WHATRECORD: A PYTHON-BASED EPICS FILE FORMAT TOOL *

Kenneth Lauer†, SLAC National Accelerator Laboratory, Menlo Park, CA

Abstract

whatrecord is a Python-based parsing tool for interacting with a variety of EPICS (Experimental Physics and Industrial Control System) file formats, including V3 and V7 database files. The project aims for compliance with epics-base by using Lark [5] grammars that closely reflect the original Lex/Yacc grammars.

whatrecord offers a suite of tools for working with its supported file formats, with convenient Python-facing dataclass object representations and easy JSON (JavaScript Object Notation) serialization. A prototype backend web server for hosting IOC (Input/Output Controller) and record information is also included as well as a Vue.js-based frontend, an EPICS build system Makefile dependency inspector, a static analyzer-of-sorts for startup scripts, and a host of other things that the author added at whim to this side project.

BACKGROUND

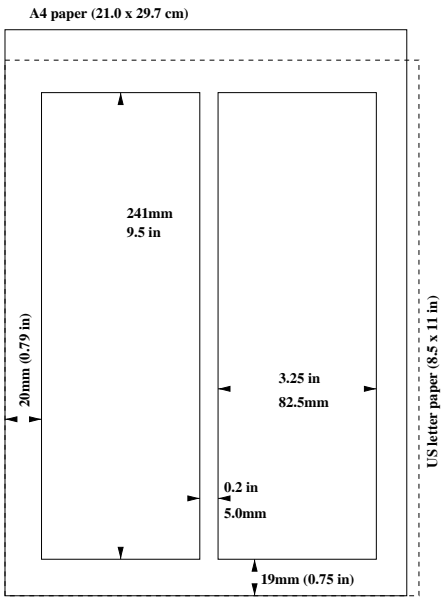

Figure 1: Layout of papers.

The problem - or the inspiration

At the LCLS, the accelerator and photon side control systems include approximately 3000 IOC instances in total, with hundreds of modules and dozens of versions per module.

In general, these EPICS IOCs, modules, and extensions are comprised of a conglomeration of unique file formats. Some examples include:

- Process database files (.db)
- Database definition files (.dbd)
- Template / substitutions files
- IOC shell scripts (st.cmd)
- StreamDevice protocols (.proto)
- State notation language programs (.st)
- Gateway configuration (.pvlist)
- Access security files (.acf)
- Facility-specific things like LCLS's IOC manager configuration
- Build system Makefiles

Combined, this makes for an enormous code base with a mix of these EPICS-specific file formats.

Links between these files are often implicit. Take, for example, that an EPICS IOC record has a specific record type name alongside its name in a database file (.db). An EPICS PV (Process Variable) name, in a traditional IOC, starts with the record name defined in a database file. This PV name acts as a global identifier that allows for clients on the same network subnet to access - and potentially modify - related data.

A record is made up of fields which can contain metadata like engineering units or user-specified descriptions, references to other records, relevant data values, and so on.

An example record instance, defining a single AI (analog input) record named `IOC:RECORD:NAME` is as follows:

```
record(ai, "IOC:RECORD:NAME") {}
```

This file does not define what the fields of the record type; that is the responsibility of the database definition file (.dbd). A simplified excerpt from a database definition file, defining a single field for the "ai" record type is as follows:

```
recordtype(ai) {
    ...
    field(NAME, DBF_STRING) {
        special(SPC_NOMOD)
        size(61)
        prompt("Record Name")
    }
    ...
}
```

Note that there is no explicit link between the database file and the database definition file: neither reference the other by filename. Rather, one can only infer the link by examining a third file, the IOC-specific IOC shell script (.cmd) file, line-by-line.

An excerpt from such a startup script could look like:

```
dbLoadDatabase("path/to/the.dbd",0,0)
IOC_registerRecordDeviceDriver(pdbbase)
dbLoadRecords("records.db")
```

* Work supported by U.S. D.O.E. Contract DE-AC02-76SF00515.
† klauer@slac.stanford.edu

Each line of this script includes up to one command. Each of those commands has been registered by either EPICS itself, the modules included in the IOC, or the IOC source code itself. Typically, the available commands would be found either in documentation or by executing the IOC and invoking the built-in help system. Alternatively, the most reliable fallback ends up being the source code itself.

Other direct or indirect references may be found inside fields. For example, depending on the `DTYP` (device type) field, the `INP` (input specification) field may be a custom string defined at the device support layer. Interpretation of this field requires knowledge of how these are formatted. Take StreamDevice [4], a generic support module for communicating with controllers that use simple byte streams for communication, for example:

```
record(ai, "IOC:RECORD:NAME") {
  field(DTYP, "stream")
  field(INP, "@ProtocolFilename.proto getValue
      PS1")
}
```

The device type here is set to `"stream"`, a custom identifier that StreamDevice has hard-coded. This instructs EPICS to use StreamDevice and interpret the `INP` field with it. It us up to the IOC developer to understand the format of these strings and set them appropriately, in order to reference back to the protocol file that defines the byte string to send and the expected response format. Here, a StreamDevice protocol file for the above record indicates that a simple string `WHAT:IS:THE:VALUE?` is to be sent, and a floating point value (`%f`, as in the C scanf format specifiers) is to be sent from the controller:

```
getValue {
    out "WHAT:IS:THE:VALUE?"; in "%f";
}
```

This section is a small but important part of what makes up an IOC: the build system surrounding all of these files, other modules with their own standards, access security configuration for intra-subnet access, gateway configuration controlling inter-subnet access, facility-specific tools that rely on PV names, and so on further complicate the number of files and references one needs to be aware of.

While those familiar with EPICS IOC development may find that the above is obvious and simple, it can be opaque at best to newer users and those unable to dedicate the time to reading through esoteric manuals (if lucky, source code otherwise).

Goals and the emergence of whatrecord

The previous section's problem led the author over the years to desire a tool that could somehow unify these file formats and provide the ability to inspect the implicit links.

The overall goals of the project could be summarized as:

- Allow for easy parsing of all the special formats outside, and represent them in a widely-used interchange format like JSON.
- Aid the user in the understanding of existing IOCs, whether they are deployed and running or not.
- Provide a method to see how different records, different IOCs, all relate to one another, without requiring the IOC to be running.
- Provide a method for cross-referencing a PV name to its database file, record definition, startup script, and IOC.

These initial goals led to the creation of this new Python package, whatrecord. Taking it a step further and exploring possibilities led to linking records to PLC code, to StreamDevice protocol information, to gateway access rules, and even shell commands to their respective source code.

Features

whatrecord will parse any of the following into intuitive Python dataclasses using lark:

- Database files (V3 or V4/V7), database definitions, template/substitution files
- Access security configuration files
- Autosave .sav files
- Gateway pvlist configuration files
- StreamDevice protocol files
- snlseq/sequencer state machine parsing

whatrecord can also interpret IOC shell scripts (i.e., st.cmd) and track contextual information during the loading process:

- What files were loaded during startup?
- What records are available?
- What errors were found?
- What file and line did record X get loaded?
- Inter- or intra-IOC record relationships

Additionally, whatrecord offers tools for:

- EPICS build system Makefile introspection, a sumo [7]-inspired implementation.
- GDB Python script that inspects binary symbols to find IOC shell commands, variables and source code context

```
dbLoadRecords [str: fname] [str: subs]

.../src/ioc/db/dbIocRegister.c:53
```

- Accurate EPICS macro handling (epics-base macLib, wrapped with Cython in epicsmacrolib [6]).
- Linting startup scripts.
- Plugins for loading happi devices, TwinCAT PLC projects, IOC information from LCLS's IOC manager, ...
- Process database record to Beckhoff TwinCAT PLC source code definition (when used in conjunction with pytmc [**?**])
- Python API, command-line tools for some of the above things
- A web-based API/backend server to monitor IOC scripts and serve IOC/record information.
- Vue.js-based frontend single-page application

Parsing with jq

```
$ whatrecord parse
    whatrecord/tests/iocs/db/pva/iq.db |
   jq '.records[] | [.name, .record_type,
       .fields.OUT.value]'
[
  "$(PREFIX)Rate",
  "ao",
  "$(PREFIX)dly_.ODLY NPP"
]
[
  "$(PREFIX)Delta",
  "ao",
  null
]
...
```

```
$ whatrecord parse
    whatrecord/tests/iocs/db/pva/iq.db |
   jq '.records[] | [ .name, .info["Q:group"]]'
[
  "$(PREFIX)Rate",
  null
]
[
  "$(PREFIX)Phase:I",
  {
    "$(PREFIX)iq": {
      "phas.i": {
        "+type": "plain",
        "+channel": "VAL"
      }
    }
  }
]
...
```

CORE FUNCTIONALITY

Parsing with lark

Backend serverParsing with lark

- Load up all EPICS IOCs (either user-specified or those listed in LCLS's IOC manager tool)
- Load the startup scripts
- Load all the databases and supported files
- Monitor loaded files for changes
- Provide a backend service for querying the information
- Based on the backend server, provide a frontend for easy access to that information

Command-line tools

whatrecord deps Makefile-derived dependency graph tool

whatrecord graph Intra/inter-IOC record graphs State notation language transition diagrams

whatrecord server

VUE.JS WEB FRONTEND

- Search for records/IOCs/etc by name and dig into the details...

Record searching

IOC listing

Inter-IOC PV Map

Gateway

LCLS-Specific Tools

Happi

LDAP / netconfig settings viewer

epicsArch settings

References

The whatrecord source code is available on GitHub [1] and documentation is available on GitHub Pages [2].

REFERENCES

[1] whatrecord source code repository, `http://www.github.com/pcdshub/whatrecord`

[2] whatrecord documentation, `http://pcdshub.github.io/whatrecord`

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

[4] StreamDevice, `https://paulscherrerinstitute.github.io/StreamDevice/index.html`

[5] Lark - a parsing toolkit for Python, https://github.com/lark-parser/lark/

[6] epicsmacrolib, https://github.com/pcdshub/epicsmacrolib

[7] epics-sumo, https://epics-sumo.sourceforge.io/

[8] pytmc, https://github.com/pcdshub/pytmc/