ADDING FLEXIBLE SUBSCRIPTION OPTIONS TO EPICS*

Ralph Lange, Helmholtz-Zentrum Berlin / BESSY II, 12489 Berlin, Germany
Andrew Johnson, Argonne National Laboratory, Argonne, IL 60439, USA
Leo Dalesio, Brookhaven National Laboratory, Upton, NY 11973, USA

Abstract

The need for a mechanism to control and filter subscriptions to control system variables by the client was described in a paper at the ICALEPCS2009 conference [1]. The implementation follows a plug-in design that allows the insertion of plug-in instances into the event stream on the server side. The client can instantiate and configure these plug-ins when opening a subscription, by adding modifiers to the channel name using JSON notation [2]. This paper describes the design and implementation of a modular server-side plug-in framework for Channel Access, and shows examples for plug-ins as well as their use within an EPICS control system.

MOTIVATION

Over the years, more and more powerful front end computers have lead to increasing data processing rates. Event and timing systems nowadays allow to attach accurate synchronous nanosecond resolution time stamps to data.

Many clients will not need every data update available. Those clients will either want to reduce the update rate for their subscription, or have the updates being correlated to events from the event system, so that they are only getting updates during phases they are specifically interested in.

The EPICS (Experimental Physics and Industrial Control System) toolkit [3] and its network protocol, Channel Access [4], only provide a small set of server-configurable update rates, and only very limited correlation between updates and a timing/event system.

SERVER-SIDE PLUG-IN FRAMEWORK

Considerations

When designing an extension to a widely-used, highly scalable system, a number of considerations have to be taken into account:

- Network compatibility: changing the on-the-wire protocol of Channel Access should be avoided.
- Footprint: the extension should be modular and only add minimal resource use on embedded systems.
- API compatibility: internal APIs should be left intact to avoid breaking externally developed code.

Existing Update Mechanism

The existing update mechanism of the EPICS database has been described in detail in [1] and is shown in Fig. 1. At connection time, a set of per-client tasks and event queues is set up. When the database processes an EPICS record, events are generated from the subscription definitions linked to the record, and put into the event queue. The Channel Access event tasks take events off the queue and ship it to the network.

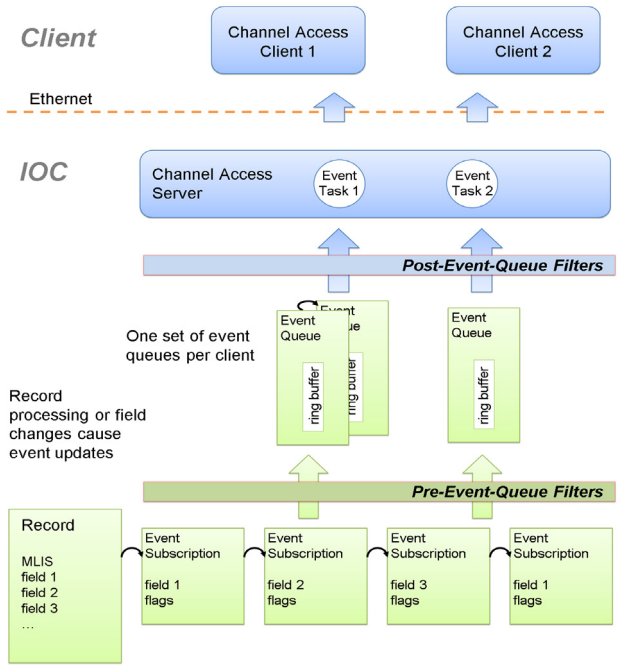

Figure 1: Event Update Mechanism.

Plug-In Framework

The required additional processing of update events is done in plug-ins, that a Channel Access client can request to be inserted into the event stream when the connection is made. These plug-ins have an API that allows for receiving and generating event data, so they can be stacked.

When update events are moving through the plug-in stack, they are handed over to each of the plug-ins. This allows the plug-in to change the event's data as well as the meta data (type, size, time stamp, alarm information etc.) The plug-in may also drop or create additional update events.

Instantiation and configuration of plug-ins is controlled by the client through JSON (JavaScript Object Notation) channel name modifiers, that are transparently forwarded through Channel Access to the IOC [2]. A JSON parser, that has been added to the IOC core software recently, is used for parsing the modifier.

*Work supported by U.S. Department of Energy (under contracts DE-AC02-06CH11357 and DE-AC02-98CH10886), German Bundesministerium für Bildung und Forschung and Land Berlin.

The plug-in framework offers two levels for integration of plug-ins:

A *channel filter* directly implements callbacks for the JSON parser, and thus can use any transmitted JSON structure as configuration.

A *channel filter plug-in* uses a simpler convenience API provided by the framework. Configuration data is restricted to key-value pairs of basic types, but the framework does the parsing for the plug-in, and the simple API allows for easy, efficient, and safe addition of plug-in modules.

The event-related part of both APIs is equivalent.

Adding Plug-Ins to an IOC

All plug-ins have to register themselves with the framework before use, specifying their name (that clients will use for instantiation) and a pointer to their API implementation. The plug-in registration can take place at any time, even after IOC initialization. That would allow plug-in code to be loaded and registered on-demand in a running IOC.

Two Target Layers

The first part of the update mechanism, between record and the event queues (shown *green* in Fig. 1), is done as part of the EPICS database processing, i.e. at high priority. CPU use in this context may keep other records from processing and affect the IOC's real time behavior.

The second part of event generation, between the event queue and the network (shown *blue* in Fig. 1), is done as part of the Channel Access tasks, i.e. at low priority. Database processing will not be affected by CPU usage in this context.

Some plug-in functionality needs to be instantiated on the database side. Plug-ins that reduce the update rate by averaging or limiting should drop surplus events before putting them on the event queue. Plug-ins that correlate updates with an external event system have to do that within the context of record processing.

Other plug-in functionality should be executed in the low priority networking context. Plug-ins that manipulate the data of a single update without relation to other updates, e.g. extracting parts of an array, doing CPU intensive mathematical operations, should avoid affecting the real time database operation, and only do these expensive operations on updates that are actually shipped to the client.

The framework offers instantiation in both layers. Fig. 1 shows the location of the two plug-in layers in the existing update scheme.

PLUG-IN COLLECTION

A number of plug-ins of different complexity have been implemented, to test function and validity of the framework APIs.

ts – Timestamp "Now"

This pre-event-queue plug-in sets the time stamp of the update to "now" (the current time).

Usage: myPV.A{"ts":{}}

When subscribing to a field that does not cause record processing, the data updates are stamped with the time stamp of the record's last processing, which does not show the time the field was updated. This plug-in solves the issue.

dbnd – Deadband Throttling

This pre-event-queue plug-in implements deadband based update throttling, similar to the mechanism in the EPICS records.

Usage: myPV.RVAL{"dbnd":{"m":"rel","d":7.5}}

As a plug-in it is available to any record field, offers per-client configuration, and adds relative deadband specification.

arr – Array Subset

This plug-in inserts itself in the post-event-queue layer. It creates a sub-array with the specified start, increment, and stop index.

Usage: myArray.VAL{"arr"{"s":-5,"i":2}}
myArray.VAL[-5:2:]

The second form is a shorthand notation added for convenience.

sync – Synchronize with Timing System

This pre-event-queue plug-in synchronizes data updates with internal or external timing systems by pushing updates only under certain conditions.

Usage: myPV.VAL{"sync":{"m":"while","s":"red"}}

Synchronization is done with respect to *system state*. This is implemented by means of a small library that offers an API through which named states can be defined and set or reset. Figure 2 shows the effect of the six available synchronization options with respect to a state:

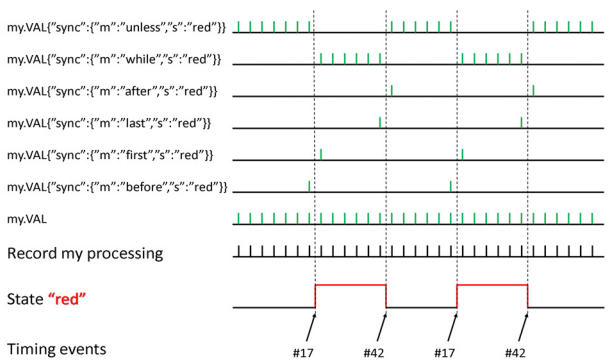

Figure 2: Options of sync Plug-In.

- *unless*: All updates while state is false are being sent.
- *while*: All updates while state is true are being sent.

- *after*: Only the first update after the transition of state from true to false is being sent.
- *last*: Only the last update before the transition of state from true to false is being sent.
- *first*: Only the first update after the transition of state from false to true is being sent.
- *before*: Only the last update before the transition of state from false to true is being sent.

IMPLEMENTATION EXPERIENCES

The major part of the framework implementation was related to reorganization of the subscription definition and data update structures. The subscription structure changes allow per-plug-in and per-plug-in-instance configuration data to be linked to a subscription. The data updates were changed from statical allocation in fixed size lists inside the event queue to heap-allocated structures that use free lists to minimize memory allocation and deallocation at run time.

While implementing the plug-ins described above, the channel filter plug-in API was carefully extended and optimized. In the current state, the individual plug-ins use very little code (40-150 lines of C), and mainly implement the code for their core functionality, i.e. their data update modifications. Thus, implementation of the set of plug-ins described above was fast and uncomplicated.

The two target layer design has proven to be the right approach, as the plug-in designer can freely select the appropriate context for a plug-in.

FOOTPRINT

The framework itself does not add much to the EPICS record structures and code. When no plug-ins are used, the additional memory and computing power consumption is negligible.

Small and embedded EPICS systems will not run into problems because of the plug-in framework.

STATUS

The framework has been developed based on EPICS base version 3.14. The code base is complete and working, unit test code has been written to verify most of the functionality.

Next steps are merging the code into the version 3.15 development trunk, combined with a careful review of the framework's memory allocation techniques and error management – two areas where standards have changed for EPICS base version 3.15.

FURTHER PLANS

Additional plug-ins will allow to throttle data updates by setting the maximum update rate, execute atomic put/get operations, and perform statistical operations on array data.

Records will be able to set default configuration values for certain plug-ins through info tags in the EPICS database.

CONCLUSION

The addition of the server-side plug-in framework to the EPICS toolkit adds valuable functionality to the IOC. It opens yet another way to customize the system, allowing to add exactly the operations needed for a specific installation, and configure them at run time individually for the appropriate connections.

This framework helps to overcome limitations of the existing design, and allows the future addition of new functionality without creating a major impact on performance, memory footprint, or overall complexity.

REFERENCES

[1] R. Lange, A.N. Johnson, “Advanced Monitor/Subscription Mechanisms for EPICS”, THP090, Proceedings of ICALEPCS2009, Kobe, Japan, pp. 847-849.

[2] A.N. Johnson, R. Lange, “Evolutionary Plans for EPICS Version 3”, WEA003, Proceedings of ICALEPCS2009, Kobe, Japan, pp. 364-366.

[3] Experimental Physics and Industrial Control System, http://www.aps.anl.gov/epics.

[4] J. Hill, R. Lange, “EPICS R3.14 Channel Access Reference Manual”, http://www.aps.anl.gov/epics/base/R3-14/12-docs/CAref.html.