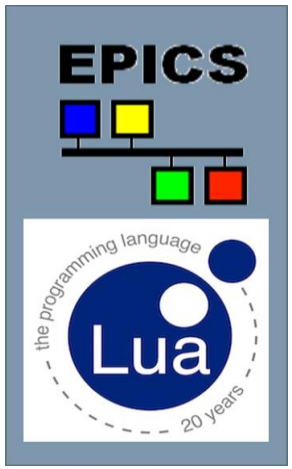
User's Guide for EPICS Lua–Based Data Processing Subscription Update Filters



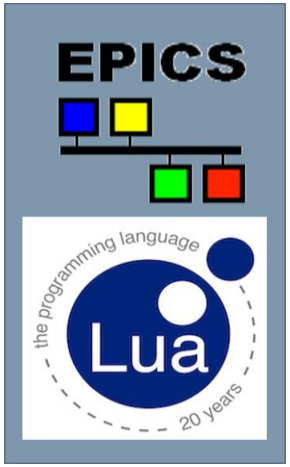
Jeffrey O. Hill

October 2nd 2019

User's Guide for EPICS Lua–Based Data Processing Subscription Update Filters

October 2nd 2019

ICALEPCS 2019
EPICS Workshop



- Lua – A Brief Introduction
- Channel Name Postfix Syntax
- Filter Interface
- Factory Interface
- LANSCE Filter Specifications
- LANSCE Example EDM Screens
- Record Subordinate Properties
- Device Subordinate Properties
- Source Code Examples
- Filter Installation
- Conclusions

Lua – A Brief Introduction

- Lua *embeddable* language was created in 1993
  - By members of the Computer Graphics Technology Group (Tecgraf) at the Pontifical Catholic University of Rio de Janeiro, in Brazil.
- "Lua" (pronounced **LOO-ah**) means "Moon" in Portuguese
- Interpreted, compiled to byte-code, dynamic typed
- A mixture of C-like and Pascal-like syntax
- Efficient virtual machine execution, small footprint, incremental garbage collection, robust error handling, easily interfaced to C code
- Comprehensive feature set, powerful adjunct-libraries written by an active user community
- Well proven for configuration, scripting, and rapid-prototyping
  - A strong return-for-effort candidate for functionally upgrading EPICS
- Liberal MIT license

Lua – A Brief Introduction

- I have also introduced Lua at previous EPICS meetings
  - Alternative EPICS shell, Lua scripting record …
- Also a paper / poster in this conference

- 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

Lua-Based Subscription Update Filters – Channel Name Postfix Syntax

- Filters are configured with a snippet of Lua code
  - Specified within a CA channel-name postfix

- Postfixes are supplied in two basic forms
  - Channel name, followed by percent sign, followed by brackets
    - Direct acting filter
      - `<pv name>%[<lua code>]`
    - Filter or channel object factory
      - `<pv name>%{<lua code>}`

- No need for revising of CA Client general-purpose community obtained application programs

Lua-Based Subscription Update Filters – Channel Name Postfix Syntax

- Direct acting Lua filter examples

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]

- Emulating Lua long literal strings
  - Optional matching long brackets also delineate the postfix
  - For example `%[[]]`, `%{={}=}`, or `%[===[]===]`

Lua-Based Subscription Update Filters – Channel Name Postfix Syntax

- Lua factory examples

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

Lua-Based Subscription Update Filters – Filter Interface

- Filter functions are called
  - Passing the subscription update, in an argument named `val`
  - This update value argument is as an ordinary Lua variable
    - `val >= 3.2 and val <= 3.4`
  - The update value argument *also* indexes subordinate properties
    - `val.alarm.condition.severity`
  - Subordinate properties are themselves also ordinary Lua variables
    - `val.alarm.condition.severity~=0`

Lua-Based Subscription Update Filters – Filter Interface

- Filters return
  - Nil
    - Suppress update
  - False
    - Suppress update
  - True
    - Forward update unmodified
  - Data
    - Forward update replacing it
      - With the returned scalar value or vector element sequence
    - Used at LANSCE
      - To time-slice waveform
    - Used for a private protocol
      - Filter to client-side application selecting it

Lua-Based Subscription Update Filters – Factory Interface

- With Lua, functions are first class values
  - They can be stored in variables
  - They can be passed as arguments to other functions
  - They can be returned as results from functions

Lua-Based Subscription Update Filters – Factory Interface

- Channel name postfix factories are called
  - Passing the channel name, in an argument named `chanName`
- Channel name postfix factories return
  - False
    - Permanently disables *all* updates
  - True
    - Permanently enables *all* updates
  - Function
    - Becomes the direct acting subscription update filter
  - Channel object
    - If this channel object provides a `filterFactory` method
      - It will be called with the arguments below to create a channel context filter
      - `filterFactory ( channel, lowDelta, highDelta, timeout )`
        - » Returning same as channel name postfix factories

Lua-Based Subscription Update Filters – LANSCE Filter Specifications

1. Selects cycles with gate H+IP and also sans both gates H-GX and MPEG
2. Replaces the payload with elements 50 through 150 of the waveform data
3. Selects cycles that have beam gate H+IP, replacing the CA payload with the first 150 µs of the waveform.
4. Replaces the CA payload with -30 through -10 µs of waveform data before the falling edge of gate MPEG, selecting only cycles containing MPEG.
5. Replaces the CA payload with 100 µs after waveform rising edge through 150 µs before waveform falling edge selecting only cycles containing LPEG
6. Selects 100 µs after gate T0 through 15 µs before waveform end for *any* flavour.

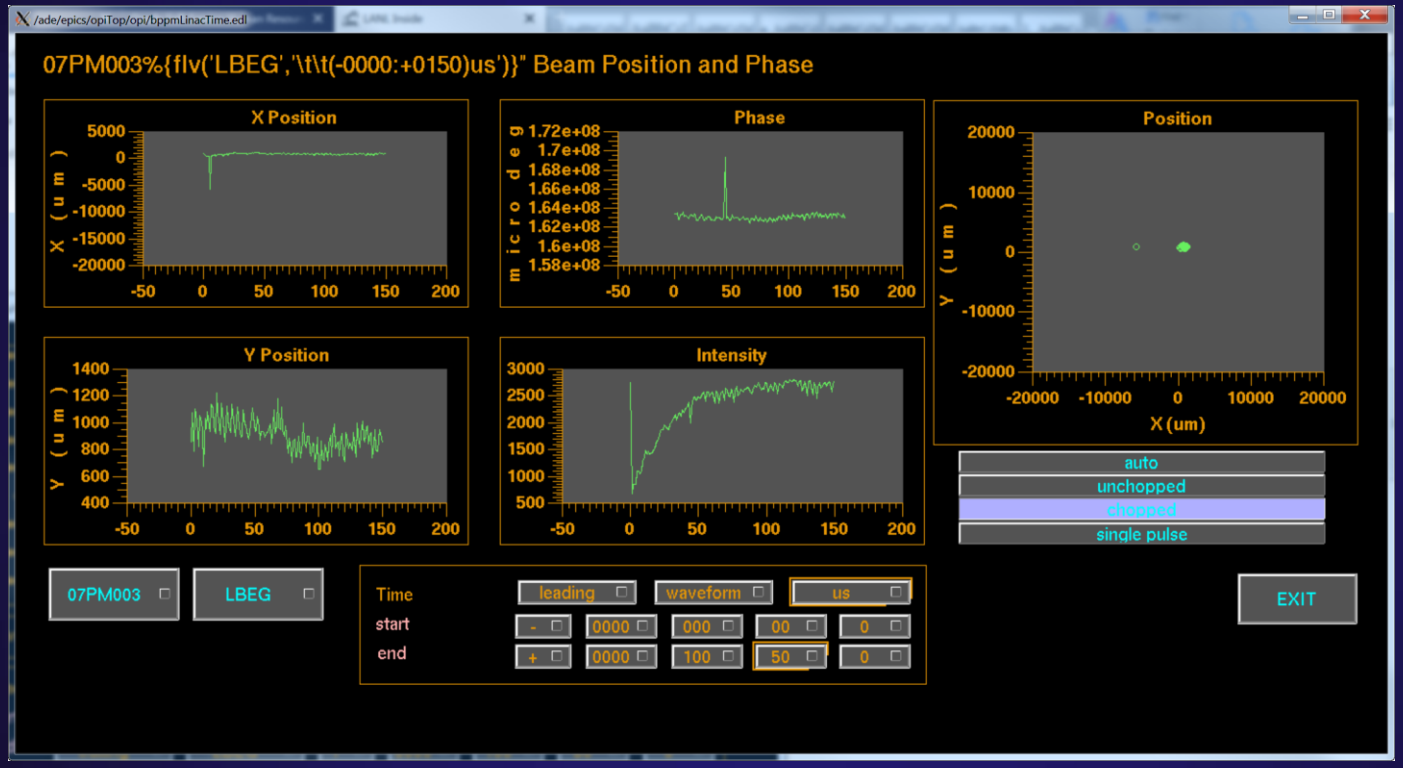
```
1   XXTDAQ001D01%{flv('H+IP no H-GX MPEG')}
2   XXTDAQ001D01%{tim('(50:150)em')}
3   XXTDAQ001D01%{flv('H+IP','(0:150)us')}
4   XXTDAQ001D01%{tim('~MPEG(-30:-10)us','MPEG')}
5   XXTDAQ001D01%{flv('LBEG','(100:~(-150))us')}
6   XXTDAQ001D01%{tim('(T0(100):~(-15))us')}
```

Lua-Based Subscription Update Filters – LANSCE Example EDM Screens

- LANSCE Linac FPGA-Embedded Beam Position and Phase IOC
  - Typical time window

Lua-Based Subscription Update Filters – LANSCE Example EDM Screens

- LANSCE Linac FPGA-Embedded Beam Position and Phase IOC
  - Start of the pulse, before beam has stabilized



Lua-Based Subscription Update Filters – LANSCE Example EDM Screens

- LANSCE Isotope Production Facility raster patterned beam position
  - Last 100 μ sec

Lua-Based Subscription Update Filters – LANSCE Example EDM Screens

- LANSCE Isotope Production Facility raster patterned beam position
  - Last 200 μ sec

Lua-Based Subscription Update Filters – Record Subordinate Properties

- The `waveformx` record provides Data Access Catalog container interface adapters for subordinate properties in the table below
  - Sourcing these properties from the fields of the record

Property	Field	Type
val.staticMetadata.revisionNumber		size_t
val.staticMetadata.filterFactory	FILT	string
val.staticMetadata.capture.samplesPerSec	SPS	double
val.staticMetadata.capture.trig.name	TRIG	string
val.staticMetadata.capture.trig.offset	OFFS	double
val.staticMetadata.capture.trig.edgeRisingNotFalling	EDGE	bool
val.device	PDEM	container interface smart-pointer

Lua-Based Subscription Update Filters – Record Subordinate Properties

- The `waveformx` record provides Data Access Catalog container interface adapters for subordinate properties
  - `val.staticMetadata.revisionNumber`
    - For detecting that static meta-data has changed
  - `val.staticMetadata.filterFactory`
    - Lua source code implementing a default filter factory
      - Used when channel name postfix is absent
  - `val.device`
    - Smart-pointer to the Data Access Catalog container interface
    - Device support supplied properties

Lua-Based Subscription Update Filters – Record Subordinate Properties

- The `waveformx` record provides Data Access Catalog container interface adapters for *capture* subordinate properties
  - `val.staticMetadata.capture.samplesPerSecond`
    - Waveform sample rate
  - `val.staticMetadata.capture.trig.name`
    - Digitizer trigger name
      - Interpreted as the gate name at LANSCE
  - `val.staticMetadata.capture.trig.edgeRisingNotFalling`
    - Digitizer trigger edge
  - `val.staticMetadata.capture.trig.offset`
    - Digitizer trigger time offset correction

Lua-Based Subscription
Update Filters – Device Subordinate Properties

- At LANSCE, we provide Data Access Catalog container interface adapters for the device subordinate properties in the table below
  - The `schedule` array contains bitmask 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

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

Lua-Based Subscription Update Filters –Source Code Examples

- The following source codes are in production use at LANSCE
- The LANSCE specific timing and flavouring Data Access Catalog container interface adapters
  - As C++ Data Access Catalog container interface derived classes
  - See timedDataApp within
    - https://git.launchpad.net/~johill-lanl/+git/lansce-filters
- The LANSCE specific timing and flavouring filter
  - As Lua C++ snap-ins
  - See timedFilterApp within
    - https://git.launchpad.net/~johill-lanl/+git/lansce-filters
- EPICS R3.15 implementing Lua subscription update filtering
  - https://code.launchpad.net/~johill-lanl/epics-base/server1

Lua-Based Subscription Update Filters – Filter Installation

- EPICS build system pre-compilation of Lua source files
  1. Lua source-code is compiled to byte-code
  2. C source-code, with the Lua byte-code embedded, is generated
     - Includes a `lua_CFunction` method for loading the embedded byte-code
  3. C source code is compiled itself into linkable object-code

Lua-Based Subscription Update Filters – Filter Installation

- Complex filters – must be preinstalled and enabled
  - Are pre-compiled, and linked, into the EPICS IOC application
  - Must be EPICS registry registered functions
    - See the application developer's guide
    - Must have `lua_CFunction` interface
      - Return LUA_OK for success, otherwise failure
  - Are specifically enabled/installed at runtime
    - Using new EPICS environment variable
      - `EPICS_CAS_LUA_STARTUP_FUNCTIONS`
        - » A white-space separated list of registry function names
  - Requires the filter factory form in the channel name postfix
    - Constructs and *configures* using appropriate arguments a pre-installed filter

Lua-Based Subscription Update Filters – Conclusions

- Lua is well proven for configuration, scripting, and rapid-prototyping
  - A strong return-for-effort candidate for functionally upgrading EPICS
- Site specific subscription update filter installation is fully supported
  - Some examples on the web
- The new EPICS capabilities are in production use at LANSCE
  - Are operationally essential at LANSCE