# CSS SCAN SYSTEM*

K. Kasemir, X. Chen, ORNL, Oak Ridge, TN 37830,
E. Berryman, MSI, Lansing, MI 60439, U.S.A.

*Abstract*

Automation of beam line experiments requires more flexibility than the control of an accelerator. The sample environment devices to control as well as requirements for their operation can change daily. Tools that allow stable automation of an accelerator are not practical in such a dynamic environment. On the other hand, falling back to generic scripts opens too much room for error. The Scan System offers an intermediate approach. Scans can be submitted in numerous ways, from pre-configured operator interface panels, graphical scan editors, scripts, the command line, or a web interface. At the same time, each scan is assembled from a well-defined set of scan commands, each one with robust features like error checking, time-out handling and read-back verification. Integrated into Control System Studio (CSS)[1], scans can be monitored, paused, modified or aborted as needed. We present details of the implementation and first usage experience.

## BEAM LINE EXPERIMENTS

A beam line control system interfaces to various devices, for instance motors, temperature controllers, detectors. Beam line users need to control these devices at a safe distance from the beam. Experiments further necessitate automation, since manual control over many hours of operation would be impractical. Finally, experiment readings need to be saved and later analyzed for scientific information.

The Python scripting language offers access to devices via Process Variables (PVs) in the Experimental Physics and Industrial Control System (EPICS)[2]. Especially for simple experiments, it is tempting to combine automation logic, data acquisition, and associated user interface within one piece of scripted software. It is easy to create such scripts from scratch. They can be modified as needed to include new automation requirements.

The monolithic script approach has considerable disadvantages: Ad-hoc scripts tend to only function satisfactorily until there are errors, for example network issues or devices responding in unexpected ways. Such potential errors tend to be overlooked when writing "a simple script". The experiment may often continue, only for users to learn much later that the acquired data is of no use.

Standalone scripts typically run until they finish. There is no way general to monitor their progress, to pause or abort them in a graceful manner. Such a basic script may close the beam line shutter at the end of an experiment, but only if the script is allowed to execute until its last line. If the user or an error aborts the script earlier, the beam line shutter may accidentally remain open!

From a software engineering standpoint, a disjunctive collection of custom, standalone scripts will also be hard to maintain in the long run.

## CSS SCAN SYSTEM

In 2011, we became aware of the "Scan Engine" design by the Software Services Group (SSG), Advanced Photon Source, Argonne National Laboratory. It proposes a robust, modular execution engine, decoupled from the user interface. We implemented its basic design for the CSS framework, resulting in the following components.

### *Scan Server*

The CSS Scan Server executes 'Scans'. Each Scan is a list of commands, including:

- *SetCommand*: Sets a Device to a value, i.e. writes to a PV. By default, it waits for the PV to return a matching value, within a configurable tolerance. It supports devices where set point and read-back have separate PVs. For EPICS, a "put-callback" operation is supported.
- *LoopCommand*: Steps a PV from an initial to a final value, with configurable step size. Each step is similar to a *SetCommand*.
- *WaitCommand*: Can wait for a PV to reach a certain value, increment by a certain amount and more.
- *LogCommand*: Add current values of PVs to the scan log.
- *DelayCommand*: Simply waits for a certain time.
- *ScriptCommand*: Executes Jython code.

The *SetCommand* illustrates the difference between a rudimentary script that writes to a PV, and a more robust system that includes error handling and read-back verification. All waiting commands support time-outs.

When a required functionality is not covered by the basic command set, the *ScriptCommand* can invoke Jython code, which allows use of a versatile programming language. Site-specific commands can be added via Java-based extension points.

The CSS support for PVs can interface to EPICS, including V4[3], as well as simulated PVs for testing. The Scan Server is packaged as a CSS/Eclipse console application. Under Linux, we typically execute it as a service with 'telnet' access for life cycle management and debugging.

The Scan Server is meant to only perform experiment automation. Via PVs, it may control the experiment data acquisition, but it is not meant to perform the actual data acquisition. Nevertheless, the Scan Server can perform basic PV logging, to track the progress of a scan or for very simple data acquisition. The log persistence layer is based on an extension point, with a default implementation for Apache Derby.

ISBN 978-3-95450-139-7

Scans are submitted, monitored and controlled via a RESTful web interface, i.e. from a web browser or other networked programs.

Scans are executed in the order of submission, which allows users to schedule follow-up experiments while in an ongoing experiment. A scan can be paused and resumed between commands, or aborted at any time.

The scan server can be configured to execute 'pre-' and 'post-scan' commands before and after each scan. For example, it could open respectively close a beam line shutter for each scan. The 'post-scan' commands will be executed no matter if the scan finishes successfully, or stops from an error or manual abort.

### *Scan Editor*

Scans are submitted as an XML-formatted list of commands to the Scan Server. The Scan Editor is a CSS tool for creating and modifying Scans, shielding users from the underlying XML format. Commands are added to a scan via drag-and-drop, and then their properties are configured within the editor. Online help describes the functionality of each command. Scans can be saved to files and later re-loaded.

The Scan Editor can submit scans to the Scan Server for execution, or download previously submitted scans, including scans submitted from other tools. When a submitted or downloaded scan is under execution, the scan server indicates the currently executed command.

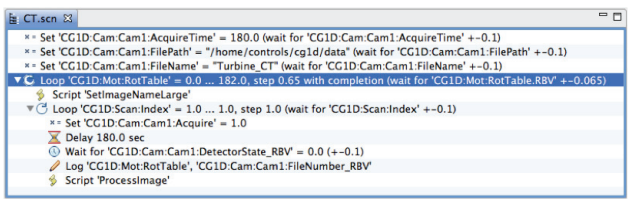


Figure 1: Scan Editor.

### *Scan Monitor*

The CSS Scan Monitor lists all scans on the server. (Fig. 2). Users can pause, resume, abort scans, open a scan in the Scan Editor, or view the logged data of a scan in the Scan Plot.

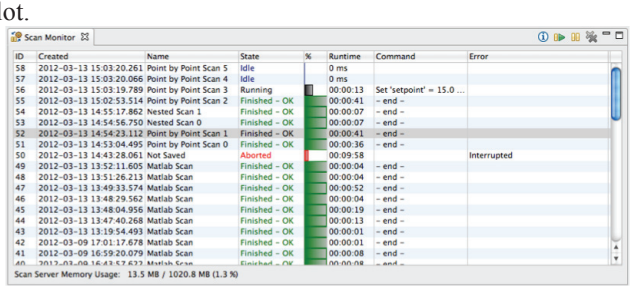


Figure 2: Scan monitor.

Executing scans are displayed with accumulated run time and an estimate for their finish time.

### *Scan Plot, Scan Data Table*

These CSS tools can display the logged data of a scan in either a plot or as a table of values, which can be exported to a file.

### *BOY Integration*

CSS BOY [4] is a tool for creating graphical user interfaces that display and control PVs. JavaScript or Jython code attached to the displays can read data from a display, create a scan and submit it, see Fig. 3 for an example.

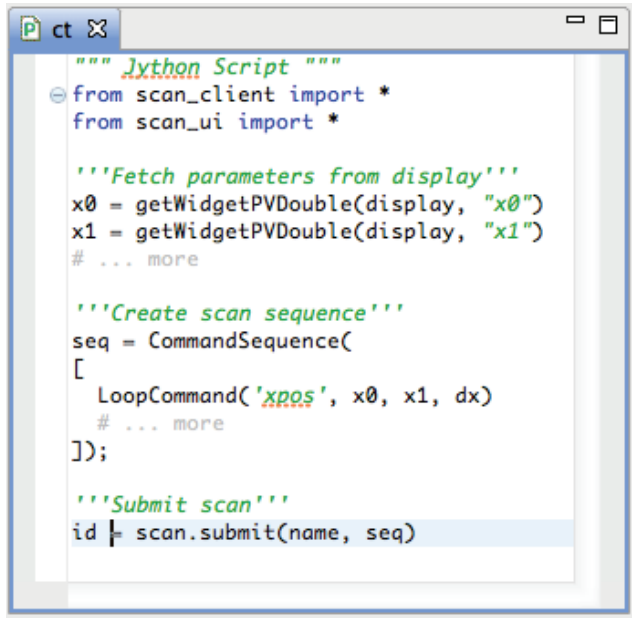


Figure 3: BOY script .

This functionality is used extensively to create operator interfaces for routine scans. Beam line visitors can thus configure and submit a scan from the same operator panels that they use to manually control the beam line devices.

## EXAMPLES

### *Computer Tomography (CT) Scan*

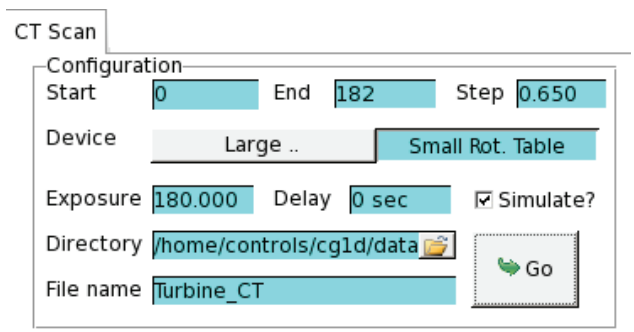


Figure 4: BOY Panel for CT scan.

Neutron computer tomography is a routine task at the ORNL CG-1D beam line. It involves rotating a sample for about 180 degrees in small steps, taking an image at each



ISBN 978-3-95450-139-7

step [6]. A BOY user interface, Fig. 4, allows experimenters to configure the desired rotation, camera exposure, and details of the saved image.

When users push the user interface "Go" button, a script similar to Fig. 3 reads the desired rotation start- and end-angles from the display panel to create a *LoopCommand*. It adds *SetCommands* to control the camera, eventually submitting a scan similar to the one shown in Fig. 1.

Instead of submitting the scan for execution, the CT Scan BOY panel also allows to first submit a scan for simulation. The simulation feature of the Scan Server returns an estimate of the required execution time. This is helpful when users need to balance desired image intensity and CT detail against the resulting execution time, which is typically 12 hours and longer.

In this CT example, the scan itself is very simple, easily handled in a standalone script. The advantage of the CSS scan system lies in the seamless integration of camera control, beam line motors, and scan control into one user interface, Fig. 5.

At the same time, the execution of typically long running scans is decoupled from the user interface. Scans can continue undisturbed for many hours while the user closes the operator interface, logs out of the computer. Later, another user may log in and open the operator panel to review the scan progress.

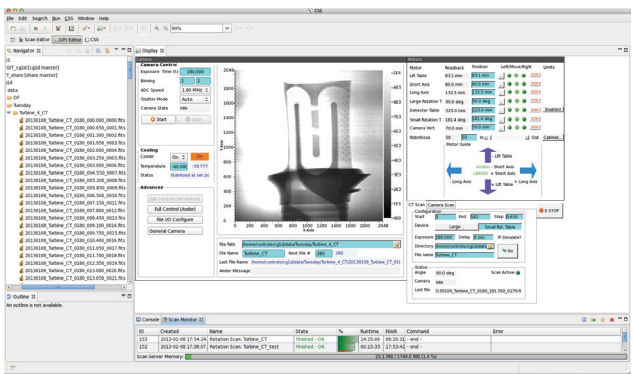

Figure 5: Overall CT Beam Line Operator Panel. Note CT section from Fig. 4 on lower right.

## *Edge Fit*

A routine beam line task is scanning a variable, often a motor, over a value range, logging some detector signal, and finally locating an 'edge' in the logged data.

Fig. 6 shows a BOY panel for configuring the desired scan and log parameters. Operators can enter arbitrary PV names, or select from lists of pre-defined PV names. At each step, the scan can be instructed to simply wait for a fixed time, await a configurable number of neutron counts, or meet other beam-line specific conditions.

The submitted scan will use *Loop*, *Wait* and *LogCommands* to collect the data, shown in Fig. 6 as a blue line. To perform the edge fit, a *ScriptCommand* invokes suitable Jython code, writing the result to PVs. These are displayed at the bottom of Fig. 6 as the "Edge Position" and "Width"; they are also indicated via re vertical plot markers.

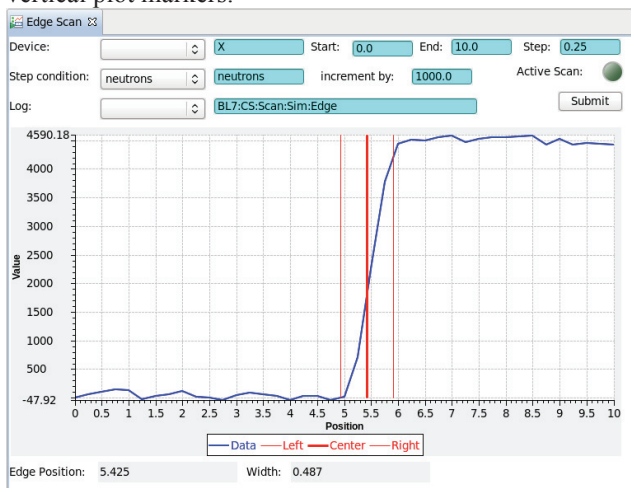


Figure 6: Scan with edge fit.

## *Table Scan*

In a table scan, users provide a list of values, for example a list of X/Y coordinates. It is configured via a table as shown in Fig. 7. The values in columns "X", "Y", and following represent the desired values for devices of this name. The scan server supports alias names, so a short device name like "X" can internally be translated into the actual PV name. The generated scan will start with *SetCommands* to move all devices to the values listed on the first line, followed by commands to move all devices to the values on the next line and so on.

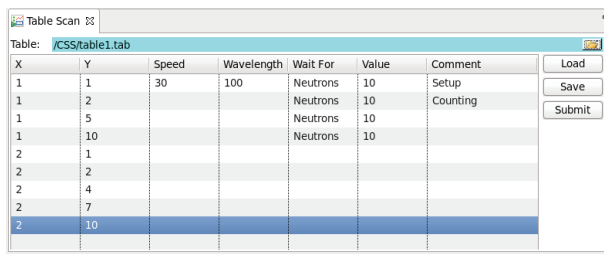


Figure 7: Overall CT Beam Line Operator Panel. Note CT section from Fig. 4 on lower right.

The "Wait For" and "Value" columns allow entering a condition on which the scan will wait via suitably generated *WaitCommands* before executing the commands for the next row.

Such a tabular representation of a scan can be especially useful if for example X/Y coordinates are not based on simple loops, but generated by other tools. Written as a spreadsheet-type, comma-separated table, the coordinates can be loaded into the Table Scan editor and submitted to the Scan Server.

## *MATLAB*

The same Java code used within the previously described CSS tools is also accessible from MATLAB [7].

ISBN 978-3-95450-139-7

This allows advanced MATLAB users to for example compute coordinates for an elaborate X/Y scan, then submitting them to the Scan Server as a list of *SetCommands*.

MATLAB can fetch logged Scan data, and then use its sophisticated numeric algorithms to perform fits beyond the basic example from Fig. 6.

## DISCUSSION

The decoupling of direct device access and user interface from the scan server is key to its robustness. All device access is via network PVs. The scan server itself has no graphical user interface. Scan execution can be monitored and controlled via networked user interfaces that are opened and closed as desired.

This separation of execution from the display allows the scan to continue without a user interface. It is also possible to open several user interfaces, for example one on the beam line and one in another office, to monitor the same scan.

A CSS BOY display can provide seamless integration to the end user who presses a "Start" button to initiate a scan, then observing its progress on the same display. There is, however, added effort for the control system engineer. The "Start" button requires a script to submit a suitable set of commands to the scan server. To report progress, the scan server cannot for example draw the path of an X/Y scan directly on the operator screen. Instead, the scan server needs to publish the X/Y positions of a progressing scan by writing to waveform PVs or the scan log, which may then be displayed in the user interface.

The Scan Server executes a set of well-tested commands, mostly *Set*, *Wait*, *Loop*, *Log*. Special requirements can be met via the *ScriptCommand* and Jython code. Such Jython code must be created with care. It is unlikely to ever be as well tested for robustness as the build-in scan commands, but the *ScriptCommand* was necessary to support for example an Edge Scan.

Jython, the Java-based implementation of Python, is an excellent general programming language, but it does not support the full numeric computing functionality of NumPy [8]. Instead, the Scan System includes CSS NumJy, a basic implementation of the NumPy array support, similar to the SciSoft Dataset of GDA [9].

## ONGOING WORK

While a seamless user interface for the scan system is possible, the intermediate steps of writing to PVs or the log results in additional effort for the engineer who creates these operator interfaces.

Recent work on the PVManager[5] simplifies the submission of scans. Instead of requiring BOY script code, basic 1-dimensional scans can be submitted by invoking a PVManager service. Future work will include adding the ability to register scan configuration files "on-the-fly" as PVManager services.

A prototype EPICS V4 network server was added to the Scan Server. This allows V4 network clients, including BOY, to display data from the scan log as a table or in plots based on PVManager expressions, again without need for scripts.

## SUMMARY

The CSS Scan Server is a modular, robust building block for experiment automation. The Scan System has been in successful operation at an ORNL beam line since January 2013 [6].

We would like to thank Claude Saunders and Tim Mooney for valuable information on the original SSG Scan Engine design.

## REFERENCES

[1] K. Kasemir, "Control System Studio Applications", ICALEPCS, Knoxville, TN, Oct. 2007.

[2] T.T. Nakamura, K. Furukawa, J-I. Odagiri, N. Yamamoto, "Development of the Software Tools Using Python for EPICS-Based Control System", ICALEPCS, Knoxville, TN, Oct. 2007.

[3] L.R. Dalesio et al, "EPICS V4 Expands Support to Physics Application, Data Acquisition, and Data Analysis", ICALEPCS, Grenoble, France, Oct. 2011.

[4] X.H. Chen, K. Kasemir, "BOY, A Modern Graphical Operator Interface Editor and Runtime", PAC2011, Brookhaven, NY, March 2011.

[5] G. Carcassi, "pvManager", EPICS Meeting, Brookhaven, NY, Oct. 2010.

[6] X. Geng, X. Chen, K. Kasemir, "First EPICS/CSS Based Instrument Control and Data Acquisition System at ORNL", ICALEPCS, San Francisco, Oct. 2013.

[7] MathWorks MATLAB software, http://www.mathworks.com/products/matlab

[8] NumPy, scientific computing in Python, http://www.numpy.org

[9] GDA, Software for Science, http://www.opengda.org



ISBN 978-3-95450-139-7