

# Using SPEC Macro Hardware for EPICS PVs

Donald A. Walko

Advanced Photon Source, Argonne National Laboratory, Argonne, IL 60439  
d-walko@anl.gov

September 28, 2006

## Introduction

Many beamlines at the Advanced Photon Source (APS) use EPICS to control all sorts of equipment, such as motors, counters, gauges, pumps, etc. The individual pieces of named data are known as Process Variables (PVs). On the other hand, most multicircle diffractometers at the APS are controlled by SPEC [1]. In these situations, SPEC is usually built upon EPICS; that is, SPEC communicates with the experiment hardware through EPICS. While it is straightforward to integrate EPICS-controlled motors and counters into SPEC, experimenters may wish to have SPEC interact with PVs which do not represent motors or counters. For example, one may want to scan a voltage or read the output of an ohmmeter. One could in principal set up the voltage as a SPEC motor, or convert the output of the meter into a frequency which can be fed into a scaler. But a much better alternative is to use SPEC's macro hardware facility. This document presents the minimal steps necessary for accomplishing this.

## spec macro hardware

SPEC is designed to be a flexible software package, capable of controlling many different types of hardware. When SPEC is built on EPICS, the hardware control is usually shifted to EPICS. In such cases, the foremost contribution of SPEC is its conversion between the reciprocal space of the sample and the angular space of the diffractometer. In the SPEC configuration file, motors controlled by EPICS have `EPICS_M2` in the controller field, and scalers have `EPICS_SC` in their device field [2]. But how can SPEC move something that is not a motor? How can SPEC measure something that is not a frequency input into a scaler card? The answer is with SPEC macro hardware. This fairly straightforward procedure requires certain settings for the “motor” or “scaler” in config as well as some parameters set in a macro file. In the following sections we will describe this procedure for macro motors and macro scalers respectively, presenting the minimal requirements to accomplish this. Depending on the particular hardware, more parameters may be needed; the examples here do not cover all of the possibilities or the full flexibility of macro hardware. Details can be found on the SPEC help website [3].

As shown in the examples below, macro hardware needs to be set up in macro files (text files which are read into SPEC, with user names such as *filename.mac*) and in the configuration

file (“config”). While particular macro motors and scalers are set up in config’s motor and counter/scaler screens respectively, the existence of macro hardware needs to be introduced in the device configuration screen. Here is an example of the screen set up for EPICS-controlled macro motors and scalers:

---

Motor and Counter Device Configuration (Not CAMAC)

MOTORS	DEVICE	ADDR	<>MODE	NUM	<>TYPE
YES	ioc1:			10	EPICS Motor Controller
YES	macmot			1	Macro Motors
NO					
NO					
NO					

  

SCALERS	DEVICE	ADDR	<>MODE	NUM	<>TYPE
YES	ioc1:scaler1			3	EPICS Scaler as Counter/Timer
YES	mac_cnt			2	Macro Counter
NO					
NO					
NO					

Type ? and H for help, ^C to quit

---

## Macro motors

Suppose an experimenter wanted to change a voltage during a SPEC scan. If the voltage source is controlled by EPICS, then the PV associated with the voltage can be treated as a “macro motor” in SPEC. First, a macro file needs to be set up to define the macro motor, and the file must be read into SPEC (using the `qdo` command). The following is an example of the contents of such a file, where the SPEC mnemonic of the macro motor is “vvv.” The PV which sets the value is `ioc1:Device:voltage.VAL` and the PV which reads the current value is `ioc1:Device:voltage.RBV`; both PVs are in units of volts.

---

```

# macromotor_vvv.mac
#
def macmot_cmd(mne, key, p1) '{
local ptemp
  if (key == "set_position") {          ### do not set position with spec
    return
  }
  if (key == "position") {             ### read position of macro motor
    if (mne == vvv)
      {ptemp = epics_get("ioc1:Device:voltage.RBV")}
    return (ptemp*1e6)}
    else return(0)
  }
  if (key == "start_one") {            ### move macro motor
    if (mne == vvv)
      {epics_put("ioc1:Device:voltage.VAL",p1/1e6)
      sleep(.1)}                       ### optional, but may help
    else
      return
  }
  if (key == "get_status") {
    return(0)
  }
  if (key == "preread_all") { return }
}'

```

---

In this example, the actual voltage from the device is on a microvolt scale, which is inconvenient for SPEC (and also for the user who must make sure to type in the correct number of leading zeros). Therefore, the value is scaled up by a factor of  $10^6$ , so vvv is in units of microvolts.

The next step is to set up the motor in config. This is what the motor screen might look like, with three “regular” motors followed by one macro motor:

---

Number: <>Controller	1:EPICS_M2	2:EPICS_M2	3:EPICS_M2	4: MAC_MOT
Unit/[Module/]Channel	0/2	0/3	0/1	0/0
Name	Theta	Chi	Phi	Vvv
Mnemonic	th	chi	phi	vvv
<>Spectrometer	fourc	fourc	fourc	fourc
Steps per degree/mm	-25000	25000	2500	1e+06
Sign of user * dial	1	1	1	1
Backlash [steps]	50	50	50	50
Steady-state rate [Hz]	2000	2000	2000	2000
Base rate [Hz]	200	200	200	200
Acceleration time [msec]	125	125	125	125
Motor accumulator	0	0	253216	0
Restrictions <>	NONE	NONE	NONE	NONE

Dial = accumulator / steps

High limit	50.0000	59.0000	500.0000	9.9500
Current	0.0000	0.0000	101.2864	0.0000
Low limit	-10.0000	-60.0000	-500.0000	0.0050

User = sign \* dial + offset

Offset	0.0000	0.0000	-40.9701	0.0000
'High' limit	50.0000	59.0000	459.0300	9.9500
Current	0.0000	0.0000	60.3164	0.0000
'Low' limit	-10.0000	-60.0000	-540.9700	0.0050

Number of motors (geo/all) 31 / 31      Type ? and H for help, ^C to quit

---

In this minimal example, several fields do not matter for the macro motor (sign, backlash, rate) and were left at their default values. The “Steps per degree/mm” field does matter, for displaying vvv in microvolts. The limits are also used, so in this case, the range of vvv is roughly 0 to 10 microvolts.

The final step is to make sure that SPEC checks the position of the macro motor before attempting to move it. This becomes an issue when the PV is changed in EPICS or elsewhere outside of SPEC. To do this, go to the second “optional motor parameters” screen in config (type “m” twice from the main motors screen) and set the “Hardware read mode” field to its most conservative, i.e., PR+AL+NQ. In this mode, SPEC always checks EPICS before moving the macro motor or reading its position, and assumes that EPICS is correct if there is a disagreement.

## Macro counters

Let’s now assume that an experimenter wants SPEC to read PVs which comes from EPICS user calculations. We can treat the PVs as macro counters. Their mnemonics are value1 and value2,

with PVs of `ioc1:userCalc1.VAL` and `ioc1:userCalc2.VAL` respectively. Again, one connects the mnemonics and PVs in a macro file, which must be read into SPEC with a `qdo` command, such as the following:

---

```
# macro_usercalcs.mac
#
def mac_cnt_cmd(mne,key, p1, p2) '{
    if (key == "counts") {
        if (mne == value1) {return 1*(epics_get("ioc1:userCalc1.VAL"))}
        else if (mne == value2){ return 1*(epics_get("ioc1:userCalc2.VAL"))}
    }
    if (key == "halt_all") return
    if (key == "halt_one") return
    if (key == "prestart_all") return
    if (key == "prestart_one") return
}'
```

---

Note that, when a count is performed (either with the `ct` command or as part of a scan), the value of the PV is returned, regardless of counting time. That is, if the PV does not change, then `ct 1` and `ct 10` would return the same values for the `value1` macro counter.

The scaler screen of the config file might look like this:

---

#### Scaler (Counter) Configuration

Numb	Name	Mnemonic	<>Spectro	<>Device	Unit	Chan	<>Use As	Scale
0	Seconds	sec	fourc	EPICS_SC		0 0	timebase	1e+07
1	Monitor	mon	fourc	EPICS_SC		0 1	monitor	1
2	Detector	det	fourc	EPICS_SC		0 2	counter	1
3	Value1	value1	fourc	MAC_CNT		0 0	counter	1
4	Value2	value2	fourc	MAC_CNT		0 1	counter	1

Number of counters (geo/all) 5 / 5 Type ? and H for help, ^C to quit

---

## References

- [1] Certified Scientific Software, Cambridge, MA.
- [2] [http://www.certif.com/spec\\_help/epics.html](http://www.certif.com/spec_help/epics.html)
- [3] [http://www.certif.com/spec\\_help/mac\\_hdw.html](http://www.certif.com/spec_help/mac_hdw.html)