CS-STUDIO SCAN SYSTEM PARALLELIZATION*

K.U. Kasemir, M.R. Pearson, ORNL, Oak Ridge, TN37831, USA

Abstract

For several years, the Control System Studio (CS-Studio) Scan System has successfully automated the operation of beam lines at the Oak Ridge National Laboratory (ORNL) High Flux Isotope Reactor (HFIR) and Spallation Neutron Source (SNS). As it is applied to additional beam lines, we need to support simultaneous adjustments of temperatures or motor positions.

While this can be implemented via virtual motors or similar logic inside the Experimental Physics and Industrial Control System (EPICS) Input/Output Controllers (IOCs), doing so requires a priori knowledge of experimenters requirements. By adding support for the parallel control of multiple process variables (PVs) to the Scan System, we can better support ad hoc automation of experiments that benefit from such simultaneous PV adjustments.

MOTIVATION

EPICS has been used with great success on the SNS accelerator, and is now a key component of on-going beam line software updates [1]. Compared to the accelerator, each beam line presents a much smaller number of devices to control. At the same time, the beam line environment is more flexible as devices are added, replaced, or operated in different ways.

The CS-Studio Scan System was developed to allow the flexible assembly and execution of "recipes", that is lists of commands [2]. After gaining operational experience on a couple of beam lines, the need for parallel command execution became obvious.

For example, the control of incident beam energy typically requires the adjustment of several neutron chopper settings as well as motor positions for slits or sample orientation. Executing these in parallel instead of sequentially allows for a significantly faster experiment preparation.

Another example is the combined movement of motors or temperatures for ad-hoc experiments. Ideally, the need for the parallel movement of motors is known in advance. Choosing suitable motor controller hardware will then allow configuring the desired motion curves into the hardware, resulting in the most efficient and accurate position control. In reality, the need for ganged motor movement can arise without prior notice. The control system software then needs to make a reasonable attempt at combined motor movement.

*This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

SCAN SYSTEM

Basic Scan Commands

While the scan system supports several commands [2], it is most important to understand the fundamental "Set" command, which writes a value to an EPICS channel:

```
Set("MotorChannel", value=14)
```

It can wait for a read-back to match the written value, with configurable tolerance and timeout in seconds:

```
Set("MotorChannel", value=14,
    readback="Motor.RBV", tolerance=0.1,
    timeout=30)
```

Alternatively, it can await "completion":

```
Set("MotorChannel", value=14,
    completion=True,
    timeout=30)
```

Finally, it can await "completion" and then check a read-back for matching the written value:

```
Set("MotorChannel", value=14,
    completion=True,
    readback="Motor.RBV", tolerance=0.1,
    timeout=30)
```

In the last case, the original implementation used the timeout both to await completion and then to wait for the read-back to match. Operationally, we found it more practical to only apply the timeout to the completion. Once the channel completes, the read-back is expected to match right away as soon as the scan server reads the most recent value of the read-back channel.

A motor-related channel will signal "completion" after the controller hardware moved the motor toward the commanded position, verified the encoder read-back, maybe performed several retries, corrected for backlash, and finally confirmed that the motor rests at the requested position.

If a device does not support "completion", it is tempting to rely on only the read-back as in the second example given above. That command will indeed wait until the read-back of the device matches the desired value, but it is misleading. The read-back may temporarily match during an overshoot or backlash

ISBN 978-3-95450-148-9

compensation while the controller is still adjusting the value.

The completion mechanism, called "put-callback" in the EPICS documentation, is therefore key to the reliable control of intelligent beam line devices. Consequently, the device support for SNS beam lines was carefully selected or updated to support "completion" confirmation, for example for the Parker 6K motion and Lakeshore 336/350 temperature controllers [3].

Commands Added for Parallelization

The "Parallel" command was added to the scan system. In the following example it concurrently moves two motors and adjusts one temperature:

```
Parallel(Set("Motor1", 13, completion=True, …),
         Set("Motor2", 24.5, completion=True, …),
         Set("Temperature3", 5.9, …) )
```

The scan server performs a "Parallel" command by submitting each command in its body to a separate thread, and then awaiting completion of all threads. Each individual command in the body may await completion and/or check read-backs as previously described, but in the example such details has been replaced by "…" for clarity.

In some cases the actions that need to be performed in parallel are not individual "Set" commands but again consist of multiple commands. The "Sequence" command allows construction such command chains:

```
Parallel(
  Sequence(Set("XYZMode", "Off"),
           Set("XYZSetpoint", 13, completion=True),
           Set("XYZMode", "Active"),
  Sequence(Set("ABCMode", "Off"),
           Set("ABCSetpoint", 13, completion=True),
           Set("ABCMode", "Active") )
        )
```

In this example, we assume that the theoretical "XYZ" and "ABC" devices can only be commanded to a new set point while turned off, and then need to be re-enabled. The scan commands will perform this sequence of commands for both devices in parallel.

Client/Scripting Library

Scan commands are submitted to the scan server network interface in an XML representation. Initially, we used a Java library, invoked by Jython from within the CS-Studio user interface, for this task [2]. To allow more software tools to interact with the scan server, we implemented the PyScanClient, a library that supports assembling scans, submitting them to the scan server, monitoring and controlling the execution, and downloading logged data [4]. The PyScanClient can be used with Jython as well as C-Python.

All code examples in this paper are based on the PyScanClient syntax. A short but complete program for submitting a scan looks like this:

```
from scan import *
client = ScanClient("localhost")
cmds = [ Set("Motor1", 43) ]
client.submit(cmds)
```

While the coordinated movement of two motors is best implemented in the controller hardware, a script like the following can approximate it by computing the desired points and assembling them into a list of parallel moves:

```
cmds = []
for pos in range(0, 100):
  # Demonstration of moving motors X, Y along
  # 0 .. 100 resp. 0 .. 200
  cmd.append(Parallel(Set("X", pos),
                      Set("Y", 2*pos) )  )
```

Site-Specific Settings

EPICS is a loosely coupled, distributed system. Given solely a channel name "XYZ", it is impossible to tell if that channel supports "completion", if there is a related read-back channel, or which timeout and tolerance would be suitable for comparing the read-back to the desired value.

A site-specific standard for channel names can often determine that for example all channels with names containing ":Mot:" are motors, so they should be controlled with completion, using a read-back channel that is based on the same channel name with an added ".RBV" suffix.

The PyScanClient library supports creating such a site-specific setup based on a combination of regular expressions for channel names with python code to handle specific channel names:

```
# Site-specific scan settings
class BeamlineScanSettings(ScanSettings):
   def __init__(self):
      super(BeamlineScanSettings, self).__init__()
      self.defineDeviceClass(".*:Mot:.*",
            completion=True, readback=True,
            tolerance=0.1, timeout=30)
      self.defineDeviceClass("BL:Mot:X42",
            completion=True,
            readback="BL:Mot:X42_Encoder",
            tolerance=0.001, timeout=60)

   def getReadbackName(self, device_name):
      if ":Mot:" in device_name:
        return device_name + ".RBV"
      return device_name
```

Based on this example, scan commands that access channels which contain ":Mot:" in their name will use

ISBN 978-3-95450-148-9

completion and read-back verification as just described with a default tolerance of 0.1 and a timeout of 30 seconds. One specific motor, "BL:Mot:X42", will use a designated encoder channel for read back instead of the default "BL:Mot:X42.RBV".

When the PyScanClient library has such site-specific settings, individual scan commands can still override them. For example, while a "Set" command for a temperature controller may default to awaiting completion, an experiment that needs to take data while the temperature changes can issue a "Set" command with "completion=False" to override the default.

Meta Commands

The site-specific settings can introduce meta-commands. For example, starting an experiment may be defined as follows to reset counts, start the data acquisition and assert that it enters the correct mode:

```
def Start():
    return Sequence(Set("Det:Counts", 0),
                    Set("BL:DAQ:Mode", "ON"),
                    Wait("DAQ:State", "Running") )
```

When assembling a scan within a python script, this "Start" command can now be used just like the predefined scan server commands. Wrapping its internal commands into a "Sequence" asserts that they will be executed sequentially even when added to the body of a "Parallel" command.

Table Scan

The PyScanClient library offers a table-based abstraction for creating scans. It its most basic form, each column of the table specifies a channel name. Cells in each row provide desired values:

Temperature3	BL:Mot:X42
50	1
100	2

This table creates the following scan:

```
Set("Temperature3", 50)
Set("BL:Mot:X42", 1)
Set(Temperature3", 100)
Set("BL:Mot:X42", 2)
```

Cells can remain empty if a device should not be changed in that row. Cells that contain ranges or lists are expanded left to right:

Temperature3	BL:Mot:X42
[50, 100]	range(1, 3)

This table creates the following scan:

```
Set("Temperature3", 50)
Set("BL:Mot:X42", 1)
Set("BL:Mot:X42", 2)
Set(Temperature3", 100)
Set("BL:Mot:X42", 1)
Set("BL:Mot:X42", 2)
```

"Wait For" and "Value" columns are used to start data acquisition, await a condition, and stop data acquisition. The specifics of how to start and stop data acquisition are provided in the form of meta-commands in the site-specific PyScanClient settings. For SNS, these consist of "Set" commands that instruct the streaming neutron event acquisition to start respectively stop. A simple yet complete scan could be expressed like this:

BL:Mot:X42	Wait For	Value
range(1, 3)	NeutronCount	1e5

This table creates the following scan, assuming that the start and stop commands in the PyScanClient are configured to write to a "DataAcq" channel:

```
Set("BL:Mot:X42", 1)
Set("DataAcq", "Start"),
WaitFor("NeutronCount", 1e5),
Set("DataAcq", "Stop")
Set("BL:Mot:X42", 2)
Set("DataAcq", "Start"),
WaitFor("NeutronCount", 1e5),
Set("DataAcq", "Stop")
```

Before taking data as just shown, a preceding table row to can adjust the sample environment The sample environment adjustments need to complete before taking data, but they can be performed in parallel. A "+p " prefix in the column header instructs the PyScanClient library to use parallel commands:

+p T	+p P	BL:Mot:X42	Wait For	Value
30	45			

This table results in the following commands:

```
Parallel(Set("T", 30),    Set("P", 45))
```

The following table describes an experiment that commands two temperature controllers to initial setting, and then takes data while the temperatures are changed to a different value:

+p T1	+p T2	Wait For
30	40	
300	330	Completion

Additional columns could previously adjust the rate-of-change on the temperature controllers to obtain the desired temperature variation over time.

User Interfaces

The primary SNS beam line user interface is CS-Studio, which includes PyDev, a Python development environment. Users can edit, inspect, debug and execute

ISBN 978-3-95450-148-9

Python scripts that assemble and submit scans in CS-Studio.

In practice, however, most users prefer a graphical interface to a script editor. For routine operations, a CS-Studio operator interface panel specifically created for the task allows adjusting the required parameters. When users push a "Submit" button, this triggers a python script to read the parameters from the display, assemble the scan commands, and submit them to the scan server.

All SNS beam lines have so far been able to utilize a common "Alignment Scan" user interface that allows moving once device from start to end, logging an EPICS channel at each point, and finally performs a Gaussian fit to the data.

Table scan support is integrated into the CS-Studio user interface by allowing users to load, edit, save and submit table scans. The editing inside CS-Studio is admittedly limited, so more complex tables are best prepared in dedicated spread sheet programs like Gnumeric [5], and only loaded into CS-Studio for a final review and submission.

While the scan system user interfaces appear similar to those described in [2], they now utilize the PyScanClient library. As a result, it is now easy to extend and combine them. For example, a custom experiment that uses a python script to assemble scan commands can now include an alignment or table scan, while adding additional commands to be executed before and after those operations.

CONCLUSION

The scan system has been very reliable since its introduction in early 2013. It is now used on seven ORNL beam lines (HFIR CG1D, SNS USANS, VULCAN, CORELLI, HYSPEC, VISION, SEQUOIUA) as the main experiment automation interface for beam line staff and visiting experimenters. In case of problems, the combination of scan system console logs and archived control system data [6] has always allowed us to determine which device had timed out or not responded as expected.

We prefer to implement support for routine beam line operations in IOCs. Beam line energy adjustments, ganged operation of temperature controllers, complex multi-motor movements are whenever possible based on EPICS database logic, EPICS sequences, or python-based IOCs, because this allows for more thorough testing, optimization and also long-term archiving of key parameters.

Nevertheless, the introduction of "Parallel" and "Sequence" scan commands makes it very easy to perform timesaving instrument adjustments, or to approximate coordinated movements for ad-hoc scenarios when such prior IOC-based implementation was not possible.

As we update more advanced SNS instrument control systems, we initially assumed that the required automation could only be performed with custom scripting. In practice, we found that the Table Scan abstraction supports most of them. The table itself may no longer be entered by a user but is instead created by a python script, and the PyScanClient library allows such scripts to directly submit the resulting scan.

ACKNOWLEDGMENT

We thank Qiu Yongxiang, Dylan Maxwell, and Guobao Shen for their collaboration in the PyScanClient development.

REFERENCES

[1] X. Geng, X.H. Chen, K. Kasemir, "First EPICS/CSS Based Instrument Control And Acquisition System at ORNL", ICALEPCS 2013, San Francisco, CA, USA (2013).

[2] K. Kasemir, X. Chen, E. Berryman, "CSS Scan System", ICALEPCS 2013, San Francisco, CA, USA (2013).

[3] X.H. Chen, K. Kasemir, "BOY, A Modern Graphical Operator Interface Editor and Runtime", PAC 11, New York, NY, USA (2011).

[4] M. Pearson, "EPICS on SNS Instruments", EPICS Collaboration Meeting, Michigan State University, East Lansing, MI (2015).

[5] https://github.com/PythonScanClient/PyScanClient

[6] http://www.pydev.org

[7] http://www.gnumeric.org

[8] K. Kasemir, "Control System Studio Data Browser", PCaPAC08, Ljubljana, Slovenia (2008).

ISBN 978-3-95450-148-9