SPEC@ Short Manual for HXMA

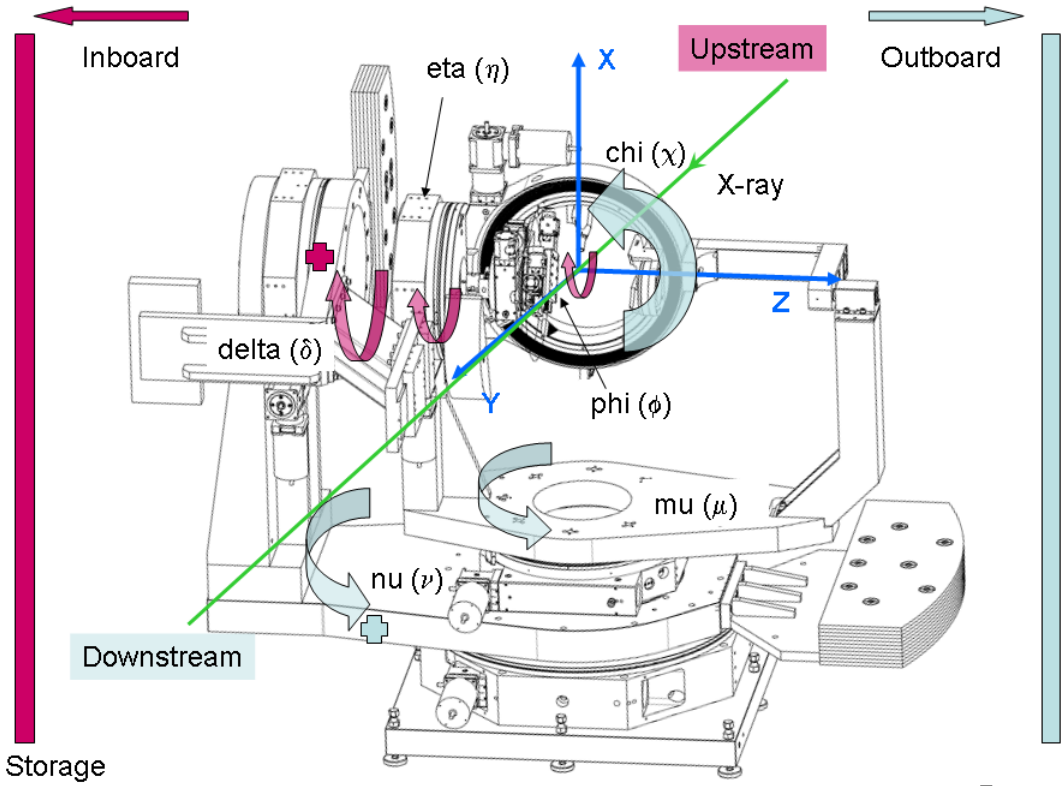
Chang-Yong Kim

Version 0.5 as of August 8, 2017

Abstract

This short manual is intended to provide quick reference for SPEC@ software which is operating the HXMA psi-8 diffractometer. More elaborate manual can be found in the official **spec** home-page http://www.certif.com.

Separate document of "Guide for HXMA diffractometer operation" covers beamline component set-up, diffractometer alignment, and various corrections to the raw data took from HXMA diffraction stage.



HXMA 8-circle psi (Ψ) diffractometer

Contents

1. Frequently Used Macros 3
2. How to use the Command Line? 4
3. Positioning Macros 5
4. Counting XIA filter box Macros 7
5. Scan Macros 8
6. Plot Macros 11
7. Monochromator Macros 12
8. Multichannel Analyzer Macros 13
9. CCD macros 17
10. UNIX/File Macros 18
11. Miscellaneous Macros 19
12. Writing and Modifying Macros 21
    12.1. Macro Commands 21
    12.2. How to write my own macros? 22
    12.3. How to modify standard macros? 23
    12.4. How to write batch files? 23
13. Where to find further macros? 25
14. Trouble Shooting 26
    14.1. I cannot start spec anymore 26
        14.1.1. You have already a valid spec session running 26
        14.1.2. You simply closed your spec window 26
        14.1.3. You have lost remote control of spec 27
    14.2. spec has crashed 27
        14.2.1. My motor does not move 27
        14.2.2. When I press the Enter key, I just get blank lines 28
        14.2.3. I cannot abort a movement or scan 28
    14.3. Workarounds to known bugs 28
15. Further Help-Resources 29

1. Frequently Used Macros

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

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]*

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

plotselect *[detector . . .]*

Selects detector(s) to be plotted.

ascan *motor start finish interv time*

Scans in absolute coordinates.

dscan *motor start_rel finish_rel interv time*

Scans relative to current position.

opsh

Opens shutter

clsh

Closes shutter

newfile *filename*

Sets a new data file

cpsetup

Menu for plot parameters

pplot *[scan nr]*

Print plot

2. How to use the Command Line?

- All inputs are saved 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 type `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*.

mv *motor1 dest1 [motor dest2…]*

Moves multiple motors to *dest*'s .

mvr *motor delta*

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

mvr *motor1 delta1 [motor delta2…]*

Moves multiple motors by *delta*'s.

umv *motor1 dest1 [motor dest2…]*

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

umvr *motor1 delta1 [motor delta2…]*

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 .recover

wu

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

tw *mot [mot2 ...] delta [delta2 ...] [count time]*

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

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.

pl_xMAX

Similar to **CEN** or **COM** but contains the peak position of the last scans display in screen plot; can be used in combination with **plotselect** and **mv**.

4. Counting and XIA filter box 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.

fb_setup *[counter-mne] [min-count]*

`counter-mne` is the mnemonic of the counter to be used

`min-count` is the minimum acceptable count for the counter specified. If the user enters a negative number, it will be interpreted as a percentage of the counter's current count

fbc

this will update the coefficient matrix for the current energy

fbu

this will provide the log10 of the attenuation coefficient for the current filterbox combination, if an argument is provided then it will display the attenuation coefficient for the combination provided

fb_on

this will enable the automatic filterbox configuration during scans

fb_auto

this finds a suitable filterbox combination to bring the counts to the threshold

5. Scan Macros

There are many different scan commands around 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 step 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.* 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 *start* to *end*. The step size for each motor is (*start-end*)/*intervals*. The number of data points collected will be *intervals.* 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.

`xascan` *motor start finish interv time [expansion] [step_ratio]*

Similar to **`ascan`**, but with extended scan ranges.

For the range of *start – expansion* * range ~ *start* the step is *interv/step_ratio*

(range = *finish –start*).

Range for *start* ~ *finish*: interv

finish ~ *finish*+*exten**range: interv/step_ratio

Example: `xascan 1.2 2.2 10 1 2 2`

xa2scan *mot1 s1 f1 mot2 s2 f2 interv time [expansion] [step_ratio]*

Expanded version of the standard a2scan macro.

xa3scan *mot1 s1 f1 mot2 s2 f2 mot3 s3 f3 interv time [exp.] [step_ratio]*

Expanded version of the standard a3scan macro.

xa4scan *m1 s1 f1 m2 s2 f2 m3 s3 f3 m4 s4 f4 interv time [exp.] [step_ratio]*

Expanded version of the standard a4scan macro.

xdscan *motor start finish interv time [expansion] [step_ratio]*

Expanded version of the standard dscan macro. xascan

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.

dmesh

Two-motor relative mesh cscan for relative mesh scans.

Syntax is like dscan but with mesh

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.

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.

`splot` *[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.

7. Monochromator Macros

getE

Returns currently set photon energy.

mve *energy*

Moves monochromator to *energy* in eV.

sa *analyzer_crystal*

Setting the analyzer to use. It will list the defualt analyzers and ask the user to choose one of them (by name). The user can enter a custom name and d-constant of a crystal of their own choosing. Use variable *d_ana* and *d_ana_n.*

amve *energy*

Moves monochromator to *energy* in keV and change analyser *ath* and *atth* to satisfy Bragg condition of the changed energy.

Escan *start end interv time[fixQ]*

Executes an energy scan from *start* to *end* energy in *interv* intervals. If *fixQ* is used diffractometer motor position will trace Bragg condition of starting *hkl* at each energy.

sEscan *start end interv time[fixQ]*

Executes an energy scan from *start* to *end* energy in *interv* intervals **without** tracking of second monochromator crystal positions (y and z positions).

aEscan *start end interv time[fixQ]*

Executes an energy scan from *start* to *end* energy in *interv* intervals **with** tracking of second monochromator crystal positions only in y and **not z positions**). The analyser theta and two theta (*ath* and *atth*) also changed to satisfy Bragg condition of analyser crystal at each energy.

8. Multichannel Analyzer Macros

8.1 Using EPICS MCA inside *spec*

`setup_mca`

Display interface to configure multiple epics MCAs in spec.

`show_mca`

Display the configured epics MCAs nicely.

`mca_rois`

Automatically check the created ROIS in each MCA and display their parameters.

`save_rois`

Turn on/off the feature of saving MCA_ROIS(defined in roi_pv[], and filled when mca_rois is called)

`rm_mca`

Remove one or all configured MCAs,also remove all rois related to that MCA

`rm_roi`

Remove one/all configured MCA ROIS.Can also rmove all rois related to one MCA by calling rm_roi mca_nme

`getandsave_mca`

Start counting and save MCAs and save their spectra in spec file.

`MCAscanpt` *`anyscan-macro`*

Save the MCA spectra in a separate file on each scan point of whatever scan.

Example:

```
MCAscanpt hklscan 0 0 0 0 0.1 1.9 180 1
```

`MCAscanend` *`anyscan-macro`*

Save the MCA spectra at the end of a scan (MCAscanend anyscan macro)

8.2 MCA data file

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*

$a+b*\text{channel}+c*\text{channel}^2$

`@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.

8.3 Notable MCA related PVs

Wherever you see "$(DXP)", replace with "dxp1606-I10-01".

$(DXP):mca1.VAL

the spectrum's histogram content; a list of the value for each bin of the histogram

$(DXP):mca1.R0

counts in Region of Interest (ROI) 0

$(DXP):mca1.R1

counts in ROI 2

...

$(DXP):mca1.R31

counts in ROI 32

$(DXP):mca1EraseStart

Erases all counters and commences compiling a new spectrum and ROI counts

$(DXP):mca1.ACQG

Indicates when integration has completed, either by a user "stop" or a timed "stop".

$(DXP):mca1.READ

Triggers the readout of data from electronics for transfer to data PVs.

$(DXP):mca1.PRTM

Sets the integration time in terms of real time rather than live time.

$(DXP):mca1.PLTM

Sets the integration time in terms of live time rather than real time.

$(DXP):mca1Status.SCAN

Sets the scan period at which EPICS scans the status of the DXP electronics, such as whether it is counting or idling, or what the current dead-time is. The shorter the period the quicker the response to incidental events, but the more overhead the DXP is burdened with. Setting this to "Passive" means you will never know when an integration period has ended. Setting this to "10 second" means you may end up waiting for an end signal up to 10 seconds after the integration ended.

$(DXP):mca1Read.SCAN

Sets the scan period at which EPICS reads data from electronics and transfers to data PVs. This uses $(DXP):mca1.READ. This is not the integration time. It does not start or stop integration, but it can interfere with data acquisition. It merely

represents the extraction of data already buffered in the electronics. If you are triggering a read externally, this should be set to "Passive".

$(DXP):dxp1.EMAX

Sets the equivalent energy of the upper end of the X-axis of the MCA histogram graph.

$(DXP):dxp1.PKTIME

Sets the Energy Peaking Time (microseconds).

9. CCD macros

Please refer user guide **SPEC_with _CCD for details.**

`ccd_setup`

Select CCD to use (only one choice for now) and start setup.

`ccd_on`

Enables CCD image acquisition through spec. Macro *ct* will take CCD image and save it. Any scan will take CCD image and save it at each scan point.

`ccd_off`

Disables CCD acuisition. Counting and scanning will be regular one without CCD acquisition. **For sample alignment (height/angle adjustment), usually it is not necessary to take CCD.**

`ccd_setup`

select CCD to use (only one choice for now) and start setup.

10. 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

11. Miscellaneous Macros

opsh

Opens sOE photon shutter. Please remember that sOE should be ready to open shutters.

clsh

Closes sOE photon shutter.

fsopen

Opens fast shutter until **fsclose** excuted. Fast shutter used in conjunction with CCD and opens during counting. The fast shutter remains closed normally. During alignment of beamstop it is desired to open the fast shutter all the time. **fsopen** is used for this purpose. Remember to revert to normal fast shutter operation by close the fast shutter with **fsclose.**

fsclose

Closes fast shutter.

print, p *expression*

prints *expression* to the screen. Expression can be also a mathematicalexpression, 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 °A 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.

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*

`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?

`/home/spec/lib/user`– where you can put your own macros

to provide them to other users

`/home/spec/lib/site.mac` – HXMA specific macros

`http://www.esrf.fr/computing/bliss/spec/local` ESRF standard macros

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. I cannot start spec anymore

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

```
Can't lock state file "/home/spec/lib/psic/userfiles/spec_ttyp#L".
Are you already running on this terminal or another virtual tty?
```

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

14.1.1. 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.

14.1.2. 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 (there will be typically 9 processes)

```
bash-2.05$ ps x | grep psic
18822 pts/3 S 0:00 psic
18823 pts/3 S 0:00 psic
.
.
```

where you need to replace the string `psic` 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*):

```
bash-2.05$ 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:

```
bash-2.05$ 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!

```
bash-2.05$ kill -9 18822
```

14.1.3. 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 14.1.2.

14.2. spec has crashed

Are you sure, that it is really **spec** which crashed?

Check out the list below for detailed help.

14.2.1. My motor does not move

There are several reasons for this phenomenon:

***spec* control the motors through soft channel.**

Check whether program for the soft channel is running.

- To check if the soft motor application is running, open shell window from any beamline linux computer and type
  'caget SMTR16062I1031'
  if the command reports a timeout, it is unlikely the application is running.
- To start the soft motor application,
  log in to ioc1606-014 as hxma
  'ssh -Y hxma@ioc1606-014'

change directory to /home/hxma/APSmotor

run the command 'runAps'

Check the communication between **spec** and your motor controller (switch on debugging with **`debug 192`**, switch back to normal operation with **`debug 0`**).

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

14.2.3. 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, communication between *spec* and soft motor application is hang-up. Ask HXMA beamline scientist for assistance!

14.3. Workarounds to known bugs

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.

`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**