



# Configuration management for *artdaq*

Gennadiy Lukhanin

DUNE-*artdaq*

11 January 2018

# Introduction

This talk focuses on the configuration management (**CM**) subsystem developed for artdaq. It's primary objective is to manage configuration parameters of artdaq processes in the online environment; nonetheless, same approach may be used for other configuration types.

*Fine print: Configuration parameters of artdaq processes include*

- *number, location, and character of DAQ processes;*
- *parameters that are used to define the dataflow;*
- *configuration of the readout electronics.*

# Key features include

- A document-oriented approach for managing artdaq configuration data
- Ability to scale up or down
  - works with a single file, or a directory
  - uses a file system, Mongo, or UconDB backends
- No upfront planing or configuration is necessary for the scale down version
- “0” learning curve
  - run `conftool.py -h` and follow examples
  - [or] use WebConfig editor
- No database schema evolution

# Required UPS products

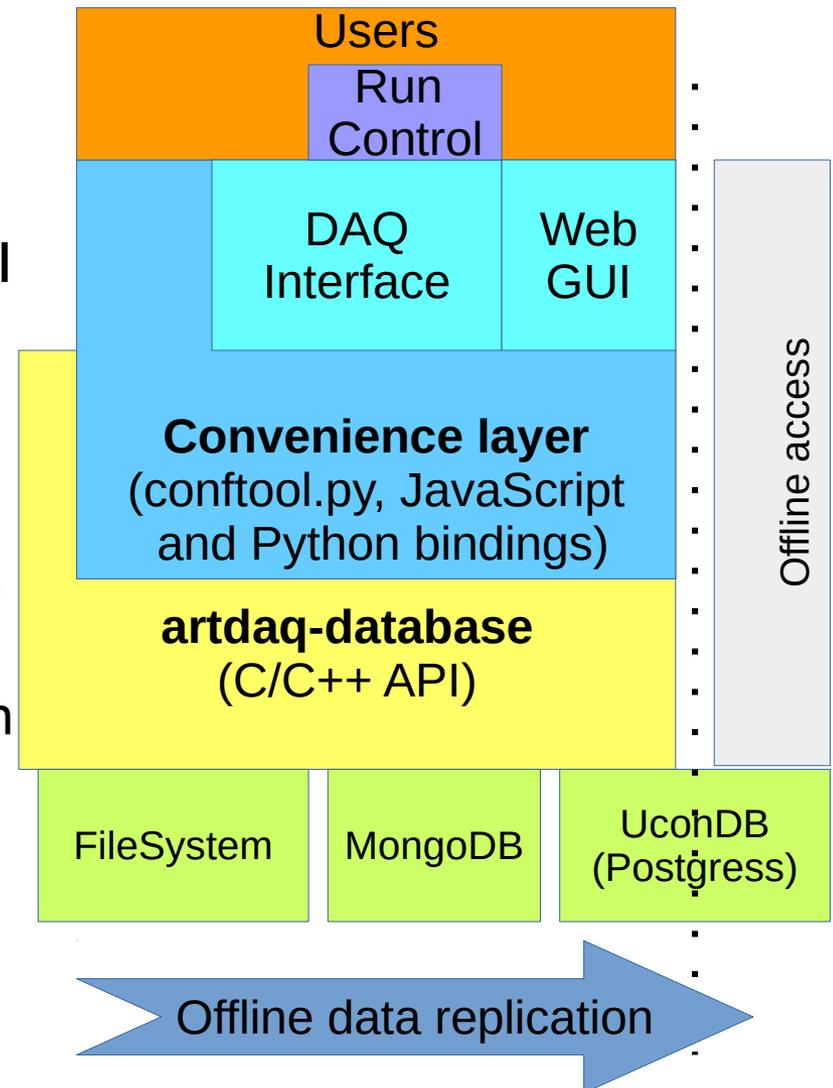
1. `artdaq_database` – CM subsystem tools and C/C++ API
2. `artdaq_node_server` – WebConfig editor
3. `artdaq_daqinterface` – experiment-specific convenience layer

All required products are available via CVMFS, and/or can be installed locally by pulling the latest `artdaq_demo` bundle.

If MongoDB or WebConfig editor is used, your CM liaison needs to run the `install_artdaq_database.sh` script, which configures MongoDB or WebConfig editor with SystemD (requires root).

# Software stack

Artdaq database is implemented as a persistence layer on top of a file system, Mongo and UconDB databases. It maintains all configuration data internally in the JSON format, and the database API performs the conversion from FHiCL and XML to JSON and reverses it back. The choice of the JSON format for internal storage was made due to its similarity to FHiCL and wide support among database vendors, which provide searching inside JSON data. The database API is written in C/C++ and includes JavaScript and Python bindings generated with SWIG. In addition, a convenience layer was added to automate experiment-specific configuration tasks.



# CM operation notes

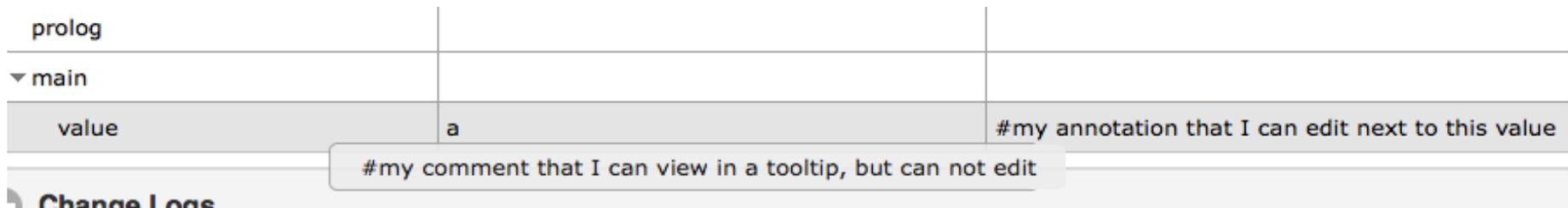
- Every configuration export must be run in an **empty** directory.
- A separate directory must be used for each configuration.
- Delete (recursively) all temporary files (\*~, \*.bak, \*.vim, \*.em, etc) in the directory with a new configuration before importing it.
- Configuration names may have numeric digits only at the end.
  - CM auto-increments them
- All FHiCL includes must precede the prolog section(s).
- Some elements of the FHiCL grammar such as **@table** and **@sequence** are not supported, so don't use them yet.

# Comments in FHiCL files

- FHiCL comments can be introduced in two ways
  - with a single `#` character, or
  - with two forward slashes (`//`).
- Only one-line FHiCL comments placed directly above a value are supported.
- Comments on the same line as a value are called **annotations** and are editable with WebConfig editor.

`#my comment that I can view in a tooltip, but can not edit`

`value: a #my annotation that I can edit next to this value`



Screenshot from WebConfig editor.

# Importance of the FHiCL file naming conventions

The C/C++ API of the CM subsystem is designed to be experiment agnostic, and hence, vastly flexible for being used directly. Experiment specific requirements are implemented in the convenience and daqinterface layers. The convenience layer (conftool.py) enables an important customization point by introducing the **schema.fcl** file, which provides a mapping between the file name and the type of data it has.

```
{collection:Components
  pattern:"(.*)(component)(\d+)(_hw_cfg)(\.fcl$)"},
#files like ./*/componentNNN_hw_cfg.fcl should be
#stored in the collection/table/folder called
#"Components"
#read conftool.py for details
```

Unmatched files will not be picked by conftool.py.

# Convenience layer -- conftool.py

```
$source ./setup_database.sh
```

*Note: setup\_database.sh is located in the /nfs/sw/database/ directory.*

```
$conftool.py -h
```

```
Usage: conftool.py [operation] [config name prefix]
```

```
Example:
```

```
conftool.py exportConfiguration demo_safemode00003
```

```
conftool.py importConfiguration demo_safemode
```

```
conftool.py archiveRunConfiguration demo_safemode 23 #where 23 is the run number
```

```
conftool.py getListOfAvailableRunConfigurationPrefixes
```

```
conftool.py getListOfAvailableRunConfigurations
```

```
conftool.py getListOfAvailableRunConfigurations demo_
```

```
conftool.py listDatabases
```

```
conftool.py listCollections
```

```
conftool.py readDatabaseInfo
```

# How to make configuration changes

- Review instructions  
<https://twiki.cern.ch/twiki/bin/view/CENF/TextConfigEditorP1>
- Quick summary
  - Setup database  

```
source /nfs/sw/database/setup_database.sh
```

```
tmpdir=$PWD/$( uuidgen );mkdir $tmpdir;cd $tmpdir
```
  - Export the existing configuration, say my\_test00077  

```
conftool.py exportConfiguration my_test00077
```
  - Edit FHiCL files
  - Import a new configuration into the database  

```
conftool.py importConfiguration my_test
```

```
rm -rf $tmpdir
```

# Troubleshooting

- Check if the expected version of `artdaq_database` is setup. – “which `conftool.py`”
- Check if the `ARTDAQ_DATABASE_URI` environment variable points to the existing directory or `database+port`.
- Query for the database info – “`conftool.py readDatabaseInfo`”.
- If TRACE-ing is enabled `artdaq_database` writes a verbose log into memory, which can be dumped and checked for errors.
- Contact your CM liaison for additional troubleshooting.

# WebConfig editor

- A nodejs app developed and maintained by Eric Flumerfelt.
- Uses JavaScript bindings to C++API.
- Managed by systemd.
- Runs on np04-srv-010 (localhost and port 8880)
  - use ssh port forwarding to connect
  - or launch firefox on np04-srv-010.
- URL <http://localhost:8880/db/client.html>

## Other notes to mention

- Daily database backups go in `/nfs/sw/database/cern_pddaq_v3x_db/backup`.
- Run configuration history is saved into a separate database; notice the “\_archive” at the end of `ARTDAQ_DATABASE_URI` below.

```
ARTDAQ_DATABASE_URI=mongodb://127.0.0.1:27037/cern_pddaq_db_archive
```

- The CM API supports writing to the offline database; however, the actual data replication procedure is not implemented.
- At the moment, the CM system does not allow choosing which available readout electronics and artdaq processes should be included into the next run. This is currently done by JCOP or by shifters who hand-edit files.