, start **spec**.

2. Start **`PyMca &`**, a GUI for the MCA, from your Linux prompt.

   Attention about the usage of the Graphical User Interface: it is at the moment only capable of monitoring the data acquired, as there is no connection between **spec** and that GUI, appart from the spectrum data itself and its calibration parameters, both stored in shared memory. For instance, the ROIs defined from **spec** or from the GUI are not connected together. They are two seperate sets of ROI definitions. The GUI is not capable of sending control commands to the hardware. Be more particularly aware that the acquisition times are not known by the GUI, so if you save the spectrum from the GUI, they wont be included in the file. rather use **spec** for saving your data. The GUI is a display.

3. Run **`macsetup`** and select the desired parameters.

4. Define the desired ROIs usinge **`mcaroi`**. The names of the ROIs are used as reference marks for the definition of pseudo counters.

```
ROI       |   Channels    | calibration                 | ROI  | Roi
names     | first | last  | min          | max          |ACTIVE| numbers
---------------------------------------------------------------------
0         | 0     | 8191  | 0.0000       | 8191.0000    |  X   | 0
one       | 0     | 1000  | 0.0000       | 1000.0000    |  -   | 1
two       | 100   | 160   | 100.0000     | 160.0000     |  -   | 2
all       | 0     | 8191  | 0.0000       | 8191.0000    |  -   | 3
```

5. Start the configuration editor (**`edconf`**). Type **`S`** (upper case S) to invoke the scalers page. Insert the defined ROIs as pseudo counters (here: `one`, `two`, `all`):

```
   Scaler (Counter) Configuration

Number        Name  Mnemonic  <>Device  Unit  Chan   <>Use As  Scale Factor
     0     Seconds       sec   ANKA_SC     0     3   timebase         1e+06
     1     Monitor       mon   ANKA_SC     0     0    counter             1
     2    Detector       det   ANKA_SC     0     1    counter             1
     3   e-Current     ecurr      NONE     0     4    counter             1
     4         One       one      NONE     0     5    counter             1
     5         Two       two      NONE     0     6    counter             1
     6         ALL       all      NONE     1     0    counter             1
```

6. Acquire data with **`ct`** and/or any scan command (*e.g.* **`ascan`**).

7. Alternatively use **`mcaacq`** *[time]* to acquire data. Depending on your settings in **`mcasetup`**, the data are saved automatically or you have to save your data manually using **`mcasave`**.

8.2 List of macros

This subsection is taken directly from the original ESRF documentation.

`mcasetup` *[adc] [tmode] [gsz] [g] [cal] [Xlbl] [Xun] [nclr] [sleep] [bgnd] [gui] [log] [dots] [lines] [ebars] [flag] [red] [synchr]*

Sets the MCA hardware and software up. You can either give all the parameters on the command line or use the menu that shows up when typing **`mcasetup`** alone.

Input arguments description. `[variable name, default value]`

adc: active adc number (1 or 2). `[MCA_ADCNO, 1]`

tmode: 1 if live time, 2 if real time. `[MCA_TMODE, 2]`

gsz: MCA memory group size in channels. `[MCA_MEMGRPSIZE, 8192]`

g: MCA selected memory group. `[MCA_MEMGRP[1] , 0]`

cal: 1 if spectrum is calibrated, 0 otherwise. `[MCA_ENERGY, 0]`

Xlbl: X plot label for calibrated spectrum. `[MCA_CP_XLBL, ‘‘Energy’’]`

Xun: X unit for calibrated spectrum. `[MCA_CP_XUNT, ‘‘KeV’’]`

nclr: 0 for memory clearing prior to acquisition, 1 otherwise. `[MCA_SAVEBUFFER, 0]`

sleep: number of sec. sleeping between run plot updates. `[MCA_DISP_SEC, 0]`

bgnd: 1 if you want background substraction in ROIs counts. `[MCA_BACKSUB, 0]`

gui: 1 for gui plot. `[MCA_GUI, 0]`

log: 1 for log plot. `[MCA_LOG, 0]`

dots: 1 for large dots in plot. `[MCA_DOTS, 0]`

lines: 1 for lines plotting. `[MCA_LINES, 0]`

ebars: 1 for error bars plotting. `[MCA_EBARS, 0]`

flag: data saving flag `[MCA_FLAG, 0]`

`0x01` save spectrum during scans to a file.

`0x02` save spectrum after each other acquisition to a file.

`0x04` save to specific MCA file, otherwise use scans file `[DATAFILE]`.

red: data reduction coefficient, to be applied at saving.`[MCA_REDUCTION, 1]`

synchr: MCA acquisition is hard-synchronized with **spec** configured timer. `[MCA_SYNCHRO, 0]`

Note: It is possible to have the last acquisition elapsed times values reported in counters by simply configuring the counters you need out of mcaLt , mcaRt and mcaDt mnemonics, respectively standing for Live, Real and Dead times.

`mcaoff`

Disables the MCA. The MCA is no longer addressed during standard **spec** operation (**`ct`**, scans ..), but still from **`mcaacq`**.

`mcaon`

Enables back the MCA.

`mcainit`

Initialises the MCA macros and hardware. Can be called at startup to re-setup things without giving any parameter, nor calling any menu. The current or default parameters are then set. Default are mentionned together with **`mcasetup`** description.

`mcaacq` *[preset_seconds] [roi | first] [last]*

Acquires data with the MCA only. The rest of counters configured in **spec** are not addressed by this command. If you need it, use standard **`ct`** command. (don't forgot, **`mcasetup`** set AUTO-RUN mode ON, or **`mcaon`**). **`mcaacq`** consists in the following sequence:

- stopping any already running acquisition.
- clearing (if setup) the active MCA memory group.
- preseting the time.
- starting the acquisition.
- polling the device till time is over.

meanwhile:

- reading the MCA buffer.
- plotting the data spectrum.

Saving: (if set-up) the acquired spectrum to disk file.

The MCA memory is read within the range specified, or if none, the active ROI. If the requested preset time is 0 or un-specified, the acquisition is started "for ever"; press CTRL-C or "s" or "q" to stop it. Pressing "c" would clear the memory on the fly. If the GUI is not active, you can also change the plot update intervals (press "u"), change the plot *x* or *y* ranges (press "x" or "y"), integrate counts on region (press "i"), toggles plot attributes such as lines (press "l"), large dots ("d") and log*y* ("g"). Meanwhile, the acquisition goes on.

`mcaclear`

Clears the active MCA memory group.

`mcastart` *[preset_seconds]*

Presets and starts an acquisition.

`mcastop`

Stops acquisition if running.

`mcahalt`

Send “stop acquisition” signal whatever is the status of the device. It might happen that the device server gives a meaningless error message, if the status is already “IDLE”.

`mcaread` *[roi | first] [last]*

Reads the MCA memory within the range specified.

`mcatimes`

Reports the MCA elapsed live, real and dead times.

`mcastat`

Prints out a detailed status of the **spec** MCA registers.

`mcaroi` *[nr | name] [first last] [name]*

ROIs definition.

detail of usage:

`mcaroi nr first last counter`

modifies ROI number *nr*.

`mcaroi nr`

sets ROI *nr* as the active one.

`mcaroi nr counter`

associates name ”counter”, which is eventually a counter mnemonic in config, with ROI number *nr*.

`mcaroi`

calls the ROI menu.

Each ROI is referrenced to by a number that is set by the software. You can use the ROI names to referrence them as well. The active ROI is used as the default operation range when no other range is specified. ROIs which names are configured counter mnemonics are automatically set as pseudo counters to report integrated counts value over the corresponding ROI.

`mcanewfile` *[file_name]*

Sets a new file prefix and initialises the spectrum run number, which appears in the file name as e.g. `PREFIX_001.mca`. But when spectrum was taken during a scan, the file name changes for `PREFIX_001_000.mca`, where first number is the scan number (`SCAN_N`) and second one the point number in the scan.

`mcasave` *[roi|min] [max] [`-silent`]*

Saves acquired data to disk file. If a range is specified, the data are saved within that range, otherwise the whole spectrum is dumped to the file. The file is either the standard scanfile or a private MCA file, depending on **`mcasetup`** entries. If standard scanfile is used, each spectrum originated from out of a scan (**`ct`** or **`mcaacq`**) is nevertheless referenced to a ”scan number”. (`SCAN_N` is incremented). If a private file is used, it must have been set a prefix using **`mcanewfile`** command. Then each spectrum is saved to a different file. The file name extensions gives the information on its origin. see mcanewfile . The file syntax is standard and MCA mixed. MCA syntax is described below:

Each MCA related line holds the character “@”.

MCA file syntax:

`#@MCA 16C`

Format string passed to **`data_dump()`** function. This format string is held by the global variable `MCA_FMT` and can then been adapted to particular needs. `16C` is the default. It dumps data on 1 line, cut every 16 points:

```
@A 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0\
 0 0 0 0 0 0 0 0 0 0 0 ...
```

`16` would do the same without any backslash, `1` would dump 1 point per line, ...

`#@CHANN 1024 0 1023 1`

number of data points, first MCA channel, last one, reduction factor.

`#@CTIME 1 17 17`

Time preset, MCA elapsed live time, MCA elapsed real time.

`#@CALIB 0 1 0`

Calibration parameters as $a\ b\ c$, in $E = a + b \times \text{channel} + c \times \text{channel}^2$.

`@A 0 0 0....`

MCA data. Each value is the content of one channel, or an integrated value over several channels if a reduction was applied.

Data reduction is useful in some cases to minimize file sizes, which might grow very fast and eventually fill up the disk. It consists of:

- averaging counts every *factor* points.
- multiplying by *factor* each integer average to get an integrated value.

When `-silent` the macro does not ask for a user comment.

`mcaload` *file scan_no [scan_point_no] [0|1]*

Loads spectrum from a file. The loaded spectrum is stored into WS memory as if it would have just been acquired. If file is a standard scan file, spectra are referrenced to by their *scan_no*, and eventually their *scan_point_no*. Both numbers must be 0 when loading data from private MCA files. 4th argument to 1 means that the calibration parameters are to be loaded from the file as well. In that case any other current calibration is overwritten. To avoid it, **`mcasave`** it with the current spectrum, before **`mcaload`**ing. Reduced data channel numbering starts from the first channel value (prior to reduction) added integer part of half the *factor*, every next point incremented by *factor*.

`mcapar`

Shows calibration fit parameters or inputs new parameters.

`mcaE`

Toggles between channel (uncalib) and calibrated mode.

`mcacal`

Computer aided energy calibration of the MCA.

`mcacalm`

Manual calibration of the MCA.

`mcapeak`

Peak search on the current spectrum.

`mcashow`

Shows peaks found.

`mcacplot` *[first|roi] [last]*

A screen plot of the current MCA data in cplot format. Part of the plot parameters are taken from **`mcasetup`**, rest from **`cpsetup`** settings.

`mcapplot` *[first|roi] [last]*

A printer plot of the current MCA data in cplot format. That follows the same rules as **`mcacplot`** macro.

`mcaguion`

Turns MCA Graphical Interface on.

`mcaguioff`

Turns MCA Graphical Interface off.

`mcasplot` *[xmin] [xmax] [ymin] [ymax]*

Plots the current MCA data spectrum. Does nothing when using GUI. Detail of usage:

`mcasplot first last [ymin ymax]`

plots acquired spectrum between the limits specified. Eventually, *ymin* and *ymax* are Y limits values so that you may Y-rescale your plot.

`mcasplot roino [ymin ymax]`

plots acquired spectrum within ROI specified.

`mcasplot`

plots the whole acquired spectrum

9 UNIX/File Macros

`newfile` *filename*

Sets a new data file name

`fon` *filename*

switches logging of all screen input/output in file *filename* on. **`foff`** switches logging off again.

`foff` *filename*

switches logging of all screen input/output in file *filename* on.

`cd` *directory*

Change directory. Without argument, cd change to the home directory

`pwd`

Print working directory

`ls` *[reg_exp]*

list files in working directory

`l` *[reg_exp]*

Abbrevation for **`ls -l`**; list files long (detailed) form in working directory

`unix("`*command***`"), u`** *command*

sends any command to the UNIX operting system

10 Miscellaneous Macros

`shopen` *[shutter]*

Tries to open shutter *shutter*. If *shutter* is omitted, the last used shutter is opened. If opening is forbidden by the safety system, a message is shown. *shutter* is one of the following mnemonics (aliases are given in parentheses):

Absorber	:	**`abs`** (**`0`**)
Front-End Shutter	:	**`fs1`** (**`1`**, **`fs`**)
Safety-Shutter	:	**`fs2`** (**`2`**, **`zs`**)

`shclose` *[shutter]*

Closes shutter; if *shutter* is omitted, the last used shutter is closed. Mnemonics are same as in **`shopen`**.

`shstate`

shows the state of all shutters.

`init_gamma`

Initializes all RST Gamma ANKA Motors on the OS/9 System. Necessary only after a system reset of the VME crate.

`restore_gamma`

Restores the RST Gamma parameters from Gamma's database. Useful when working with other measurement software which relies on Gamma's parameters: Issue a **`restore_gamma`** before leaving **spec** and starting the third-party application.

`reset_gamma`

kills and restarts all notorious problematic processes of Gamma (currently: network and CAN-bus). Requires a responding TCP/IP network on the OS/9 side, which might be disabled from the Gamma modules, too. The communication output is written to `/tmp/reset_gamma.log`

`cryforhelp`

Sends an e-mail to computing services for assistance.

`print, p` *expression*

prints *expression* to the screen. Expression can be also a mathematical expression, so that you can use **spec** as a calculator:

Example:

```
74.SPEC> p dhkl=5.431/sqrt(pow(3,2)+pow(1,2)+pow(1,2))
1.63751

75.SPEC> p deg(asin(1./(2*dhkl))), "deg"
17.7787 deg
```

Calculates the Bragg angle in degrees for the Si(311) reflection and 1 Å wavelength

`comment, com` *comment*

Writes comment to data file.

`sleep` *time*

Sleeps for *time* seconds.

`config`

Invokes the config-editor.

`history, hi`

shows the last submitted commands.

Commands can be repeated by

`!` *linenr*

or

!*abbrevation*

Examples:

`!156` Repeats command number 156

`!dsc` Repeats last command which starts with **`dsc`** (*e.g.* **`dscan ...`**)

For a full description of what is possible see `h readline`

`sync`

Synchronizes **spec**'s internal motor data base of with the controller values. If there is a discrepancy, spec prompts the user whether the controller (dial) value should be overwritten. If answered with no, **spec** aligns its database to the dial value.

`emac` *macroname*

Edits the existing macro *macroname* with the default editor (defined in the variable `EDITOR`) and loads the changed macro in **spec**..

11 Beamline-Related Macros

11.1 Absorption Beamline

`kscan` *energy_motor edge_energy time*

Executes a scan around the absorption edge at *edge_energy*. The detailed scan can be defined by using the **`kscan_setup`** macro. Default values can be used using the **`kscan_default`** macro.

Currently, only **`energy_motor`** is available: **`E`** executes a fixed exit scan.

11.2 Fluo-Topo Beamline

`fsopen` *[time]*

opens the fast-shutter at the topography station for *time* seconds. If *time* is omitted or equal zero, the shutter is opend for infinit.

`fsclose`

closes the fast-shutter at the topography station after being opend for infinit.

`fsstate`

shows the state of the fast-shutter and the safety-shutter

12 Writing and Modifying Macros

Almost all commands you issue to **spec** are macros. The **spec** programming language is very similar to the C–language; so most of the people will feel immediately at home. This section describes the differences to C and things you need to know for programming **spec**. For help, it is a good idea to look in the definition of other macros using `prdef`. For a detailed description of the **spec** language refer to the **spec**-Homepage (`www.certif.com`)

12.1 Macro Commands

`prdef` *macro*

Prints the definition of *macro*

`def` *macroname* `'` *statememts* `'`

Defines *statements* as macro.

`cdef("`*macroname*`",`*statememts [,key [,flags]]*`)`

Chain defintion: Appends *statements* to an (existing) macro *macroname*. With the optional *key* argument, the pieces can be selectively replaced or deleted, *i.e.* by using the *key*, parts can be later accessed. The *flags* argument controls whether the pieces are added to the front or to the back of the macro or whether the pieces should be selectively included in the definition based on whether key is a currently configured motor or counter mnemonic. The bit meanings for *flags* are as follows:

0x01 : only include if key is a motor mnemonic and the motor is not disabled.

0x02 : only include if key is a counter mnemonic and the counter is not disabled.

0x10 : place in the front part of the macro.

0x20 : place in the back part of the macro.

If flag is the string **”delete”**, the piece associated with *key* is deleted from the named macro, or if the name is the null string, from all the chained macros. If *key* is the null string, the FLAGS have no effect.

For an introduction in using the `cdef` function see the ESRF tutorial (`http://www.esrf.fr/computing/bliss/tutor/spec.html`).

`undef` *macro*

Undefines *macro*

`lsdef` *[reg_exp]*

lists all macros currently known to **spec**.

`qdo` *file*

Executes macro/batch/script from file *file*. Also to be used to load a macro, if *file* contains a macro definition, *i.e.* the definition of a macro is executed.

`savmac` *macro_name file_name*

Write macro *macro_name* to file *file_name*

`jtdo("`*macro***`")`**

Loads the macro *macro* from the installed standard macro collection. **`jtdo`** searches first in the system user directory, then in the beamline directory, then in the general ANKA directory, and finally in the ESRF macro directory.

Note, that we have most of the ESRF macros installed here, too. Many of them can be used out of the box, others contain special ESRF commands (**`esrf_io`**) or hardware and are of no direct use at ANKA. However, they can be useful as a template or to get an idea for your own macros.

`emac` *macroname*

Edits the existing macro *macroname* with the default editor (defined in the variable `EDITOR`) and loads the changed macro in **spec**..

`moredef` *macroname*

Like **`prdef`** but uses a pager like more to display the macro definition. You can define `PAGER` to change the page to less or something else.

`#`

Starts a comment until the end of the line

`$#`

Replaced by the number of arguments given.

`$1`

Replaced by the first argument given, when macro is invoked.

`$*`

Replaced by all arguments given, when macro is invoked.

12.2 How to write my own macros?

1. Open/create a file with your favourite editor. From the **spec**-prompt, you can use *e.g.* `vi mymacro.mac`. If you are not familiar with `vi`, try something like `nedit`, `emacs`, `xemacs`, `joe`, `pico`, . . .

2. Write the macro starting with the keyword **`def`**, then give the macro name and enclose your code in **`' '`**, *e.g.*

   ```
   def hello '
       print "Hello World!"
   '
   ```

3. Save the file

4. Load the file in **spec** with **`qdo`**, *e.g.*

   ```
   1.SPEC> qdo mymacro.mac
   ```

5. Run the macro from the **spec**-prompt:

   ```
   2.SPEC> hello
   Hello World!
   ```

12.3 How to modify standard macros?

1. Search for the desired position to be changed, using the **`prdef`** command. Example (to change the update value format when using **`umv`**):

   ```
   3.SPEC> prdef umv

   # SPECD/standard.mac
   def umv '_mv $*; _update("$1")'
   ```

2. Save the standard macro to your own file, *e.g.* using the **`savmac`** macro: Example:

   ```
   4.SPEC> savmac _update myupdate.mac
   ```

3. Proceed with Section 12.2.

Hints:

- The macro **emac** (edit macro) does all of this on the fly:

  Example:

  ```
  emac _update
  ```

  Creates a temporary file, opens an editor (defined in the variable EDITOR), and loads the macro afterwards.

- To restore the original **spec** macros, type **newmac**.

12.4 How to write batch files?

The description of how to write macros in Section12.2 describes actually already how to write a batch file. The command **qdo** *file* executes everything what is written in that given file: After writing our macro in Section 12.2 we have executed the macro definition. However, we can put anything else instead of or additionally to the macro definition. The file could read for instance like this:

```
def hello '
    print "Hello World!"
'
hello
ascan th 0 10 20 1
mv th CEN
dscan th -1 1 100 3
pplot
```

First we define the hello macro (def ...), which we execute immediately after definition (hello). After the greeting, we do a coarse absolute scan (ascan ...) of th from 0 to 10 deg. Next we move th to the center of the reflection (mv th CEN) and then we continue with a fine scan around the center position at which th is now positioned (dscan ...). And finally, we send the plot to the printer (pplot).

13 Where to find further macros?

`/usr/local/lib/spec.d/local/user` – where you can put your own macros to provide them to other users

`/usr/local/lib/spec.d/local/BL` – Beamline specific macros

`/usr/local/lib/spec.d/local/anka` – ANKA standard macros

`/usr/local/lib/spec.d/local/esrf` – ESRF standard macros

http://www.esrf.fr/computing/bliss/spec/local

14 Trouble Shooting

Please note: *__spec__ has proved to be very stable. Problems that occur are likely not to be due to __spec__, but due to the underlying Gamma control system. Moreover, __spec__ is designed to be failsafe. I.e. the idea of simply closing your __spec__ window will not help but rather bring you in deeper troubles!*

14.1 spec has crashed

Are you sure, that it is really **spec** which crashed? Probably, the underlying control system Gamma has crashed und you just see **spec** waiting for the underlying control system, which does not respond anymore. Check out the list below for detailed help.

14.1.1 My motor does not move.

There are several reasons for this phenomenon:

- **You control the motors via Gamma and have forgotten to initialize the motor driver:**

  Enter **`init_gamma`** from the **spec** prompt. Depending on your beamline, you might need to wait about one minute for the sysload on the OS/9 computer to decrease. If you continue your work too soon, Gamma will crash and you need to reset the whole system.

- **Your Gamma driver for your motor controller has crashed.**

  Reset your Gamma Motor Driver as described in Section14.1.2. Ask your beamline scientist for assistance.

- **Your McLennan PM595 Motor Controller has crashed.**

  Reset your PM595 Motor controller. Ask your beamline scientist for assistance.

- **Due to a hardware exception, both limit switches appear to be active and spec does not inform you about this state.**

  Check out, whether all cables are mounted and the opto-couppler board is switched on. Check the communication between **spec** and your motor controller (switch on debugging with **`debug 192`**, switch back to normal operation with **`debug 0`**).

14.1.2 When I press the Enter key, I just get blank lines.

Probably, your CAN-Bus interface in Gamma has crashed. Try to run **`reset_gamma`** from your Linux prompt. **`reset_gamma`** will kill all problematic processes on your underlying control system and restart them. You also have to run **`init_gamma`** from your **spec** session afterwards, in order to be able to use your motors again.

If this does not help, ask your beamline scientist for assistance.

14.1.3 I cannot abort a movement or scan

You have typed (several times) CTRL-C, but nothing happend or **spec** replies with

```
Waiting for motors to stop.
Still waiting.
```

or something similar. Probably, your motor driver in the control system or your motor controller (likely at McLennan PM595) itself hangs. In the former case, you need to reset the driver in Gamma as described in Section 14.1.2. In the latter case, you need to reset the motor controller. In both cases, ask your beamline scientist for assistance!

14.1.4 `RST Gamma error:  Can't setup connection.`

You get an error message like:

```
RST Gamma error:  Can't setup connection.
RST Gamma unavailable.
```

You have lost your network connection to the Gamma control system. (Perhaps you have closed and restarted your **spec** session without waiting at least 15 seconds.) Try the following:

- **Wait for at least 2–3 minutes** (check with a watch!) – Then try again. Sometimes the network interface becomes available again.
- **Restart the Gamma network interface automatically** (works only, if you still can make a `telnet` connection to the OS/9 computer:
  1. Run `reset_gamma` from the Linux prompt. If this does not work, skip the rest of this item and proceed with “Restart the network interface manually”.
  2. Run **`anka_par("reconnect")`** from the **spec** prompt.
  3. Run **`init_gamma`** from the **spec** prompt.
- **Restart the Gamma network interface manually** (You might need assistance from your beamline scientist):
  1. Open a terminal window on the OS/9 computer (klick with the mouse in the backdrop) and enter the password (ask your beamline scientist).
  2. Find out the node- and process number (PID) of your network interface (note that the UNIX “`|`” is a “`!`” in OS/9):

```
GAMMA: procs ! grep CN
103   0   0.0     128  112.00k   0 s 16:08:03.80 1294:21 CNetMan <>>>nil
106 103   0.0     129   80.00k   0 s 79:41:28.46 1294:21 CNETTCP_309 <>>>nil
```

Here, `103` and `106` are the PIDs and `309` is the node number.

3. Type the following, where you replace the example PIDs and Node numbers by your respective PIDs and Node number:

```
GAMMA: setenv LNODE 309
GAMMA: kill 103 106
GAMMA: cnetman <>>>/nil &
```

4. Run **`anka_par("reconnect")`** from the **spec** prompt.

14.2 I cannot start spec anymore

You want to start **spec** and you get something like

```
Can't lock state file "/usr/local/lib/spec.d/spec/userfiles/sul_ttyp#L".
Are you already running on this terminal or another virtual tty?
```

There are several causes that you can not start **spec** anymore:

14.2.1 You have already a valid spec session running

Close your running **spec** session first or continue with your running **spec** session: There are several beamline control components which can be used by one process (*i.e.* **spec** session), only. Thus you can not run more than one **spec** session at once which uses the same control components.

14.2.2 You simply closed your spec window

spec is programmed to be failsafe. Closing your **spec** window will *only* close the window while **spec** considers this as a failure on the user interface side (like the breakdown of a network connection, too) and continues the operation. In other words: when you lost your network connection, your long-run scan will not be aborted.

Unfortunately, you don't have a regular user interface to **spec** anymore. So you have to tell **spec** to terminate by operating system signal:

1. Lookup the process identifier (PID) of your **spec** session:

```
linux:~> ps x | grep spec
18822 pts/6    S      0:00 spec
```

where you need to replace the string `spec` after `grep` by the name of your **spec** session (*e.g.* `fourc`, `optics`, *etc*).

2. Tell **spec** to terminate oridinary:

   The following command will tell **spec** to terminate normally (*i.e.* saving all files *etc*):

   ```
   linux:˜> kill -HUP 18822
   ```

   and check, whether **spec** really has terminated, by repeating the `ps` command above. If **spec**has terminated, stop here. If not, try the next point:

3. Tell **spec** to end:

   ```
   linux:˜> kill 18822
   ```

   Check again whether **spec** is still running or not (*c.f.* first item). If **spec** is still running, try the last alternative:

4. Tell Linux to kill **spec**:

   You should consider this option as the very last way. It is equivalent to switching off your computer. **spec** will not learn about your termination request. Thus, your variables and history *etc* will not be saved!

   ```
   linux:˜> kill -9 18822
   ```

14.2.3 You have lost remote control of spec

You are running **spec** over a network connection (*i.e.* `ssh` or `telnet`) and you have lost/closed this connection. Now, you can not access **spec** anymore:

Log on to the computer on which **spec** is running and proceed with Section 14.2.2.

14.3 Further problems when starting spec

14.3.1 `bind(): address already in use`

You run more than one program (**spec** session? Status display?) on the very same computer which talks to the very same OS/9 computer (Gamma control system).

Sorry, that's not possible. Close one of those programs. Ask your beamline scientist for assistance.

14.4 Workarounds to known bugs

Unfortunately, ANKA's control system and hardware suffer from several well-known bugs. ANKA is striving to get these bugs removed. Until then, there are a couple of work-arounds of these bugs.

Connection to beamline: (Applies, when using RST's Gamma for beamline control) When you close your **spec** session, take care that you *wait at least 15 seconds* before you restart **spec**. Otherwise, the entire network interface of the VME computer will crash and your beamline scientist has to restart the network on location; there is no remote access possible anymore.

Small step discrepancy when moving by small distances: (Applies, when using the McLennan PM595 motor controllers) When you want your motor to move for less then five steps, the McLennan motor controller will return that he is already at this location and will not move. After the movement, **spec** will compare its own target position with the target position reached. Since this discrepancy becomes obvious here, **spec** will ask who is right and who is wrong. Usually, it seems to be a good idea to say no to the question

```
Should the ANKA Gamma Motor controller be changed [spec suggests
NO]?
```

However, it was also observed that the motor controller said that there is nothing to move, but the motor moved nevertheless. In this case, you should say yes, in order to update the controller to the physical position of the motor.

15 Further Help-Resources

- **`help`** *topic*

  Online help from the **spec**-command line: Example: `h readline` – shows the help page for the command line editor. When invoked without argument, a list of topics is shown.

- **`cryforhelp`**

  sends e-mail to computing services.

- `http://www.certif.com`

  The official **spec** home-page

- `http://www.esrf.fr/computing/bliss`

  The ESRF-BLISS Homepage

- `http://www.esrf.fr/computing/bliss/spec/local`

  ESRF-**spec** macros; help and downloading

- `http://www.esrf.fr/computing/bliss/tutor/commands/commands.html`

  spec commands reference list

- `http://www.esrf.fr/computing/bliss/tutor/spec.html`

  Tutorial in **spec**