ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-MOP018

MODERNIZING WIRE SCAN DIAGNOSTICS FOR REPRODUCIBLE, REAL-TIME BEAM MEASUREMENTS THROUGH A MODULAR PYTHON MIDDLE LAYER INTEGRATED WITH EPICS*

T. Kabana†, S. Chowdhury, B. Jacobson, N. Neveu , E. Yang, C. Zimmer
SLAC National Accelerator Laboratory, Stanford, CA, USA

Abstract

As part of a broader effort to modernize beam diagnostics at SLAC, we are developing a new middle-layer application to support wire scan measurements using Python. This tool is designed to replace aging MATLAB GUIs with a streamlined framework that interfaces directly with EPICS and Beam Synchronous Acquisition systems. The middle layer manages the complete wire scan workflow while emphasizing modularity, reproducibility, and integration with existing controls infrastructure. This paper will cover the system architecture, practical implementation details, and lessons learned in deploying a diagnostic tool suited to the high-throughput, real-time needs of accelerator operations.

INTRODUCTION

Wire scan diagnostics are a common tool in accelerator operations, providing precise measurements of transverse beam profiles [1]. Historically, these systems have relied on rigid, all-in-one software that is difficult to maintain, extend, or reuse. The existing MATLAB GUI spans several thousand lines of code and contains configuration data, scanning logic, analysis routines and display controls in a single file. As machine complexity and measurement demands grow, more flexible, modular tools are needed to enable rapid development, robust testing, and integration with diverse hardware

To address these needs, we developed a modern Python based wire scan framework with an object-oriented middle layer that encapsulates EPICS process variables (PVs) and controls logic into reusable device abstractions [2]. Motion control, data acquisition, and analysis are implemented as independent components, improving testability, simplifying debugging, and accelerating hardware integration. Configuration is defined in structured YAML files validated with Pydantic [3], allowing measurement setups to be defined without modifying core logic.

This paper describes the system architecture, implementation, and operational features, including its GUI interface, dynamic configuration, and structured data output. Designed for real-time beam measurements, the tool is in use for development and testing at SLAC, with plans for broader deployment across beamline diagnostics.

SYSTEM ARCHITECTURE

The core of the wire scan modernization effort is an object-oriented middle layer written in Python. This architecture abstracts hardware controls into modular device classes, each encapsulating EPICS PVs, device metadata, and control logic. The design is built around a set of abstract base classes that enforce a consistent interface across hardware types, enabling flexible behavior and extensive code reuse.

Each physical device, such as a wire scanner, BPM, or beam loss detector, is represented by a Python class instance. Device classes define PVs as class attributes using standardized descriptors, making PV access and manipulation intrinsic to the object. For example, reading a wire position or writing a motion command is expressed naturally as *wire_device.position* or *wire_device.start_scan()*, abstracting away lower-level PV handling and promoting readable, maintainable code.

In parallel, device metadata and PV definitions are also declared in external YAML files, a human-readable, structured data format. These files define the beampath layout, region groupings, and all associated devices and PVs in a hierarchical, easy-to-maintain structure. Upon initialization, the system dynamically loads this configuration and binds the defined PVs to the appropriate device classes. This dual-layered approach of external YAML for structure and internal Python for logic enables rapid deployment of new measurement setups without modifying application code.

The result is a flexible, declarative system in which devices are defined through a combination of structured metadata and object-oriented logic. This design allows developers to test, extend, or swap out devices with minimal effort, streamlining both prototyping and production use. It also supports unit testing through mock device classes that inherit from the same base interfaces.

CONFIGURATION AND METADATA MANAGEMENT

External YAML configuration files support flexible deployment and eliminate the need for code changes. Well-suited for complex beamline configurations, YAML's hierarchical nature defines each device's PVs, name, location, and initialization data. At runtime, this metadata is loaded and bound to the corresponding device classes, making PVs intrinsic to each object and eliminating hard-coded logic.

All configurations are validated using Pydantic, which enforces strict schema compliance, catching errors at startup with clear messages indicating the exact field, expected type, and problematic value. It also provides strong typing and structured serialization allowing measurement results to be exported to YAML, JSON, or HDF5 formats.

* Work supported by U.S. Department of Energy, Contract DEAC02-76SF00515
† kabanaty@slac.stanford.edu

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-MOP018

This ensures safe integration with downstream tools such as emittance reconstruction and beta matching without custom parsing.

Separating configuration from control logic streamlines updates and promotes modular reuse. New measurement setups or devices can be added entirely through YAML edits with no changes to business logic.

MODULAR MEASUREMENT PIPELINE

The wire scan system is architected around a modular pipeline that cleanly separates each stage of the measurement process into independently testable components. The pipeline consists of four primary stages: Management, Motion, Acquisition, and Analysis. This separation simplifies debugging, improves maintainability, and allows for targeted development and reuse across different diagnostics.

Management

Configuration management constructs appropriate device objects at runtime. This layer serves as the entry point for defining beampaths, regions, and scan configurations, ensuring all components operate from a consistent set of structured inputs. This data and the associated constructor functions are part of the larger middle-layer library in use at SLAC and not exclusive to wire scan operations.

Motion

The motion stage of the measurement pipeline manages the physical positioning of the wire scanner. This includes initializing the wire device, configuring scan parameters, such as speeds or ranges, issuing motion commands via EPICS PVs, and monitoring device status for faults during travel. The use of an abstraction layer ensures that motion logic remains consistent and reusable across different software implementations.

Acquisition

The acquisition stage manages buffer reservations and data collection. It interfaces directly with SLAC's Beam Synchronous Acquisition (BSA) system to capture wire position and detector signals with precise timing alignment [4]. BSA associates each measurement with a unique pulse ID, enabling high-fidelity correlation between device readbacks and beam conditions. During a scan, acquisition components reserve buffers, configure timing masks and destinations, and monitor completion flags to ensure accurate, synchronized capture

Beam loss is measured using two complementary methods: localized loss from scattered electrons collected by PMTs and beam loss monitors near the wire, and transmitted beam loss (TMITLOSS) is measured as a percentage change in beam charge using BPM data [5].

All collected data, regardless of source, is returned as structured Pydantic models, providing a uniform format across detectors, device, and diagnostics. This eliminates detector-specific handling in later analysis and, through collaboration with downstream tool developers, ensures compatibility and seamless data flow through the diagnostic pipeline.

Analysis

Acquired data is processed in real time, transforming raw detector signals and wire positions into meaningful beam size measurements. After acquisition, the system separates detector responses into individual beam profiles based on expected profile ranges. Each profile is normalized and fit using a selected model to extract the transverse beam size.

The default fitting method applies a Gaussian curve to the profile, returning both the full fit parameters and derived quantities such as the RMS beam size (Fig. 1). In addition to the Gaussian model, additional fit strategies are under development to support non-Gaussian beam distributions and improve robustness across a wider range of operating conditions.

Fitting results, raw data, and metadata are returned in a structured *MeasurementResult* object, which is defined using Pydantic for strict validation and portability. This result object includes all data captured during the scan, as well as the processed results, and metadata. The advantages of this approach will be outlined in the next section.

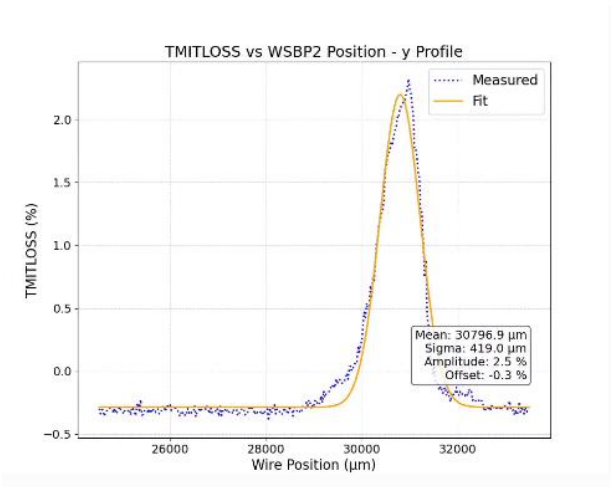

Figure 1: Vertical beam profile measurement with Gaussian fit overlay.

DATA HANDLING AND RESULT STRUCTURING

To ensure consistency and ease of integration, all wire scan outputs are returned using a structured *MeasurementResult* object defined with Pydantic. This model serves as a single container for the complete result of a scan, containing raw data, processed signals, fit results, and metadata in a validated and portable format.

A *MeasurementResult* object includes:

- Raw data: wire positions and corresponding detector signals.
- Processed data: data separated by beam profile position as determined by scan configuration.
- Fit results: parameters from Gaussian or other fitting methods, including beam centroid, amplitude, offset, and RMS beam size.

ISBN: 978-3-95450-261-5 ISSN: 2673-7000 doi: 10.18429/JACoW-NAPAC2025-MOP018

- Metadata: information about the scan configuration, available detectors, beam profile ranges, fit method used, and other supplementary notes as determined by the user.

By enforcing strict typing and validation at the model level, the result structure guarantees that all data returned from the measurement pipeline adheres to a consistent schema. This not only facilitates real-time display and logging but also ensures compatibility with downstream tools without requiring custom parsing or translation layers. As these follow-up tools continue to develop, the *MeasurementResult* format provides a stable and extensible interface for seamless integration across the diagnostic pipeline.

GRAPHICAL USER INTERFACE

The wire scan system includes a modular graphical user interface (GUI) built with Python Display Manager (PyDM) designed to provide operators with a responsive and intuitive control interface (Fig. 2) [6]. The GUI supports scan configuration and execution, live plotting of results, and saving output data. Select widgets, such as those for configuration and plotting, are independently maintained and can be repurposed across different tools, promoting code reuse and simplifying interface development.

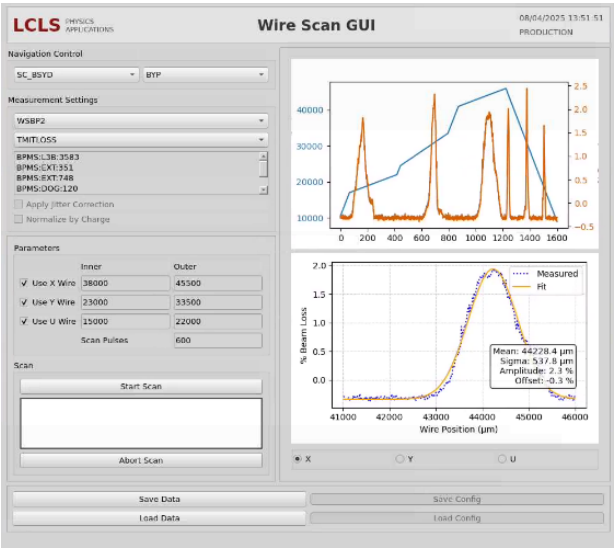

Figure 2: The PyDM Wire Scan GUI displays a successful scan with beam profile fit. Six peaks are visible, corresponding to three profile wires intercepting the beam in both direction of travel.

All displayed data is sourced dynamically from the middle layer, with no hardcoded PVs or embedded business logic. This separation of concerns ensures that the GUI remains lightweight and easy to extend while all critical control logic and device interactions remain testable in Python. The result is a maintainable interface that can adapt quickly to new devices or scan types without requiring a rework of the frontend. All functionality is also available through a command-line interface, allowing users to perform all tasks without the GUI, if preferred or required.

The GUI provides real-time feedback during each stage of the scan process, including live beam profile visualization, fit overlays, and fault/error logging display. Operators can monitor the scan status and view the latest analysis results immediately upon scan completion. The layout is designed with operations in mind, prioritizing clarity, responsiveness, and accessibility of key controls and indicators.

A custom logging module is tightly integrated into the GUI and backend, capturing messages from all stages of the pipeline. Logs are written to multiple destinations, including the GUI display, console output, and persistent log files, enabling both live monitoring and post-mortem analysis. This unified logging approach improves visibility during scans and simplifies debugging across both development and production environments.

CONCLUSION AND FUTURE WORK

This modular Python-based wire scan system improves maintainability, accelerates development, and supports a wide range of hardware through device abstraction, dynamic configuration, and structured data handling. The separation of logic across acquisition, motion, analysis, and visualization components supports robust testing and flexible development.

Future work includes expanding the toolset to additional diagnostics and downstream integration of results into emittance reconstructions.

REFERENCES

[1] N. Balakrishnan *et al.*, "Control Systems Design for LCLS-II Fast Wire Scanners at SLAC National Accelerator Laboratory", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 1075-1078. doi:10.18429/JACoW-ICALEPCS2019-WEPHA010

[2] EPICS – Experimental Physics and Industrial Control System, https://epics-controls.org

[3] S. Samuel, Pydantic: Data validation and settings management using Python type annotations, https://docs.pydantic.dev

[4] K. H. Kim, S. Allison, T. Straumann, and E. Williams, "Real-Time Performance Improvements and Consideration of Parallel Processing for Beam Synchronous Acquisition (BSA)", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 992-994. doi:10.18429/JACoW-ICALEPCS2015-WEPGF122

[5] F.-J. Decker, B. Jacobson, S. Hoobler, T. Kabana, and W. Colocho, "Beam position monitors (BPMs), using their charge information at SLAC", in *Proc. LINAC'24*, Chicago, IL, USA, Aug. 2024, pp. 762-764. doi:10.18429/JACoW-LINAC2024-THPB064

[6] H. H. Slepicka and M. L. Gibbs, "PyDM - Extension Points", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 539-543. doi:10.18429/JACoW-ICALEPCS2019-MOPHA135