THE FRIB RUN PERMIT SYSTEM*

D. Chabot[†], M. Ikegami, M. G. Konrad, D. Maxwell
Facility for Rare Isotope Beams, Michigan State University, MI 48824, USA

Abstract

The Facility for Rare Isotope Beams (FRIB) will accelerate many different ion species and charge states defining a wide spectrum of operating modes and parameters. The role of the Run Permit System (RPS) here is to examine if a requested state is suitable for the production of beam. The decision to permit beam is based on input from configuration management databases, machine and personnel protection systems, and beam characteristics and destination. From this information an appropriate set of operating parameters are deployed to hardware to support the requested mode. This paper will describe the interfaces, implementation, and behavior of the RPS at FRIB.

INTRODUCTION

The Facility for Rare Isotope Beams, currently under initial commissioning, will be capable of accelerating heavy ions up to a beam power of 400 kW [1]. Capable of accelerating a wide variety of ion species at variable beam energies, machine protection risks are of significant concern. In one of several parallel efforts to address these risks, a Run Permit System is under development and testing, which operates in concert with Personnel Protection System (PPS), Machine Protection System (MPS), and the Global Timing System (GTS).

The FRIB Run Permit System software performs several functions, acting over a set of *Critical Signals* defined here as the set of control system channels that the RPS writes or subscribes to in order to affect beam production:

- Determines if conditions are such that beam production may commence
    - Configuration management systems indicates approval of hardware for use
    - Critical Signal thresholds are distributed (eg: power supply operating ranges)
    - The absence of PPS, MPS, and Critical Signal alarms or faults
- Indicates to dependent systems that beam production may commence or continue by issuance of the *Run Permit* signal
- Determines if conditions are such that beam operations may continue in concert with PPS and MPS mechanisms - failure of Critical Signals revoke the Run Permit
- Prevents modifications to Critical Signals during beam operations

RPS COMPONENTS

Technologies

The control system software of FRIB is implemented using the EPICS toolkit, CS Studio is utilized to construct graphical user interfaces, and the RPS itself is a Python application relying on *pyepics*, *pcaspy*, *pymongo*, and *transitions* [2-7]. The database backend of the system uses *MongoDB* [8].

Machine and Beam Modes

Two of the more important concepts employed by the RPS are the notions of *Machine Modes* and *Beam Modes*. Machine Modes are hosted by the PPS and define the geographic scope of permissible beam propagation. Beam Modes are properties hosted by the Run Permit System and define the permissible range of beam power or energy, timing structure, and ramping strategies. Table 1 and Table 2 itemize samples of both.

Table 1: Machine Mode samples

ID	Description	Beam Modes
M0	Maintenance	B0
M1	Beam delivery up to linac	B0, B8
M4	Beam delivery up to experimental systems	B0, B1, B2, B5

Table 2: Beam Mode samples

ID	Time Structure/Power	Scope
B0	No Beam	Entire Machine
B1	CW/10-400 kW	Entire Machine
B8	Variable/2-650 eμA	Front End

Machine Mode, Beam Mode, ion species, and charge state together form a *state-set*, which is utilized by the RPS to query a database for the list of control system process variables (PV) and values curated for that state. Armed with this list, the RPS may then distribute the values to the hardware in preparation for beam delivery.

* Work supported by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661
† chabot@frib.msu.edu

Finite State Machine Model

The RPS models accelerator operations as a finite set of states. A diagram for this finite state machine (FSM) is shown below [Fig. 1], along with the events causing transitions between states. This design choice provides a simple, robust model with deterministic source and destination states, user and control system-based event triggers, and conditionally guarded transitions. The FSM is implemented using the *transitions* library [7].

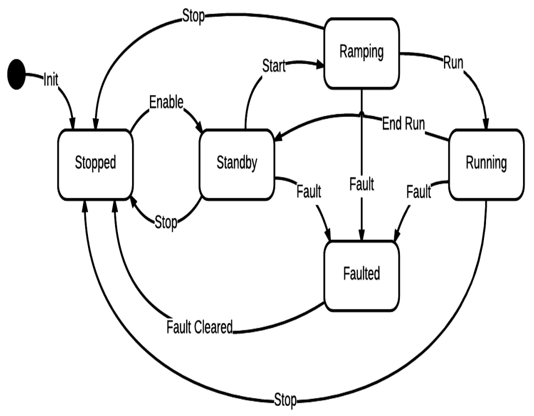


Figure 1: FSM model of the RPS.

The nature of each state in this model is explained in more detail below:

- **Stopped**: this is the idle state, where the RPS is not actively managing or monitoring. Beam may be present up to the first Faraday cup following the ion sources.

- **Standby**: while the system is in the Standby state, beam may present up to and including the Chopper.

A request to change the Machine State from Stopped to Standby is affected by operator access via the RPS' Channel Access interface. This is the Enable transition. If no fault conditions are present, the RPS will then collect the process variable values for Machine and Beam Mode, ion species, and charge state. If fault conditions are present, no transition will occur.

Using the collected state-set to query the RPS database, a set of values, ranges, and masks are returned and distributed to the Critical Signals. At the conclusion of a successful distribution, the RPS will complete the transition to the Standby state, and enable the Run Permit control system signal.

It is the Run Permit EPICS variable that systems external to the RPS must observe to govern their own behaviour. The transition of this variable between DISABLED and ENABLED controls the deactivation or activation of the following Channel Access Security rules:

  - **RPS Threshold**: EPICS Records with this Access Security Group (ASG) will prohibit writes to metadata fields (DRVH, DRVL) while the Run Permit is ENABLED.

  - **RPS Lock**: EPICS Records with this ASG will prohibit writes to their VAL fields while the Run Permit is ENABLED.

- **Ramping**: A Start transition is affected at operator command via the Channel Access interface. The GTS is then configured by the RPS to support the chosen Beam Mode and the GTS outputs are enabled.

- **Running**: The Run transition is affected when the GTS indicates its ramping sequence has concluded, and the RPS enters the Running state. If *single-shot* mode is in effect, the RPS will automatically transition back to the Standby mode in preparation for the next shot. If the chosen Beam Mode features continuous delivery, the system will remain in the Running state until instructed to issue a Stop transition (via the CA interface), or a Fault transition has been triggered.

- **Faulted**: Faults are indicated by changes to the EPICS Status and/or Severity of Critical Signals managed by the RPS. From the Standby, Ramping, or Running states, any indication of abnormal conditions will cause a transition to the Faulted state.

Exiting from the Faulted state depends on having all fault conditions cleared. When that condition is satisfied, the RPS will transition to the Stopped state, and must be Enabled again to resume operations.

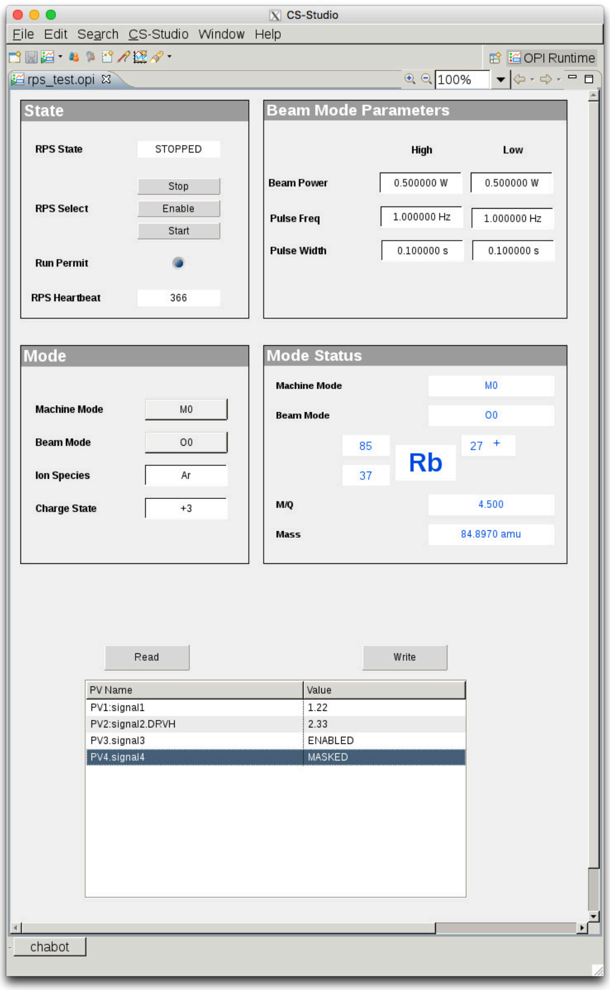
If the RPS detects a fault condition that is not a part of the set monitored by MPS or PPS, it will signal the FPS to dump the beam and issue a Fault transition (provided the RPS is in the Standby, Ramping, or Running states. If a fault condition is detected by the MPS or PPS systems prior to detection by the RPS, then those systems are responsible for initiating mitigating action and signalling the RPS to make a Fault transition.

Database

Stateful information is persisted in a backing store. Here, that store is MongoDB, a non-relational database. The primary purpose of this database is to store Critical Signals and their values as a function of accelerator state. That is, Critical Signals and their values may be queried according to the Beam Mode, ion species, charge state, or time.

There are two clients of this backing store: machine operators and the RPS software itself. Operators may create, retrieve, update, or delete from the set of Critical Signals for a given set of search keys. The interface for these user-facing operations is a tabular Control System Studio graphical interface. The RPS software client then simply retrieves the Critical Signals names and values for a given machine state, and distributes those values to the corresponding Process Variables.

The CS Studio interface is a two-column table, similar to a spreadsheet. This table is backed by two small Python scripts, one to populate the table following a *Read* command, and another to serialize and transfer the table contents following a *Write* command. This command structure helps enforce transaction atomicity. See Fig. 2:



RPS State	STOPPED
RPS Select	Stop / Enable / Start
Run Permit	●
RPS Heartbeat	366

	High	Low
Beam Power	0.500000 W	0.500000 W
Pulse Freq	1.000000 Hz	1.000000 Hz
Pulse Width	0.100000 s	0.100000 s

Machine Mode	M0
Beam Mode	O0
Ion Species	Ar
Charge State	+3

Machine Mode	M0
Beam Mode	O0
	85 Rb 27+ 37
M/Q	4.500
Mass	84.8970 amu

PV Name	Value
PV1:signal1	1.22
PV2:signal2.DRVH	2.33
PV3.signal3	ENABLED
PV4.signal4	MASKED

Figure 2: RPS user interface panel

The interface between the user-facing table and the MongoDB backend is an EPICS waveform PV, with communication naturally being Channel Access between the user interface and the RPS server hosting the waveform. A critical constraint imposed by this arrangement is that EPICS waveform data-types must be homogeneous. Practically, this is addressed by enforcing PV names and values to be ASCII-encoded strings and configuring the waveform to be of type *CHAR*.

Further formatting is imposed on the PV name/value strings in that they are formatted as a JSON object. This common format was chosen to minimize any conversion that may be required between the RPS and the MongoDB backend. In other words, JSON is a natural format for objects stored in and transferred to/from the backend.

MongoDB naturally stores *Document objects*, and these map well to Python dictionaries (associative arrays). This means that, ideally, one could store PV name/values as simple Python dictionary as well.

However, PV names often contain a '.' character to delimit EPICS Record Fields. MongoDB does not permit '.' characters in Document keys as that is a reserved syntactical element. As a workaround for this constraint, the JSON object representing all persisted Critical Signals for a given state set is simply a string. One drawback of this scheme is that searching by Critical Signals may be somewhat complicated, as the search must be performed not over database keys, but on their values. However, at this time such a search capability is not required.

STATUS AND SUMMARY

Development of the RPS software library is on going, with testing sequestered from the production network and devices for the time being. Full testing is expected to begin near the end of October 2017, with the initial scope limited to the Front End up to and include the Medium Energy Beam Transport section just prior to the superconducting linac sections.

Integration with MongoDB backend continues to be developed to expose more functionality. This includes searching for Critical Signal sets based on a range of dates, observing differences between selected state-sets, and editing values. Additionally, it is also desired to integrate RPS logging with the digital operations logbook. This automation will permit recording of critical RPS event entries in the official facility logs.

Given the potential impact and sensitivity of the Run Permit System, a test suite is being developed to automate the measurement of documented RPS requirements. This will permit rapid turnaround of changes to the code base, while still capturing regressions.

REFERENCES

[1] M. Ikegami *et al*, "Operation Mode and Machine State Control For FRIB Driver Linac Operation", in *Proc. of Linear Accelerator Conf. (LINAC '16)*, East Lansing, USA, Sept. 2016, paper THPLR045, pp. 956-958.

[2] EPICS, `http://www.aps.anl.gov/epics`

[3] CS-Studio, `http://controlsystemstudio.org`

[4] Python, `https://www.python.org`

[5] pyepics, `https://github.com/pyepics/pyepics`

[6] pcaspy, `https://github.com/paulscherrerinstitute/pcaspy`

[7] transitions, `https://github.com/pytransitions/transitions`

[8] MongoDB, `https://www.mongodb.com`