

# DUNEDAQ-V2.2.0

JOHN FREEMAN  
(WITH MANY SLIDES FROM PENGFEI)  
FEB-1-2020

# A BIT OF BACKGROUND

- The dunedaq-v2.2.0 software suite was released Jan. 20
  - I'll cover some of its features
  - Where to find all of this documented
  - Other topics

# OUR DOCUMENTATION

- Important links for users of dunedaq-v2.2.0:
  - *How to set up a work area:* <https://github.com/DUNE-DAQ/appfwk/wiki/Compiling-and-running-under-v2.2.0>
  - *How to create a new DUNE DAQ package:* <https://github.com/DUNE-DAQ/appfwk/wiki/Creating-a-new-package-under-v2.2.0>
  - *C++ coding guidelines:* <https://github.com/DUNE-DAQ/styleguide/blob/develop/dune-daq-cppguide.md>
  - *Our git workflow:* [https://github.com/DUNE-DAQ/daq-release/blob/develop/doc/development\\_workflow\\_gitflow.md](https://github.com/DUNE-DAQ/daq-release/blob/develop/doc/development_workflow_gitflow.md)
- We take our documentation seriously. **The goal is for anyone with a basic knowledge of Linux to be able to create DUNE DAQ packages by following the instructions.** If there's a problem with the documentation please let me know.

# 30,000 FOOT VIEW (~ 10K METERS)

- dunedaq-v2.2.0 is a tag which aliases to tagged versions of all standard DUNE DAQ packages (dfmodules, etc.). Idea of a “release tag” vs. a “version tag”.
- A subset of those packages are dedicated to development, as opposed to DAQ function:
  - daq-buildtools: scripts which let you set up a work area and build your package
  - daq-cmake: CMake functions which simplify and standardize the building of a package’s apps, plugins, etc.
  - daq-release: the package versions to which dunedaq-vX.Y.Z is an alias
- Idea is for developers of “functional” packages to use the most recent release of the development packages, even while working on develop/feature branches of their own packages

# DAQ-BUILDTOOLS RECENT DEVELOPMENTS

- Support for scripts/, python/ and schema/ directories in an installed package (PR #81)
- Allowing unit testing and/or code linting just one of the repos in your build area, rather than all of them (PR #79)
- Factoring out all package version and dependency info (PRs #71 and #77)
- Remove automatic cloning of certain packages (e.g. appfwk) when setting up a work area (PR #64)
- Standardize daq-buildtools script names with `dbt-` suffixes (PR #78)

# DAQ-CMAKE RECENT DEVELOPMENTS

- Good news: function signatures have been stable recently
- Automatic installation of scripts/, python/ and schema/ directories in a package inside daq\_install()
- Improving support for application of moo to schema files (see PRs #7 and #19, and active Issue #27)
- Info about a given build saved in <pkgname>\_build\_info.txt (PR #9)

# DAQ-RELEASE RECENT DEVELOPMENTS

DONEC QUIS NUNC

- Let's look at Pengfei's slides...

# The structure of DAQ release (I)

We have created dedicated directories in cvmfs for each release, and deployed the following pieces in them:

- dbt-build-order.cmake: used by CMake for build orders of DAQ packages;
- dbt-setting.sh: lists of external dependencies and DAQ packages in the release;
- pyvenv\_requirements.txt: list of python packages used when setting up a python virtual environment for DAQ;
- externals: a directory for external dependencies;
- packages: a directory for pre-built DAQ packages.

```
docker-bd dingpf ~ tree -L 1 /cvmfs/dune.opensciencegrid.org/dunedaq/DUNE/releases/dunedaq-v2.2.0
/cvmfs/dune.opensciencegrid.org/dunedaq/DUNE/releases/dunedaq-v2.2.0
├── dbt-build-order.cmake
├── dbt-settings.sh
├── externals
├── packages
└── pyvenv_requirements.txt

2 directories, 3 files
```

# The structure of DAQ release (II)

- The products area in cvmfs is a pool of many versions of packages ever used by any historical release;
- It is hard to navigate through it when digging into details of a release;
- The “externals” and “packages” in each release directory contains only the packages and their corresponding versions used by the given release.

```
docker-bd dingpf ~ tree -L 2 /cvmfs/dune.opensciencegrid.org/dunedaq/D
DUNE/releases/dunedaq-v2.2.0/packages/
/cvmfs/dune.opensciencegrid.org/dunedaq/DUNE/releases/dunedaq-v2.2.0/packages/
├── appfwk
│   ├── v2_1_0 -> ../../../../products/appfwk/v2_1_0
│   └── v2_1_0.version -> ../../../../products/appfwk/v2_1_0.version
├── cmdlib
│   ├── v1_0_2b -> ../../../../products/cmdlib/v1_0_2b
│   └── v1_0_2b.version -> ../../../../products/cmdlib/v1_0_2b.version
├── daq_cmake
│   ├── v1_2_3 -> ../../../../products/daq_cmake/v1_2_3
│   └── v1_2_3.version -> ../../../../products/daq_cmake/v1_2_3.version
├── dataformats
│   ├── v1_0_0 -> ../../../../products/dataformats/v1_0_0
│   └── v1_0_0.version -> ../../../../products/dataformats/v1_0_0.version
├── dfmessages
│   ├── v1_0_0 -> ../../../../products/dfmessages/v1_0_0
│   └── v1_0_0.version -> ../../../../products/dfmessages/v1_0_0.version
├── dfmodules
│   ├── v1_1_1 -> ../../../../products/dfmodules/v1_1_1
│   └── v1_1_1.version -> ../../../../products/dfmodules/v1_1_1.version
├── ipm
│   ├── v1_1_0 -> ../../../../products/ipm/v1_1_0
│   └── v1_1_0.version -> ../../../../products/ipm/v1_1_0.version
├── listrev
│   ├── v2_0_1b -> ../../../../products/listrev/v2_0_1b
│   └── v2_0_1b.version -> ../../../../products/listrev/v2_0_1b.version
├── minidaqapp
│   ├── v1_2_0 -> ../../../../products/minidaqapp/v1_2_0
│   └── v1_2_0.version -> ../../../../products/minidaqapp/v1_2_0.version
├── readout
│   ├── v1_0_1 -> ../../../../products/readout/v1_0_1
│   └── v1_0_1.version -> ../../../../products/readout/v1_0_1.version
├── restcmd
│   ├── v1_0_3 -> ../../../../products/restcmd/v1_0_3
│   └── v1_0_3.version -> ../../../../products/restcmd/v1_0_3.version
├── setup -> ups/v6_0_8/Linux64bit+3.10-2.17/ups/setup
├── setups -> ups/v6_0_8/Linux64bit+3.10-2.17/ups/setups
├── setups_layout
├── trigemu
│   ├── v1_0_0 -> ../../../../products/trigemu/v1_0_0
│   └── v1_0_0.version -> ../../../../products/trigemu/v1_0_0.version
├── ups
│   ├── current.chain -> ../../../../products/ups/current.chain
│   ├── v6_0_8 -> ../../../../products/ups/v6_0_8
│   └── v6_0_8.version -> ../../../../products/ups/v6_0_8.version
```

# The structure of pre-built DAQ package

- Pre-built DAQ packages are in the form of UPS packages:
  - **bin**: executables (binary files or scripts);
  - **include**: header files;
  - **lib64**: shared libraries, python modules and cmake config files;
  - **share**: jsonnet schema files and other configuration files;
  - pkg\_name\_build\_info.txt: information only, contains commit hash, build time etc.

```
docker-bd dingpf ~ tree -L 6 /cvmfs/dune.opensciencegrid.org/dunedaq/DUNE/products/dfmodules/
/cvmfs/dune.opensciencegrid.org/dunedaq/DUNE/products/dfmodules/
├── v1_1_1
│   ├── slf7.x86_64.e19.prof
│   └── dfmodules
│       ├── bin
│       │   ├── hdf5dump
│       │   │   ├── hdf5_dump.py
│       │   │   └── sample.hdf5
│       │   └── dfmodules_build_info.txt
│       ├── include
│       │   └── dfmodules
│       │       ├── DataStore.hpp
│       │       ├── KeyedDataBlock.hpp
│       │       └── StorageKey.hpp
│       ├── lib64
│       │   ├── dfmodules
│       │   │   └── cmake
│       │   ├── libdfmodules_DataGenerator_duneDAQModule.so
│       │   ├── libdfmodules_DataTransferModule_duneDAQModule.so
│       │   ├── libdfmodules_DataWriter_duneDAQModule.so
│       │   ├── libdfmodules_FakeDataProd_duneDAQModule.so
│       │   ├── libdfmodules_FakeReqGen_duneDAQModule.so
│       │   ├── libdfmodules_FakeTrigDecEmu_duneDAQModule.so
│       │   ├── libdfmodules_FragmentReceiver_duneDAQModule.so
│       │   ├── libdfmodules_HDF5DataStore_duneDataStore.so
│       │   ├── libdfmodules_RequestGenerator_duneDAQModule.so
│       │   └── libdfmodules.so
│       ├── share
│       │   └── schema
│       │       ├── data-combiner-demo.jsonnet
│       │       ├── data-generator-demo.jsonnet
│       │       ├── dfmodules-DataCombiner-make.jsonnet
│       │       ├── dfmodules-DataGenerator-make.jsonnet
│       │       ├── dfmodules-DataGenerator-schema.jsonnet
│       │       ├── dfmodules-DataTransferModule-schema.jsonnet
│       │       ├── dfmodules-DataWriter-schema.jsonnet
│       │       ├── dfmodules-FakeDataProd-schema.jsonnet
│       │       ├── dfmodules-FakeTrigDecEmu-schema.jsonnet
│       │       ├── dfmodules-FragmentReceiver-schema.jsonnet
│       │       ├── dfmodules-HDF5DataStore-schema.jsonnet
│       │       ├── dfmodules-RequestGenerator-schema.jsonnet
│       │       └── fake-minidaq-app.jsonnet
│       └── ups
│           └── dfmodules.table
├── v1_1_1.version
└── Linux64bit+3.10-2.17_e19_prof

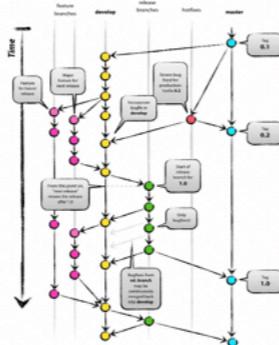
14 directories, 31 files
```

# DAQ development workflow (gitflow)

- We introduced “gitflow” as our development workflow;
- It has been used on many repos in the last three DAQ releases;
- Details can be found at:
  - [https://github.com/DUNE-DAQ/daq-release/blob/develop/doc/development\\_workflow\\_gitflow.md](https://github.com/DUNE-DAQ/daq-release/blob/develop/doc/development_workflow_gitflow.md)
  - Or in this presentation given to the Dataflow working group: [https://indico.fnal.gov/event/47339/contributions/206370/attachments/139027/174355/DUNE\\_DAQ\\_Dataflow\\_Gitflow.pdf](https://indico.fnal.gov/event/47339/contributions/206370/attachments/139027/174355/DUNE_DAQ_Dataflow_Gitflow.pdf)
- We are still evaluating how much of gitflow to formally adopt.

## Branches of DAQ repositories

- Required long-lived branches: develop, master;
- Default branch: develop;
- Short-lived branches:
  - feature branches
    - branch off from develop,
    - merge back to develop;
  - hotfix branches
    - branch off from master,
    - merge back to develop and master
  - release branches
    - branch off from develop,
    - merge back to develop and master



## Development workflow (feature branches)

The following workflow should be followed as much as possible (regardless of the amount of committed code change):

1. Create a GitHub issue in the repo describe the bugfix or proposed feature (optional for non-significant bugfixes);
2. Create a topic branch;

```
git checkout develop #
git checkout -b dingpf/issue_12_feature_dev_demo #
```
3. Make code development, commit, and push the topic branch to GitHub;

```
git push -u origin dingpf/issue_12_feature_dev_demo #
```
4. Create pull request to develop branch when the topic branch is ready to be reviewed and merged, link the issue created in step 1 to the pull request;
5. The pull request gets reviewed by other developers who can:
  - comment on the commits in the PR;
  - request changes;
  - approve pull requests and merge to develop;
  - delete the pull request branch once it's merged (optional), and close the linked issue.

## Tagging and releasing workflow (release branches)

Package maintainers are the primary developers who make version tags of a package. The following workflow should be used when doing so:

1. Check the state of the develop branch: verify all pull requests related to the planned release have been reviewed and merged;
2. Create a release branch;

```
git checkout -b release-v2.2.0 develop #
```
3. Make necessary changes such as bump versions in CMakeLists.txt in the release branch, commit and push;
4. Optional: (especially if protection rules are in place for the master branch) create a pull request of the release branch against both the master branch;
5. If not using step 4, merge the release branch to master, otherwise review & merge the pull requests (preferably done by other developers, protection rules can be set to enforce reviewing rules) git checkout master;

```
git merge --no-ff release-v2.2.0 # always use the --no-ff option) #
```
6. Tag the master branch;

```
git tag -a v2.2.0 # use annotated tag #
```
7. Merge the release branch to develop (if protection rules are in place for develop, one may need to create another pull request in this case);

```
git checkout develop; git merge --no-ff release-v2.2.0 #
```
8. Optional: delete the release branch.

# Prototyping features (I)

- Built and deployed dunedaq-v2.0.0 for CentOS 8:
  - Details can be found at [DUNE-DAQ/daq-release#11](#);
  - Good to demonstrate building the release a newer Linux kernel;
  - No plans on officially supporting CentOS 8 yet.
- Docker images for building/running the DAQ release:
  - Dockerfiles are in [DUNE-DAQ/daq-docker](#)
  - Three images so far:
    - [dunedag/sl7](#) – Scientific Linux 7 based, close to full desktop environment;
    - [dunedag/sl7-minimal](#) – Scientific Linux 7 based, minimalist image capable of building and running the DAQ release;
    - [dunedag/c8](#) – CentOS 8 based, close to full desktop environment.
  - Images are automatically built on [DockerHub](#), triggered by new push to the develop branch of the GitHub repo.

# Prototyping features (II)

- Tarballs of DAQ release:
  - This is helpful for deploying the release outside of cvmfs, e.g. use with dunedag/sl7-minimal docker image by GitHub CI jobs, or deploy on NP04 DAQ nodes;
  - Instructions on how to use it;
  - Note: these tarballs eliminate the need of cvmfs on client nodes, but not the public internet access, as client nodes still need to access pypi.org when creating the python virtual environment. We plan to make a tarball of local pypi mirror or a configured venv soon.
- GitHub CI:
  - With the release tarballs and dunedag/sl7-minimal docker image, we plan to provide a template CI configuration to be used with each repo.

# A FEW WORDS ABOUT OUR STYLE GUIDE

## DONEC QUIS NUNC

- Forked from the Google C++ Style Guide, but greatly simplified and with some rules changed
- “Style” is a bit of a misnomer - it doesn’t refer (only) to whitespace and brace placement, but rules for making code logically organized and less bug-prone
- If “--lint” is passed to dbt-build.sh, tools will be run which catch *some but not all* guideline violations
- Not ironclad - occasionally a rule needs to be broken
- Thanks for all the effort people have put in to get the code compliant with the guide, particularly in light of the new naming conventions

# THE BOTTOM DRAWER

- Configuration of DAQ modules using Python now available
- The documentation linked to on the first slide is the Software Coordination Group's responsibility. Individual package README.md files aren't, and can sometimes get out-of-date.
- Code for testing is an essential part of the development process. It's a good idea to write unit tests early on, and to occasionally pass the "--unittest" argument to dbt-build.sh to regression test your code during development [https://github.com/DUNE-DAQ/daq-release/blob/develop/doc/development\\_workflow\\_gitflow.md](https://github.com/DUNE-DAQ/daq-release/blob/develop/doc/development_workflow_gitflow.md). Integration testing also a good idea (see, e.g., the "Developer Testing" section of <https://github.com/DUNE-DAQ/ipm> as an example of this)
- A develop branch's code should have been reviewed, and should compile

# CONCLUSIONS

- The DUNE DAQ software development ecosystem is stabilizing
- We want to the DUNE DAQ developer experience to be as frictionless as possible
  - Easy to set up a new development area
  - Easy to begin writing a package
  - Easy to build that package
  - Easy to collaborate
  - Etc.
- Still work to do, however

# BACKUP SLIDES

# EXAMPLE OF A PACKAGE'S BUILD INFO

```
user for build:      jcfree
hostname for build:  mu2edaq13
build time:         Sun Jan 31 15:21:22 CST 2021
local repo dir:     /home/jcfree/daqbuild_pywork/sourcecode/appfwk
git branch:         eflumerf/Styleguide_Updates
git commit hash:    c28d78884d87ddd4d465e12189c55b9906839145
git commit time:    Thu Jan 28 17:10:21 2021 -0600
git commit description: JCF: change function call so appfwk builds against current head of cmdlib's johnfreema
n/Styleguide_Conformance branch (ef608349fa2547)
git commit author:  John Freeman
uncommitted changes:
```