pyScan user's manual

Masamitsu Aiba

December, 2015

pyScan is a python class that performs a scan for single or multiple given knobs. The user may specify single or multiple observables. The knobs and observables should be available in Epics channels. Concerning the screen image, it is planned to implement a feature that the analyzed values (beam size, coupling correlation, etc.) can be the observables and the screen images are saved on the disk. For now, only raw screen images are returned as measured observable. For this temporal implementation, one has to be careful with the number of images to be returned since it may cost significant amount of memory and time.

pyScan can be used as a base class when 'Scan server' is to be implemented.

Type of scan

The simplest scan would be varying single knob. This is referred to as 'single-knob scan' (SKS) in this manual. When several knobs are varied at the same time, it is 'multi-knob scan' (MKS). The input parameters for SKS and MKS are given in the form of python dictionary as described below. The observable can be single or multiple for any type of scan.

More complicated scan can be built by combining SKSs and MKSs. This is achieved by defining multiple input dictionaries. For example, 'N-dimensional scan' can be defined by a combination of *N* SKSs. The observable is specified only in the last dictionary that is nested most inside of the measurement loop.

Input dictionary

Input of the scan is given in the form of python dictionary. This may allow the developer to include more features in the future, keeping the backward compatibility. The presently implemented fields are shown below.

Indict['Knob']: The name of knob (Epics channel) given as a string for SKS while as a list of strings for MKS.

Indict['KnobReadback']: Knob may be Epics channel to set the value to the machine (for example, Quad:I-SET) while there is normally a corresponding read-back channel (Quad:I-READ). The user can specify the read-back channel to check if the knob reached to the desired value within a tolerance. It is given as a string for SKS and as a list for MKS. If not given, Knob list is copied to KnobReadback (and thus no check on the readback).

Indict['KnobTolerance']: The tolerance for the knob read-back described above. If not given, it is set to be 1. It is given as single value for SKS and a list for MKS.

Indict['KnobWaiting']: Waiting (or timeout) for setting the knob in units of seconds. It is given as single value for SKS and a list for MKS. If not given, it is set to be 10 seconds.

Indict['KnobWaitingExtra']: The user may wish to set a waiting time even when it is confirmed that the knobs have reached to the set value, for example, the case where the observable is selected to be an Epics channel that shows a value averaged by the hardware. It is given as single value for SKS and MKS (counting starts after all the knobs reaching to the set value within the tolerance). If not given, it is set to be zero.

Indict['Additive']: If True (or 1), the Knob(s) is(are) varied with respect to the present value. If not given, the default is False.

Indict['ScanRange']: The range of the scan given as a list containing the initial and final values for SKS. Give a list of the scan ranges for MKS.

Indict['Nstep']: The number of scan points. It is given as an integer number for SKS and MKS.

Indict['StepSize']: The step size of the scan. It is given as a float for SKS. For MKS, give Nstep instead. StepSize is ignored when Nstep is given.

Indict['ScanValues']: Instead of 'ScanRange', the knob values to be scanned can be given as a list for SKS and as a list of value lists for MKS. The order of the nested list corresponds to the order of Knob list. This allows the user to set arbitrary knob setting rather than equidistant steps within a scan range. For MKS, the number of elements in each value list should be the same. Otherwise, the measurement is to be terminated at the end of the shortest list. ScanRange, Nstep and StepSize are ignored when this field is given.

Indict['Observable']: The observable (Epics channel) is given here. Multiple observables could be given in a list. Valid only for the most inside input dictionary.

Indict['Validation']: Channels to validate the observable can be given here. For example, BPM:POS-VALID is a channel to show if BPM:X/Y channels are valid or not. The format is a list pf channels to be recorded. Valid only for the most inside input dictionary. If not given, no validation is performed. Note that this validation does not stop or abort the scan and simply give a report, if the observables are valid or not, in the output. If the user wishes to include any action in the case where the observables are invalid, use Monitor described below.

Indict['NumberOfMeasurements']: The number of measurements for each setting of the knobs. Give as an integer. Valid only for the most inside input dictionary. If not given, it is set to 1.

Indict['PreAction']: The setting that the user needs to put into the machine before the scan can be passed to the scan function. The format is [[Ch-set, Ch-read, Value, Tolerance, Timeout=10],…], where Ch-set is the Epics channel to be set, Ch-read is the corresponding readback channel (can be the same to Ch-set).

Indict['PreActionWaiting']: Waiting after PreAction, given as a float in units of seconds. It will be zero if not given. The user may want to set this, for example, if PreAction includes magnet cycling.

Indict['PreActionOrder']: The pre-actions may be needed to be done in a defined order. For example, a series of actions, closing beam shutter -> magnet cycling -> opening beam shutter. The order can be specified as [0,1,2] for this example. If not given, all the actions are invoked almost at the same time.

Indict['PostAction']: The action to be taken after the measurement. Actions can be given in the same format to PreAction. String 'Restore' can be used instead of the list to restore the setting of the knobs as before the scan.

Indict['Monitor']: During the scan, there may be the channels that must be being watched. A typical channel would be the bunch charge at a BPM to confirm if the beam is on. Single or multiple channels can be given as a list. Valid only for the most inside input dictionary.

Indict['MonitorValue']: The nominal value for the cannel to be monitored. It is given as a list with the same order to the list of Monitor. Valid only for the most inside input dictionary. If not given, the values at the time of initialization are taken from the machine.

Indict['MonitorTolerance']: The tolerance for the cannel to be monitored. It is given as a list with the same order to the list of Monitor. Valid only for the most inside input dictionary. If not given, 10% of the MonitorValue is taken. For string (waveform) channels, this field does not make sense and put None (or whatever). Valid only for the most inside input dictionary.

Indict['MonitorAction']:The action to be taken when the value under monitoring is out of the tolerance. It is given as a list with the same order to Monitor list. The action can be 'Abort' to immediately stop the measurement, 'Wait' to wait until the monitored value comes back to the nominal value, and 'WaitAndAbort' to wait for a specified timeout and then abort the measurement if the value does not come back. The action can/should be given to each monitored channel as a list. The measurement continues, taking one step back. 'WaitAndNoStepback', however, continues the measurement without taking one step back. In the case of Abort, PostActions are taken place before leaving the scan. Valid only for the most inside input dictionary.

Indict['MonitorTimeout']: Specify the timeout to abort the measurement in units of seconds. Single value (float) would be enough for our purpose even when several channels are under monitoring. If not given, it is set to be 30 seconds. Valid only for the most inside input dictionary.

Output dictionary

The output of the scan is returned as a dictionary. The same implementation philosophy to the input is applied. The presently implemented fields are shown below.

Outdict['ErrorMessage']: Error message describing the reason for failure when the initialization or the scan was not successful. When successful, it will be *None*.

Outdict['KnobReadback']: The actual value of the knob(s) at the time of measurement. For an SKS, it will be a list of read-back values. For an MKS, it will be a nested list of read-back values. For a complicated scan, for example a 3-dimensional scan, the user extracts the read-back values as Outdict['KnobReadback'][i][j][k] for the measurement identified by the index (*i, j, k*).

Outdict['Observable']: The measured value of the observable. The results are stored with the same order to KnobReadback. When multiple observables are assigned, each result is give as a list with the order defined in Indict['Observable'] . Note that the nesting is one level deeper, including the repeated measurement result for the same knob setting.

Outdict['Validation']: The result of the validation during the scan is reported here as a nested list with the same order to Observable.

Launch a scan

After the input dictionary is prepared, the scan is launched as follows:

```
################################

from pyScan import *

# Define your input dictionary here

pscan=pyScan()

outdict=pscan.initializeScan(indict)

outdict=pscan.startScan()

pscan.finalizeScan() # Needed after every scan..

################################
```

For a nested scan, the initialization is done by

```
pscan.initializeScan([indict0, indict1, indict2, …, indictN])
```

where *indictN* is defining the scan nested most inside of the measurement loop. Therefore the observable is defined only in *indictN*. The other input dictionaries are the input for the outer measurement loop.

Schematically, the measurement loop looks like

```
for indict0

    for indict1

        for indict2…
```