Science and Technology Facilities Council

ISIS Neutron and Muon Source

isis.stfc.ac.uk

PVEcho: Design of a VISTA/EPICS Bridge for the ISIS Control System Transition

MOPV019

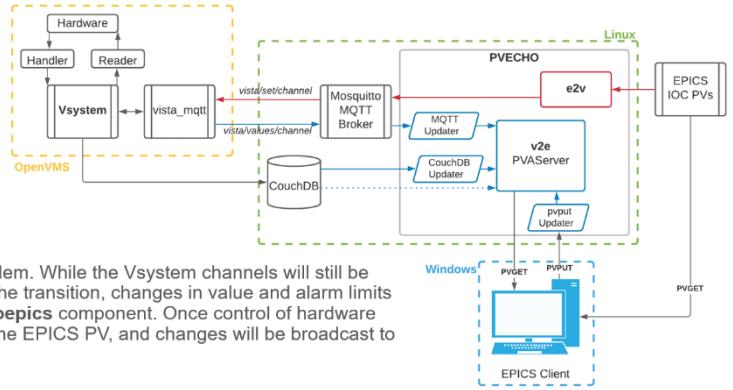
k.baker@stfc.ac.uk

K.R.L. Baker, I.D. Finch, G.D. Howells, A.Saoulis, ISIS, RAL, Didcot, United Kingdom, M.Romanovschi, University of Manchester

ICALEPCS 2021

Control of the ISIS accelerators is being migrated from Vsystem (colloquially referred to as **Vista**) to EPICS using a phased porting of control of hardware, and a hybrid UI during the transition.

This requires bridging software, referred to as **PVEcho** and developed in **Python**, that will map more than 33,000 live channels in Vsystem to an equivalent EPICS Process Variable (PV).



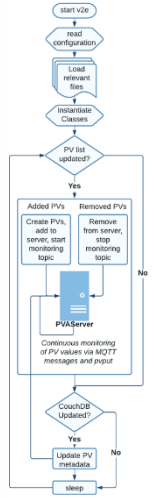
PVEcho consists of two programs operating in tandem. While the Vsystem channels will still be acting as the source of truth in the early stages of the transition, changes in value and alarm limits will be broadcast to EPICS PVs through the **vistatoepics** component. Once control of hardware has been migrated, the source of truth will shift to the EPICS PV, and changes will be broadcast to Vsystem through the **epicstovista** component.

Both applications interface with Vsystem through **MQTT** [1] messages (see WEPV049). Vistatoepics uses the read-only **'values' topic** to track changes to channel values, while epicstovista uses the **'set' topics** for each channel to publish changes in PV values. Metadata associated with each live channel (e.g. channel type, alarm limits and display configuration) is accessed by both programs through a **CouchDB** [2] database. The **pvapy** [3] library is used by both programs to connect to or maintain EPICS PVs.

PVEcho replicates the entire control system, rather than individual devices. We need to be able to dynamically add, remove or edit PVs on the server without having to restart the program (which would interrupt operation of the accelerator). Therefore, a `PVAServer` from the **pvapy** library was chosen to host the PVs rather than a traditional IOC.

In this application, **Vsystem channels are acting as the source of truth** and changes in values are being broadcast to corresponding EPICS PVs.

A snapshot of the existing configuration of each Vsystem channel (e.g. alarm limits and display configuration) is captured using the CouchDB database and used to construct an equivalent EPICS PV for all of the channels listed in the monitor list.



Once the PVs have been initialized, changes in their value or configuration are tracked through:

- **MQTT messages** – subscribe to the channel's MQTT 'values' topic to track live changes to the channel's value
- **CouchDB updates** – track changes to Vsystem database files through alterations to CouchDB using and modify the PV's metadata to match the updated values
- **pvput commands** – PV values and metadata may be changed by operators through `pvput` commands triggered from Phoebus control screens

Each source produces updates in a different json format, so a fleet of `Updater` classes are used to manipulate the data from the raw input into a structure appropriate for use with the `pvapy.PvObject.set()` method.

Exact reproduction of live Vsystem channels also encompasses alarm management. Some alarm types such as **range** alarms and **binary match** alarms are easy to map to the new control system as EPICS defines these alarm types natively. Vsystem also offers **integer match** alarms and **reference** alarms, which do not have direct EPICS equivalent.

The following alarm types are currently implemented and managed by vistatoepics:

- **Range alarms –** With each new update to the value or the meta-data, the current value is compared to the most recent alarm limits to determine its alarm state.
- **Binary match alarms –** Raises an alarm if the current value matches that of the defined alarm match value.
- **Integer match alarms –** An alarm is raised if the current value of the PV matches the alarm match value defined in Vsystem. To differentiate these from range alarms, all alarm limits (high/low, alarm/warning) are set to equal the alarm match value from Vsystem and the incoming value is compared only with the highAlarmLimit.
- **Reference alarms –** The relevant alarm is raised if the value of the PV sits within the high/low alarm/warning ranges, reset every time the setter channel's value changes.

Integer match alarm:

An example of where an integer match alarm might exist is a pump that operates in multiple modes. These could include pump operating states such as:

`0=off, on=1, 2=fault, 3=standby, 4=turbo`

where only state 2 should trigger an alarm.

Reference alarm:

A channel where the alarm limits are determined by the value of a different Vsystem channel according to some calculation. The channel that dictates the reference channel's alarm limits is referred to as its 'setter' channel.

EPICS to Vista

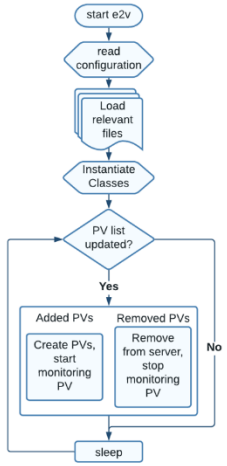
Once control of hardware has been migrated to EPICS, the **PV becomes the source of truth** for the value and alarm limits of a given channel. Updates are published in a JSON format to a "set" topic on the MQTT broker. This topic is subscribed to by code running on OpenVMS that sets the value in Vsystem.

Development Environment

PVEcho has been developed using **Docker** [4] containers with Linux as the base OS. Separate containers will be used to run vistatoepics and epicstovista simultaneously. During development, containers replicating the ISIS MQTT broker and CouchDB database are used to prevent premature interaction with the live control system. Unit and integration testing has been applied throughout using Python's unittest framework, as well as the use of an in-house virtual logging system designed to track CPU and memory performance of the application.

Future Work

- **Channel access** – ISIS will adopt the pvaccess protocol for majority of migrated hardware so focus has been placed on developing pvaccess PVs initially, but the intention is to also broadcast changes to a channel access PV equivalent as well
- **Bulk transition of channels from v2e to e2v** – Currently PVs are added/removed to the monitor list by means of a file which is susceptible to human error. In future we want to be able to allow bulk transfer of channels from v2e to e2v through a flag in a database (e.g. as a device is migrated)
- **Stress testing** – ensure the software can cope with day-to-day frequency of messages



Science and Technology Facilities Council
ISIS Neutron and Muon Source

PVEcho Overview

Control of the ISIS accelerators is being migrated from Vsystem (colloquially referred to as **Vista**) to EPICS using a phased porting of control of hardware, and a hybrid UI during the transition.

This requires bridging software, referred to as **PVEcho** and developed in **Python**, that will map more than 33,000 live channels in Vsystem to an equivalent EPICS Process Variable (PV).

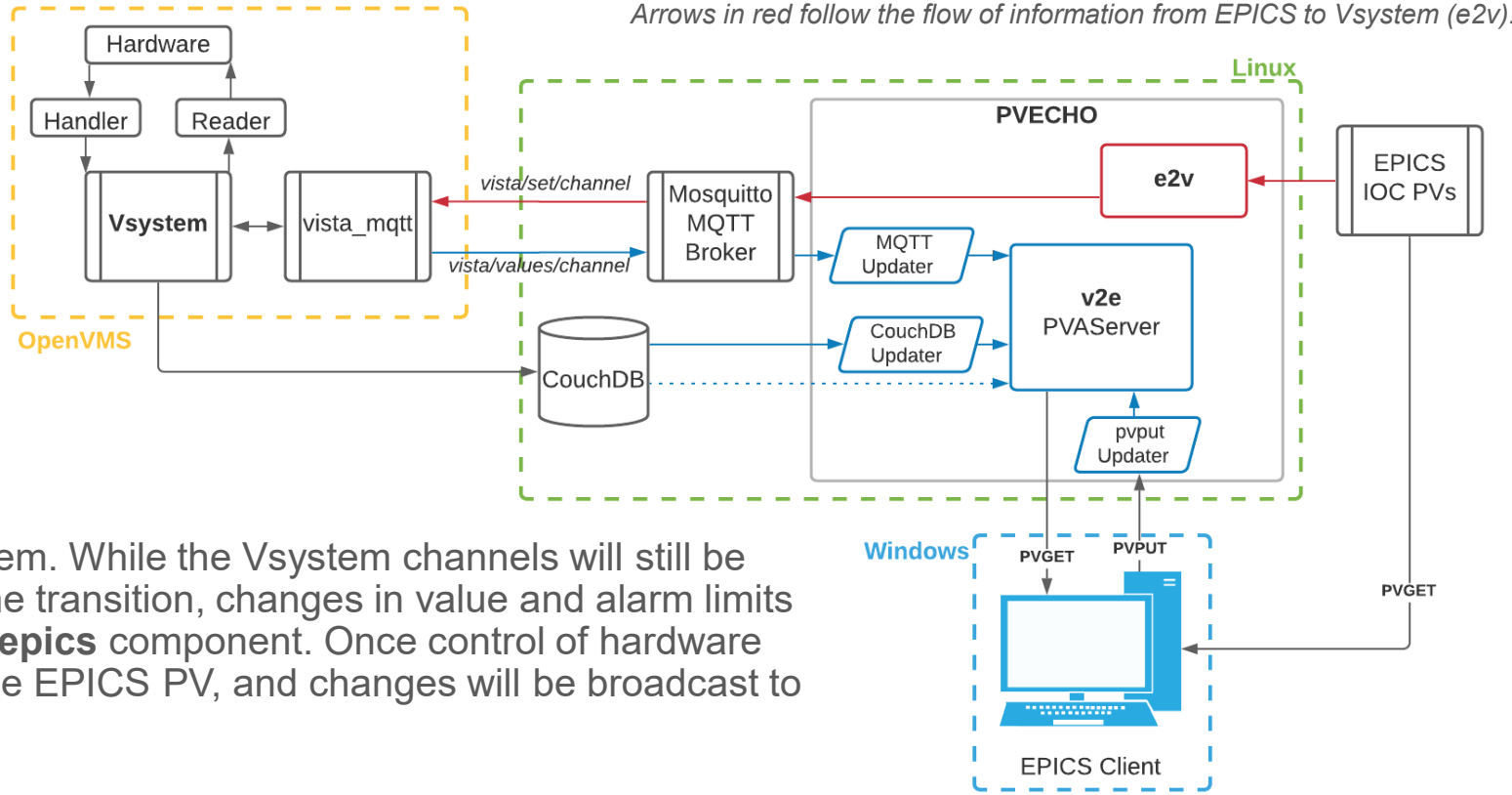


Figure 1: *An overview of how PVEcho will integrate into the control system. Arrows in dark blue indicate the flow of data from a Vsystem channel through to EPICS (v2e). Dotted lines represent an initialization step in the creation of PVs. Arrows in red follow the flow of information from EPICS to Vsystem (e2v).*

PVEcho consists of two programs operating in tandem. While the Vsystem channels will still be acting as the source of truth in the early stages of the transition, changes in value and alarm limits will be broadcast to EPICS PVs through the **vistatoepics** component. Once control of hardware has been migrated, the source of truth will shift to the EPICS PV, and changes will be broadcast to Vsystem through the **epicstovista** component.

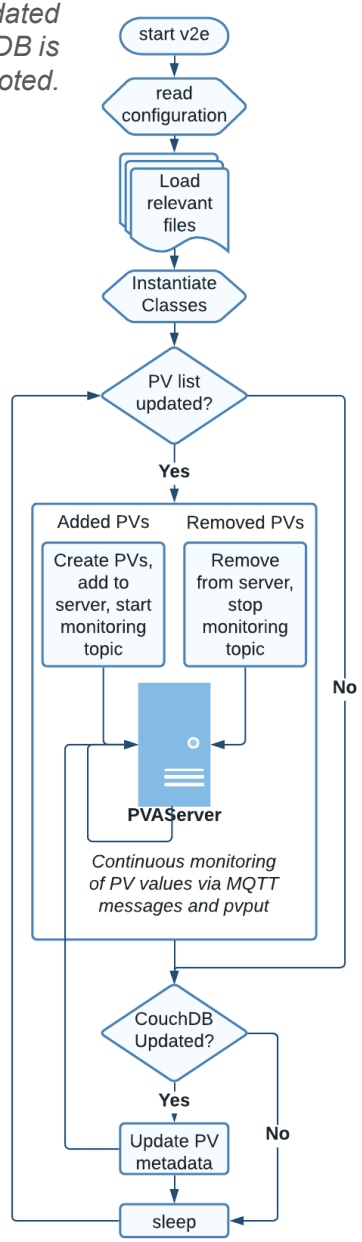
Both applications interface with Vsystem through **MQTT** [1] messages (see WEPV049). Vistatoepics uses the read-only **'values' topic** to track changes to channel values, while epicstovista uses the **'set' topics** for each channel to publish changes in PV values. Metadata associated with each live channel (e.g. channel type, alarm limits and display configuration) is accessed by both programs through a **CouchDB** [2] database. The **pvapy** [3] library is used by both programs to connect to or maintain EPICS PVs.

[1] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html [2] http://couchdb.apache.org/ [3] https://epics.anl.gov/extensions/pvapy/production/index.html

Science and Technology Facilities Council
ISIS Neutron and Muon Source

Vista to EPICS (v2e)

Figure 2: *Flowchart indicating the steps taken during the main loop of the vistatoepics program. The value of the PVs are continuously updated via pvput commands and callbacks within the connection to the MQTT broker. PV metadata is updated whenever a change to CouchDB is noted.*



PVEcho replicates the entire control system, rather than individual devices. We need to be able to dynamically add, remove or edit PVs on the server without having to restart the program (which would interrupt operation of the accelerator). Therefore, a `PVAServer` from the **pvapy** library was chosen to host the PVs rather than a traditional IOC.

In this application, **Vsystem channels are acting as the source of truth** and changes in values are being broadcast to corresponding EPICS PVs.

A snapshot of the existing configuration of each Vsystem channel (e.g. alarm limits and display configuration) is captured using the CouchDB database and used to construct an equivalent EPICS PV for all of the channels listed in the monitor list.

Once the PVs have been initialized, changes in their value or configuration are tracked through:

- **MQTT messages** – subscribe to the channel's MQTT 'values' topic to track live changes to the channel's value
- **CouchDB updates** – track changes to Vsystem database files through alterations to CouchDB using and modify the PV's metadata to match the updated values
- **pvput commands** – PV values and metadata may be changed by operators through `pvput` commands triggered from Phoebus control screens

Each source produces updates in a different json format, so a fleet of `Updater` classes are used to manipulate the data from the raw input into a structure appropriate for use with the `pvapy.PvObject.set()` method.

Science and Technology Facilities Council
ISIS Neutron and Muon Source

v2e Alarm Management & EPICS to Vista (e2v)

Figure 3: Example Vsystem screens in the ISIS main control room where alarm status of channels / PVs will be displayed

Exact reproduction of live Vsystem channels also encompasses alarm management. Some alarm types such as **range** alarms and **binary match** alarms are easy to map to the new control system as EPICS defines these alarm types natively. Vsystem also offers **integer match** alarms and **reference** alarms, which do not have direct EPICS equivalent.

The following alarm types are currently implemented and managed by vistatoepics:

- **Range alarms –** With each new update to the value or the meta-data, the current value is compared to the most recent alarm limits to determine its alarm state.
- **Binary match alarms –** Raises an alarm if the current value matches that of the defined alarm match value.
- **Integer match alarms –** An alarm is raised if the current value of the PV matches the alarm match value defined in Vsystem. To differentiate these from range alarms, all alarm limits (high/low, alarm/warning) are set to equal the alarm match value from Vsystem and the incoming value is compared only with the highAlarmLimit.
- **Reference alarms –** The relevant alarm is raised if the value of the PV sits within the high/low alarm/warning ranges, reset every time the setter channel's value changes.

Integer match alarm:

An example of where an integer match alarm might exist is a pump that operates in multiple modes. These could include pump operating states such as:

`0=off, on=1, 2=fault, 3=standby, 4=turbo`

where only state 2 should trigger an alarm.

Reference alarm:

A channel where the alarm limits are determined by the value of a different Vsystem channel according to some calculation. The channel that dictates the reference channel's alarm limits is referred to as its 'setter' channel.

EPICS to Vista (e2v), Development Environment and Future Work

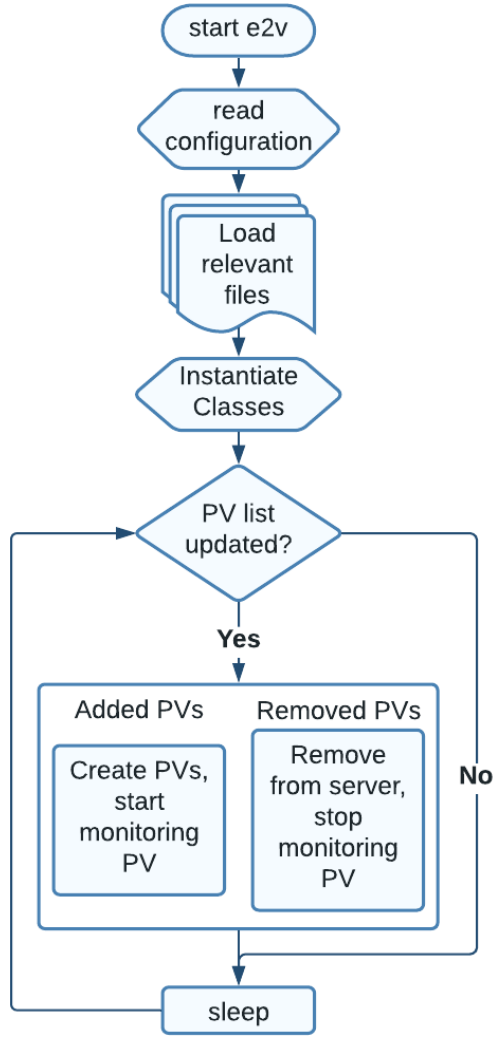
Figure 2: Flowchart indicating the steps taken during the main loop of the epicstovista program. Updates to the values of the PVs are not illustrated as they are updated via callbacks within the pvapy.Channel connection, likewise for updates via pvputs.

EPICS to Vista

Once control of hardware has been migrated to EPICS, the **PV becomes the source of truth** for the value and alarm limits of a given channel. Updates are published in a JSON format to a "set" topic on the MQTT broker. This topic is subscribed to by code running on OpenVMS that sets the value in Vsystem.

Development Environment

PVEcho has been developed using **Docker** [4] containers with Linux as the base OS. Separate containers will be used to run vistatoepics and epicstovista simultaneously. During development, containers replicating the ISIS MQTT broker and CouchDB database are used to prevent premature interaction with the live control system. Unit and integration testing has been applied throughout using Python's unittest framework, as well as the use of an in-house virtual logging system designed to track CPU and memory performance of the application.



Future Work

- **Channel access** – ISIS will adopt the pvaccess protocol for majority of migrated hardware so focus has been placed on developing pvaccess PVs initially, but the intention is to also broadcast changes to a channel access PV equivalent as well
- **Bulk transition of channels from v2e to e2v** – Currently PVs are added/removed to the monitor list by means of a file which is susceptible to human error. In future we want to be able to allow bulk transfer of channels from v2e to e2v through a flag in a database (e.g. as a device is migrated)
- **Stress testing** – ensure the software can cope with day-to-day frequency of messages

[4] https://www.docker.com/