Web Monitor Documentation

Release 3.2.0

SCSE@ORNL

Feb 28, 2024

CONTENTS

CHAPTER

ONE

DATA WORKFLOW MANAGER

SNS data workflow manager and reporting app

Dependencies:

- stomp
- django
- MySQLdb if using MySQL
- psycopg2 if using PostgreSQL

It consists of 3 applications (Workflow Manager, Web Monitor, and DASMON Listener) which are deployed via docker-compose.

CHAPTER

TWO

WORKFLOW MANAGER

Data workflow manager based on ActiveMQ

The database abstraction is done through django.db and the communication layer is done using stomp.

The `icat_activemq.xml` file is the ActiveMQ configuration used to set up the cluster of brokers.

Use `sns_post_processing.py` to start the workflow manager.

Use `test/test_consumer.py` to simulate the worker nodes.

CHAPTER

THREE

WEB MONITOR

Reporting application for the workflow manager. The reporting app is built using django. It connects to the reporting database used by the workflow manager. Look at the `docker-compose.yml` to see how to configure and run it.

CHAPTER

FOUR

DASMON LISTENER

DB logging daemon.

The DASMON listener watches for DASMON message sent to ActiveMQ and logs them. See the `src/dasmon_app/dasmon_listener/README.md` file for more details.

PostgreSQL optimization: If using PostgreSQL, you can install the stored procedure in `reporting/report/sql/stored_procs.sql`. Those will speed up the run rate and error rate queries. No change needs to be made to the settings when installing those stored procedures and the application will use them automatically.

CHAPTER

FIVE

DEVELOPER'S ENTRY POINT

Start at the Developer's documentation. This requires building the docs:

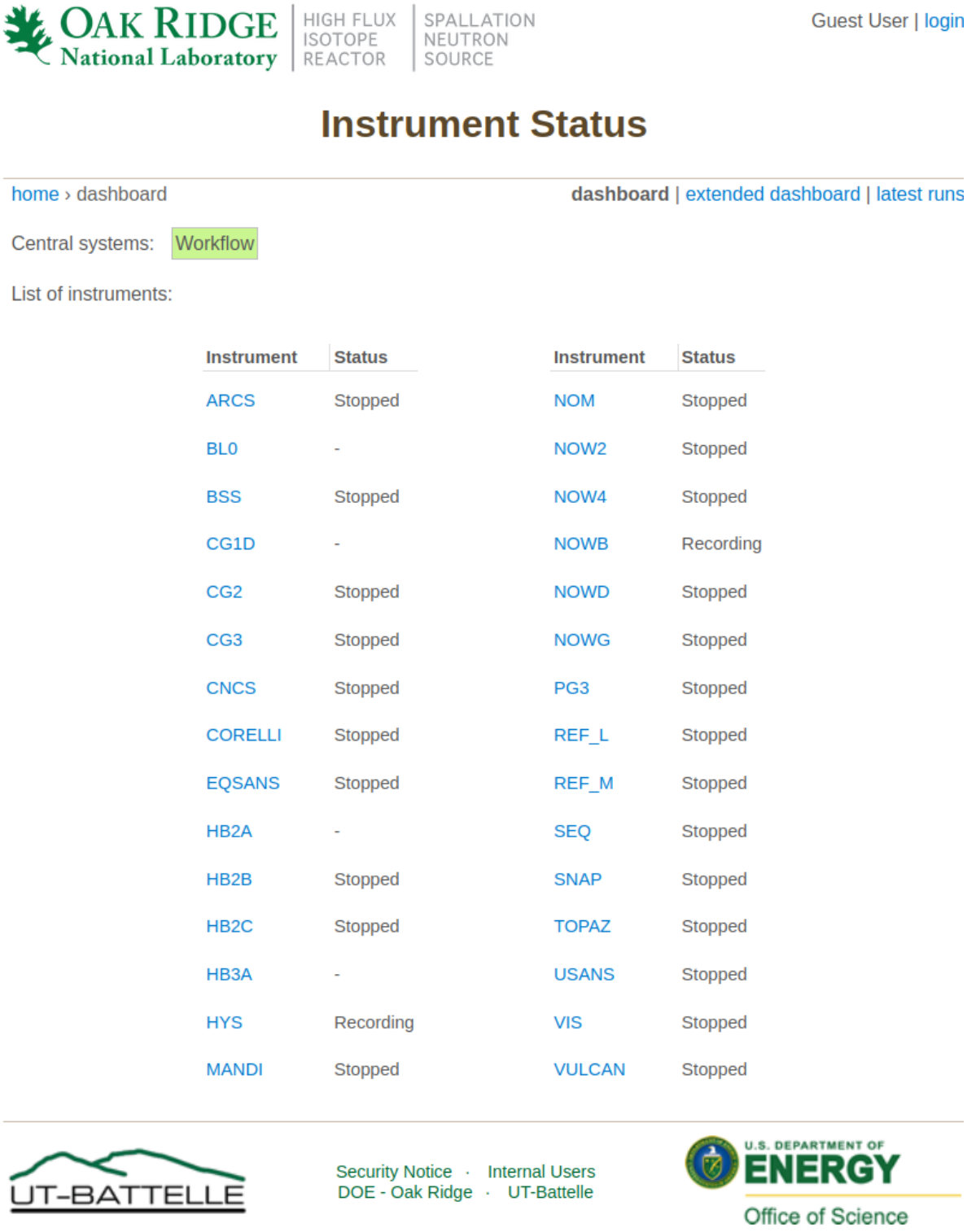
```
$ make create/conda
$ make docs
```

5.1 User Documentation

The https://monitor.sns.gov is a portal for the users to monitor the status of data aquisition and reduction during experiments. Different views are described below, sorted by various access levels.

5.1.1 Guest User View

Anyone can see a list of instruments



When clicking on any instrument one can see the status of the instrument

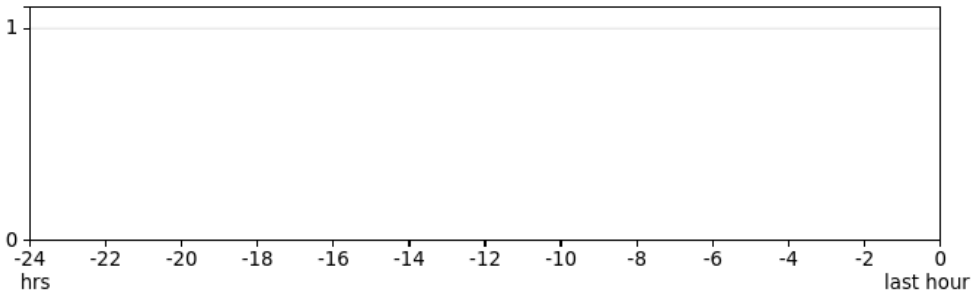
OAK RIDGE National Laboratory | HIGH FLUX ISOTOPE REACTOR | SPALLATION NEUTRON SOURCE

Guest User | login

ARCS Monitor

home › arcs › monitor

live monitoring: **status** | runs | PVs



test

Proposal: IPTS-27800 Run: 0

Status: Stopped Count rate: 0

Systems: Workflow

Last run: 214583 from IPTS-27800 created on Nov. 8, 2021, 11:16 a.m.

Key	Value	Last Updated
count_rate	0	Dec. 6, 2021, 4:57 p.m.
has_states_count	0	Nov. 8, 2021, 11:16 a.m.
monitor_count_1	0	Dec. 6, 2021, 4:57 p.m.
monitor_count_2	0	Nov. 28, 2021, 4:24 p.m.
paused	false	Nov. 19, 2021, 3:35 p.m.
recording	false	Nov. 19, 2021, 3:35 p.m.
scan_index	0	Nov. 19, 2021, 3:35 p.m.
scanning	false	Nov. 19, 2021, 3:35 p.m.
system_dasmon	0	Dec. 6, 2021, 4:57 p.m.
system_postprocessing	Created ARCS reduction script	Oct. 4, 2021, 4:13 p.m.
system_pvsd	0	Dec. 6, 2021, 4:57 p.m.
total_charge	0	Nov. 8, 2021, 11:16 a.m.
total_counts	0	Nov. 8, 2021, 11:16 a.m.
total_time	27.1008	Nov. 8, 2021, 11:16 a.m.

UT-BATTELLE

Security Notice · Internal Users
DOE - Oak Ridge · UT-Battelle

U.S. DEPARTMENT OF ENERGY
Office of Science

In the top right hand side, there are links to see the runs in the current experiment, together with the processing status.

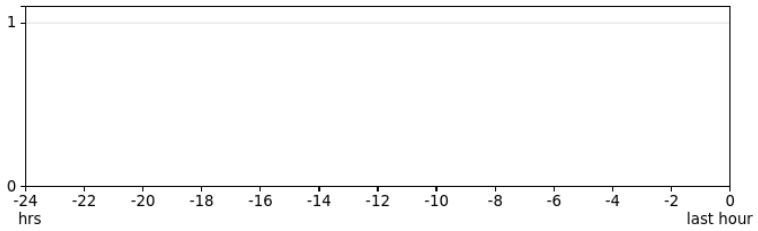
OAK RIDGE National Laboratory | HIGH FLUX ISOTOPE REACTOR | SPALLATION NEUTRON SOURCE

Guest User | login

ARCS Monitor

home › arcs › monitor

live monitoring: status | **runs** | PVs



test

Proposal: IPTS-27800 Run: 0

Status: Stopped Count rate: 0

Systems: Workflow

Last run: 214583 from IPTS-27800 created on Nov. 8, 2021, 11:16 a.m.

List of latest runs:

Show: 25

Search:

Run	Created on ▼	Status
214583	Nov. 8, 2021, 11:16 a.m.	error
214582	Nov. 8, 2021, 11:01 a.m.	error
214581	Oct. 20, 2021, 6:50 a.m.	complete
214580	Oct. 20, 2021, 6:47 a.m.	complete
214579	Oct. 20, 2021, 6:44 a.m.	complete
214578	Oct. 20, 2021, 6:41 a.m.	complete
214577	Oct. 20, 2021, 6:37 a.m.	complete
214576	Oct. 20, 2021, 6:34 a.m.	complete
214575	Oct. 20, 2021, 6:31 a.m.	complete
214574	Oct. 20, 2021, 6:28 a.m.	complete
214573	Oct. 20, 2021, 6:25 a.m.	complete
214572	Oct. 20, 2021, 6:22 a.m.	complete
214571	Oct. 20, 2021, 6:19 a.m.	complete
214570	Oct. 20, 2021, 6:16 a.m.	complete
214569	Oct. 20, 2021, 6:13 a.m.	complete
214568	Oct. 20, 2021, 6:10 a.m.	complete
214567	Oct. 20, 2021, 6:07 a.m.	complete
214566	Oct. 20, 2021, 6:03 a.m.	complete
214565	Oct. 20, 2021, 6 a.m.	complete
214564	Oct. 20, 2021, 5:57 a.m.	complete
214563	Oct. 20, 2021, 5:54 a.m.	complete
214562	Oct. 20, 2021, 5:51 a.m.	complete
214561	Oct. 20, 2021, 5:48 a.m.	complete
214560	Oct. 20, 2021, 5:45 a.m.	complete
214559	Oct. 20, 2021, 5:42 a.m.	complete

Showing 25 of 25 records

Previous 1 Next

UT-BATTELLE

Security Notice · Internal Users
DOE - Oak Ridge · UT-Battelle

U.S. DEPARTMENT OF ENERGY
Office of Science

In addition, there is a list of process variables (PVs). These are a list of sample environment and instrument parameter logs. If one clicks on a PV link, it will show the history of that PV in the last 15 minutes or 2 hours. The y scale can be switched between linear and logarithmic.

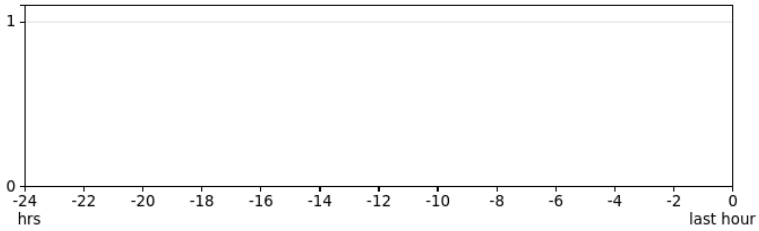
OAK RIDGE National Laboratory | HIGH FLUX ISOTOPE REACTOR | SPALLATION NEUTRON SOURCE

Guest User | logi

ARCS Process Variables

home › arcs › monitor

live monitoring: status | runs | **PV**



test

Proposal: IPTS-27800 Run: 0

Status: Stopped Count rate: 0

Systems: Workflow

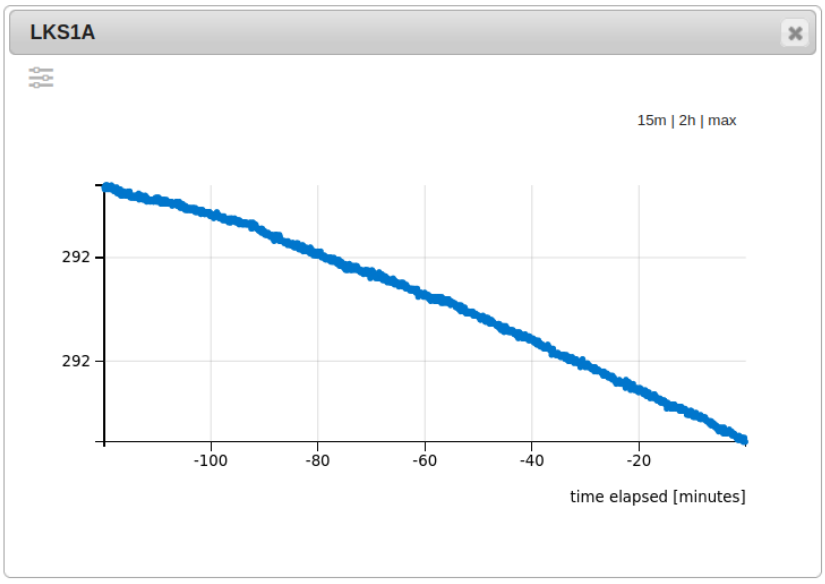
Last run: 214583 from IPTS-27800 created on Nov. 8, 2021, 11:16 a.m.

Key	Value	Last Updated
BL18:SE:SampleTemp	291.511	Dec. 6, 2021, 4:59 p.m.
chopper3_TDC	8.2445e+06	Dec. 6, 2021, 4:59 p.m.
HeartbeatPVSD	18820	Dec. 6, 2021, 4:54 p.m.
LKS1A	291.712	Dec. 6, 2021, 4:59 p.m.
LKS1B	291.502	Dec. 6, 2021, 4:59 p.m.
LKS1C	293.165	Dec. 6, 2021, 4:59 p.m.

UT-BATTELLE

Security Notice · Internal Users
DOE - Oak Ridge · UT-Battelle

U.S. DEPARTMENT OF ENERGY
Office of Science



Clicking on the instrument name in the breadcrumbs will display a list of IPTS experiments

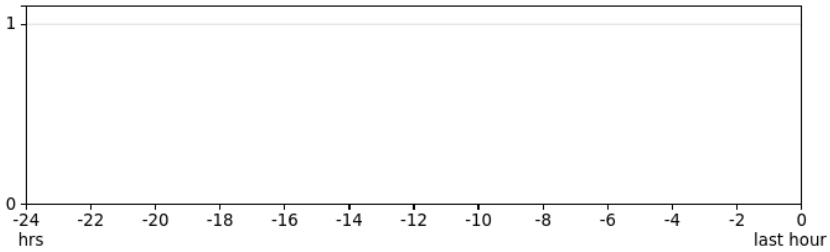
OAK RIDGE National Laboratory | HIGH FLUX ISOTOPE REACTOR | SPALLATION NEUTRON SOURCE

Guest User | login

ARCS Experiments

home › arcs

live monitoring: status | runs | PVs



test

Proposal: IPTS-27800 Run: 0

Status: Stopped Count rate: 0

Systems: Workflow

Last run: 214583 from IPTS-27800 created on Nov. 8, 2021, 11:16 a.m.

List of ARCS experiments (view errors)

Show: 25 Search:

Experiment	No. of runs	Created on ▼
IPTS-27278	872	Oct. 16, 2021, 2:15 p.m.
IPTS-28826	370	Oct. 13, 2021, 5:38 p.m.
IPTS-27442	142	Oct. 5, 2021, 9:24 p.m.
IPTS-27235	1519	Sept. 21, 2021, 9:49 p.m.
IPTS-27456	1359	Sept. 14, 2021, 4:41 p.m.
IPTS-25602	164	Sept. 7, 2021, 5:55 p.m.
IPTS-27032	2726	Sept. 7, 2021, 3:57 p.m.
IPTS-26972	1751	Sept. 1, 2021, 5:53 p.m.
IPTS-27906	13	Aug. 25, 2021, 2:09 p.m.
IPTS-27298	94	Aug. 17, 2021, 4:51 p.m.
IPTS-27071	210	Aug. 16, 2021, 9:42 a.m.
IPTS-27035	304	Aug. 10, 2021, 5:17 p.m.
IPTS-26919	2167	Aug. 4, 2021, 1:36 a.m.
IPTS-27473	173	July 29, 2021, 7:04 p.m.
IPTS-27751	10	July 27, 2021, 3:05 p.m.
IPTS-27502	120	July 26, 2021, 4:26 p.m.
IPTS-27339	121	July 24, 2021, 8:35 p.m.
IPTS-26063	102	July 20, 2021, 7:35 p.m.
IPTS-27800	398	July 12, 2021, 1:43 p.m.
IPTS-24271	116	May 27, 2021, 10:51 a.m.
IPTS-26418	180	May 26, 2021, 1:44 p.m.
IPTS-24207	3409	May 18, 2021, 5:05 p.m.
IPTS-26201	240	May 11, 2021, 3:06 p.m.
IPTS-26616	1397	April 23, 2021, 2:28 p.m.
IPTS-26513	1803	April 16, 2021, 11:10 a.m.

Showing 1 to 25 of 342 records

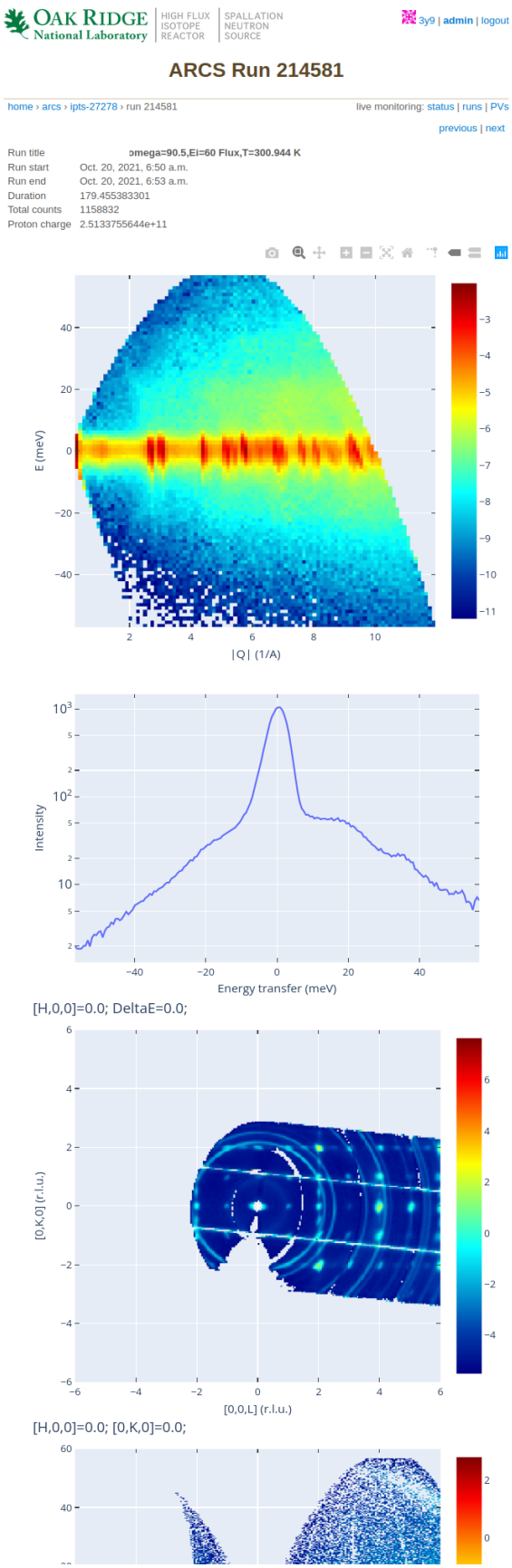
Previous 1 2 3 … 14 Next

Note that guest users don't have access to the data, so clicking on any run will prompt the user to log in with their UCAMS/XCAMS credentials

5.1.2 General User View

In addition to the *Guest User View*, there are additional privileges for which the user must be logged in. To to that, click on the *login* link at the top right of the page.

As opposed to a guest user, when clicking on a specific run, a logged in user will get more information, and plots (if set up that way by the instrument scientist/CIS).


OAK RIDGE National Laboratory
HIGH FLUX ISOTOPE REACTOR
SPALLATION NEUTRON SOURCE
3y9 | admin | logout
ARCS Run 214581
home › arcs › ipts-27278 › run 214581
live monitoring: status | runs | PVs
previous | next
Run title ɔmega=90.5,Ei=60 Flux,T=300.944 K
Run start Oct. 20, 2021, 6:50 a.m.
Run end Oct. 20, 2021, 6:53 a.m.
Duration 179.455383301
Total counts 1158832
Proton charge 2.5133755644e+11
E (meV)
|Q| (1/A)
Intensity
Energy transfer (meV)
[H,0,0]=0.0; DeltaE=0.0;
[0,K,0] (r.l.u.)
[0,0,L] (r.l.u.)
[H,0,0]=0.0; [0,K,0]=0.0;

At the bottom of the page there is a table describing the processing status of the particular data file. Any error messages will show up in red. Please communicate with your local contact about any such occurrences.

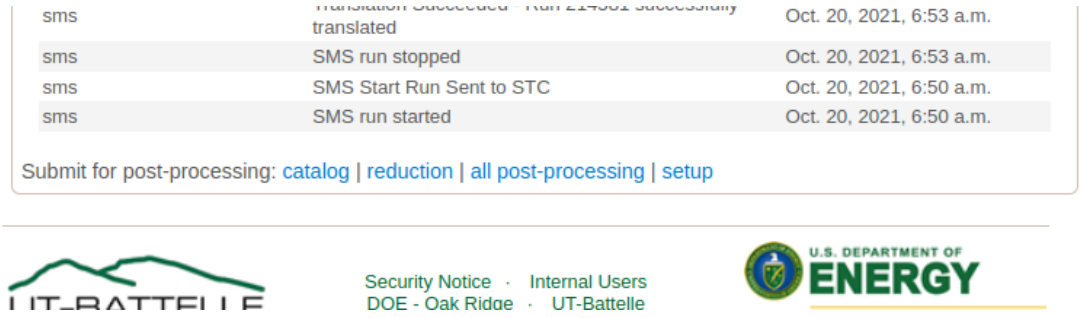
Data files:

/SNS/ARCS/IPTS-27278/nexus/ARCS_214581.nxs.h5

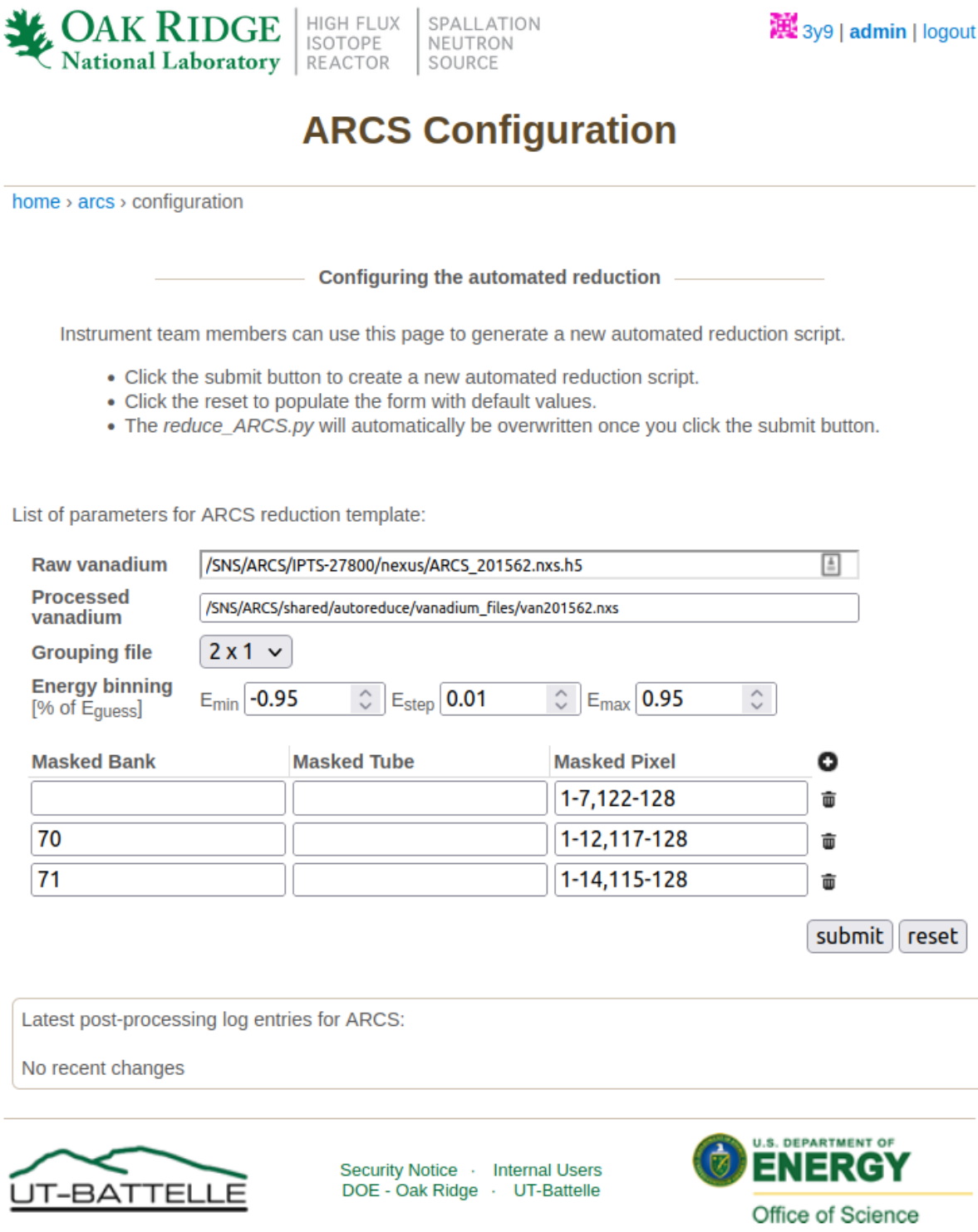
Message	Information	Time
reduction_catalog.complete	autoreducer1.sns.gov	Oct. 20, 2021, 6:54 a.m.
reduction_catalog.started	autoreducer1.sns.gov	Oct. 20, 2021, 6:54 a.m.
reduction_catalog.data_ready		Oct. 20, 2021, 6:54 a.m.
reduction.complete	Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings	Oct. 20, 2021, 6:54 a.m.
catalog.oncat.complete	autoreducer4.sns.gov	Oct. 20, 2021, 6:53 a.m.
catalog.oncat.started	autoreducer4.sns.gov	Oct. 20, 2021, 6:53 a.m.
reduction.started	autoreducer2.sns.gov	Oct. 20, 2021, 6:53 a.m.
catalog.complete		Oct. 20, 2021, 6:53 a.m.
catalog.oncat.data_ready		Oct. 20, 2021, 6:53 a.m.
catalog.complete		Oct. 20, 2021, 6:53 a.m.
reduction.data_ready		Oct. 20, 2021, 6:53 a.m.
postprocess.data_ready		Oct. 20, 2021, 6:53 a.m.
sms	Translation Succeeded - Run 214581 successfully translated	Oct. 20, 2021, 6:53 a.m.
sms	SMS run stopped	Oct. 20, 2021, 6:53 a.m.
sms	SMS Start Run Sent to STC	Oct. 20, 2021, 6:50 a.m.
sms	SMS run started	Oct. 20, 2021, 6:50 a.m.

5.1.3 Instrument Scientist View

The instrument scientist (local contact) can send individual runs back for cataloging, or reduction, by clicking on the links at the bottom of the page.



In addition, several beamlines can open a setup page for auto-reduction (click on `setup`).



The exact view is instrument specific. Modifying the configuration page will generate a new auto-reduction script. There is a `reduction_INSTRUMENT.py` template file. In several places in the file the variables on this page are inserted, and the file is saved as `/SNS/INSTRUMENT/shared/autoreduce/reduce_INSTRUMENT.py`.

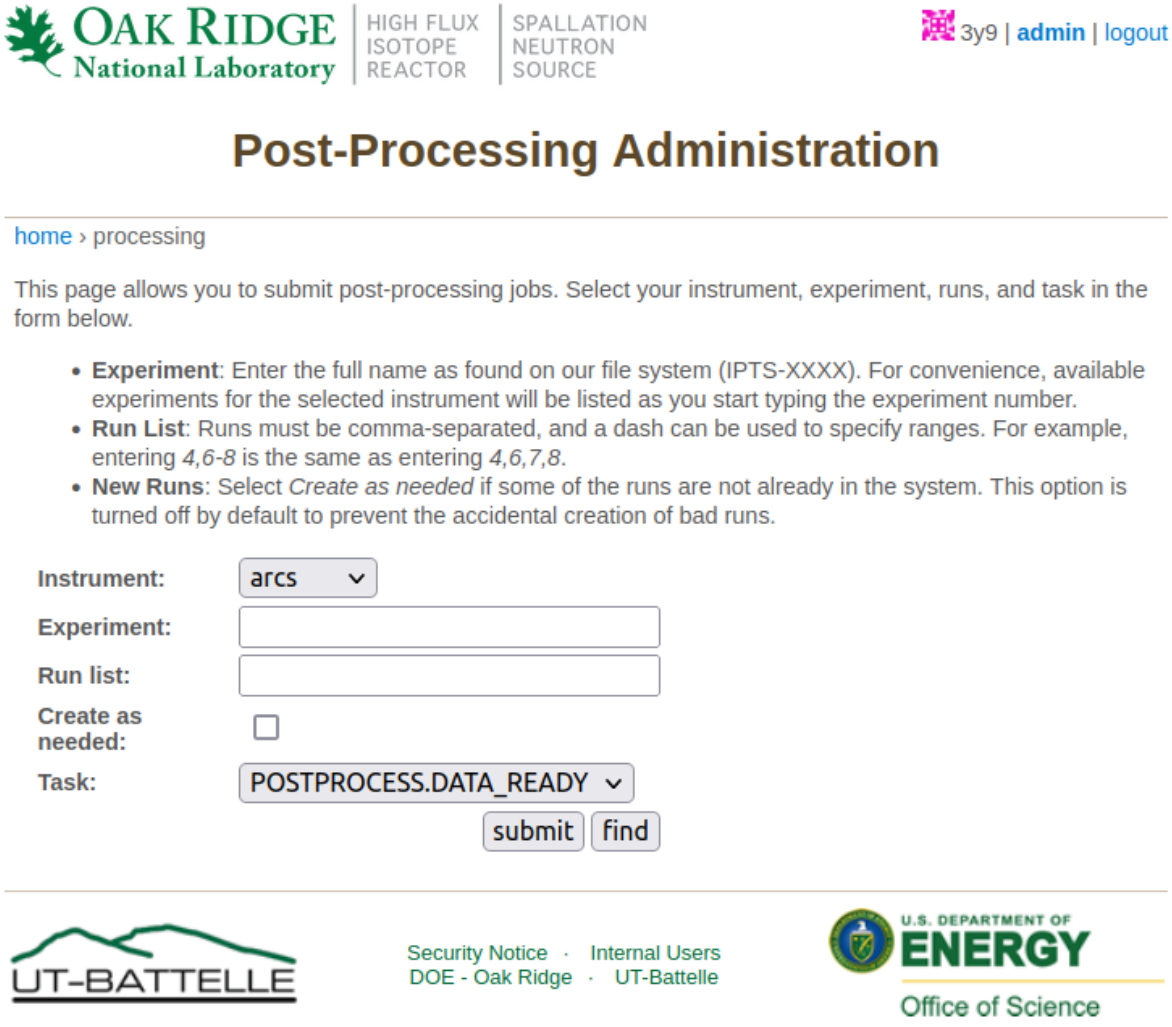
For example, in the case of ARCS, `RawVanadium="${raw_vanadium}"` is changed to `RawVanadium="/SNS/ARCS/IPTS-27800/nexus/ARCS_201562.nxs.h5"`. A special case is for mask, where `${mask}` is changed to

- `MaskBTPParameters.append({'Pixel': '1-7,122-128'})`
- `MaskBTPParameters.append({'Pixel': '1-12,117-128', 'Bank': '70'})`
- `MaskBTPParameters.append({'Pixel': '1-14,115-128', 'Bank': '71'})`

The current settings on the configuration page are stored in a database. Changing them and clicking submit should yield a message at the bottom of the page that contains the change time.

Admin View

Certain people can run a batch postprocessing. On top, next to the username, click on `admin`. It will open a page like below



One can then submit several runs for re-reduction. This is useful if there is a mistake in the reduction script. The messages for postprocessing are not necessarily handled in the order of run numbers.

Note: Most instrument scientists do not have this option.

5.1.4 Release Notes

Full release notes are on github

5.2 Release notes

5.2.1 v3.0.0

The development team is very happy to announce this release which focused on modernization, dockerization, environment setup, and documentation.

Dependency changes

- python 2.7 to 3.10
- apache to nginx
- django 1.11 to 3.2
- jquery 1.72 to 3.6
- postgess 9.6.23 to 14

Other modernization changes include a heavily increased test coverage (currently 77%), building python wheels, and a sphinx site .

In addition to the various other changes since the last release of 2.8.0 in 2015.

5.3 Developer documentation

5.3.1 Developer Guide

How to Build a Local Instance

Note: This document is updated, however, it may be good to read the continuous integration scripts as well.

Running static analysis

This repository uses pre-commit framework to run the static analysis checks. After installing pre-commit the checks can be run using

```
pre-commit run --all-files
```

Running unit tests

The unit tests exist next to the code it is testing. They are run inside a conda environment and pointing at the correct directory with the configuration inside the root-level `setup.cfg`. Replace `conda` with `mamba` for the faster dependency resolver. This is based on what is run in .github/workflow/ci.yml

```
make create/conda  # substitute with "create/mamba" when using mamba
conda activate webmon
DJANGO_SETTINGS_MODULE=reporting.reporting_app.settings.unittest \
   python -m pytest src
```

If the environment already exists, `conda_environment.yml` can be used to update it as well.

```
conda activate webmon
conda env update --file conda_development.yml
```

Running system test

The system test are run via .github/workflow/system.yml .

```
make all # wheels and test data
LDAP_SERVER_URI=. LDAP_DOMAIN_COMPONENT=. docker-compose up --build
```

Wait for a time for everything to get up and running. This is normally noted by seeing a collection of worker threads starting. Once started tests can be run via

```
DJANGO_SETTINGS_MODULE=reporting.reporting_app.settings.envtest \
   python -m pytest tests
```

Setup and Deployment for Development

Most of the shell commands used when working in the developer setup (a.k.a "localdev") are encapsulated in `make` targets. Type `make help` for a list of `make` targets and a brief description.

When starting for scratch, open a shell where the following secret environment variables have been initialized:

```
DJANGO_SETTINGS_MODULE=reporting.reporting_app.settings.develop
GENERAL_USER_USERNAME=GeneralUser
GENERAL_USER_PASSWORD=GeneralUser
LDAP_SERVER_URI=*****
LDAP_USER_DN_TEMPLATE=*****
LDAP_DOMAIN_COMPONENT=*****
CATALOG_URL=*****
CATALOG_ID=*****
CATALOG_SECRET=*****
```

It is recommended to store these variables in an `.envrc` file and manage their loading/unloading into the shell with the direnv command-line utility.

Description of settings

The settings are split into a couple of bundled options that can be selected by specifying `DJANGO_SETTINGS_MODULE`

- `reporting.reporting_app.settings.unittest` for running outside of docker in the conda environment
- `reporting.reporting_app.settings.develop` for local docker containers. This can be connected to production ldap server in a read only mode and will ignore TLS errors
- `reporting.reporting_app.settings.envtest` for the remote testing environment
- `reporting.reporting_app.settings.prod` for production

The environment variables `LDAP_SERVER_URI` and `LDAP_DOMAIN_COMPONENT` are shown above with no-op values. Senior developers can provide the values to use, then the developer setup can work with Neutron Scattering Division's (NSD) LDAP instance.

The environment variables `CATALOG_URL`, `CATALOG_ID` and `CATALOG_SECRET` can be set to allow run metadata to be retrieved from ONCat.

Special users

While one can connect to the production LDAP, in a developer environment there are listed below as username:password

- `GeneralUser` : `GeneralUser` has permissions to pages similar to a general beamline users. The username and password can be set using the `GENERAL_USER_USERNAME` and `GENERAL_USER_PASSWORD` environment variables. The credentials are stored in `unittest.py` settings file
- `InstrumentScientist` : `InstrumentScientist` has permissions similar to an instrument scientist

After setting the environment variables, run the following `make` targets in the shell:

```
make conda/create
make wheel/all  # create python packages for dasmon, webmon, and workflow
make SNSdata.tar.gz  # create fake SNS data for testing
make localdev/up  # build all the services
```

The site is served at http://localhost by default.

After making changes to the source code, it is necessary to:

1. stop the running containers
2. recreate the python wheel(s) if the source code of the apps has changed
3. rebuild the images

Stop the Running Containers

Stoping and deleting the running containers as well as deleting the images and docker volumes:

```
docker-compose down --volumes
```

this command will delete the database. Omit `--volumes` if preservation of the database is desired.

Recreate the Python Wheels

The selected format to inject `dasmon`, `webmon`, and `workflow` apps into their corresponding services is python wheels, thus any changes for the python source code requires rebuilding the python wheel(s).

For instance, if the source code of `dasmon` is changed, run at this point `make wheel/dasmon` to rebuild the `dasmon` wheel.

If necessary, delete all existing wheels with `make wheel/clean`

Rebuild the Images

Run `make localdev/up`. This `make` target builds the services with command `docker-compose up --build` using settings in `docker-compose.yml`.

More information on docker commands for this project can be found *here*.

Uploading a Database Dump

Make target `localdev/dbup` contains the shell command to load the database dump and start the service. Assuming that:

- we started the `webmon` Conda environment
- the full path to the dump file is `./database_dump_file.sql`:
- the current working directory is the root of the source tree (containing file `.env`):

```
(webmon) $> dbdumpfile=./database_dump_file.sql make DATABASE_PASS=$(dotenv get DATABASE_
↪PASS) localdev/dbup
```

Target `localdev/dbup` sets `LOAD_INITIAL_DATA="false"`, thus preventing loading the default database dump (file "db_init.json")

Description of Test Fixtures

The docker files for each test fixture is in the root of the repository with the name `Dockerfile.<fixturename>`. The overall design of the test fixture is captured here . One must be authenticated to view this repository.

`activemq` fixture is built from rmohr/activemq with the configuration from `src/workflow_app/workflow/icat_activemq.xml`. It is the activemq broker.

`autoreducer` fixture runs a copy of post processing agent. This is given a fake filesystem with the contents of `tests/data` in the location `/SNS/` (at the root level of the filesystem).

`amq_test_gen` fixture creates pretend messages associated with runs being saved. It contains the code in the web monitor test AMQ message generator repository . One must be authenticated to view this repository.

`amq_pv_gen` fixture creates fake process variables (PVs) that the data aquisition would make. It contains the code in the web monitor test pv generator repository . One must be authenticated to view this repository.

`catalog_process` fixture is running the script located in `src/catalog/catalog_process.py` which responds with the messages in a similar way to how ONCAT would. The script creates a *`Listener`* and responds accordingly.

Docker information

Note: This document is updated, however, it may be good to read the `docker-compose` and `Dockerfile.*` in the repository themselves for the most up-to-date information.

This guide assumes that `docker` and docker-compose are present on your system.

Starting and Stopping

While docker can be used to start each individual container separately, using `docker-compose up --build` is the preferred method because it starts all services in the correct order. Pressing `ctrl-c` will cleanly shutdown interactive docker. Pressing `ctrl-c` multiple times will kill the running images and leave docker in a somewhat funny state that likely requires running `docker-compose down` before starting again An additional flag `-d` can be supplied to run docker in detached mode.

Note: Use `docker-compose --file <filename>` to select a different configuration

To start a single image, supply its name as an additional argument to `docker-compose up`. To stop all images, including in detached mode, run `docker-compose down`.

Cleaning docker

The *build instructions* suggest using the `--build` flag which will build images before starting the containers. Additionally, one may want to use the `--force-recreate` flag to recreate images even if ther configuration and images haven't changed. The following commands can be used (in this order) to further clean out docker and start with a cleaner state (`-f` with get rid of the confirmation):

- Use `docker container prune` to prune all stopped containers
- [Optional] Use `docker image prune` to remove all unused images
- Use `docker volume prune` to prune all unused volumes

if explicit pruning does not work, use `docker system prune -f -a --volumes` to purge all.

Misc

1. Several things to keep in mind while running Web monitor via docker:
   - The option `-d` will start the web-monitor in the background. Remove it if you want to run the web-monitor in the foreground.
   - The command `docker container logs CONTAINER_NAME` will provide the runtime log for given container, where `CONTAINER_NAME` can be found via `docker ps`.
   - Add option `--build` to force rebuild the container if the local changes are not reflected in the container.
   - Add option `--force-recreate` to recreate all images if `--build` does not work.
   - **If all fails (e.g. the local changes are not showing up in the runtime instances):**
     - stop the instance with `docker-compose down`.
     - prune caches of images, container and volumes.

- restart the instance with `docker-compose up -d --build --force-recreate`.

2. If you cannot find web-monitor at `localhost`, it is possible that the standard http port 80 is used by another application. Here are two possible solutions:
   - Stop the service running at port 80 and restart the instance.
   - Modify the port of nginx in the docker compose file to use a different port (e.g. change to `81:80`).

How to Modify an Instrument Autoreduction Configuration Page

Note: This document is a draft. If you use the instructions, please consider correcting/improving them.

Instrument scientists can set up a form where Web Monitor users with staff status can modify the autoreduction script for the instrument. The web form lets the instrument scientists change parameter values in the autoreduction script and create a new version on the analysis cluster at

```
/SNS/<instrument>/shared/autoreduce/reduce_<instrument>.py
```

from where it is fetched by the postprocessing agent during autoreduction. Here is an example of the form for the instrument CNCS (can only be viewed by logged in users with staff or superuser status).

How to add a parameter to the configuration page

The following describes how an instrument scientist can add new parameters to an existing autoreduction configuration form. The instrument CNCS will be used as an example.

1. When the instrument scientist submits new parameter values in Web Monitor, a message containing a dictionary of key-value pairs is sent to the post-processing agent. The post-processing agent replaces the template variables (e.g. `${motor_names}`) in `reduce_<instrument>.py.template` with values from the dictionary and writes a new version of `reduce_<instrument>.py`. To add a new parameter, first add the template variable to the template script:

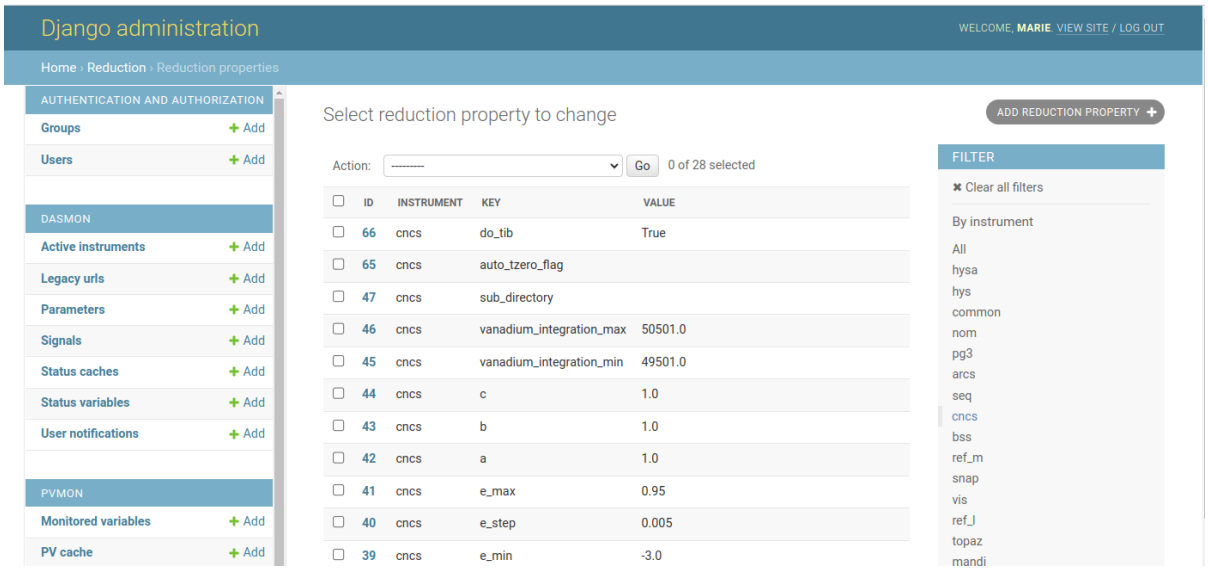
On the analysis cluster, modify the template Python script for the instrument

```
/SNS/<instrument>/shared/autoreduce/reduce_<instrument>.py.template
```

to define the new templated variable and the logic using it, e.g.:

```
Motor_names="${motor_names}"
...
elog=ExperimentLog()
elog.setSERotOptions(Motor_names)
```

2. Add an entry with instrument, a parameter key/name and default value to the reduction properties model in the database using the Django administrative interface.



3. Make the parameter part of the configuration form by adding it as a field (e.g. `FloatField`) to the class `ReductionConfigurationCNCSForm` found in forms.py.

4. Make the parameter visible on the web page by adding the parameter and a descriptive label to the HTML template used to display the form in configuration_cncs.html.

5. Create a pull request to https://github.com/neutrons/data_workflow/tree/next.

6. When the change has been merged, a new release of Web Monitor should be created in GitHub. Once the release is available, work with the DevOps team to deploy the new release into production, see the *deployment instructions*.

How to test changes locally

Set up a local instance of Web Monitor by following the *build instructions*.

How to test in the test environment

TODO

Guide to Contributing

Contributions to this project are welcome. All contributors agree to the following:

Contributing as a User

- Bug report and feature request:

  Please use the issue tracker to submit bug report and feature requests.

- Pull requests:

  Please fork the webmon project and submit a PR to the `next` branch. When submitting the PR, please make sure to include the following information in the description:

  - A link to the corresponding issue (if applicable).
  - A summary of changes introduced in the PR.
  - Attach a screenshot of the affected page (if applicable).
  - Select a reviewer from the SAE team to review the PR.

  Once the PR is approved and merged into `next`, the feature will be available to the `main` branch at the start of the next release cycle.

Contribute as a Developer of SAE

As a developer of Software Engineering Aplication Engineering (SAE), you should request access to the internal issue tracking system for project webmon. When making a PR to introduce bug fixes or new features, please follow the guidelines below:

- Use the `next` branch as the base.
- Select the `next` branch as the target branch for your PR.
- Add a link to the internal issue at the top of the PR description.
- Provide a summary of the changes in the PR description.
- Attach a screenshot of the affected page (if applicable).
- Select a reviewer from the SAE team to review the PR.

Once the PR is approved and all tests pass, a senior developer will take care of the merging. The feature will be available to the `main` branch at the start of the each release cycle.

- It is assumed that the contributor is an ORNL employee and belongs to the development team. Thus the following instructions are specific to ORNL development team's process.
- You have permission and any required rights to submit your contribution.
- Your contribution is provided under the license of this project and may be redistributed as such.
- All contributions to this project are public.

All contributions must be "signed off" in the commit log and by doing so you agree to the above.

Getting access to the main project

Direct commit access to the project is currently restricted to core developers. All other contributions should be done through pull requests.

Development procedure

1. A developer is assigned with a task during neutron status meeting and changes the task's status to **In Progress**.
2. The developer creates a branch off `next` and completes the task in this branch.
3. The developer creates a merge request (MR) off `next`.
4. The developer asks for another developer as a reviewer to review the MR. An MR can only be approved and merged by the reviewer.
5. The developer changes the task's status to **Complete** and closes the associated issue.

Contacting the Team

The best mechanism for a user to request a change is to contact the project managers. Please email Peter F. Peterson or John Hetrick with your request.

A change needs to be in the form of a:

- Story for any enhancement request
- Defect for any bug fix request.

Manual testing

Guest View

Web Monitor should be tested for the *Guest User View* use case.

The complications comes from that the behavior is dependent on the domain you are accessing from.

The setting `ALLOWED_DOMAIN` defines which domains have access to the Guest View, which allows you to see limited information without logging in. The default domains are `"ornl.gov"`, `"sns.gov"`.

If connecting from the correct domain without logging in you should be greeted with the list of instruments like is shown in the use case, otherwise you will be redirected to login.

To test the behavior you can try any of the following

- If connecting to ORNL with VPN, try with and without VPN to see the difference.
- Change the `ALLOWED_DOMAIN` settings to a domain that matches you current domain and then to one that doesn't match
- Connect from two different devices, one within the domain and one outside.

General User View

Web Monitor should be tested for the *General User View* use case.

To test these views from a general user's perspective you must login with an account that has the appropriate permissions. The account must have access to a run populated with enough data to confirm the elements in the above doc appear and are functional.

For instance, the example in the docs uses https://monitor.sns.gov/report/arcs/214581/ Please ensure you can access it or a run like it on monitor.sns.gov before proceeding with the test.

Please confirm:

- Catalog information appears at the top of the page
- Then the plot data appears next and is interactable

Instrument Scientist View

Web Monitor should be tested for the *Instrument Scientist View* use case.

Please confirm the UI elements that appeared in the General User View also appear when logged in as an Instrument Scientist.

In addition, please ensure the post-processing buttons that appear in the first screenshot are interactable and return no error when submitted.

Follow the directions in the linked Instrument Scientist View doc to open the autoreduction page. Fill in the information listed, including clicking the small round plus icon to add additional mask info, and submit. Ensure no error is returned or appears on the page. Click the reset button, ensure this also does not result in an error.

Finally, follow the directions to open the postprocessing page. Again, fill in the form and hit submit. Ensure no error is returned. Then attempt to find the job you just submitted by filling in the exact same information and instead clicking the 'find' button.

Deployment

To deploy the web-monitor, you require **web-monitor**, **workflow-db** and **workflow-mgr**. For test deployments there are also **catalog**, **testfixtures**, **amqbroker** and **autoreducer** to fake external services.

Configuration

web-monitor environment variables

Variable	Secret	Description
AMQ_BROKER		List of ActiveMQ brokers
APP_SECRET	yes	Django SECRET_KEY
CATALOG_ID	yes	ONCat client ID
CATALOG_SECRET	yes	ONCat client secret
CATALOG_URL	yes	ONCat URL
DATABASE_HOST		PostgreSQL hostname
DATABASE_NAME		Database name
DATABASE_PASS	yes	PostgreSQL Owner password
DATABASE_PORT		PostgreSQL post
DATABASE_USER		PostgreSQL Owner username
DJANGO_SETTINGS		Description of settings
ICAT_PASS	yes	ActiveMQ password
ICAT_USER		ActiveMQ username
LDAP_CERT_FILE	yes	ldap.OPT_X_TLS_CACERTFILE
LDAP_DOMAIN_CO	yes	Use in AUTH_LDAP_USER_DN_TEMPLATE
LDAP_SERVER_URI	yes	AUTH_LDAP_SERVER_URI
TIME_ZONE		Time zone to use
LOAD_INITIAL_DA		In non-production environments, loads default database “db_init.json” when set to “true”. Set to “false” if loading a database dump instead

workflow-db environment variables

Variable	Secret	Description
POSTGRES_DB		Database name
POSTGRES_PASSWORD	yes	PostgreSQL Owner password
POSTGRES_USER		PostgreSQL Owner username

workflow-mgr environment variables

Variable	Secret	Description
AMQ_BROKER		List of ActiveMQ brokers
AMQ_QUEUE		List of ActiveMQ queues dasmon should listen to|
APP_SECRET	yes	Django SECRET_KEY
DATABASE_HOST		PostgreSQL hostname
DATABASE_NAME		Database name
DATABASE_PASS	yes	PostgreSQL Owner password|
DATABASE_PORT		PostgreSQL post
DATABASE_USER		PostgreSQL Owner username
ICAT_PASS	yes	ActiveMQ password
ICAT_USER		ActiveMQ username
TIME_ZONE		Time zone to use
WORKFLOW_USER		ActiveMQ workflow username
WORKFLOW_PASS	yes	ActiveMQ workflow password

catalog environment variables (TEST only)

Variable	Secret	Description
ACTIVE_MQ_HOST		ActiveMQ hostname
ACTIVE_MQ_PORTS		ActiveMQ port
ICAT_PASS	yes	ActiveMQ password
ICAT_USER		ActiveMQ username

testfixtures environment variables (TEST only)

Variable	Secret	Description
BROKER		ActiveMQ broker address
DATABASE_HOST		PostgreSQL hostname
DATABASE_NAME		Database name
DATABASE_PASS	yes	PostgreSQL Owner password|
DATABASE_PORT		PostgreSQL post
DATABASE_USER		PostgreSQL Owner username
ICAT_PASS	yes	ActiveMQ password
ICAT_USER		ActiveMQ username

Additional configuration files

- **amqbroker-deploy** (TEST only) -> icat_activemq.xml
  - icat and workflow username and passwords are set in here
- **autoreducer-deploy** (TEST only)-> post_processing.conf
  - ActiveMQ server address needs to be set in here
  - icat username and password needs to be set in here
- **web-monitor-deploy** -> nginx conf

Notes

You need to make sure the following variables match:

- `DATABASE_*` in **web-monitor**, **workflow-mgr** and **testfixtures**, and **POSTGRES_*** in database
- `ICAT_USER` and `ICAT_PASS` in **web-monitor**, **workflow-mgr**, **catalog** and **testfixtures**, and **amqbroker** (`icat_activemq.xml`) and **autoreducer** (`post_processing.conf`)
- `WORKFLOW_USER` and `WORKFLOW_PASS` in **workflow-mgr** and in **amqbroker** (`icat_activemq.xml`)

5.3.2 Modules

The web-monitor contains three independent Django applications

- *dasmon_listener*: to interface with the data acquisition system (DAS).
- *webmon*: user facing web interface, visit the production version at monitor.sns.gov.
- *workflow*: backend manager.

and a mocked catalog services.

dasmon_app

dasmon_listener package

Submodules

dasmon_listener.amq_consumer module

DASMON ActiveMQ consumer class

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

class dasmon_listener.amq_consumer.**Client**(*brokers*, *user*, *passcode*, *queues=None*, *consumer_name='amq_consumer'*)

Bases: `object`

ActiveMQ client Holds the connection to a broker

connect()

Connect to a broker

get_connection(*listener=None*)

Establish and return a connection to ActiveMQ

Parameters
listener – listener object

listen_and_wait(*waiting_period=1.0*)

Listen for the next message from the brokers. This method will simply return once the connection is terminated.

Parameters
waiting_period – sleep time between connection to a broker

send(*destination*, *message*, *persistent='false'*)

Send a message to a queue. This send method is used for heartbeats and doesn't need AMQ persistent messages.

Parameters

- **destination** – name of the queue
- **message** – message content
- **persistent** – true, to set persistent header

set_listener(*listener*)

Set the listener object that will process each incoming message.

Parameters
listener – listener object

stop()

Disconnect and stop the client

class dasmon_listener.amq_consumer.**Listener**

Bases: ConnectionListener

Base listener class for an ActiveMQ client

A fully implemented class should overload the on_message() method to process incoming messages.

on_message(*frame*)

Process a message.

Parameters
frame – stomp.utils.Frame

retrieve_instrument(*instrument_name*)

Retrieve or create an instrument given its name

retrieve_parameter(*key*)

Retrieve of create a Parameter entry

dasmon_listener.amq_consumer.**notify_users**(*instrument_id*, *signal*)

Find users who need to be notified and send them a message

Parameters

- **instrument_id** – Instrument object
- **signal** – Signal object

dasmon_listener.amq_consumer.**process_SMS**(*instrument_id*, *headers*, *data*)

Process SMS process information The message content looks like this:

{u'start_sec': u'1460394343',
u'src_id': u'SMS_32162', u'msg_type': u'2686451712', u'facility': u'SNS', u'timestamp': u'1460394348', u'dest_id': u'', u'start_nsec': u'554801929', u'instrument': u'BL16B', u'reason': u'SMS run stopped', u'run_number': u'3014'}

Parameters

- **instrument_id** – Instrument object
- **data** – data dictionary

dasmon_listener.amq_consumer.**process_ack**(*data=None*, *headers=None*)

Process a ping request ack

Parameters
data – data that came in with the ack

dasmon_listener.amq_consumer.**process_signal**(*instrument_id*, *data*)

Process and store signal messages.

Asserted signals look like this: {

"msg_type": "2147483648", "src_name": "DASMON.0", "timestamp": "1375464085", "sig_name": "SID_SVP_HIGH", "sig_source": "DAS", "sig_message": "SV Pressure too high!", "sig_level": "3"

}

Retracted signals look like this: {

"msg_type": "2147483649", "src_name": "DASMON.0", "timestamp": "1375464079", "sig_name": "SID_SVP_HIGH"

}

Parameters

- **instrument_id** – Instrument object
- **data** – data dictionary

dasmon_listener.amq_consumer.**send_message**(*sender*, *recipients*, *subject*, *message*)

Send an email message

Parameters

- **sender** – email of the sender
- **recipients** – list of recipient emails
- **subject** – subject of the message
- **message** – content of the message

dasmon_listener.amq_consumer.**store_and_cache**(*instrument_id*, *key_id*, *value*, *timestamp=None*, *cache_only=False*)

Protected store and cache process. Store and cache a DASMON parameter

Parameters

- **instrument_id** – Instrument object

- **key_id** – key Parameter object
- **value** – value for the given key
- **cache_only** – only update cache

dasmon_listener.amq_consumer.**store_and_cache_**(*instrument_id*, *key_id*, *value*, *timestamp=None*, *cache_only=False*)

Store and cache a DASMON parameter

Parameters

- **instrument_id** – Instrument object
- **key_id** – key Parameter object
- **value** – value for the given key
- **cache_only** – only update cache

dasmon_listener.listener_daemon module

DASMON listener daemon

class dasmon_listener.listener_daemon.**DasMonListenerDaemon**(*pidfile*, *stdin='/dev/null'*, *stdout='/dev/null'*, *stderr='/dev/null'*)

Bases: *Daemon*

DASMON listener daemon

run()

Run the dasmon listener daemon

dasmon_listener.listener_daemon.**run**()

Entry point for command line script

Module contents

webmon_app

reporting package

Subpackages

reporting.dasmon package

Subpackages

reporting.dasmon.templatetags package

Submodules

reporting.dasmon.templatetags.dasmon_tags module

reporting.dasmon.templatetags.dasmon_tags.**is_number**(*value*)

reporting.dasmon.templatetags.dasmon_tags.**strip**(*value*)

Module contents

Submodules

reporting.dasmon.admin module

class reporting.dasmon.admin.**ActiveInstrumentAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument_id', 'is_alive', 'is_adara')

list_editable = ('is_alive', 'is_adara')

property media

class reporting.dasmon.admin.**LegacyURLAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument_id', 'url', 'long_name')

property media

class reporting.dasmon.admin.**ParameterAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'name', 'monitored')

list_editable = ('monitored',)

property media

class reporting.dasmon.admin.**SignalAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument_id', 'name', 'message', 'level', 'timestamp')

property media

class reporting.dasmon.admin.**StatusVariableAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument_id', 'key_id', 'value', 'timestamp', 'msg_time')

list_filter = ('instrument_id', 'key_id')

property media

msg_time(*obj*)

class reporting.dasmon.admin.**UserNotificationAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'user_id', 'email')

property media

reporting.dasmon.legacy_status module

Get the status of legacy instruments

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2015 Oak Ridge National Laboratory

reporting.dasmon.legacy_status.**get_legacy_url**(*instrument_id*, *include_domain=True*)

Generate URL for legacy instrument status data

Parameters

- **instrument_id** – Instrument object
- **include_domain** – True if we need to return a complete URL

reporting.dasmon.legacy_status.**get_ops_status**(*instrument_id*)

Pull the legacy status information

Parameters

instrument_id – Instrument object

reporting.dasmon.models module

Dasmon app models @author: M. Doucet, Oak Ridge National Laboratory @copyright: 2015 Oak Ridge National Laboratory

class reporting.dasmon.models.**ActiveInstrument**(**args*, ***kwargs*)

Bases: Model

Table containing the list of instruments that are expecting to have their DAS turned ON

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

has_pvsd

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

has_pvstreamer

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id_id

is_adara

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_alive

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <reporting.dasmon.models.ActiveInstrumentManager object>

class reporting.dasmon.models.**ActiveInstrumentManager**(**args*, ***kwargs*)

Bases: Manager

Table of options for instruments

has_pvsd(*instrument_id*)

Returns True if the instrument is running pvsd Defaults to False

has_pvstreamer(*instrument_id*)

Returns True if the instrument is running PVStreamer Defaults to True

is_adara(*instrument_id*)

Returns True if the instrument is running ADARA

is_alive(*instrument_id*)

Returns True if the instrument should be presented as part of the suite of instruments

class reporting.dasmon.models.**LegacyURL**(**args*, ***kwargs*)

Bases: Model

Table of URLs pointing to the legacy instrument status service

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id_id

long_name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

url

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.dasmon.models.**Parameter**(**args*, ***kwargs*)

Bases: Model

Table holding the names of the measured quantities

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

monitored

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

statuscache_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

statusvariable_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

class reporting.dasmon.models.**Signal**(**args*, ***kwargs*)

Bases: Model

Table of signals received from DASMON

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_timestamp(***, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(***, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id_id

level

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

message

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

source

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.dasmon.models.**StatusCache**(**args*, ***kwargs*)

Bases: Model

Table of cached status variable values

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id_id

key_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

key_id_id

objects = <django.db.models.manager.Manager object>

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.dasmon.models.**StatusVariable**(**args*, ***kwargs*)

Bases: Model

Table containing key-value pairs from the DASMON

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id_id

key_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

key_id_id

objects = <django.db.models.manager.Manager object>

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.dasmon.models.**UserNotification**(**args*, ***kwargs*)

Bases: Model

Table of users to notify

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

email

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instruments

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

objects = <django.db.models.manager.Manager object>

registered

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

user_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

reporting.dasmon.urls module

Define url structure

reporting.dasmon.view_util module

Status monitor utilities to support 'dasmon' views

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

class reporting.dasmon.view_util.**SignalEntry**(*name=''*, *status=''*, *assert_time=''*, *key=''*, *ack_url=''*)

Bases: object

Utility class representing a DASMON signal

reporting.dasmon.view_util.**add_status_entry**(*instrument_id*, *message_channel*, *value*)

Add a status message for a given instrument.

Parameters

- **instrument_id** – Instrument object
- **message_channel** – name of the AMQ channel used to report updates
- **value** – value for the entry (string)

reporting.dasmon.view_util.**dasmon_diagnostics**(*instrument_id*, *timeout=None*)

Diagnostics for DASMON

Parameters

- **instrument_id** – Instrument object
- **timeout** – number of seconds of silence before declaring a problem

reporting.dasmon.view_util.**fill_template_values**(*request*, ***template_args*)

Fill a template dictionary with information about the instrument

reporting.dasmon.view_util.**get_cached_variables**(*instrument_id*, *monitored_only=False*)

Get cached parameter values for a given instrument

Parameters

- **instrument_id** – Instrument object
- **monitored_only** – if True, only monitored parameters are returned

reporting.dasmon.view_util.**get_completeness_status**(*instrument_id*)

Check that the latest runs have successfully completed post-processing

Parameters
instrument_id – Instrument object

reporting.dasmon.view_util.**get_component_status**(*instrument_id*, *red_timeout=1*, *yellow_timeout=None*, *process='dasmon'*)

Get the health status of an ADARA component

Parameters

- **red_timeout** – number of hours before declaring a process dead
- **yellow_timeout** – number of seconds before declaring a process slow

reporting.dasmon.view_util.**get_dashboard_data**()

Get all the run and error rates

reporting.dasmon.view_util.**get_instrument_status_summary**()

Create an instrument status dictionary that can be used to fill out the summary page template or the summary update response.

reporting.dasmon.view_util.**get_instruments_for_user**(*request*)

Get the list of instruments for a given user

reporting.dasmon.view_util.**get_latest**(*instrument_id*, *key_id*)

Returns the latest entry for a given key on a given instrument

Parameters

- **instrument_id** – Instrument object

- **key_id** – Parameter object

reporting.dasmon.view_util.**get_latest_updates**(*instrument_id*, *message_channel*, *timeframe=2.0*, *number_of_entries=10*, *start_time=None*)

Return a list of recent status messages received on a given channel.

Parameters

- **instrument_id** – Instrument object
- **message_channel** – name of the AMQ channel used to report updates
- **timeframe** – number of days to report on
- **number_of_entries** – number of recent entries to return if nothing was found in the desired time frame
- **start_time** – earliest time of returned entries. Takes precedence over timeframe and number.

reporting.dasmon.view_util.**get_live_runs**(*timeframe=12*, *number_of_entries=25*, *instrument_id=None*, *as_html=True*)

Get recent runs for all instruments. If no run is found in the last few hours (defined by the timeframe parameter), we return the last few runs (defined by the number_of_entries parameter).

Parameters

- **timeframe** – number of hours going back from now, defining the period of time for the runs
- **number_of_entries** – number of entries to return if we didn't find any run in the defined period
- **instrument_id** – if provided, results will be limited to the given instrument

reporting.dasmon.view_util.**get_live_runs_update**(*request*, *instrument_id*, *ipts_id*, ***data_dict*)

Get updated information about the latest runs

Parameters

- **request** – HTTP request so we can get the 'since' parameter
- **instrument_id** – Instrument model object
- **ipts_id** – filter by experiment, if provided
- **data_dict** – dictionary to populate

reporting.dasmon.view_util.**get_live_variables**(*request*, *instrument_id*)

Create a data dictionary with requested live data

Parameters

- **request** – HttpRequest object
- **instrument_id** – Instrument object

reporting.dasmon.view_util.**get_monitor_breadcrumbs**(*instrument_id*, *current_view='monitor'*)

Create breadcrumbs for a live monitoring view

Parameters

- **instrument_id** – Instrument object
- **current_view** – name to give this view

reporting.dasmon.view_util.**get_pvstreamer_status**(*instrument_id*, *red_timeout=1*, *yellow_timeout=None*)

Get the health status of PVStreamer

Parameters

- **red_timeout** – number of hours before declaring a process dead
- **yellow_timeout** – number of seconds before declaring a process slow

reporting.dasmon.view_util.**get_run_list**(*run_list*)

Get a list of run object and transform it into a list of dictionaries that can be used as a simple dictionary that can be shipped as json.

Parameters

run_list – list of run object (usually a QuerySet)

reporting.dasmon.view_util.**get_signals**(*instrument_id*)

Get the current list of signals/alarms for a given instrument

Parameters

instrument_id – Instrument object

reporting.dasmon.view_util.**get_system_health**(*instrument_id=None*)

Get system health status. If an instrument_id is provided, the sub-systems relevant to that instrument will also be provided, otherwise only common sub-systems are provided.

Parameters

instrument_id – Instrument object

reporting.dasmon.view_util.**get_workflow_status**(*red_timeout=1*, *yellow_timeout=None*)

Get the health status of Workflow Manager

Parameters

- **red_timeout** – number of hours before declaring a process dead
- **yellow_timeout** – number of seconds before declaring a process slow

reporting.dasmon.view_util.**is_running**(*instrument_id*)

reporting.dasmon.view_util.**postprocessing_diagnostics**(*timeout=None*)

Diagnostics for the auto-reduction and cataloging

Parameters

timeout – number of seconds of silence before declaring a problem

reporting.dasmon.view_util.**pvstreamer_diagnostics**(*instrument_id*, *timeout=None*, *process='pvstreamer'*)

Diagnostics for PVStreamer

Parameters

- **instrument_id** – Instrument object
- **timeout** – number of seconds of silence before declaring a problem
- **process** – name of the process to diagnose (pvsd or pvstreamer)

reporting.dasmon.view_util.**workflow_diagnostics**(*timeout=None*)

Diagnostics for the workflow manager

Parameters

timeout – number of seconds of silence before declaring a problem

reporting.dasmon.views module

Live monitoring

reporting.dasmon.views.**notifications**(*request*)

Let an instrument team member register for a DASMON signal

Module contents

reporting.pvmon package

Submodules

reporting.pvmon.admin module

class reporting.pvmon.admin.**AddForm**(**args*, ***kwargs*)

Bases: ModelForm

class Meta

Bases: object

exclude = ()

model

alias of *MonitoredVariable*

base_fields = {'instrument': <django.forms.models.ModelChoiceField object>, 'pv_name': <django.forms.models.ModelChoiceField object>, 'rule_name': <django.forms.fields.CharField object>}

declared_fields = {}

property media

Return all media required to render the widgets on this form.

class reporting.pvmon.admin.**MonitoredVariableAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

form

alias of *AddForm*

list_display = ('id', 'instrument', 'pv_name', 'rule_name')

list_editable = ('pv_name', 'rule_name')

property media

class reporting.pvmon.admin.**PVAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

get_timestamp(*pv*)

list_display = ('id', 'instrument', 'name', 'value', 'status', 'update_time', 'get_timestamp')

list_filter = ('instrument', 'name', 'status')

property media

class reporting.pvmon.admin.**PVNameAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'name', 'monitored')

list_editable = ('monitored',)

property media

class reporting.pvmon.admin.**PVNameCharField**(*, *max_length=None*, *min_length=None*, *strip=True*, *empty_value=''*, ***kwargs*)

Bases: CharField

to_python(*value*)

Return a string.

reporting.pvmon.models module

Models for PV monitor app

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

class reporting.pvmon.models.**MonitoredVariable**(**args*, ***kwargs*)

Bases: Model

Table of PVs that need special monitoring and might have DASMON rules associated with them

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id

objects = <django.db.models.manager.Manager object>

pv_name

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

pv_name_id

rule_name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.pvmon.models.**PV**(**args*, ***kwargs*)

Bases: `Model`

Table holding values

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

instrument_id

name

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

name_id

objects = <django.db.models.manager.Manager object>

status

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

update_time

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.pvmon.models.**PVCache**(**args*, ***kwargs*)

Bases: Model

Table holding the latest values

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id

name

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

name_id

objects = <django.db.models.manager.Manager object>

status

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

update_time

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.pvmon.models.**PVName**(**args*, ***kwargs*)

Bases: Model

Table holding the Process Variable names

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

monitored

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

monitoredvariable_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

pv_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

pvcache_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

pvstring_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

pvstringcache_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class reporting.pvmon.models.**PVString**(**args*, ***kwargs*)

Bases: `Model`

Table holding string values

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id

name

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

name_id

objects = <django.db.models.manager.Manager object>

status

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

update_time

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.pvmon.models.**PVStringCache**(**args*, ***kwargs*)

Bases: Model

Table holding the latest string values

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id

name

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

name_id

objects = <django.db.models.manager.Manager object>

status

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

update_time

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

reporting.pvmon.urls module

Define url structure

reporting.pvmon.view_util module

Utilities to compile the PVs stored in the web monitor DB.

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

`reporting.pvmon.view_util.`**get_cached_variables**(*instrument_id*, *monitored_only=False*)

Get cached PV values for a given instrument

Parameters

- **instrument_id** – Instrument object
- **monitored_only** – if True, only monitored PVs are returned

`reporting.pvmon.view_util.`**get_live_variables**(*request*, *instrument_id*, *key_id=None*)

Create a data dictionary with requested live data

Parameters

- **request** – HTTP request object
- **instrument_id** – Instrument object
- **key_id** – key to return data for, if request is None

reporting.pvmon.views module

Live PV monitoring

Module contents

reporting.reduction package

Submodules

reporting.reduction.admin module

class reporting.reduction.admin.**ChoiceAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument', 'property', 'description', 'value')

list_filter = ('instrument',)

property media

class reporting.reduction.admin.**PropertyDefaultAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'property', 'value', 'timestamp')

property media

class reporting.reduction.admin.**PropertyModificationAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'property', 'value', 'user', 'timestamp')

list_filter = ('property', 'user')

property media

class reporting.reduction.admin.**ReductionPropertyAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument', 'key', 'value', 'timestamp')

list_filter = ('instrument', 'key')

property media

reporting.reduction.forms module

Forms for auto-reduction configuration

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

class reporting.reduction.forms.**BaseReductionConfigurationForm**(**args*, ***kwargs*)

Bases: Form

Base class for reduction form

base_fields = {}

declared_fields = {}

property media

Return all media required to render the widgets on this form.

set_instrument(*instrument*)

Populate instrument-specific options.

Parameters
instrument – instrument short name

to_db(*instrument_id*, *user=None*)

Store the form data

Parameters

- **instrument_id** – Instrument object
- **user** – user that made the change

to_template()

Return a dictionary

class reporting.reduction.forms.**MaskForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: Form

Simple form for a mask entry. A combination of banks, tubes, pixels can be specified.

base_fields = {'bank': <django.forms.fields.CharField object>, 'pixel': <django.forms.fields.CharField object>, 'remove': <django.forms.fields.BooleanField object>, 'tube': <django.forms.fields.CharField object>}

declared_fields = {'bank': <django.forms.fields.CharField object>, 'pixel': <django.forms.fields.CharField object>, 'remove': <django.forms.fields.BooleanField object>, 'tube': <django.forms.fields.CharField object>}

classmethod from_dict_list(*param_value*)

Return a list of dictionaries that is compatible with our form

Parameters
param_value – string representation of the dictionary

property media

Return all media required to render the widgets on this form.

classmethod to_dict_list(*mask_list*)

Create a list of mask dictionary from a set of mask forms

Parameters
mask_list – list of MaskForm objects

classmethod to_python(*mask_list*, *indent=' '*)

Take a block of Mantid script from a list of mask forms

Parameters

- **mask_list** – list of MaskForm objects
- **indent** – string indentation to add to each line

classmethod to_tokens(*value*)

Takes a block of Mantid script and extract the dictionary argument. The template should be like

MaskBTPParameters({'Bank':'', 'Tube':'', 'Pixel':''})

Parameters
value – string value for the code snippet

class reporting.reduction.forms.**PlottingForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Bases: Form

Simple form for a mask entry. A combination of banks, tubes, pixels can be specified.

base_fields = {'maximum': <django.forms.fields.FloatField object>, 'minimum': <django.forms.fields.FloatField object>, 'perpendicular_to': <django.forms.fields.ChoiceField object>, 'remove': <django.forms.fields.BooleanField object>}

declared_fields = {'maximum': <django.forms.fields.FloatField object>, 'minimum': <django.forms.fields.FloatField object>, 'perpendicular_to': <django.forms.fields.ChoiceField object>, 'remove': <django.forms.fields.BooleanField object>}

classmethod from_dict_list(*param_value*)

Return a list of dictionaries that is compatible with our form

Parameters
param_value – string representation of the dictionary

property media

Return all media required to render the widgets on this form.

classmethod to_dict_list(*opt_list*)

Create a list of option dictionary from a set of plotting forms

Parameters
optlist – list of PlottingForm objects

class reporting.reduction.forms.**ReductionConfigurationCNCSForm**(**args*, ***kwargs*)

Bases: *BaseReductionConfigurationForm*

Generic form for DGS reduction instruments

base_fields = {'a': <django.forms.fields.FloatField object>, 'alpha': <django.forms.fields.FloatField object>, 'auto_tzero_flag': <django.forms.fields.BooleanField object>, 'b': <django.forms.fields.FloatField object>, 'beta': <django.forms.fields.FloatField object>, 'c': <django.forms.fields.FloatField object>, 'create_elastic_nxspe': <django.forms.fields.BooleanField object>, 'create_md_nxs': <django.forms.fields.BooleanField object>, 'do_tib': <django.forms.fields.BooleanField object>, 'e_max': <django.forms.fields.FloatField object>, 'e_min': <django.forms.fields.FloatField object>, 'e_pars_in_mev': <django.forms.fields.BooleanField object>, 'e_step': <django.forms.fields.FloatField object>, 'gamma': <django.forms.fields.FloatField object>, 'grouping': <django.forms.fields.ChoiceField object>, 'mask': <django.forms.fields.CharField object>, 'motor_names': <django.forms.fields.CharField object>, 'processed_vanadium': <django.forms.fields.CharField object>, 'raw_vanadium': <django.forms.fields.CharField object>, 'sub_directory': <django.forms.fields.CharField object>, 't0': <django.forms.fields.CharField object>, 'temperature_names': <django.forms.fields.CharField object>, 'tib_max': <django.forms.fields.CharField object>, 'tib_min': <django.forms.fields.CharField object>, 'u_vector': <django.forms.fields.CharField object>, 'v_vector': <django.forms.fields.CharField object>, 'vanadium_integration_max': <django.forms.fields.FloatField object>, 'vanadium_integration_min': <django.forms.fields.FloatField object>}

declared_fields = {'a': <django.forms.fields.FloatField object>, 'alpha': <django.forms.fields.FloatField object>, 'auto_tzero_flag': <django.forms.fields.BooleanField object>, 'b': <django.forms.fields.FloatField object>, 'beta': <django.forms.fields.FloatField object>, 'c': <django.forms.fields.FloatField object>, 'create_elastic_nxspe': <django.forms.fields.BooleanField object>, 'create_md_nxs': <django.forms.fields.BooleanField object>, 'do_tib': <django.forms.fields.BooleanField object>, 'e_max': <django.forms.fields.FloatField object>, 'e_min': <django.forms.fields.FloatField object>, 'e_pars_in_mev': <django.forms.fields.BooleanField object>, 'e_step': <django.forms.fields.FloatField object>, 'gamma': <django.forms.fields.FloatField object>, 'grouping': <django.forms.fields.ChoiceField object>, 'mask': <django.forms.fields.CharField object>, 'motor_names': <django.forms.fields.CharField object>, 'processed_vanadium': <django.forms.fields.CharField object>, 'raw_vanadium': <django.forms.fields.CharField object>, 'sub_directory': <django.forms.fields.CharField object>, 't0': <django.forms.fields.CharField object>, 'temperature_names': <django.forms.fields.CharField object>, 'tib_max': <django.forms.fields.CharField object>, 'tib_min': <django.forms.fields.CharField object>, 'u_vector': <django.forms.fields.CharField object>, 'v_vector': <django.forms.fields.CharField object>, 'vanadium_integration_max': <django.forms.fields.FloatField object>, 'vanadium_integration_min': <django.forms.fields.FloatField object>}

property media

Return all media required to render the widgets on this form.

set_instrument(*instrument*)

Populate instrument-specific options.

Parameters
instrument – instrument short name

class reporting.reduction.forms.**ReductionConfigurationCorelliForm**(**args*, ***kwargs*)

Bases: *BaseReductionConfigurationForm*

Generic form for Corelli reduction instruments

base_fields = {'mask': <django.forms.fields.CharField object>, 'plot_requests': <django.forms.fields.CharField object>, 'ub_matrix_file': <django.forms.fields.CharField object>, 'useCC': <django.forms.fields.BooleanField object>, 'vanadium_SA_file': <django.forms.fields.CharField object>, 'vanadium_flux_file': <django.forms.fields.CharField object>}

declared_fields = {'mask': <django.forms.fields.CharField object>, 'plot_requests': <django.forms.fields.CharField object>, 'ub_matrix_file': <django.forms.fields.CharField object>, 'useCC': <django.forms.fields.BooleanField object>, 'vanadium_SA_file': <django.forms.fields.CharField object>, 'vanadium_flux_file': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

class reporting.reduction.forms.**ReductionConfigurationDGSForm**(**args*, ***kwargs*)

Bases: *BaseReductionConfigurationForm*

Generic form for DGS reduction instruments

base_fields = {'e_max': <django.forms.fields.FloatField object>, 'e_min': <django.forms.fields.FloatField object>, 'e_step': <django.forms.fields.FloatField object>, 'grouping': <django.forms.fields.ChoiceField object>, 'mask': <django.forms.fields.CharField object>, 'processed_vanadium': <django.forms.fields.CharField object>, 'raw_vanadium': <django.forms.fields.CharField object>}

declared_fields = {'e_max': <django.forms.fields.FloatField object>, 'e_min': <django.forms.fields.FloatField object>, 'e_step': <django.forms.fields.FloatField object>, 'grouping': <django.forms.fields.ChoiceField object>, 'mask': <django.forms.fields.CharField object>, 'processed_vanadium': <django.forms.fields.CharField object>, 'raw_vanadium': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

set_instrument(*instrument*)

Populate instrument-specific options.

Parameters
instrument – instrument short name

class reporting.reduction.forms.**ReductionConfigurationREFMForm**(**args*, ***kwargs*)

Bases: *BaseReductionConfigurationForm*

Generic form for REF_M reduction instruments

base_fields = {'bck_max': <django.forms.fields.IntegerField object>, 'bck_min': <django.forms.fields.IntegerField object>, 'bck_width': <django.forms.fields.IntegerField object>, 'const_q_cutoff': <django.forms.fields.FloatField object>, 'fit_peak_in_roi': <django.forms.fields.BooleanField object>, 'force_background': <django.forms.fields.BooleanField object>, 'force_peak': <django.forms.fields.BooleanField object>, 'peak_max': <django.forms.fields.IntegerField object>, 'peak_min': <django.forms.fields.IntegerField object>, 'plot_in_2D': <django.forms.fields.BooleanField object>, 'q_step': <django.forms.fields.FloatField object>, 'skip_quicknxs': <django.forms.fields.BooleanField object>, 'use_const_q': <django.forms.fields.BooleanField object>, 'use_roi_bck': <django.forms.fields.BooleanField object>, 'use_sangle': <django.forms.fields.BooleanField object>, 'use_side_bck': <django.forms.fields.BooleanField object>}

declared_fields = {'bck_max': <django.forms.fields.IntegerField object>, 'bck_min': <django.forms.fields.IntegerField object>, 'bck_width': <django.forms.fields.IntegerField object>, 'const_q_cutoff': <django.forms.fields.FloatField object>, 'fit_peak_in_roi': <django.forms.fields.BooleanField object>, 'force_background': <django.forms.fields.BooleanField object>, 'force_peak': <django.forms.fields.BooleanField object>, 'peak_max': <django.forms.fields.IntegerField object>, 'peak_min': <django.forms.fields.IntegerField object>, 'plot_in_2D': <django.forms.fields.BooleanField object>, 'q_step': <django.forms.fields.FloatField object>, 'skip_quicknxs': <django.forms.fields.BooleanField object>, 'use_const_q': <django.forms.fields.BooleanField object>, 'use_roi_bck': <django.forms.fields.BooleanField object>, 'use_sangle': <django.forms.fields.BooleanField object>, 'use_side_bck': <django.forms.fields.BooleanField object>}

property media

Return all media required to render the widgets on this form.

class reporting.reduction.forms.**ReductionConfigurationSEQForm**(**args*, ***kwargs*)

Bases: *ReductionConfigurationDGSForm*

Reduction form for SEQ

base_fields = {'create_elastic_nxspe': <django.forms.fields.BooleanField object>, 'e_max': <django.forms.fields.FloatField object>, 'e_min': <django.forms.fields.FloatField object>, 'e_step': <django.forms.fields.FloatField object>, 'grouping': <django.forms.fields.ChoiceField object>, 'mask': <django.forms.fields.CharField object>, 'processed_vanadium': <django.forms.fields.CharField object>, 'raw_vanadium': <django.forms.fields.CharField object>}

declared_fields = {'create_elastic_nxspe': <django.forms.fields.BooleanField object>, 'e_max': <django.forms.fields.FloatField object>, 'e_min': <django.forms.fields.FloatField object>, 'e_step': <django.forms.fields.FloatField object>, 'grouping': <django.forms.fields.ChoiceField object>, 'mask': <django.forms.fields.CharField object>, 'processed_vanadium': <django.forms.fields.CharField object>, 'raw_vanadium': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

reporting.reduction.forms.**validate_float_list**(*value*)

Parameters
value – string value to parse

reporting.reduction.forms.**validate_integer_list**(*value*)

Allow for "1,2,3" and "1-3"

Parameters
value – string value to parse

reporting.reduction.models module

class reporting.reduction.models.**Choice**(**args*, ***kwargs*)

Bases: Model

Table of choices for forms

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

description

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id

objects = <django.db.models.manager.Manager object>

property

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

property_id

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.reduction.models.**PropertyDefault**(**args*, ***kwargs*)

Bases: `Model`

Table of default values

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

get_next_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

property

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

property_id

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.reduction.models.**PropertyModification**(**args*, ***kwargs*)

Bases: Model

Table of actions taken by users to modify the reduction property table.

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_timestamp(***, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(***, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

property

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

property_id

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

user_id

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.reduction.models.**ReductionProperty**(**args*, ***kwargs*)

Bases: Model

Table of template properties for reduction scripts

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

choice_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

get_next_by_timestamp(**, field=<django.db.models.fields.DateTimeField: timestamp>, is_next=True, **kwargs*)

get_previous_by_timestamp(**, field=<django.db.models.fields.DateTimeField: timestamp>, is_next=False, **kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id

key

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

propertydefault_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

propertymodification_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

value

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

reporting.reduction.urls module

Define url structure

reporting.reduction.view_util module

Utilities for reduction configuration views

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

`reporting.reduction.view_util.`**reduction_setup_url**(*instrument*)

Return a URL for the reduction setup if it's enabled for the given instrument

Parameters
instrument – instrument name

`reporting.reduction.view_util.`**reset_to_default**(*instrument_id*)

Reset reduction properties for a given instrument to their default value. If no default has been set for a property, it will not be changed.

Parameters
instrument_id – Instrument object

reporting.reduction.view_util.**send_template_request**(*instrument_id*, *template_dict*, *user='unknown'*)

Send an ActiveMQ message to request a new script

Parameters

- **instrument_id** – Instrument object
- **template_dict** – dictionary of peroperties
- **user** – user that created the change

reporting.reduction.view_util.**store_property**(*instrument_id*, *key*, *value*, *user=None*)

Store a reduction property

Parameters

- **instrument_id** – Instrument object
- **key** – name of the property
- **value** – value of the property (string)
- **user** – user that created the change

reporting.reduction.views module

Automated reduction configuration view

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

Module contents

reporting.report package

Subpackages

reporting.report.management package

Subpackages

reporting.report.management.commands package

Submodules

reporting.report.management.commands.add_stored_procs module

class reporting.report.management.commands.add_stored_procs.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Bases: BaseCommand

handle(**args*, ***options*)

The actual logic of the command. Subclasses must implement this method.

help = 'add additional stored procedures to backend database'

reporting.report.management.commands.clearcache module

class reporting.report.management.commands.clearcache.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Bases: BaseCommand

handle(**args*, ***kwargs*)

The actual logic of the command. Subclasses must implement this method.

reporting.report.management.commands.ensure_adminuser module

class reporting.report.management.commands.ensure_adminuser.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Bases: BaseCommand

add_arguments(*parser*)

Entry point for subclassed commands to add custom arguments.

handle(**args*, ***options*)

The actual logic of the command. Subclasses must implement this method.

help = "Creates an admin user non-interactively if it doesn't exist"

Module contents

Module contents

Submodules

reporting.report.admin module

class reporting.report.admin.**DataRunAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'run_number', 'instrument_id', 'ipts_id', 'file', 'created_on')

list_filter = ('instrument_id', 'ipts_id')

property media

class reporting.report.admin.**ErrorAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'run_status_id', 'time', 'description')

list_filter = ('run_status_id__run_id__instrument_id',)

property media

readonly_fields = ('run_status_id',)

search_fields = ['description']

time(*obj*)

class reporting.report.admin.**IPTSAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('expt_name', 'created_on', 'show_instruments')

property media

search_fields = ['instruments__name']

show_instruments(*ipts*)

class reporting.report.admin.**InformationAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'run_status_id', 'time', 'description')

property media

readonly_fields = ('run_status_id',)

search_fields = ['description']

time(*obj*)

class reporting.report.admin.**InstrumentStatusAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument_id', 'last_run_id')

property media

readonly_fields = ('last_run_id',)

class reporting.report.admin.**RunStatusAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'run_id', 'queue_id', 'created_on')

list_filter = ('run_id__instrument_id', 'queue_id')

property media

readonly_fields = ('run_id',)

search_fields = ['run_id__run_number']

class reporting.report.admin.**StatusQueueAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'name', 'is_workflow_input')

list_filter = ('is_workflow_input',)

property media

class reporting.report.admin.**TaskAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'instrument_id', 'input_queue_id', 'task_class', 'task_queues', 'success_queues')

list_filter = ('instrument_id', 'input_queue_id')

property media

search_fields = ['instrument_id__name', 'input_queue_id__name']

class reporting.report.admin.**WorkflowSummaryAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

actions = [<function reduction_not_needed>, <function reduction_needed>, <function reduction_complete>, <function reduction_incomplete>]

date(*summary*)

list_display = ('run_id', 'date', 'complete', 'catalog_started', 'cataloged', 'reduction_needed', 'reduction_started', 'reduced', 'reduction_cataloged', 'reduction_catalog_started')

list_editable = ('reduction_needed',)

list_filter = ('run_id__instrument_id', 'complete', 'catalog_started', 'cataloged', 'reduction_needed', 'reduction_started', 'reduced', 'reduction_cataloged', 'reduction_catalog_started')

property media

readonly_fields = ('run_id',)

search_fields = ['run_id__run_number']

reporting.report.admin.**reduction_complete**(*modeladmin*, *request*, *queryset*)

reporting.report.admin.**reduction_incomplete**(*modeladmin*, *request*, *queryset*)

reporting.report.admin.**reduction_needed**(*modeladmin*, *request*, *queryset*)

reporting.report.admin.**reduction_not_needed**(*modeladmin*, *request*, *queryset*)

reporting.report.catalog module

Optional utilities to communicate with ONcat. ONcat is an online data catalog used internally at ORNL.

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2018 Oak Ridge National Laboratory

reporting.report.catalog.**decode_time**(*timestamp*)

Decode timestamp and return a datetime object :param timestamp: timestamp to decode

reporting.report.catalog.**get_run_info**(*instrument*, *ipts*, *run_number*)

Legacy issue: Until the facility information is stored in the DB so that we can retrieve the facility from it, we'll have to use the application configuration. :param str instrument: instrument short name :param str ipts: experiment name :param str run_number: run number :param str facility: facility name (SNS or HFIR)

reporting.report.detect_mobile module

reporting.report.detect_mobile.**is_mobile**(*request*)

Returns true if a request appears to be coming from a mobile device. The request.mobile data member is also filled in.

Parameters
request – django request object

reporting.report.forms module

Forms for auto-reduction configuration

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2016 Oak Ridge National Laboratory

class reporting.report.forms.**ProcessingForm**(**args*, ***kwargs*)

Bases: Form

Form to send a post-processing request

base_fields = {'create_as_needed': <django.forms.fields.BooleanField object>, 'experiment': <django.forms.fields.CharField object>, 'instrument': <django.forms.fields.ChoiceField object>, 'run_list': <django.forms.fields.CharField object>, 'task': <django.forms.fields.ChoiceField object>}

declared_fields = {'create_as_needed': <django.forms.fields.BooleanField object>, 'experiment': <django.forms.fields.CharField object>, 'instrument': <django.forms.fields.ChoiceField object>, 'run_list': <django.forms.fields.CharField object>, 'task': <django.forms.fields.ChoiceField object>}

property media

Return all media required to render the widgets on this form.

process()

Process the completed form

set_initial(*initial*)

Set the initial values after cleaning them up

Parameters
initial – initial dictionary

reporting.report.forms.**validate_integer_list**(*value*)

Allow for "1,2,3" and "1-3"

Parameters
value – string value to parse

reporting.report.models module

The definition of the models is shared by the workflow manager and the reporting app. The models are defined in the workflow manager and should be installed on the system before running the app.

reporting.report.urls module

Define url structure

reporting.report.view_util module

Status monitor utilities to support 'report' views

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

reporting.report.view_util.**append_key**(*input_url*, *instrument*, *run_id*)

Append a live data secret key to a url

Parameters

- **input_url** – url to modify
- **instrument** – instrument name
- **run_id** – run number

reporting.report.view_util.**error_rate**(*instrument_id*, *n_hours=24*)

Returns the rate of errors for the last n_hours hours.

Parameters

- **instrument_id** – Instrument model object
- **n_hours** – number of hours to track

reporting.report.view_util.**extract_ascii_from_div**(*html_data*, *trace_id=None*)

Extract data from an plot . Only returns the first one it finds.

Parameters
html_data – string

#TODO: allow to specify which trace to return in cases where we have multiple curves

reporting.report.view_util.**extract_d3_data_from_json**(*json_data*)

DEPRECATED

For backward compatibility, extract D3 data from json data for the old-style interactive plots.

Parameters
json_data – json data block

reporting.report.view_util.**fill_template_values**(*request*, ***template_args*)

Fill the template argument items needed to populate side bars and other satellite items on the pages.

Only the arguments common to all pages will be filled.

reporting.report.view_util.**find_skipped_runs**(*instrument_id*, *start_run_number=0*)

Find run numbers that were skipped for a given instrument

Parameters

- **instrument_id** – Instrument object
- **start_run_number** – run number to start from

reporting.report.view_util.**generate_key**(*instrument: str*, *run_id: int*)

Generate a secret key for a run on a given instrument

Parameters

- **instrument** – instrument name
- **run_id** – run number

reporting.report.view_util.**get_current_status**(*instrument_id*)

Get current status information such as the last experiment/run for a given instrument.

Used to populate AJAX response, so must not contain Model objects

Parameters
instrument_id – Instrument model object

reporting.report.view_util.**get_plot_data_from_server**(*instrument*, *run_id*, *data_type='json'*)

Get json data from the live data server

Parameters

- **instrument** – instrument name
- **run_id** – run number
- **data_type** – data type, either 'json' or 'html'

reporting.report.view_util.**get_plot_template_dict**(*run_object=None*, *instrument=None*, *run_id=None*)

Get template dictionary for plots

Parameters

- **run_object** – DataRun object
- **instrument** – instrument name
- **run_id** – run_number

reporting.report.view_util.**get_post_processing_status**(*red_timeout=0.25*, *yellow_timeout=120*)

Get the health status of post-processing services :param red_timeout: number of hours before declaring a process dead :param yellow_timeout: number of seconds before declaring a process slow

reporting.report.view_util.**get_run_list_dict**(*run_list*)

Get a list of run object and transform it into a list of dictionaries that can be used to fill a table.

Parameters
run_list – list of run object (usually a QuerySet)

reporting.report.view_util.**get_run_status_text**(*run_id*, *show_error=False*, *use_element_id=False*)

Get a textual description of the current status for a given run

Parameters

- **run_id** – run object
- **show_error** – if true, the last error will be whow, otherwise "error"

reporting.report.view_util.**is_acquisition_complete**(*run_id*)

Determine whether the acquisition is complete and post-processing has started

Parameters
run_id – run object

reporting.report.view_util.**needs_reduction**(*request*, *run_id*)

Determine whether we need a reduction link to submit a run for automated reduction

Parameters

- **request** – HTTP request object
- **run_id** – DataRun object

reporting.report.view_util.**processing_request**(*request*, *instrument*, *run_id*, *destination*)

Process a request for post-processing

Parameters

- **instrument** – instrument name
- **run_id** – run number [string]
- **destination** – outgoing AMQ queue

reporting.report.view_util.**retrieve_rates**(*instrument_id*, *last_run_id*)

Retrieve the run rate and error rate for an instrument. Try to get it from the cache if possible.

Parameters

- **instrument_id** – Instrument object
- **last_run_id** – DataRun object

reporting.report.view_util.**run_rate**(*instrument_id*, *n_hours=24*)

Returns the rate of new runs for the last n_hours hours.

Parameters

- **instrument_id** – Instrument model object
- **n_hours** – number of hours to track

reporting.report.view_util.**send_processing_request**(*instrument_id*, *run_id*, *user=None*, *destination=None*, *is_complete=False*)

Send an AMQ message to the workflow manager to reprocess the run

Parameters

- **instrument_id** – Instrument object
- **run_id** – DataRun object

reporting.report.views module

Report views

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014-2015 Oak Ridge National Laboratory

reporting.report.views.**download_reduced_data**(*request*, *instrument*, *run_id*)

Download reduced data from live data server

Parameters

- **request** – http request object
- **instrument** – instrument name
- **run_id** – run number

reporting.report.views.**processing_admin**(*request*)

Form to let admins easily reprocess parts of the workflow

reporting.report.views.**submit_for_cataloging**(*request*, *instrument*, *run_id*)

Send a run for cataloging

Parameters

- **instrument** – instrument name
- **run_id** – run number

reporting.report.views.**submit_for_post_processing**(*request*, *instrument*, *run_id*)

Send a run for complete post-processing

Parameters

- **instrument** – instrument name
- **run_id** – run number

reporting.report.views.**submit_for_reduction**(*request*, *instrument*, *run_id*)

Send a run for automated reduction

Parameters

- **instrument** – instrument name
- **run_id** – run number

Module contents

reporting.reporting_app package

Subpackages

Submodules

reporting.reporting_app.urls module

Define url structure

reporting.reporting_app.view_util module

Utilities common to the whole web application.

@copyright: 2014 Oak Ridge National Laboratory

reporting.reporting_app.view_util.**reduction_setup_url**(*instrument*)

Check whether the reduction app is installed, and if so return a URL for the reduction setup if it's enabled for the given instrument

Parameters
instrument – instrument name

reporting.reporting_app.view_util.**send_activemq_message**(*destination*, *data*)

Send an AMQ message to the workflow manager.

Parameters

- **destination** – queue to send the request to
- **data** – JSON data payload for the message

reporting.reporting_app.wsgi module

WSGI config for reporting_app project.

This module contains the WSGI application used by Django's development server and any production WSGI deployments. It should expose a module-level variable named `application`. Django's `runserver` and `runfcgi` commands discover this application via the `WSGI_APPLICATION` setting.

Usually you will have the standard Django WSGI application here, but it also might make sense to replace the whole Django WSGI application with a custom one that later delegates to the Django one. For example, you could introduce WSGI middleware here, or combine a Django application with an application of another framework.

Module contents

reporting.users package

Submodules

reporting.users.admin module

class reporting.users.admin.**DeveloperNodeAdmin**(*model*, *admin_site*)

Bases: `ModelAdmin`

get_host(*view*)

list_display = ('id', 'ip', 'get_host')

property media

class reporting.users.admin.**NonDeveloperUsers**(*request*, *params*, *model*, *model_admin*)

Bases: `SimpleListFilter`

lookups(*request*, *model_admin*)

Returns a list of tuples. The first element in each tuple is the coded value for the option that will appear in the URL query. The second element is the human-readable name for the option that will appear in the right sidebar.

parameter_name = 'user_type'

queryset(*request*, *queryset*)

Returns the filtered queryset based on the value provided in the query string and retrievable via *self.value()*.

title = 'User type'

class reporting.users.admin.**PageViewAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

get_host(*view*)

list_display = ('user', 'view', 'ip', 'get_host', 'path', 'timestamp')

list_filter = ('user', 'view', <class 'reporting.users.admin.NonDeveloperUsers'>)

property media

class reporting.users.admin.**SNSUserAdmin**(*model*, *admin_site*)

Bases: UserAdmin

get_groups(*user*)

list_display = ('username', 'first_name', 'last_name', 'get_groups', 'is_staff', 'is_superuser')

property media

class reporting.users.admin.**SiteNotificationAdmin**(*model*, *admin_site*)

Bases: ModelAdmin

list_display = ('id', 'message', 'is_active')

list_editable = ('message', 'is_active')

property media

reporting.users.models module

class reporting.users.models.**DeveloperNode**(**args*, ***kwargs*)

Bases: Model

Table of IP names recognized as developer nodes

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ip

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

class reporting.users.models.**PageView**(*id*, *user*, *view*, *path*, *ip*, *timestamp*)

Bases: Model

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

get_next_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=True*, ***kwargs*)

get_previous_by_timestamp(*, *field=<django.db.models.fields.DateTimeField: timestamp>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ip

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

path

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

timestamp

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

user

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

user_id

view

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class reporting.users.models.**SiteNotification**(*id*, *message*, *is_active*)

Bases: Model

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_active

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

message

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

class reporting.users.models.**TruncatingCharField**(**args*, *db_collation=None*, ***kwargs*)

Bases: CharField

get_prep_value(*value*)

Perform preliminary non-db specific value checks and conversions.

reporting.users.urls module

Define url structure

reporting.users.view_util module

View utility functions for user management

reporting.users.view_util.**fill_template_values**(*request*, ***template_args*)

Fill the template argument items needed to populate side bars and other satellite items on the pages.

Only the arguments common to all pages will be filled.

reporting.users.view_util.**is_experiment_member**(*request*, *instrument_id*, *experiment_id*)

Determine whether a user is part of the given experiment.

Parameters

- **request** – request object
- **instrument_id** – Instrument object
- **experiment_id** – IPTS object

reporting.users.view_util.**is_instrument_staff**(*request*, *instrument_id*)

Determine whether a user is part of an instrument team

Parameters

- **request** – HTTP request object
- **instrument_id** – Instrument object

reporting.users.view_util.**login_or_local_required**(*fn*)

Function decorator to check whether a user is allowed to see a view.

reporting.users.view_util.**login_or_local_required_401**(*fn*)

Function decorator to check whether a user is allowed to see a view.

Usually used for AJAX calls.

reporting.users.view_util.**monitor**(*fn*)

Function decorator to monitor page usage

reporting.users.views module

User management

reporting.users.views.**perform_login**(*request*)

Perform user authentication

reporting.users.views.**perform_logout**(*request*)

Logout user

Module contents

Submodules

reporting.manage module

Module contents

workflow_app

workflow package

Subpackages

workflow.database package

Subpackages

workflow.database.report package

Submodules

workflow.database.report.models module

class workflow.database.report.models.**DataRun**(**args*, ***kwargs*)

Bases: Model

TODO: run number should be unique for a given instrument

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

classmethod create_and_save(*run_number*, *ipts_id*, *instrument_id*, *file*)

Create a database entry for this run and update the instrument status

created_on

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

file

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_created_on(*, *field=<django.db.models.fields.DateTimeField: created_on>*, *is_next=True*, ***kwargs*)

get_previous_by_created_on(*, *field=<django.db.models.fields.DateTimeField: created_on>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id_id

instrumentstatus_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

ipts_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

ipts_id_id

is_complete()

Return completion status

json_encode()

Encode the object as a JSON dictionnary

last_error()

Return last error

objects = <workflow.database.report.models.DataRunManager object>

run_number

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

runstatus_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

workflowsummary_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class workflow.database.report.models.**DataRunManager**(**args*, ***kwargs*)

Bases: `Manager`

get_last_cached_run(*instrument_id*)

Try to get the last run from the InstrumentStatus table. If we can't find it, find it the long way and add the result to the cache.

Parameters

- **instrument_id** – Instrument object
- **ipts_id** – IPTS object

get_last_run(*instrument_id*, *ipts_id=None*)

Get the last run for a given instrument and experiment. Returns None if nothing was found.

Parameters

- **instrument_id** – Instrument object
- **ipts_id** – IPTS object

class workflow.database.report.models.**Error**(**args*, ***kwargs*)

Bases: Model

Details of a particular error event

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

description

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

run_status_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

run_status_id_id

class workflow.database.report.models.**IPTS**(**args*, ***kwargs*)

Bases: Model

Table holding IPTS information

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

created_on

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

datarun_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

expt_name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_next_by_created_on(*, *field=<django.db.models.fields.DateTimeField: created_on>*, *is_next=True*, ***kwargs*)

get_previous_by_created_on(*, *field=<django.db.models.fields.DateTimeField: created_on>*, *is_next=False*, ***kwargs*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instruments

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

number_of_runs(*instrument_id=None*)

Returns the total number of runs for this IPTS on the given instrument.

Parameters
instrument_id – Instrument object

objects = <workflow.database.report.models.IPTSManager object>

class workflow.database.report.models.**IPTSManager**(**args*, ***kwargs*)

Bases: `Manager`

get_last_ipts(*instrument_id*)

Get the last experiment object for a given instrument. Returns None if nothing was found.

Parameters
instrument_id – Instrument object

ipts_for_instrument(*instrument_id*)

class workflow.database.report.models.**Information**(**args*, ***kwargs*)

Bases: Model

Extra information associated with a status update

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

description

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

run_status_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

run_status_id_id

class workflow.database.report.models.**Instrument**(*id*, *name*)

Bases: Model

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

activeinstrument_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

choice_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

datarun_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrumentstatus_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

legacyurl_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

monitoredvariable_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

number_of_expts()

Returns the total number of experiments for this instrument

number_of_runs()

Returns the total number of runs for this instrument

objects = <workflow.database.report.models.InstrumentManager object>

pv_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

pvcache_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

pvstring_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

pvstringcache_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

reductionproperty_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

signal_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

statuscache_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

statusvariable_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

task_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class workflow.database.report.models.**InstrumentManager**(**args*, ***kwargs*)

Bases: `Manager`

find_instrument(*instrument*)

Get the object associated to an instrument name

sql_dump()

class workflow.database.report.models.**InstrumentStatus**(**args*, ***kwargs*)

Bases: `Model`

Cache the latest information for each instrument. This can be used to quickly access status information.

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

instrument_id_id

last_run_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

last_run_id_id

objects = <django.db.models.manager.Manager object>

class workflow.database.report.models.**RunStatus**(**args*, ***kwargs*)

Bases: Model

Map ActiveMQ messages, which have a header like this: headers: {'expires': '0', 'timestamp': '1344613053723',

'destination': '/queue/POSTPROCESS.DATA_READY', 'persistent': 'true', 'priority': '5', 'message-id': 'ID:mac83086.ornl.gov-59780-1344536680877-8:2:1:1:1'}

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

created_on

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

error_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

get_next_by_created_on(*, *field=<django.db.models.fields.DateTimeField: created_on>*, *is_next=True*, ***kwargs*)

get_previous_by_created_on(*, *field=<django.db.models.fields.DateTimeField: created_on>*, *is_next=False*, ***kwargs*)

has_errors()

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

information_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

last_error()

Return the last available error object for this status

last_info()

Return the last available information object for this status

message_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <workflow.database.report.models.RunStatusManager object>

queue_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

queue_id_id

run_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

run_id_id

class workflow.database.report.models.**RunStatusManager**(**args*, ***kwargs*)

Bases: `Manager`

get_last_error(*run_id*)

last_timestamp(*run_id*)

Returns the last timestamp for this run

Parameters
run_id – DataRun object

status(*run_id*, *status_description*)

Returns all database entries for a given run and a given status message.

Parameters

- **run_id** – DataRun object
- **status_description** – status message, as a string

class workflow.database.report.models.**StatusQueue**(**args*, ***kwargs*)

Bases: Model

Table containing the ActiveMQ queue names used as status

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_workflow_input

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

runstatus_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

task_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class workflow.database.report.models.**Task**(**args*, ***kwargs*)

Bases: Model

Define a task

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

input_queue_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

input_queue_id_id

instrument_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

instrument_id_id

json_encode()

Encode the object as a JSON dictionary

objects = <workflow.database.report.models.TaskManager object>

success_queue_ids

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

success_queues()

task_class

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

task_queue_ids

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

task_queues()

class workflow.database.report.models.**TaskManager**(**args*, ***kwargs*)

Bases: `Manager`

sql_dump()

Get the object associated to an instrument name

class workflow.database.report.models.**WorkflowSummary**(**args*, ***kwargs*)

Bases: `Model`

Overall status of the workflow for a given run

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

catalog_started

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

cataloged

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

complete

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <workflow.database.report.models.WorkflowSummaryManager object>

reduced

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

reduction_catalog_started

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

reduction_cataloged

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

reduction_needed

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

reduction_started

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

run_id

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

run_id_id

update()

Update status according the messages received

class workflow.database.report.models.**WorkflowSummaryManager**(**args*, ***kwargs*)

Bases: `Manager`

get_summary(*run_id*)

Get the run summary for a given DataRun object

Parameters
run_id – DataRun object

incomplete()

Returns the query set of all incomplete runs

Module contents

Submodules

workflow.database.manage module

workflow.database.transactions module

Perform DB transactions

workflow.database.transactions.**add_status_entry**(*headers*, *data*)

Populate the reporting database with the contents of a status message of the following format:

headers: {'expires': '0', 'timestamp': '1344613053723',
'destination': '/queue/POSTPROCESS.DATA_READY', 'persistent': 'true', 'priority': '5', 'message-id': 'ID:mac83086.ornl.gov-59780-1344536680877-8:2:1:1:1'}

The data is a dictionary in a JSON format.

data: {"instrument": tokens[2],
"ipts": tokens[3], "run_number": run_number, "data_file": message}

Parameters

- **headers** – ActiveMQ message header dictionary
- **data** – JSON encoded message content

workflow.database.transactions.**add_task**(*instrument*, *input_queue*, *task_class=''*, *task_queues=None*, *success_queues=None*)

Add a task entry

workflow.database.transactions.**add_workflow_status_entry**(*destination*, *message*)

Add a database entry for an event generated by the workflow manager. This represents additional information regarding interventions by the workflow manager.

Parameters

- **destination** – string representing the StatusQueue
- **message** – JSON encoded data dictionary

workflow.database.transactions.**get_message_queues**(*only_workflow_inputs=True*)

Get the list of message queues from the DB

Parameters
only_workflow_inputs – if True, only the queues that the workflow manager listens to will be returned

workflow.database.transactions.**get_task**(*message_headers*, *message_data*)

Find the DB entry for this queue

Parameters

- **headers** – message headers
- **message** – JSON-encoded message content

workflow.database.transactions.**sql_dump_tasks**()

Dump the SQL necessary to insert the current task definitions

Module contents

Submodules

workflow.amq_client module

ActiveMQ workflow manager client

@author: M. Doucet, Oak Ridge National Laboratory @copyright: 2014 Oak Ridge National Laboratory

class workflow.amq_client.**Client**(*brokers*, *user*, *passcode*, *queues=None*, *workflow_check=False*, *check_frequency=24*, *workflow_recovery=False*, *flexible_tasks=False*, *consumer_name='amq_consumer'*, *auto_ack=True*)

Bases: `object`

ActiveMQ client Holds the connection to a broker

connect()

Connect to a broker

get_connection(*consumer_name=None*)

Get existing connection or create a new one.

listen_and_wait(*waiting_period=1.0*)

Listen for the next message from the brokers. This method will simply return once the connection is terminated.

Parameters
waiting_period – sleep time between connection to a broker

new_connection(*consumer_name=None*)

Establish and return a connection to ActiveMQ

Parameters
consumer_name – name to give the new connection

set_listener(*listener*)

Set the listener object that will process each incoming message.

Parameters
listener – listener object

verify_workflow()

Verify that the workflow has completed for all the runs and recover if it hasn't

workflow.amq_listener module

ActiveMQ listener class for the workflow manager

class workflow.amq_listener.**Listener**(*use_db_tasks=False*, *auto_ack=True*)

Bases: `ConnectionListener`

AMQ listener for the workflow manager

on_message(*frame*)

Process a message. Example of an ActiveMQ header: headers: {'expires': '0', 'timestamp': '1344613053723',

'destination': '/queue/POSTPROCESS.DATA_READY', 'persistent': 'true', 'priority': '5', 'message-id': 'ID:mac83086.ornl.gov-59780-1344536680877-8:2:1:1:1'}

Parameters
frame – stomp.utils.Frame

set_amq_user(*brokers*, *user*, *passcode*)

Set the ActiveMQ credentials to use when created a new connection

set_connection(*connection*)

Set a AMQ connection

workflow.daemon module

Code taken from: http://www.jejik.com/articles/2007/02/a_simple_unix_linux_daemon_in_python/

class workflow.daemon.**Daemon**(*pidfile*, *stdin='/dev/null'*, *stdout='/dev/null'*, *stderr='/dev/null'*)

Bases: object

A generic daemon class.

Usage: subclass the Daemon class and override the run() method

daemonize()

do the UNIX double-fork magic, see Stevens' "Advanced Programming in the UNIX Environment" for details (ISBN 0201563177) http://www.erlenstar.demon.co.uk/unix/faq_2.html#SEC16

delpid()

restart()

Restart the daemon

run()

You should override this method when you subclass Daemon. It will be called after the process has been daemonized by start() or restart().

start()

Start the daemon

status()

stop()

Stop the daemon

workflow.sns_post_processing module

Workflow manager process

class workflow.sns_post_processing.**WorkflowDaemon**(*pidfile*, *stdin='/dev/null'*, *stdout='/dev/null'*, *stderr='/dev/null'*, *check_frequency=None*, *workflow_recovery=False*, *flexible_tasks=False*)

Bases: *Daemon*

Workflow daemon

run()

Run the workflow manager daemon

workflow.sns_post_processing.**run**()

Interactive run command

workflow.sns_post_processing.**run_daemon**(*pid_file*, *stdout_file*, *stderr_file*, *check_frequency*, *recover*, *flexible_tasks*, *command*)

Start daemon

workflow.state_utilities module

workflow.state_utilities.**decode_message**(*message*)

Decode message and turn it into a dictionnary we can understand.

Messages from streaming translation are expected to be an absolute path of the following type:

Old system: /SNS/EQSANS/IPTS-1234/…/EQSANS_5678_event.nxs ADARA: /SNS/EQSANS/IPTS-1234/nexus/EQSANS_5678.nxs.h5

Calibration runs, etc… have 2009_06_24_CAL instead of IPTS-xxxx

workflow.state_utilities.**logged_action**(*action*)

Decorator used to log a received message before processing it

workflow.states module

Action classes to be called when receiving specific messages.

To add an action for a specific queue, add a StateAction class with the name of the queue in lower-case, replacing periods with underscores.

class workflow.states.**Catalog_request**(*connection=None*, *use_db_task=False*)

Bases: *StateAction*

Default action for CATALOG.REQUEST messages

class workflow.states.**Postprocess_data_ready**(*connection=None*, *use_db_task=False*)

Bases: *StateAction*

Default action for POSTPROCESS.DATA_READY messages

class workflow.states.**Reduction_complete**(*connection=None*, *use_db_task=False*)

Bases: *StateAction*

Default action for REDUCTION.COMPLETE messages

class workflow.states.**Reduction_request**(*connection=None*, *use_db_task=False*)

Bases: *StateAction*

Default action for REDUCTION.REQUEST messages

class workflow.states.**StateAction**(*connection=None*, *use_db_task=False*)

Bases: object

Base class for processing messages

send(*destination*, *message*, *persistent='true'*)

Send a message to a queue

Parameters

- **destination** – name of the queue
- **message** – message content

workflow.workflow_process module

Actual process that each data run must go through.

class workflow.workflow_process.**WorkflowProcess**(*connection=None*, *recovery=True*, *allowed_lag=3600*)

Bases: *StateAction*

verify_workflow()

Walk through the data runs and make sure they have gone through the whole workflow.

Module contents

catalog

This is source code for a fake ONCAT service.

catalog_process module

This is source code for a fake ONCAT service.

class catalog_process.**Listener**

Bases: ConnectionListener

on_message(*frame*) → None

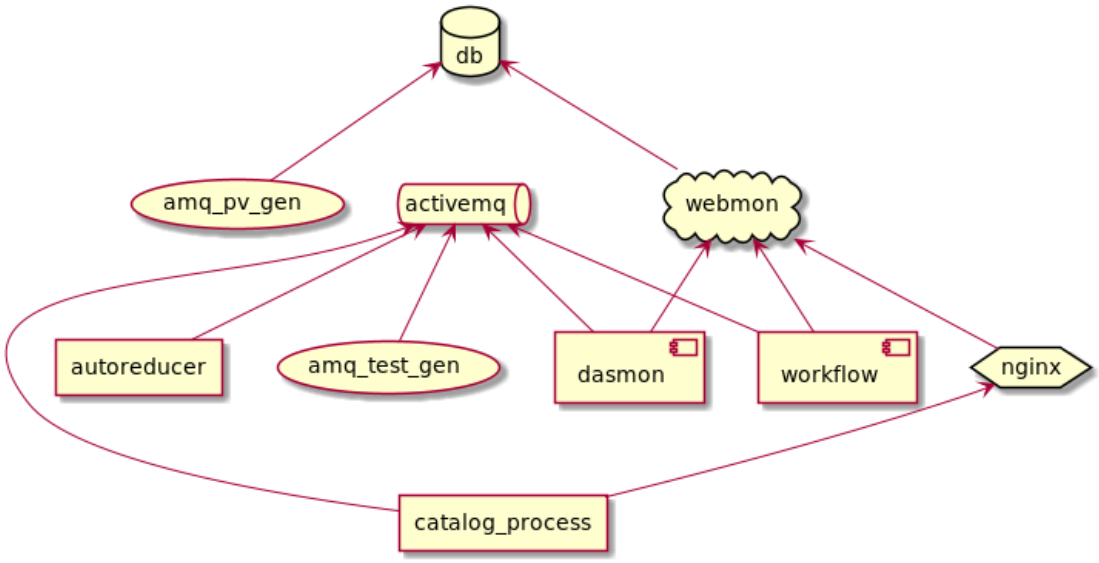
Called by the STOMP connection when a MESSAGE frame is received.

Parameters

frame (*Frame*) – the stomp frame

5.3.3 Services

The components making up the infrastructure of Web Monitor have dependencies. In the diagram below *service1* –> *service2* is to be read as *service1* depends on *service2*. For instance, *amq_pv_gen* depends on *db*.



5.3.4 Related software

- post_processing_agent is the system that runs on the autoreducer nodes
- live_data_server is the system that contains database for holding the plots/divs produced by live reduction and autoreduction
- livereduce is the `sysctl` daemon that runs on instrument computers and generates plots of the active acquisition

5.4 Other use cases

5.4.1 Database View

As a admin user (person with proper role rather than a specific account), I would like to type `https://monitor.sns.gov/database` and see the "django administration site administration" page. Since it is an external app, being able to login and go there is all that is necessary.

5.5 Admin settings

This is a collection of settings stored in the database that are manually edited through the Django admin dashboard found at `/database` sub-directory.

For the production deployment it can be found at https://monitor.sns.gov/database You are required to be a *superuser* to modify the settings, see *Users*.

5.5.1 Active Instrument

URL: https://monitor.sns.gov/database/dasmon/activeinstrument/

Some options can be modified in the full list, but clicking on the `ID` for the instrument allows for modifying the state of the following values

- `Is alive` An instrument can be hidden from the Web Monitor by deselecting this `Is alive` option in the model. This is useful to hide testing or fake instruments.
- `Is adara` for instruments that have switched to the adara DAS
- `Has pvsd` for instruments that have pvstreamer
- `Has pvstreamer` which is not turned on for instruments and appears to be covered by the `Has pvsd` parameter already

5.5.2 Choice

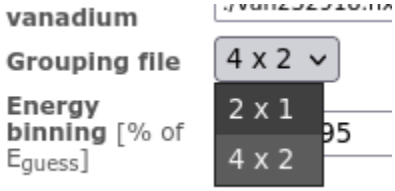
URL: https://monitor.sns.gov/database/reduction/choice/

The `Choice` model controls selection options for instrument reduction templates.

e.g. the `arcs.grouping` properties

ID	INSTRUMENT	PROPERTY	DESCRIPTION	VALUE
7	arcs	arcs.grouping	4 x 2	/SNS/ARCS/shared/autoreduce/ARCS_4X2_grouping.xml
6	arcs	arcs.grouping	2 x 1	/SNS/ARCS/shared/autoreduce/ARCS_2X1_grouping.xml

corresponds to the `Grouping File` selection at https://monitor.sns.gov/reduction/arcs/

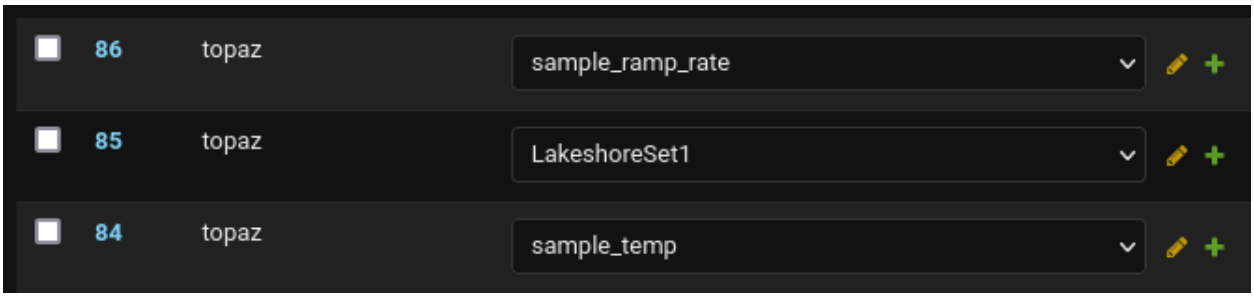


5.5.3 Monitored Variables

URL: https://monitor.sns.gov/database/pvmon/monitoredvariable/

The `Monitored Variable` model control which PVs appear on the instrument status page. To promote a PV from the PV page to the status page just create a model with the correct `Instrument` and `PV Name`. To remove a PV from the status page just delete that model.

For example, the following `Monitored Variables` exist for TOPAZ,



which results in the following PVs appearing on https://monitor.sns.gov/dasmon/topaz/

Signal/PV	Value	History	Last Updated
LakeshoreSet1	300		Aug. 22, 2022, 1:03 p.m.
sample_ramp_rate	0		Sept. 13, 2022, 8:18 a.m.
sample_temp	0		Sept. 13, 2022, 8:19 a.m.

5.5.4 PV Name

URL: https://monitor.sns.gov/database/pvmon/pvname/

A `PV Name` model is created whenever a new PV is received. By default it will be visible on the instrument `PVs` page but if you deselect the `MONITORED` on a particular `PV Name` it will no longer be visible.

5.5.5 Users

URL: https://monitor.sns.gov/database/auth/user/

This allows you to control who is staff or superuser. Being a superuser gives you access to the admin settings while being staff gives you access to the *Admin View*.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W