ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA058

Lua–LANGUAGE–BASED DATA ACQUISITION PROCESSING EPICS SUBSCRIPTION FILTERS*

J. O. Hill, Los Alamos National Laboratory, Los Alamos, USA

Abstract

A previous paper described an upgrade to EPICS enabling client side tools at LANSCE to receive subscription updates filtered selectively to match a logical configuration of LANSCE beam gates, as specified dynamically by control room application programs. This update paper will examine evolving enhancements enabling Lua–language based data acquisition processing subscription update filters, specified by snippets of Lua-language source-code embedded within the EPICS channel-name's postfix. We will discuss the generalized utility of this approach across a wide range of data acquisition applications, projects, and platforms; the performance and robustness of our production implementation; and our operational experience with the software at LANSCE.

LANSCE

The Los Alamos Neutron Science Centre (LANSCE) was originally designed to be a versatile machine for medium-energy (800 MeV) nuclear physics experiments. It had three injectors and could simultaneously accelerate positive hydrogen ions (H+), negative hydrogen ions (H-) and polarized negative hydrogen ions (P-). These three beams could all have different intensities, duty factors, and even different energies – depending on experimental needs. Today LANSCE can simultaneously generate four H- beam types and two H+ beam types. It services several experimental facilities including a proton storage ring, a low-intensity neutron research facility, proton radiography, ultra-cold neutron source, isotope production, and a proposed materials test-station.

Developed during the infancy of computer control systems, the architecture of the original LANSCE control system (LCS) had elements of data-acquisition along with elements of traditional computer control system architectures. One of the more interesting and useful features of the legacy LCS system was its ability to do "Timed" and "Flavoured" reads. A "Timed Read" sampled typically relative to the leading or trailing edge of a beam gate. A "Flavoured Read" refers to the ability to schedule the read for a particular machine cycle containing a desired configuration of beam-gates. A "Flavour" is configured by specifying for each of 13 timing system beam gates whether it must be present, must be absent, or is not relevant. Therefore, there can be up to 3^{13} possible flavour combinations. In practice, only a few (meaningful) combinations of the six beam destination beam-gates along with a handful of diagnostic-trigger-gates are used, but more esoteric flavours for diagnostic and experimental purposes are considered to be essential.

* Work supported by US Department of Energy under contract DE-AC52-06NA25396.

At LANSCE the 120 slot super-cycle of regularly scheduled beam gates repeats at a 1 Hz rate, but there is also a cycle-stealing scheduling anomaly allowing multiple incompatible beam species to be scheduled in the same cycle, and if more than one of them is currently enabled *only* the gates for the highest-priority species will be emitted during that cycle. This allows, for example, a beam species associated with one-shot enabled proton-radiography, to modally assume a cycle assigned also to the beam species used for high repetition rate production-oriented neutron experiments.

A pivotal requirement, imposing unique constraints on implementation of the lowest levels of the real-time embedded system-software, is that configuration of "Timed" and "Flavoured" data acquisition cycles must be dynamically selected by application programs at the situational compulsion of LANSCE operations and tuning staff.

EPICS CONTROL SYSTEM

An EPICS Input Output Controller (IOC) is configured with Database Records implementing function blocks for various purposes including logical IO, numerical calculation, and ordered sequencing. The EPICS Channel Access (CA) internet communication subsystem is based on a publish-and-subscribe communication model where clients subscribe for updates, servers publish updates to subscribed clients, and records post state change events to servers. A channel is a virtual communication link between a client side application program and a process variable (PV) exported by a service. EPICS clients issue asynchronous read, write, and subscribe requests to the process variable in the service. Clients are notified when the network connectivity of a channel changes.

An essential tenet of the original EPICS design was that regular periodic processing of EPICS Records isn't disturbed by influences outside of an IOC thereby guaranteeing time-periodic algorithms such as PID loops, and time-deterministic response by EPICS Records to state changes detected within sensors, are properly maintained. The load induced by Record Processing is constant and predictable. In contrast, externally induced load from network clients is variable. Therefore record processing executes at higher priorities, and CA network services execute at relatively lower priorities. Multiple event-queues, containing subscription updates, communicate in between record-processing, and the server's per-client dedicated threads.

LUA – A BRIEF INTRODUCTION

"Lua", a language designed specifically to be embeddable within other software, was created in 1993 by members of the Computer Graphics Technology Group (Tecgraf) at the Pontifical Catholic University of Rio de Janeiro, in Bra-

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA058

zil. "Lua" (pronounced LOO-ah) means "Moon" in Portuguese. It is a dynamic typed language, allowing automated conversion between string and numeric types, with a mixture of C-like and Pascal-like syntax. Lua is easily interfaced with C-language software.

LUA – OUR PERSPECTIVE

Lua provides unique features suitable for its embedding within the core of EPICS, and for improving the overall utility of EPICS. Lua provides efficient, compiled to bytecode virtual machine execution, a compact footprint, a portable implementation, and incremental garbage collection. Lua exception handling ensures that the sequence of nested function calls conveying execution to a failure-source-code-line might be reported. Lua has been successfully deployed into many industrial applications, and based on this reputation it is expected to be robust. Lua has a comprehensive set of features, and powerful adjunct-libraries written by an active user community. Lua is well proven for configuration, scripting, and rapid-prototyping, and is a strong return-for-effort candidate functionally upgrading weak areas in the pre-existing implementation of EPICS. Finally, Lua has a liberal MIT license, compatible with EPICS.

There are some negatives. In particular, with Lua the default scope of variables is global, arrays start at one although storing data at index zero isn't prohibited, and there is ambiguity between nil-valued contrasted with non-existent table elements. Lua lacks support for user-defined-type dedicated memory allocators appropriate within memory constrained systems. See below.

MOTIVATION – DATA ACQUISITION PROCESSING SUBSCRIPTION FILTERS

For review, at LANSCE an essential requirement is that the flavour of a particular channel's subscription update stream currently viewed by the operations and tuning staff, must be dynamically selectable. It is not practical for all useful permutations to be a priori instantiated as flavour-dedicated EPICS Records, and instead we must specify flavouring when subscribing. Subscription update rates must be managed due to our 120 Hz cycle rate to reduce loading, while simultaneously facilitating cycle aligning of updates originating from multiple IOCs.

Furthermore, at LANSCE we have additional requirements that starting and ending waveform process-variable element-sequence indexes need to be specified as time offsets from timing system gates, and or waveform edges.

Another important requirement at LANSCE is to convey to specialized application programs the beam-gates actually occurring within a particular data capture's cycle subject to the LANSCE cycle-stealing scheduling anomaly. The data-processing subscription update filter must insert an additional word at the beginning of the waveform element sequence identifying the actually occurring set of beam gates. This type of filter establishes a private protocol between itself and the application program that selected it.

Moreover, it is possible at LANSCE for the super-cycle beam gate scheduling and the relative time offsets of all of the timing gates, to be reconfigured by operators at any time, and therefore we have additional requirement that flavour selection matching and timing gate edge references are computed against the particular configuration of the timing system that generated the data, and not against the current configuration of the timing system which might be very different considering that the filter execution must be postponed until it is running in the server's downstream lower-priority event queue consumer, processing on behalf of a particular client.

Finally, practical considerations dictate that the configuration of filters used by a particular subscription can be specified without revising the source code of the CA Client general purpose application programs, obtained from the wider EPICS user community.

Dynamically configured subscription filtering is a site specific feature required at LANSCE, but our goal was to provide general purpose infrastructure easily customized across a wide range of sites and projects, effectively paradigm shifting EPICS from its general-purpose process-control-system origins into a wider utility also as a flexible data-acquisition-system.

MOTIVATION – ENHANCED EVENT QUEUE FUNCTIONALITY

The original EPICS IOC event queue implementation has some flaws. First, there was a very limited set of data transported on the event-queue; they were the process variable's scalar value, its alarm state, and its time-stamp. Furthermore, it wasn't possible for a single copy of device support allocated constant data associated with a particular update to be shared in the event queues of multiple clients. This type of sharing, and avoidance of memory copying, becomes essential when waveforms, and or other types of complex site specific data, are stored on the event queues of multiple clients. Second, suppression of intermediate updates *is* normal and expected in memory-constrained systems, when event production exceeds consumption rates [1]. However, the original implementation's memory management allowed reordering of events on the queue. This was observed comparing arrivals of subscription updates of one channel versus another, but not within updates for a single channel. Third, real-time atomic bindings of the data with site specific attributes must be preserved as data updates flow through the EPICS database and event queue conduits.

IMPLEMENTATION – FILTER SYNTAX

Our design configures the subscription update filters with a snippet of Lua code specified within a CA channel-name postfix. This approach avoids revising the source code of CA Client general-purpose community obtained application programs. Such postfixes are recognized and removed by the CA Server before passing the channel name to its service layer. Two basic forms of this channel-name postfix both begin with a percent character followed

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA058

by Lua source code enclosed by square or curly brackets specifying respectively a direct-action filter or a factory. Emulating Lua long literal strings, optional matching long brackets also delineate the postfix, for example `[[]]`, `{={}=}`, or `[===[]===]`. Long brackets allow an unlimited character set unambiguously within the postfix. Some examples are provided in Tables 1 and 2 below, with `myPV` serving as the channel's name.

Table 1: Lua Filter, Channel Name Postfix

`myPV%[val >= 3.2 and val <= 3.4]`
`myPV%[val.alarm.condition.severity~=0]`
`myPV%[3.4<val]`
`myPV%[==[val%3.4 [[nested comment]] ]==]`
`myPV %[val==3.2]`

Table 2: Lua Factory, Channel Name Postfix

`myPV%{myFilterFactory ('blue')}`
`myPV%{myChannelFactory()}`
`myPV%{ myApplicationsFactory(10,2)}`
`myPV % {flavour('savoury')}`

The channel name postfixes are themselves implicitly prefixed, just prior to compilation into a callable Lua chunk, as in table 3.

Table 3: Implicit Prefix

filter	`local val=...; return`
factory	`local chanName=...; return`

Filters are called, passing the subscription update payload, in an argument named `val`. Filters return `nil`, `false`, `true`, or a `data-object` for conveying suppress, suppress, send, or send replacing the update payload's value with the returned data object respectively. With Lua, functions are first class values meaning they can be stored in variables, passed as arguments to other functions, and returned as results from functions. A factory may return type Boolean, a direct-acting filter function, or a channel object. A channel object may provide a method named `filterFactory` returning type Boolean or a channel specific direct-acting filter function. See table 4 for a description of that method's interface. Factories return Boolean `false` or `true` for when *all* subscription updates will be permanently disabled or enabled respectively.

Table 4: Channel's Filter Factory Interface

`filterFactory ( channel, lowDelta, highDelta, timeout )`

A subscription update subordinate property optionally supplies a default filter to be used when the channel name postfix is absent.

IMPLEMENTATION – DATA ACQUISITION PROCESSING SUBSCRIPTION FILTERS

The server creates a private Lua context for each of its clients, thereby eliminating client-to-client side effects, and also reducing mutual-exclusion overhead. Our implementation provides a set of Lua classes, as proxies for each of the Lua primitive types. These objects enclose, in addition to one of the Lua primitive types, also a C++ 11 smart pointer to the `Data Access Catalog` container introspection interface, providing efficient access to a hierarchy of the object's subordinate properties. Furthermore, these wrapper objects implement appropriate Lua operators so as to behave transparently as proxies for the enclosed Lua primitive type. Finally, the Lua indexing operator is also implemented, providing access to subordinate properties. For example, `Data Access` interfaced subordinate properties are conveniently accessed simply as ordinary variables within Lua source code as shown in table 5.

Table 5: Lua Property Index Syntax

`val.alarm.condition.severity`

The EPICS CA protocol has been enhanced so that, when asynchronous requests fail within the server, a detailed multi-line diagnostic message is conveyed to the application's response call-back method. A diagnostic message is essential when providing the sequence of nested function calls leading up to the source line of a Lua execution failure. Such diagnostics are operationally essential improving productivity when users misconfigure waveform indexes out of bounds, misconfigure non-existent gate-names, or for any other Lua exceptions occurring unexpectedly during rapid-prototyping. Detailed diagnostics messages are also forwarded to clients when there are Lua compilation errors.

IMPLEMENTATION – WAVEFORMX RECORD

A new `DBF_VARIANT` database field type has been added to the EPICS database. The variant field type is implemented as a class enclosing two C++ 11 smart pointers to the Data Access `Catalog`, introspection, and `Mutator`, modifying, C++ abstract base classes. The `DBF_VARIANT` value field in the new `waveformx` record provides therefore a polymorphic reference to *any* scalar and vector primitive data type, and significantly also a polymorphic reference to any hierarchical set of subordinate properties. Furthermore, when processed, a shared immutable reference to its value field's `Data Access Catalog` container introspection interface is placed on the private EPICS event queue of each subscribed client. The database is also now upgraded to range-check precarious type conversions.

This record is essential at LANSCE, allowing the real-time binding between the data and the LANSCE specific timing and flavouring properties to be transparently preserved, when transporting data through the EPICS database event-queue conduits.

In contrast to the `waveform` record, the `waveformx` record provides also new-functionality fields specifying the default filter, samples-per-second, trigger name, trigger edge, and trigger edge offset correction metadata. The additional fields are used when calculating a waveform update element sequence's starting and ending element, and

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA058

for supplying a default filter when the user does not specify one in the channel name postfix.

IMPLEMENTATION – LANSCE SPECIFIC SUBORDINATE PROPERTIES

At LANSCE, we provide polymorphic `Data Access` container interface adapters for the subordinate properties described in table 6. The `schedule` array contains bit-mask elements encoding the set of gates scheduled in each of the 120 slots of the LANSCE super-cycle. The `cycleIndex` provides the current index in the `schedule` array. The `gateSet` array provides time-offset delay and width information for each of the LANSCE gates. The `updateVersion` is incremented whenever either of the `schedule` array or the `gateSet` array is modified by the timing system.

Table 6: LANSCE Device Specific Properties

`val.device.timing.cycleIndex`
`val.device.timing.updateVersion`
`val.device.timing.schedule[i]`
`val.device.timing.gateSet[i].width`
`val.device.timing.gateSet[i].delay`

We emphasize, that *site-specific polymorphic `Data Access` container interface adapters are fully supported and easily implemented for site specific properties*.

IMPLEMENTATION – LANSCE FILTERS

At LANSCE site-specific flavour filters and time-slice filters have been implemented. Example filter syntax is provided in table 7. Filter one selects cycles with gate `H+IP` and also sans both gates `H-GX` and `MPEG`. Filter two replaces the CA payload with elements 50 through 150 of the waveform data. Filter three selects cycles that have beam gate `H+IP`, replacing the payload with the first 150 µs of the waveform. Filter four replaces the CA payload with -30 through -10 µs of waveform data before the falling edge of gate `MPEG`, selecting only cycles containing `MPEG`. Filter five, replaces the CA payload with 100 µs after waveform rising edge through 150 µs before waveform falling edge selecting only cycles containing `LPEG`. Filter six, selects 100 µs after gate `T0` through 15 µs before waveform end for *any* flavour.

Table 7: LANSCE Filter Examples

1	`XXTDAQ001D01%{flv('H+IP no H-GX MPEG')}`
2	`XXTDAQ001D01%{tim('(50:150)em')}`
3	`XXTDAQ001D01%{flv('H+IP','(0:150)us')}`
4	`XXTDAQ001D01%{tim('~MPEG(-30:-10)us','MPEG')}`
5	`XXTDAQ001D01%{flv('LBEG','(100:~(-150))us')}`
6	`XXTDAQ001D01%{tim('(T0(100):~(-15))us')}`

Figures 1 and 2 show the position of raster patterned beam at the LANSCE Isotope Production Facility selected for two different time-slices. Figure 3 shows a LANSCE linac oscilloscope-style beam-position screen push-button selecting the flavour and the gate-relative time-slice. Our perspective is that data acquisition filters, along with recently improved immediacy of diagnostic screen update response to set-point adjustments, has improved productivity of our operations and tuning staff.

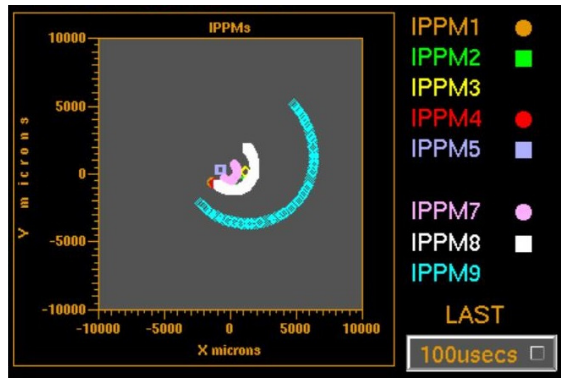

Figure 1: LANSCE isotope production beam position last 100 µs, raster patterned beam.

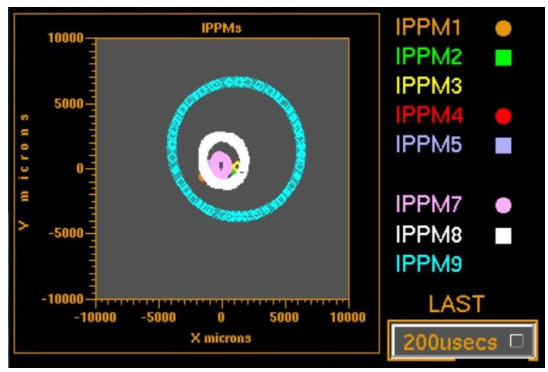

Figure 1: LANSCE isotope production beam position last 200 µs, raster patterned beam.

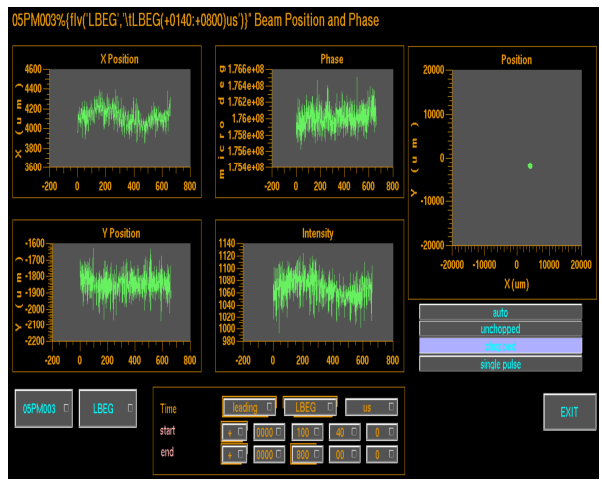

Figure 3: LANSCE linac beam position oscilloscope-style button flavoured and time-sliced.

The LANSCE flavour filters also rate-govern, and synchronize, high-repetition-rate flavours. This is accomplished when, for each flavour, at most four evenly distributed indexes into the `schedule` array are selected for enabling subscription updates. Index selections are identical across all IOCs, as synchronized by the timing system. The same rate-governing mechanisms are utilized when users specify that the flavour isn't relevant. This type of index

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA058

match enabling of subscription updates greatly reduces load on upstream IOC-to-IOC cycle correlation engines. Additionally, our design anticipated, and we now observe in practice, greatly improved probability of operator display updates being perceived stationary and synchronized in between updates, across multiple IOCs.

We emphasize, that *site-specific Lua filters are fully supported and easily implemented for site specific purposes*. The LANSCE specific Lua filters, implemented as C-language source code Lua snap-ins, along with LANSCE specific polymorphic `Data Access` container interface adapters are provided as examples on the world-wide-web [2].

IMPLEMENTATION – EVENT QUEUE MODIFICATIONS

A fully order-correct event queue sharing immutable payloads in between multiple clients, and based on C++ 11 shared pointers to the Data Access polymorphic container introspection interface, has been implemented.

IMPLEMENTATION – MEMORY MANAGEMENT

Lua allocates, and resizes, many small random-sized memory blocks, typically from the C-language runtime's dynamic memory pool. We expect an inevitable design trade-off blending in between increased fragmentation or increased CPU consumption, depending on allocation strategies. Lua lacks support for user-defined-type dedicated memory allocators, facilitating fixed-sized-block based allocators [3], a standard approach in the memory-constrained EPICS IOC. Nevertheless, the Lua allocator *is* replaceable, and our replacement allocates blocks sized less than a threshold value from a free-list. We set this threshold so that all of our Lua user-defined-types allocate from this free-list.

IMPLEMENTATION – INSTALLATION

Site specific Lua filter installation requires only registry function registration as described in the EPICS Application Developer's Guide [4]. A new configuration environment-variable, see Table 8, specifies a white-space separated set of registry function names for Lua state initialization, called once per each new client attaching to the server. The interface to these functions is the `lua_CFunction` in the Lua reference manual, returning `LUA_OK` for success, otherwise failure. The EPICS build system has been modified so Lua source files are first compiled to byte-code, next to a C-source-code with a `lua_CFunction` callable method for loading its embedded Lua byte-code, and finally into object-code. EPICS base R3.15 source-code, upgraded as described herein, and in use operationally at LANSCE implementing FPGA embedded advanced signal processing diagnostic and feedback control systems, can be found on the world-wide-web [5][6].

Table 8: Filter Install Environment Variable

`EPICS_CAS_LUA_STARTUP_FUNCTIONS`

IMPLEMENTATION – FUTURE WORK

We will soon implement specialized filtering based on the actual flavours subject to cycle-stealing. Furthermore, at LANSCE our filters process every update at 120 Hz, for multiple signals, for multiple clients. Therefore the filter execution efficiency is significant, and we have some additional optimization ideas, allowing more clients and subscriptions per installed EPICS IOC. We also anticipate need for additional new record types consistent with the *waveformx* record's paradigm.

OTHER USES OF LUA WITHIN EPICS

We have also, as an original-concept presented at multiple EPICS user-group meetings, leveraged the embedded Lua interpreter supplying an alternative EPICS IOC shell, and a Lua scripting record. Lua offers a significant functionality upgrade compared to the EPICS IOC shell, and has been plumbed capable of calling any of the EPICS IOC shell commands. The Lua scripting record provides an alternative in between the EPICS C-language based `subroutine` record and the EPICS `calc` expression interpreter. It provides rapid prototyping similar to `calc`, Lua's improved full-language-functionality compared to `calc`, but reduced efficiency compared to the `subroutine` function-block. Rapid-prototyping is available via modifications to the Lua record fields specifying Lua source file names.

CONCLUSIONS

EPICS base has been enhanced supporting Lua–language–based data acquisition processing subscription update filters. Such filters determine if a particular subscription update may be forwarded or not, and might also post-process the subscription update's data replacing the subscription update message's data payload. Site-specific hierarchical process-variable subordinate properties and site-specific Lua filters are fully supported and easily implemented. The new features paradigm shift EPICS from its original process-control-system origins into a much wider utility as also a flexible data-acquisition-system. These new features are in operational use, improving productivity, at LANSCE.

REFERENCES

[1] Queueing Theory, `https://en.wikipedia.org/wiki/Queueing_theory`

[2] LANSCE Filters, `https://git.launchpad.net/~johill-lanl/+git/lansce-filters`

[3] Memory Management – Fixed-size blocks allocation, `https://en.wikipedia.org/wiki/Memory_management#Fixed-size_blocks_allocation`

[4] EPICS Application Developer's Guide, `https://epics.anl.gov/base/R3-15/5-docs/AppDevGuide/AppDevGuide.html`

[5] `https://code.launchpad.net/~johill-lanl/epics-base/server1`

[6] J. O. Hill, "The LANSCE FPGA Embedded Signal Processing Framework", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, paper THHA2O02, pp. 1079-1082.