

---

# **SPECpy Documentation**

*Release 1*

**Lakshmipriya Sukumar and Brian Toby**

October 05, 2012



# CONTENTS

<b>1</b>	<i>Module spec: SPEC-like emulation</i>	<b>1</b>
<b>2</b>	<i>Module spec: Global variables</i>	<b>3</b>
<b>3</b>	<i>Module spec: All Functions</i>	<b>5</b>
<b>4</b>	<i>Module macros: SPEC-like emulation</i>	<b>11</b>
<b>5</b>	<i>Module macros: All Functions</i>	<b>13</b>
	<b>Index</b>	<b>15</b>



---

# ***MODULE SPEC: SPEC-LIKE EMULATION***

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

*Motor interface routines.*

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

*Scaler routines*

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



## ***MODULE SPEC: GLOBAL VARIABLES***

**COUNT** defines the default counting time (sec) when ct is called without an argument. Defaults to 1 sec.

**MAX\_RETRIES** Number of times to retry an EPICS operation (that are nominally expected to work on the first try) before generating an exception.

**DEBUG** Set to True for code development/testing use only. Causes lots of print statements to be executed.

**ENABLE** Initialized as False, which indicates that indicates that EPICS PV access should be simulated (also allows module to be imported for documentation generation, etc. without importing PyEpics). Use function EnableEPICS() to set ENABLE to True.



# MODULE SPEC: ALL FUNCTIONS

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

Type	Description
str	interpreted as a global symbol
int	value references an entry in mtrDB

**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 (int)` – a value 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.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 value 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 1 through 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 1 through 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.**count\_em** (*count=None, index=0*)

Cause scaler to start counting for specified period.

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*)

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 variables are set after count is done:** S (list) is set to the count values for the n channels (set in `DefineScaler()`) Time (float) is set to the counting time

#### Parameters

- **count-time** (*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.

**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(4)
>>> 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.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 emulate similar macros in SPEC.

<b>macro</b>	<b>Description</b>
<code>beep_dac()</code>	Causes a beep to sound
<code>specdate()</code>	Returns the date/time formatted like Spec
<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>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
<code>Cclose()</code>	Close IID fast shutter in B hutch
<code>Copen()</code>	Open IID fast shutter in B hutch
<code>shutter_sweep()</code>	Set IID fast shutter to external control
<code>shutter_manual()</code>	Set IID fast shutter to manually control
<code>Sopen()</code>	Open IID Safety shutter to bring beam into 1-ID-C
<code>check_beam_shutterA()</code>	Open IID Safety shutter to bring beam into 1-ID-A



# MODULE MACROS: ALL FUNCTIONS

The functions available in this module are listed below.

`macros.Cclose()`

Close IID fast shutter in B hutch

`macros.Copen()`

Open IID fast shutter in B hutch

`macros.MakeMtrDefaults` (*file*='/home/beams/SIIDUSER/specpy/IID/IID\_stages.csv')

Development routine to create an initialization file from a spreadsheet describing the IID beamline motor assignments

opens file /home/beams/SIIDUSER/specpy/IID/IID\_stages.csv

writes file /home/beams/SIIDUSER/specpy/IID/mtrsetup.py.new

The output file is renamed before use to mtrsetup.py

`macros.Sopen()`

Open IID Safety shutter to bring beam into 1-ID-C

`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.beep_dac (beetime=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** **beetime** (*float*) – time to sound the beeper (sec), defaults to 1.0

`macros.check_beam_shutterA ()`

Check if A shutter is open and if not, open it

`macros.init_logging ()`

Initialize the list of data items to be logged

`macros.shutter_manual ()`

Set IID 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 IID fast shutter so that it will be controlled by 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”

`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()`, or

`add_logging_motor()`

**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), 13  
add\_logging\_motor() (in module macros), 14  
add\_logging\_PV() (in module macros), 13  
add\_logging\_PVobj() (in module macros), 13

## B

beep\_dac() (in module macros), 14

## C

Cclose() (in module macros), 13  
check\_beam\_shutterA() (in module macros), 14  
Copen() (in module macros), 13  
COUNT, 3  
count\_em() (in module spec), 8  
ct() (in module spec), 8

## D

DefineMtr() (in module spec), 5  
DefineScaler() (in module spec), 6

## E

EnableEPICS() (in module spec), 6  
ExplainMtr() (in module spec), 6

## G

get\_counts() (in module spec), 8  
GetDet() (in module spec), 6  
GetMon() (in module spec), 7  
GetMtrInfo() (in module spec), 7

## I

init\_logging() (in module macros), 14

## L

ListMtrs() (in module spec), 7

## M

MakeMtrDefaults() (in module macros), 13  
MAX\_RETRIES, 3

mv() (in module spec), 9  
mvr() (in module spec), 9

## P

PositionMtr() (in module spec), 7

## R

ReadMtr() (in module spec), 7

## S

SetDet() (in module spec), 7  
SetMon() (in module spec), 8  
shutter\_manual() (in module macros), 14  
shutter\_sweep() (in module macros), 14  
Sopen() (in module macros), 13  
specdate() (in module macros), 14

## U

umv() (in module spec), 9  
umvr() (in module spec), 10

## W

wa() (in module spec), 10  
wait\_count() (in module spec), 10  
wm() (in module spec), 10  
write\_logging\_header() (in module macros), 14  
write\_logging\_parameters() (in module macros), 14