
SPECpy Documentation

Release 1

Lakshmipriya Sukumar and Brian Toby

October 11, 2012

CONTENTS

1	<i>Module spec: SPEC-like emulation</i>	3
2	<i>Module spec: Global variables</i>	5
3	<i>Module spec: Function Descriptions</i>	7
4	<i>Module macros: SPEC-like emulation</i>	15
5	<i>Module macros: All Functions</i>	17
	Index	23

Note that this package requires the Python NumPy and PyEpics packages be installed order to control an instrument. However, if PyEpics is not installed, all routines documented here can still be run. However, in this case EPICS interactions will be simulated and print statements will report what the Python code is attempting to do. Likewise, if PyEpics is installed, but :func:'EnableEPICS' is not called (or is called with a value of False), again no communication with EPICS is attempted. This allows scripts to be developed and tested without access to the instrument.

MODULE SPEC: SPEC-LIKE EMULATION

The Python functions listed below are designed to emulate similar commands/macros in SPEC.

Motor interface routines

Description	Relative	Absolute
move motor	mvr ()	mv ()
move motor with wait	umvr ()	umv ()
where is this motor?		wm ()
where are all motors?		wa ()

Scaler routines

description	command
start and readout scaler after completion	ct ()
start scaler and return	count_em ()
wait for scaler to complete	wait_count ()
read scaler	get_counts ()

Other routines in module spec

other routines	Description
EnableEPICS ()	Turns simulation mode on or off
ShowEnabled ()	Show if use of use of EPICS is available
DefineMtr ()	Define a motor to be accessed
GetMtrInfo ()	Retrieves all motor info from a key
DefineScaler ()	Define a scaler to be accessed
GetScalerInfo ()	Retrieves all scaler info from an index
ListMtrs ()	Returns a list of motor symbols
Sym2MtrVal ()	Retrieves the motor entry key from a symbol
ExplainMtr ()	Retrieves the motor description from a key or symbol
ReadMtr ()	Returns the motor position from a key
PositionMtr ()	Moves a motor
GetScalerLastCount ()	Returns the last set of counts that have been read for a scaler
GetScalerLastTime ()	Returns the counting time for the last use of a scaler
GetScalerLabels ()	Returns the labels that have been retrieved for a scaler
SetMon ()	Set the monitor channel for the scaler
GetMon ()	Return the monitor channel for the scaler
SetDet ()	Set the main detector channel for the scaler
GetDet ()	Return the main detector channel for the scaler
setCOUNT ()	Sets the default counting time
setRETRIES ()	Sets the maximum number of EPICS retries
setDEBUG ()	Sets debugging mode on (printing lots of stuff) or off

MODULE SPEC: GLOBAL VARIABLES

COUNT defines the default counting time (sec) when `ct` is called without an argument. Defaults to 1 sec. Use `setCOUNT()` to set this when using `from spec import *`, as setting the variable directly has problems.

This will sort-of work:

```
>>> from spec import *
>>> import spec
>>> spec.COUNT=3
```

however, `COUNT` in the local namespace will still have the old value.

but this will not work:

```
>>> from spec import *
>>> COUNT=3
```

This fails because the local copy of `COUNT` gets replaced, but the copy of `COUNT` actually in the `spec` module is left unchanged.

S `S` is a list that contains the last count values measured during the last call to `ct()` or `get_counts()`.

MAX_RETRIES Number of times to retry an EPICS operation (that are nominally expected to work on the first try) before generating an exception. Use `setRETRIES()` to set this or care when changing this (see comment on `COUNT`, above.)

DEBUG When set to `True` lots of print statements to be executed. Use for code development/testing. Use `setDEBUG()` to set this or care when changing this (see comment on `COUNT`, above.)

MODULE SPEC: FUNCTION DESCRIPTIONS

The functions available in this module are listed below.

`spec.DefineMtr` (*symbol*, *prefix*, *comment*='')

Define a motor for use in this module. Adds a motor to the motor table.

Parameters

- **symbol** (*string*) – a symbolic name for the motor. A global variable is defined in this module's name space with this name, This must be unique; exception `specException` is raised if a name is reused.
- **prefix** (*string*) – the prefix for the motor PV (`ioc:mnnn`). Omit the motor record field name (`.VAL`, etc.).
- **comment** (*string*) – a human-readable text field that describes the motor. Suggestion: include units and define the motion direction.

Returns key of entry created in motor table (str).

If you will use the “ `from <module> import *` ” python command to import these routines into the current module's name space, it is necessary to repeat this command after `DefineScaler()` to import the globals defined within in the top namespace:

Example (recommended for interactive use):

```
>>> from spec import *
>>> EnableEPICS()
>>> DefineMtr('mtrXX1', 'ioc1:mtr98', 'Example motor #1')
>>> DefineMtr('mtrXX2', 'ioc1:mtr99', 'Example motor #2')
>>> from spec import *
>>> mv(mtrXX1, 0.123)
```

Note that if the second `from ... import *` command is not used, the variables `mtrXX1` and `mtrXX2` cannot be accessed and the final command will fail.

Alternate example (this is a cleaner way to code scripts, since namespaces are not mixed):

```
>>> import spec
>>> spec.EnableEPICS()
>>> spec.DefineMtr('mtrXX1', 'ioc1:mtr98', 'Example motor #1')
>>> spec.DefineMtr('mtrXX2', 'ioc1:mtr99', 'Example motor #2')
>>> spec.mv(spec.mtrXX1, 0.123)
```

It is also possible to mix the two styles:

```
>>> import spec
>>> spec.EnableEPICS()
>>> spec.DefineMtr('mtrXX1', 'ioc1:mtr98', 'Example motor #1')
>>> spec.DefineMtr('mtrXX2', 'ioc1:mtr99', 'Example motor #2')
>>> from spec import *
>>> mv(mtrXX1, 0.123)
```

`spec.DefineScaler` (*prefix*, *channels=8*, *index=0*)

Defines a scaler to be used for this module

Parameters

- **prefix** (*string*) – the prefix for the scaler PV (ioc:mnnn). Omit the scaler record field name (.CNT, etc.)
- **channels** (*int*) – the number of channels associated with the scaler. Defaults to 8.
- **index** (*int*) – an index for the scaler, if more than one will be defined. The default (0) is used to define the scaler that will be used when `ct()` is called with one or no arguments.

Example (recommended for interactive use):

```
>>> from spec import *
>>> EnableEPICS()
>>> DefineScaler('idl:scaler1', 16)
>>> DefineScaler('idl:scaler2', index=1)
>>> ct()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

Alternate example (preferred for use in code):

```
>>> import spec as s
>>> s.EnableEPICS()
>>> s.DefineScaler('ioc1:3820:scaler1', 16)
>>> s.DefineScaler('ioc1:3820:scaler2', index=1)
>>> s.ct()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
>>> s.ct(index=1)
[1, 2, 3, 4, 5, 6, 7, 8]
```

`spec.EnableEPICS` (*state=True*)

Call to enable communication with EPICS.

If not called then the module will function in simulation mode only. If the PyEpics module cannot be loaded, then simulation will also be used.

Parameters *state* (*bool*) – if False is specified, then simulation mode is used (default value, True)

`spec.ExplainMtr` (*mtr*)

Show the description for a motor, as defined in `DefineMtr()`

Parameters *mtr* (*various*) – symbolic name for the motor, can take two forms: a motor key or a motor symbol.

Returns motor description (str) or '?' if not defined

`spec.GetDet` (*index=0*)

Return the main detector channel for the scaler or none if not defined. (See `SetDet()`) This is used for ASCAN, etc.

Parameters *index* (*int*) – an index for the scaler, if more than one will be defined (see `DefineScaler()`). The default (0) is used if not specified.

Returns the channel number of the Detector

`spec.GetMon` (*index=0*)

Return the monitor channel for the scaler or none if not defined. (See `SetMon()`) This is used for counting on the Monitor.

Parameters *index* (*int*) – an index for the scaler, if more than one will be defined (see `DefineScaler()`). The default (0) is used if not specified.

Returns the channel number of the Monitor

`spec.GetMtrInfo` (*mtr*)

Return a dictionary with motor information.

Parameters *mtr* (*str*) – a key corresponding to an entry in the motor table. If the value does not correspond to a motor entry, an exception is raised.

Returns dictionary with motor information

`spec.GetScalerInfo` (*index=0*)

returns information about a scaler based on the index

Parameters *index* (*int*) – an index for the scaler, if more than one is be defined (see `DefineScaler()`). The default (0) is used if not specified.

Returns a dictionary with information on the scaler

`spec.GetScalerLabels` (*index=0*)

returns the labels that have been retrieved for a scaler

Parameters *index* (*int*) – an index for the scaler, if more than one is be defined (see `DefineScaler()`). The default (0) is used if not specified.

Returns a list of labels

`spec.GetScalerLastCount` (*index=0*)

returns the last set of counts that have been read for a scaler

Parameters *index* (*int*) – an index for the scaler, if more than one is be defined (see `DefineScaler()`). The default (0) is used if not specified.

Returns a list of the last counts

`spec.GetScalerLastTime` (*index=0*)

returns the count time for the last read from a scaler

Parameters *index* (*int*) – an index for the scaler, if more than one is be defined (see `DefineScaler()`). The default (0) is used if not specified.

Returns a single float with the last elapsed time for that scaler (initialized at 0) of the last counts

`spec.ListMtrs` ()

Returns a list of the variables defined as motor symbols.

Returns a python list of defined motor symbols (list of str values).

`spec.PositionMtr` (*mtr*, *pos*, *wait=True*)

Move a motor

Position a motor associated with *mtr* to position *pos*, wait for the move to complete if *wait* is `True`, or else return immediately. The function attempts to verify the move command has been acted upon.

Parameters

- **mtr** (*int*) – a value corresponding to an entry in the motor table, as defined in `DefineMtr()`. If the value does not correspond to a motor entry, an exception is raised.
- **pos** (*float*) – a value to position the motor. If the value is invalid or outside the limits an exception occurs (todo: are hard limits checked?).
- **wait** (*bool*) – a flag that specifies if the move should be completed before the function returns. If False, the function returns immediately.

`spec.ReadMtr(mtr)`

Return the motor position associated with the passed motor value.

Parameters **mtr** (*int*) – a key corresponding to an entry in the motor table. If the value does not correspond to a motor entry, an exception is raised.

Returns motor position (float).

`spec.SetDet(Detector=None, index=0)`

Set the main detector channel for the scaler. The default is to restore this to the initial setting, where this is undefined. This is used for ASCAN, etc.

Parameters

- **Monitor** (*int*) – channel number. If omitted the Monitor is set as undefined. The valid range for this parameter is 0 through one less than the number of channels.
- **index** (*int*) – an index for the scaler, if more than one will be defined (see `DefineScaler()`). The default (0) is used if not specified.

`spec.SetMon(Monitor=None, index=0)`

Set the monitor channel for the scaler. The default is to restore this to the initial setting, where this is undefined. This is needed for counting on the Monitor.

Parameters

- **Monitor** (*int*) – channel number. If omitted the Monitor is set as undefined. The valid range for this parameter is 0 through one less than the number of channels.
- **index** (*int*) – an index for the scaler, if more than one will be defined (see `DefineScaler()`). The default (0) is used if not specified.

`spec.ShowEnabled()`

Show if use of EPICS is allowed or disabled, see `EnableEPICS()`.

Returns True if PyEpics has been loaded, False otherwise

`spec.Sym2MtrVal(mtrsym)`

Converts a motor symbol (as a string) to the motor value (key) as assigned in `DefineMtr()`

Parameters **mtrsym** (*str*) – a motor symbol as supplied in `DefineMtr()`. If the value does not correspond to a motor entry, an exception is raised.

Returns motor value (str).

`spec.count_em(count=None, index=0)`

Cause scaler to start counting for specified period, but return immediately. On the first use, this will take the scaler out of autocount mode and put it into one-shot mode (this is because if one does not read the scaler shortly after a count when in autocount mode, the scaler returns to autocount and the values are lost.) If put in one-shot mode, then autocount will be restored when the python interpreter is exited.

Counting is on time if count is 0 or positive; Counting is on monitor if count < 0

Parameters

- **count-time** (*float*) – time (sec) to count, if omitted COUNT is used

- **index** (*int*) – an index for the scaler, if more than one will be defined (see `DefineScaler()`). The default (0) is used if not specified.

Returns None

Example:

```
>>> count_em()
>>> # do other commands
>>> wait_count()
>>> get_counts()
```

`spec.ct` (*count=None, index=0, label=False*)

Cause scaler to count for specified period or to a specified number of counts on a prespecified channel (see `SetMon()`)

Counting is on time if count is 0 or positive; Counting is on monitor if count < 0

Global variable S is set to the count values for the n channels (set in `DefineScaler()`) to provide functionality similar to `spec`.

Parameters

- **count** (*float*) – time (sec) to count, if omitted COUNT is used
- **index** (*int*) – an index for the scaler, if more than one is defined (see `DefineScaler()`). The default (0) is used if not specified.
- **label** (*bool*) – indicates if counts should be printed along with their labels The default (False) is to not print counts

Returns count values for the channels (see `DefineScaler()`)

Example:

```
>>> ct()
[10000000.0, 505219.0, 359.0, 499.0, 389.0, 356.0, 114.0, 53.0]
>>> SetMon(3)
>>> ct(-1000)
[20085739.0, 1011505.0, 719.0, 1000.0, 781.0, 715.0, 226.0, 105.0]
```

`spec.get_counts` (*wait=False*)

Read scaler with optional delay, must follow `count_em`

reads count values for the channels (see `DefineScaler()`)

Parameters **wait** (*bool*) – True causes the routine to wait for the scaler to complete; False (default) will read the scaler instantaneously

Returns a list of channels values

Example:

```
>>> get_counts()
[1, 2, 3, 4, 5, 6, 7, 8]
```

`spec.mv` (*mtr, pos*)

Move motor without wait

If the move cannot be made, an exception is raised.

Parameters

- **mtr** (*int*) – a value corresponding to an entry in the motor table, as defined in `DefineMtr()`. If the value does not correspond to a motor entry, an exception is raised.
- **pos** (*float*) – a value to position the motor. If the value is invalid or outside the limits, an exception occurs.

Example:

```
>>> mv (samX, 0.1)
```

`spec.mvr (mtr, delta)`

Move motor relative to current position without wait.

If the move cannot be made, an exception is raised.

Parameters

- **mtr** (*int*) – a value corresponding to an entry in the motor table, as defined in `DefineMtr()`. If the value does not correspond to a motor entry, an exception is raised.
- **delta** (*float*) – a value to offset the motor. If the resulting value is invalid or outside the limits, an exception occurs.

Example:

```
>>> mvr (samX, 0.1)
```

`spec.setCOUNT (count)`

Sets the default counting time, see global variable `COUNT` or `ct()`.

Parameters **count** (*float*) – default time (sec) to count.

`spec.setDEBUG (state=True)`

Sets the debug state on or off, see global variable `DEBUG`.

Parameters **state** (*bool*) – `DEBUG` is initialized as `False`, but the default effect of `setDEBUG`, if no parameter is specified is to turn the debug state on.

`spec.setRETRIES (count=20)`

Sets the maximum number of times to retry an EPICS operation (that would nominally be expected to work on the first try) before generating an exception. See global variable `MAX_RETRIES`.

Parameters **count** (*float*) – maximum number of times to retry an EPICS operation. Defaults to 20.

`spec.umv (mtr, pos)`

Move motor with wait.

If the move cannot be completed, an exception is raised.

Parameters

- **mtr** (*int*) – a value corresponding to an entry in the motor table, as defined in `DefineMtr()`. If the value does not correspond to a motor entry, an exception is raised.
- **pos** (*float*) – a value to position the motor. If the value is invalid or outside the limits, an exception occurs.

Example:

```
>>> umv (samX, 0.1)
```

spec.**umvr** (*mtr*, *delta*)

Move motor relative to current position with wait.

If the move cannot be completed, an exception is raised.

Parameters

- **mtr** (*int*) – a value corresponding to an entry in the motor table, as defined in `DefineMtr()`. If the value does not correspond to a motor entry, an exception is raised.
- **delta** (*float*) – a value to offset the motor. If the resulting value is invalid or outside the limits, an exception occurs.

Example:

```
>>> umvr (samX, 0.1)
```

spec.**wa** (*label=False*)

Print positions of all motors defined using `DefineMtr()`.

Parameters **label** (*bool*) – a flag that specifies if the list should include the motor descriptions. If omitted or `False`, the descriptions are not included.

Example:

```
>>> wa ()
samX          1.0
samZ          0.0
>>> wa (True)
samX          1.0          sample X position (mm) + outboard
samZ          0.0          sample Z position (mm) + up
```

spec.**wait_count** ()

Wait for scaler to finish, must follow `count_em`

Returns `None`

Example:

```
>>> wait_count ()
```

spec.**wm** (**mtrs*)

Read out specified motor(s).

Arguments one or more motor table entries that are defined in `DefineMtr()`.

Returns a single float if a single argument is passed to `wm`. Returns a list of floats if more than one argument is passed.

Example:

```
>>> wm (samX, samZ)
[1.0, 0.0]
```


MODULE MACROS: SPEC-LIKE EMULATION

Python functions listed below are designed to implement functionality similar to that in spec.

General purpose routines	Description
<code>specdate()</code>	Returns the date/time formatted like Spec
<code>SetScanFile()</code>	Open a file for scan output
<code>ascan()</code>	Scan a single motor on a fixed range
<code>dscan()</code>	Scan a single motor on a range relative to current position
<code>FitLastScan()</code>	Fit a user-supplied function to a user-supplied function

Logging

An important set of configuration parameters is that which determine what values are recorded. During data collection, for example, after each `ascan()` or `dscan()` data point. Also, for use in defining macros, the values can also be saved to a log file using `write_logging_parameters()`.

Logging routines	Description
<code>init_logging()</code>	Initializes the list of items to be reported
<code>show_logging()</code>	Displays a list of the items that will be logged
<code>add_logging_PV()</code>	Adds a PV to the list of items to be reported
<code>add_logging_Global()</code>	Adds a Global variable to the list of items to be reported
<code>add_logging_PVobj()</code>	Adds a PV object to the list of items to be reported
<code>add_logging_motor()</code>	Adds a motor reference to the list of items to be reported
<code>add_logging_scaler()</code>	Adds a scaler channel to the list of items to be reported
<code>write_logging_header()</code>	Writes a header line with labels for each logged item
<code>write_logging_parameters()</code>	Write the current value of each logged variable

Example for setting up logging:

```
>>> import macros
>>> macros.init_logging()
>>> GE_prefix = 'GE2:cam1:'
>>> macros.add_logging_PV('GE_fname', GE_prefix+"FileName", as_string=True)
>>> macros.add_logging_PV('GE_fnum', GE_prefix+"FileNumber")
>>> macros.add_logging_motor(spec.samX)
>>> macros.add_logging_scaler(9)
>>> macros.add_logging_Global('var S9', 'spec.S[9]')
>>> macros.add_logging_PV('p1Vs', "1dc:m64.RBV")
```

Note that the `add_logging_scaler` and `add_logging_Global` calls above will record the same value (though with different headings), but the `add_logging_scaler` is a better choice as the second option could produce the wrong value if use of a second scaler is later added to a script.

Example for use of logging in a script:

```
>>> mac.write_logging_header(logname)
>>> spec.umv(spec.mts_y, stY)
>>> for iLoop in range(nLoop):
>>>     spec.umvr(spec.mts_y, dY)
>>>     count_em(Nframe*tframe)
>>>     GE_expose(fname, Nframe, tframe)
>>>     wait_count()
>>>     get_counts()
>>>     mac.write_logging_parameters(logname)
>>> mac.beep_dac()
```

This code step-scans motor `mts_y`. It writes a header to the log file at the beginning of the operation and then logs parameters after each measurement. Measurements are done in `GE_expose` and the default scaler, which are run at the same time.

Note that it can be useful to put differing sets of logging configurations into files where they can be invoked as needed using `execfile(xxx.py)` [where `xxx.py` is the name of the file to be read]. Do not use `import` for this task because `import` will process the file when it is referenced first, but will not do anything if one attempts to import the file again (to reset values back after a different setting has been used). One must use `reload` to force that.

Macros defined specifically for 1-ID

These macros reference 1-ID PV's or are customized for 1-ID in some other manner.

1-ID specific routines	Description
<code>beep_dac()</code>	Causes a beep to sound
<code>Cclose()</code>	Close 1-ID fast shutter in B hutch
<code>Copen()</code>	Open 1-ID fast shutter in B hutch
<code>shutter_sweep()</code>	Set 1-ID fast shutter to external control
<code>shutter_manual()</code>	Set 1-ID fast shutter to manually control
<code>check_beam_shutterA()</code>	Open 1-ID Safety shutter to bring beam into 1-ID-A
<code>check_beam_shutterC()</code>	Open 1-ID Safety shutter to bring beam into 1-ID-C
<code>Sopen()</code>	Same as <code>check_beam_shutterC()</code> , bring beam into 1-ID-C
<code>MakeMtrDefaults()</code>	Create a file with default motor assignments
<code>SaveMotorLimits()</code>	Create a file with soft limits for off-line simulations

MODULE MACROS: ALL FUNCTIONS

The functions available in this module are listed below.

`macros.Cclose()`

Close 1-ID fast shutter in B hutch

`macros.Copen()`

Open 1-ID fast shutter in B hutch

`macros.FitLastScan(StartEq, EvalEq)`

Fit and plot an arbitrary equation to data from the last ascan

Parameters

- **StartEq** (*function*) – a function that accepts two arguments, a list of x values and a list of y values and returns rough starting values for the fitting function. This function returns a list of n parameters, which must match the number of parameters needed by the fitting function
- **EvalEq** (*function*) – a fitting function that accepts two arguments, a list of n parameters and a list of x values. This function must return a list or tuple of y values that computed based on those starting parameters. By convention, the first parameter is assumed to be the peak maximum location; the last is a background value, added to all points; and the second is the intensity at the peak relative to the background. The parameters are initially generated by function StartEq, but then are optimized via least squares.

Returns an optimized list of parameters or None if the fit fails

Example:

```
>>> macros.FitLastScan(macros.startSymSawt, macros.evalSymSawt)
array([ 1.44999999, 28.50005241, 2.10525894, 1.4999749 ])
```

`macros.MakeMtrDefaults(fil=None, out=None)`

Routine in Development: Creates an initialization file from a spreadsheet describing the 1-ID beamline motor assignments

Parameters

- **fil** (*str*) – input file to read. By default opens file `../1ID/1ID_stages.csv` relative to the location of the current file.
- **out** (*str*) – output file to write. By default writes file `../1ID/mtrsetup.py.new` Note that if the default file name is used, the output file must be renamed before use to `mtrsetup.py`

`macros.SaveMotorLimits(out=None)`

Routine in Development: Creates an initialization file for simulation use with the limits for every motor PV that

is found in the current 1-ID beamline motor assignments. import mtrsetup.py or equivalent first. Scans each PV from 1 to the max number defined.

Parameters **out** (*str*) – output file to write, writes file motorlimits.dat.new in the same directory as this file by default. Note that if the default file name is used, the output file must be renamed before use to motorlimits.dat

`macros.SetScanFile (outfile=None)`

Set a file for output from ascan, etc. The output is intended to closely mimic what spec produces in ascan and dscan.

Parameters **outfile** (*str*) – the file name to be opened. If not specified, output is sent to the terminal. If the file is new (or is the not specified) a header listing all motors, etc. is printed

`macros.Sopen ()`

If not already open, open 1-ID-C Safety shutter to bring beam into 1-ID-C. Keep trying in an infinite loop until the shutter opens.

`macros.add_logging_Global (txt, var)`

Define a global variable to be recorded when `write_logging_parameters ()` is called

Parameters

- **txt** (*str*) – defines a text string, preferably short, to be used when `write_logging_header ()` is called as a header for the item to be logged.
- **var** (*str*) – defines a Python variable that will be logged each time `write_logging_parameters ()` is called. Note that this is read inside the macros module so the variable must be defined inside that module or must be prefixed by a reference to a module referenced in that module, e.g. `spec.S[0]`

`macros.add_logging_PV (txt, PV, as_string=False)`

Define a PV to be recorded when `write_logging_parameters ()` is called.

Parameters

- **txt** (*str*) – defines a text string, preferably short, to be used when `write_logging_header ()` is called as a header for the item to be logged.
- **PV** (*str*) – defines an EPICS Process Variable that will be read and logged each time `write_logging_parameters ()` is called.
- **as_string** (*bool*) – if True, the PV will be translated to a string. When False (default) the native data type will be used. Use of True is of greatest for waveform records that are used to store character strings as a series of integers.

`macros.add_logging_PVobj (txt, PVobj, as_string=False)`

Define a PVobj to be recorded when `write_logging_parameters ()` is called

Parameters

- **txt** (*str*) – defines a text string, preferably short, to be used when `write_logging_header ()` is called as a header for the item to be logged.
- **PV** (*epics.PV*) – defines a PyEpics PV object that is connected to an EPICS Process Variable. The PV method `.get()` will be used to read that PV to log it each time `write_logging_parameters ()` is called.
- **as_string** (*bool*) – if True, the PV value will be translated to a string. When False (default) the native data type will be used. Use of True is of greatest for waveform records that are used to store character strings as a series of integers.

`macros.add_logging_motor (mtr)`

Define a motor object to be recorded when `write_logging_parameters()` is called. Note that the heading text string is defined as the motor's symbol (see `spec.DefineMtr()`).

Parameters `mtr (str)` – a reference to a motor object, returned by `spec.DefineMtr()` or defined in the motor symbol. The position of the motor will be read and logged each time `write_logging_parameters()` is called.

`macros.add_logging_scaler (channel, index=0)`

Define a scaler channel to be recorded when `write_logging_parameters()` is called. Note that the heading text string is defined as the scaler's label (which is read from the scaler when `spec.DefineScaler()` is run).

Parameters

- **channel (str)** – a channel number for a scaler, which can be any value between 0 and one less than the number of channels. The last-read value of that scaler logged each time `write_logging_parameters()` is called.
- **index (int)** – an index for the scaler, if more than one is to be defined (see `DefineScaler()`). The default (0) is used if not specified.

`macros.ascan (mtr, start, finish, npts, count, index=0, settle=0.0, _func='ascan')`

Scan one motor and record parameters set with logging to the scanfile (see `func:SetScanFile`).

Parameters

- **mtr (str)** – a reference to a motor object, returned by `spec.DefineMtr()` or defined in the motor symbol.
- **start (float)** – starting position for scan
- **finish (float)** – ending position for scan
- **npts (int)** – number of points for scan
- **count (float)** – count time. Counting is on time (sec) if count is 0 or positive; Counting is on monitor if count < 0
- **index (int)** – an index for the scaler, if more than one will be defined (see `DefineScaler()`). The default (0) is used if not specified.
- **settle (float)** – a time to wait (sec) after the motor has been moved before counting is starting. Default is 0.0 which means no delay

Example:

```
>>> spec.SetDet(2)
>>> macros.ascan(spec.samX, 1, 2, 21, 1, settle=.1)
```

It is recommended that if `ascan` will be run in command line, where python commands are typed into a console window, that `ipython` be used in `pylab` mode (`ipython --pylab`).

`macros.beep_dac (beep_time=1.0)`

Set the 1-ID beeper on for a fixed period, which defaults to 1 second uses PV object beeper (defined as `1id:DAC1_8.VAL`) makes sure that the beeper is actually turned on and off throws exception if beeper fails

Parameters `beep_time (float)` – time to sound the beeper (sec), defaults to 1.0

`macros.check_beam_shutterA ()`

If not already open, open 1-ID-A Safety shutter to bring beam into 1-ID-A. Keep trying in an infinite loop until the shutter opens.

`macros.check_beam_shutterC()`

If not already open, open 1-ID-C Safety shutter to bring beam into 1-ID-C. Keep trying in an infinite loop until the shutter opens.

`macros.dscan(mtr, start, finish, npts, count, index=0, settle=0.0)`

Relative scan of motor, see func:*ascan*,

Parameters

- **mtr** (*str*) – a reference to a motor object, returned by `spec.DefineMtr()` or defined in the motor symbol.
- **start** (*float*) – starting position for scan, relative to current motor position
- **finish** (*float*) – ending position for scan, relative to current motor position
- **npts** (*int*) – number of points for scan
- **count** (*float*) – count time. Counting is on time (sec) if count is 0 or positive; Counting is on monitor if count < 0
- **index** (*int*) – an index for the scaler, if more than one will be defined (see `DefineScaler()`). The default (0) is used if not specified.
- **settle** (*float*) – a time to wait (sec) after the motor has been moved before counting is starting. Default is 0.0 which means no delay

Example:

```
>>> spec.SetDet(2)
>>> macros.dscan(spec.samX, -1, 1, 21, 1, settle=.1)
```

It is recommended that if dscan will be run in command line, where python commands are typed into a console window, that ipython be used in pylab mode (`ipython --pylab`).

`macros.evalGauss(parm, x)`

Evaluate a Gaussian function at positions *x*, returns a list of intensities

Parameters

- **parm** (*list*) – parameters used in Gaussian

parm[0]	is location of peak
parm[1]	is function value at maximum relative to parm[3]
parm[2]	is width as FWHM
parm[3]	is added to all points

- **x** (*np.array*) – numpy array of locations for computation

Returns a `np.array` of intensity values for each *x* value

`macros.evalSawt(parm, x)`

Evaluate a asymmetric Sawtooth function at positions *x*, returns a list of intensities

Parameters

- **parm** (*list*) – parameters used in in this function:

parm[0]	is location of peak
parm[1]	is function value at maximum
parm[2]	is slope on negative side of peak (+ is rising)
parm[3]	is slope on positive side of peak (+ is falling)
parm[4]	is added to all points

- **x** (*np.array*) – numpy array of locations for computation

Returns a np.array of intensity values for each x value

`macros.evalSymSawt` (*parm, x*)

Evaluate a Symmetric Sawtooth function at positions x, returns a list of intensities

Parameters

- **parm** (*list*) – parameters used in this function

parm[0]	is location of peak
parm[1]	is function value at maximum
parm[2]	is slope
parm[3]	is added to all points

- **x** (*np.array*) – numpy array of locations for computation

Returns a np.array of intensity values for each x value

`macros.init_logging` ()

Initialize the list of data items to be logged

`macros.show_logging` ()

Show the user the current logged items

`macros.shutter_manual` ()

Set 1-ID fast shutter so that it will not be controlled by the GE TTL signal and can be manually opened and closed with Copen() and Cclose()

`macros.shutter_sweep` ()

Set 1-ID fast shutter so that it will be controlled by an external electronic control (usually the GE TTL signal)

`macros.specdate` ()

format current date/time as produced in Spec

Returns the current date/time as a string, formatted like “Thu Oct 04 18:24:14 2012”

Example:

```
>>> macros.specdate()
'Thu Oct 11 16:16:39 2012'
```

`macros.startGauss` (*x, y*)

Estimate starting parameters for a Gaussian fit

Parameters

- **x** (*list*) – locations for computation
- **y** (*list*) – intensity at each location

Returns

a list of four parameters:

parm[0]	is location of maximum
parm[1]	is intensity at maximum
parm[2]	is width as FWHM, starts at spacing of two points
parm[3]	is added to all points, starts at 0

`macros.startSawt` (*x, y*)

Estimate starting parameters for an asymmetric Sawtooth fit

Parameters

- **x** (*list*) – locations for computation
- **y** (*list*) – intensity at each location

Returns

a list of five parameters:

parm[0]	is location of maximum
parm[1]	is intensity at maximum
parm[2]	is slope on negative side of peak (+ is rising)
parm[3]	is slope on positive side of peak (+ is falling)
parm[4]	is added to all points

macros.**startSymSawt** (*x, y*)

Estimate starting parameters for a Sawtooth fit

Parameters

- **x** (*list*) – locations for computation
- **y** (*list*) – intensity at each location

Returns

a list of four parameters:

parm[0]	is location of maximum
parm[1]	is intensity at maximum
parm[2]	is slope on negative side of peak (+ is rising)
parm[3]	is added to all points

macros.**write_logging_header** (*filename=''*)

Write a header for parameters recorded when `write_logging_parameters()` is called.

Parameters filename (*str*) – a filename to be used for output. If not specified, the output is sent to the terminal window.

macros.**write_logging_parameters** (*filename=''*)

Record the current value of all items tagged to be recorded in `add_logging_PV()`, `add_logging_Global()`, `add_logging_PVobj()`, `add_logging_motor()` or `add_logging_scaler()`.

Parameters filename (*str*) – a filename to be used for output. If not specified, the output is sent to the terminal window.

INDEX

A

add_logging_Global() (in module macros), 18
add_logging_motor() (in module macros), 18
add_logging_PV() (in module macros), 18
add_logging_PVobj() (in module macros), 18
add_logging_scaler() (in module macros), 19
ascan() (in module macros), 19

B

beep_dac() (in module macros), 19

C

Cclose() (in module macros), 17
check_beam_shutterA() (in module macros), 19
check_beam_shutterC() (in module macros), 19
Copen() (in module macros), 17
COUNT, 5
count_em() (in module spec), 10
ct() (in module spec), 11

D

DEBUG, 5
DefineMtr() (in module spec), 7
DefineScaler() (in module spec), 8
dscan() (in module macros), 20

E

EnableEPICS() (in module spec), 8
evalGauss() (in module macros), 20
evalSawt() (in module macros), 20
evalSymSawt() (in module macros), 21
ExplainMtr() (in module spec), 8

F

FitLastScan() (in module macros), 17

G

get_counts() (in module spec), 11
GetDet() (in module spec), 8
GetMon() (in module spec), 9
GetMtrInfo() (in module spec), 9

GetScalerInfo() (in module spec), 9
GetScalerLabels() (in module spec), 9
GetScalerLastCount() (in module spec), 9
GetScalerLastTime() (in module spec), 9

I

init_logging() (in module macros), 21

L

ListMtrs() (in module spec), 9

M

MakeMtrDefaults() (in module macros), 17
MAX_RETRIES, 5
mv() (in module spec), 11
mvr() (in module spec), 12

P

PositionMtr() (in module spec), 9

R

ReadMtr() (in module spec), 10

S

S, 5
SaveMotorLimits() (in module macros), 17
setCOUNT() (in module spec), 12
setDEBUG() (in module spec), 12
SetDet() (in module spec), 10
SetMon() (in module spec), 10
setRETRIES() (in module spec), 12
SetScanFile() (in module macros), 18
show_logging() (in module macros), 21
ShowEnabled() (in module spec), 10
shutter_manual() (in module macros), 21
shutter_sweep() (in module macros), 21
Sopen() (in module macros), 18
specdate() (in module macros), 21
startGauss() (in module macros), 21
startSawt() (in module macros), 21
startSymSawt() (in module macros), 22

Sym2MtrVal() (in module spec), 10

U

umv() (in module spec), 12

umvr() (in module spec), 13

W

wa() (in module spec), 13

wait_count() (in module spec), 13

wm() (in module spec), 13

write_logging_header() (in module macros), 22

write_logging_parameters() (in module macros), 22