IceProd Tutorial 2024

IceProd/SimProd Workshop
Summer 2024

Overview

- What is IceProd? Why use it?
- Writing a configuration file
- Dataset lifecycle
- Troubleshooting
- Advanced Topics

Down the rabbit hole?

What is IceProd?

Data provenance

- Configuration for how a file was generated or processed
- Which software, what versions, when/where it ran, …

Dataset submission

- Monitor job status, resource usage
- Auto-retry failed jobs for non-physics errors

Use cases:

- Simulation production
- Experimental data processing
- Common analysis processing
- Other large-scale workloads

Why not DagMan, Snakemake, etc

They are great tools for running a set of jobs once, but bad for keeping history of what exactly was run

- This is not just a nice thing - we regularly get questions about how a dataset was configured, sometimes years later

If your jobs have variable resource requirements, or requirements not known at submit time, these tools won’t work very well

- IceProd can resubmit jobs with higher resource requirements if they fail and HTCondor identifies resource usage as the issue

IceProd Scalability and Resources

IceProd is all about scale. It is specifically designed to handle millions of jobs

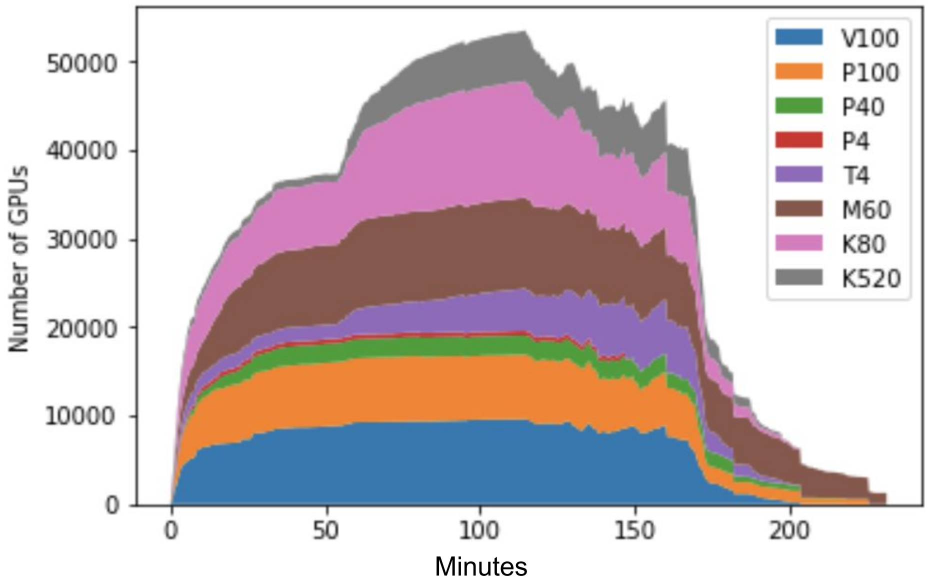
Regular resource pool:

- 20k CPUs, 700 GPUs

Peak resources tested:

- 100k CPUs, 50k GPUs

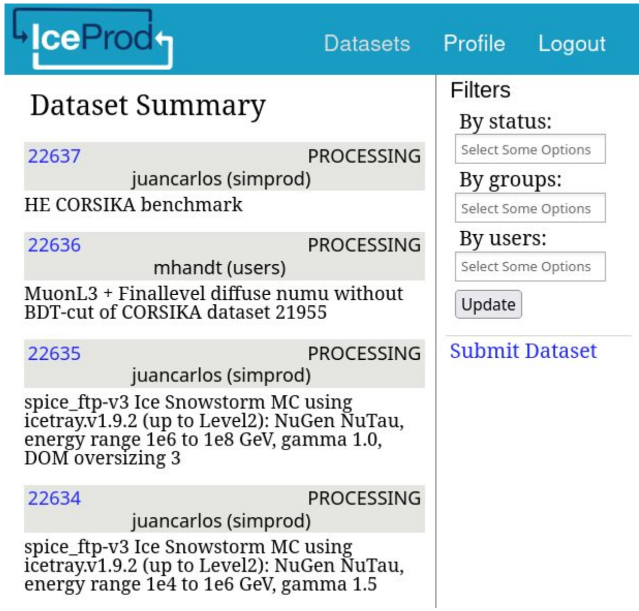
Pre-exascale compute in the cloud



The IceProd Website

Go to https://iceprod2.icecube.wisc.edu

- Log in via IceCube SSO
- Search for a Dataset
- View Dataset details and configuration
- View Job and Task status, logs, and statistics
- Submit new Datasets



Dataset Summary

22637 PROCESSING
juancarlos (simprod)
HE CORSIKA benchmark

22636 PROCESSING
mhandt (users)
MuonL3 + Finallevel diffuse numu without BDT-cut of CORSIKA dataset 21955

22635 PROCESSING
juancarlos (simprod)
spice_ftp-v3 Ice Snowstorm MC using icetray.v1.9.2 (up to Level2): NuGen NuTau, energy range 1e6 to 1e8 GeV, gamma 1.0, DOM oversizing 3

22634 PROCESSING
juancarlos (simprod)
spice_ftp-v3 Ice Snowstorm MC using icetray.v1.9.2 (up to Level2): NuGen NuTau, energy range 1e4 to 1e6 GeV, gamma 1.5

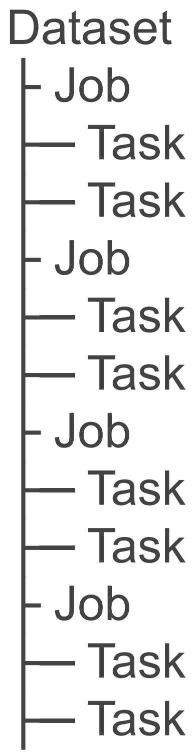
Writing a Configuration File

IceProd Terms

Dataset – a collection of Jobs with a single configuration. They are commonly referred to by number, such as 22634

Job – One of many parallel instances running the configuration, with a job “index” starting at 0 and counting to the number of jobs submitted - 1

Task – Translates to an HTCondor job. Contains multiple Trays, resource requirements, dependencies, and file input / output



IceProd Terms

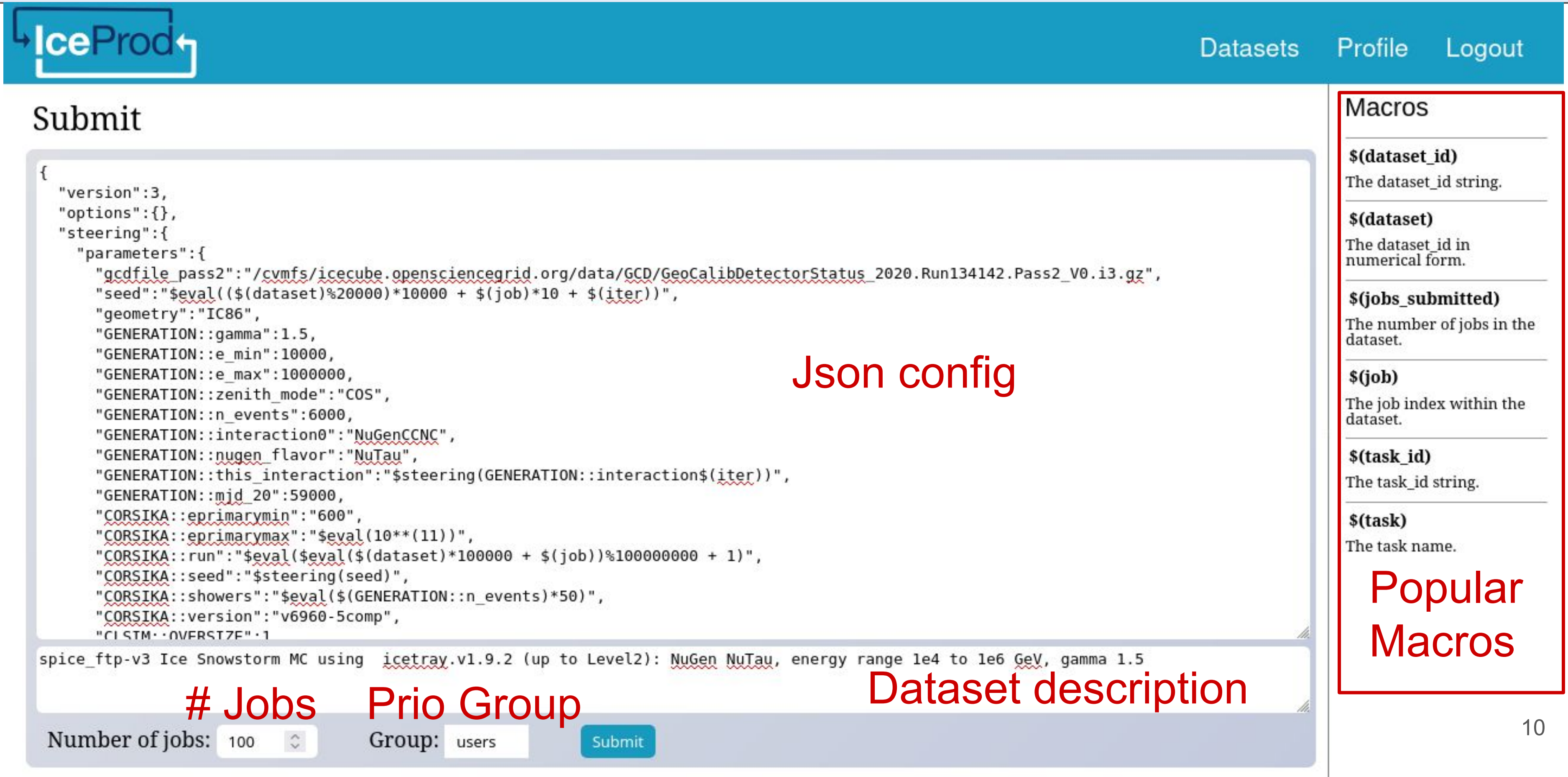
Tray – Like in IceTray, a grouping of Modules inside a Task. Usually just one is used

Iteration – Repeat a Tray multiple times if desired. Works well varying a single parameter

Module – Runs a single script with arguments in a specific environment. Designed for Python scripts, but can run bash or compiled programs

```
Task
 -  Dependencies
 -  Requirements
 -  Files
├ Tray
├── Module
├── Module
├ Tray
└── Module
```

IceProd Configuration File - Json



Submit

```
{
  "version":3,
  "options":{},
  "steering":{
    "parameters":{
      "gcdfile_pass2":"/cvmfs/icecube.opensciencegrid.org/data/GCD/GeoCalibDetectorStatus_2020.Run134142.Pass2_V0.i3.gz",
      "seed":"$eval(($(dataset)%20000)*10000 + $(job)*10 + $(iter))",
      "geometry":"IC86",
      "GENERATION::gamma":1.5,
      "GENERATION::e_min":10000,
      "GENERATION::e_max":1000000,
      "GENERATION::zenith_mode":"COS",
      "GENERATION::n_events":6000,
      "GENERATION::interaction0":"NuGenCCNC",
      "GENERATION::nugen_flavor":"NuTau",
      "GENERATION::this_interaction":"$steering(GENERATION::interaction$(iter))",
      "GENERATION::mjd_20":59000,
      "CORSIKA::eprimarymin":"600",
      "CORSIKA::eprimarymax":"$eval(10**(11))",
      "CORSIKA::run":"$eval($eval($(dataset)*100000 + $(job))%100000000 + 1)",
      "CORSIKA::seed":"$steering(seed)",
      "CORSIKA::showers":"$eval($(GENERATION::n_events)*50)",
      "CORSIKA::version":"v6960-5comp",
      "CLSIM::OVERSIZE":1
```

Json config

```
spice_ftp-v3 Ice Snowstorm MC using  icetray.v1.9.2 (up to Level2): NuGen NuTau, energy range 1e4 to 1e6 GeV, gamma 1.5
```

Dataset description

# Jobs   Prio Group

Number of jobs: 100   Group: users   Submit

Macros

$(dataset_id)	The dataset_id string.
$(dataset)	The dataset_id in numerical form.
$(jobs_submitted)	The number of jobs in the dataset.
$(job)	The job index within the dataset.
$(task_id)	The task_id string.
$(task)	The task name.

Popular Macros

IceProd Configuration File - Json

```
{
  "steering": {
    "parameters": {
      "subdirectory": "$sprintf('%07d-%07d',$eval($(job)//1000*1000),$eval($(job)//1000*1000+999))",
      "TARGET": "gsiftp://gridftp.icecube.wisc.edu/data/sim/IceCube/2023/filtered/level2/
neutrino-generator/$(dataset)/$steering(subdirectory)",
      "outfile": "NuGen_$(job).i3.zst",
      "env_path": "/cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/icetray-env icetray/v1.9.2"
    }
  },
```

Here we define a few global parameters

- Note the use of macros like $(dataset) and $(job), as well as functions like $sprintf and $eval

IceProd Configuration File - Json

```
  "tasks": [ {
    "name": "generation",
    "requirements": {"memory": 2, "time": 1.5},
    "data": [ {
      "type": "permanent",
      "movement": "output",
      "remote": "$steering(TARGET)/$steering(outfile)"
    } ],
    "trays": [ {
      "iterations": 1,
      "modules": [ {
        "name": "NuGen",
        "src": "/cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/metaprojects/icetray/v1.9.2/
simprod-scripts/resources/scripts/nugen.py",
        "args": {
          "FromEnergy": 10000,
          "ToEnergy": 1000000,
          "nevents": 10000,
          "outputfile": "$steering(outfile)"
        },
        "env_shell": "$steering(env_path)"
      } ]
    } ]
  } ]
}
```

Define resource requirements

Transfer output to Madison

Iterations - can use $(iter) inside here

Run NuGen

/cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/icetray-env icetray/v1.9.2 python /cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/metaprojects/icetray/v1.9.2/simprod-scripts/resources/scripts/nugen.py --FromEnergy 10000 --ToEnergy 1000000 --nevents 10000 --outputfile NuGen_0.i3.zst

Task Requirements

Resource requirements:

- cpu: integer, >= 1, default=1
- gpu: integer, >= 0, default=0
- memory: float, > 0, default=1, unit=GB
- disk: float, > 0, default=1, unit=GB
- time: float, > 0, default=1, unit=hour

Other requirements:

- os: list, default=[], uses CVMFS OS string (RHEL_7_x86_64)
- site: string, default=None, used to select IceProd site to run on

Task Dependencies

Like with Dagman, tasks can depend on other tasks within the same job

```
"tasks": [ {
  "name": "generation",
  …
}, {
  "name": "propagation",
  "depends": ["generation"],
  …
} ]
```

Tasks can also depend on tasks from other datasets, using a dataset id and colon before the task name. This is a 1:1 match between job indexes

```
 "tasks": [ {
   "name": "propagation",
   "depends":
["632bbe3ecb8611eea1dd00505684797b:generation"],
 } ]
```

Data Transfer

Data transfer can be defined at the Task level

- type: `permanent` or `job_temp`
- movement: `input`, `output`, or `both`
- transfer: `true` or `false`
- remote: url path
- local: file name

```
  "tasks": [ {
    "name": "generation",
    "data": [ {
      "type": "permanent",
      "movement": "output",
      "remote":
"$steering(TARGET)/$steering(outfile)"
    } ],
    …
  } ]
```

When the type is `permanent` and `local` is not defined, it is assumed to be the basename of the remote path

Data Transfer

When the type is `job_temp` and `remote` is not defined, it will be stored in the global IceProd scratch storage and deleted when a job completes

`job_temp` is primarily used to transfer temporary files between tasks, such as between cpu and gpu tasks

```
"tasks": [ {
  "name": "generation",
  "data": [ {
   "type": "job_temp",
   "movement": "output",
   "local": "$steering(corsika_file)"
  } ],
  …
}, {
  "name": "propagation",
  "data": [ {
   "type": "job_temp",
   "movement": "input",
   "local": "$steering(corsika_file)"
  } ],
  …
} ]
```

Modules

First, define the environment to run in with `env_shell`

This is typically a CVMFS environment + metaproject

```
/cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/icetray-env <metaproject-dir>
```

Then define the `src`

- `src` can be any of:
  - python script
  - bash script
  - linux executable file

```
{
  "name": "NuGen",
  "env_shell":
"/cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/icetray-env
icetray/v1.9.2"
  "src":
"/cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/metaproject
s/icetray/v1.9.2/simprod-scripts/resources/scripts/nugen.p
y",
  "args": {
    "FromEnergy": 10000,
    "ToEnergy": 1000000,
    "nevents": 10000,
    "outputfile": "$steering(outfile)"
  },
}
```

Module Arguments

Args can be either a dict, list, or string. These are all the same:

```
"args": {
  "FromEnergy": 10000,
  "ToEnergy": 1000000,
  "nevents": 10000,
  "outputfile": "$steering(outfile)"
}
```

```
"args": [
  "--FromEnergy=10000",
  "--ToEnergy=1000000",
  "--nevents=10000",
  "--outputfile=$steering(outfile)"
}
```

```
"args": "--FromEnergy=10000 --ToEnergy=1000000 --nevents=10000 --outputfile=$steering(outfile)"
```

You can even define the parsed structure directly:

```
"args": {
  "args": ["$steering(outfile)"],
  "kwargs": {
    "FromEnergy": 10000,
    "ToEnergy": 1000000,
    "nevents": 10000,
  }
}
```

$\Rightarrow$

```
$steering(outfile) --FromEnergy=10000
--ToEnergy=1000000 --nevents=10000
```

Dataset Lifecycle

(for IceProd 3.x)

Dataset Status

- processing: the starting status
- suspended: the dataset is manually suspended
- errors: the dataset has jobs in an error state
- complete: all jobs are complete

A dataset can be reset back to processing:

- A regular reset will reset any non-complete jobs and tasks
- A “hard reset” will reset all jobs and tasks

Job Status

- processing: the starting status
- suspended: the job is manually suspended, or a task has been suspended and no tasks are running
- errors: the job has at least one task that has failed
- complete: all tasks are complete

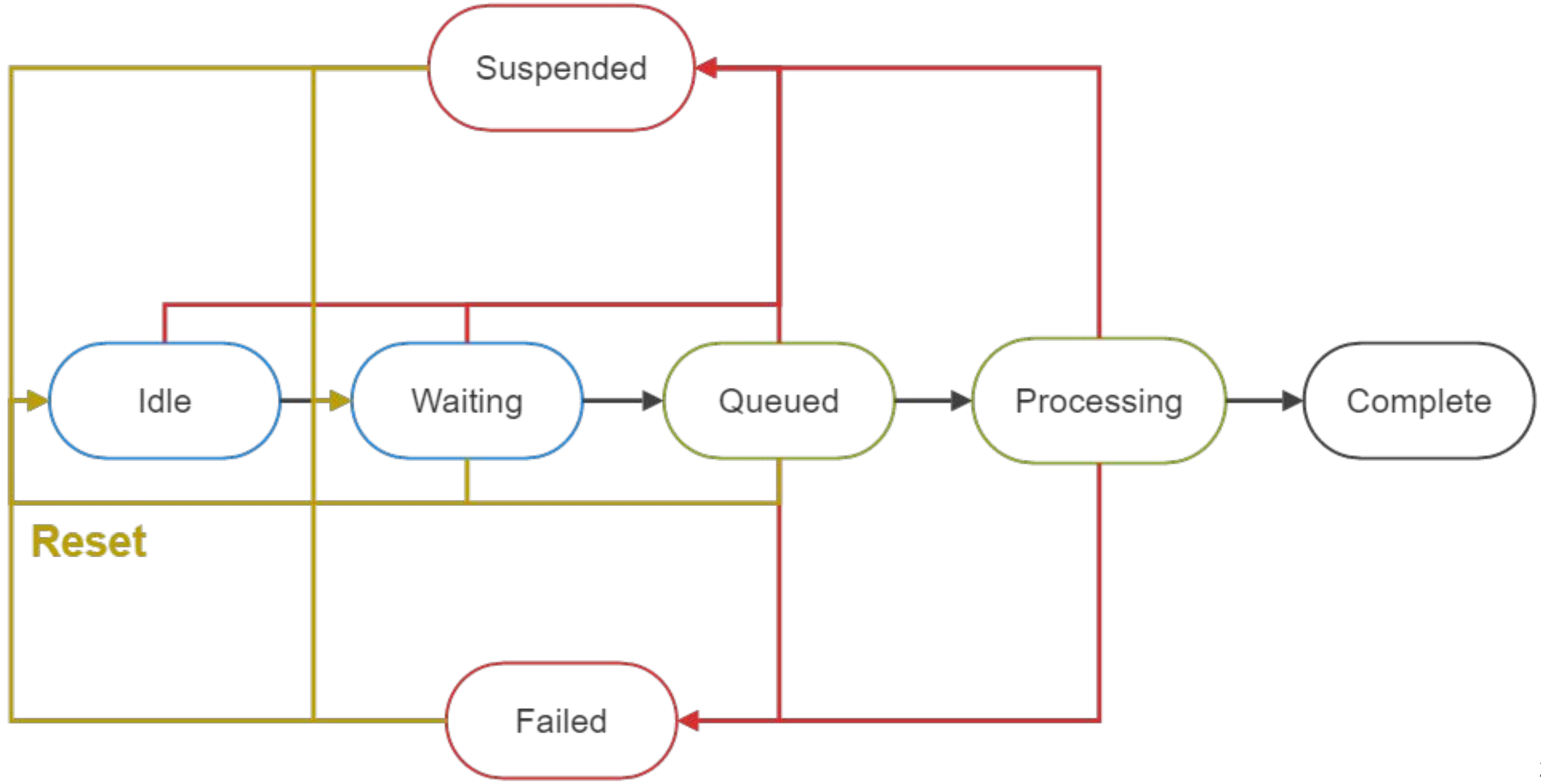
A job can be reset back to processing:

- A regular reset will reset any non-complete tasks
- A “hard reset” will reset all tasks

Task Status

- idle: the starting status, task is waiting on a dependency or priority
- waiting: the task is ready to queue
- queued: the task is queued to HTCondor
- processing: the task is running in HTCondor
- suspended: the task is manually suspended
- failed: the task has a physics error, or 11 non-physics errors
- complete: the task was successful

Task Status


Suspended
Idle
Waiting
Queued
Processing
Complete
Reset
Failed

A note on “physics” errors

A non-physics error is defined as a random or transient error, where it is expected that retrying will fix the problem

Some examples of recoverable errors:

- CVMFS errors
- Illegal instruction
- No disk space remaining
- Ran out of memory

All other errors are assumed to be “physics” errors

Note: IceProd used to retry on all errors, but this can bias clsim in icetray version 1.7.2 or greater, so this newer policy was instituted

Troubleshooting

Initial Steps

1. Look at task logs for any obvious problems

   stdlog is from iceprod itself, stdout and stderr are your script

2. If you’re not sure, reset it to see if it is a transient error
3. Are all the tasks failing? It might be a configuration error

Common Errors

```
RuntimeError: OpenCL error: could build the OpenCL program!
```

Something is wrong with OpenCL on this node

```
RuntimeError: Internal error: unknown particle id from OpenCL
```

This is a bug in CLSim

```
env-shell.sh: line 189: 8208 Segmentation fault
```

There’s a problem in the C++ code that needs to be fixed

Getting Help

```
unknown failure
```

The automated error collection failed, but something went wrong

If you can’t find a suspicious error in the rest of your dataset and only get this, it’s time to ask for help

The best way is to post a message on Slack in the #iceprod channel

- Add details like the dataset and task (links are good), and what you know so far

Advanced Topics

Using the REST API

If you want to aggregate information from a dataset (or multiple datasets), you will need to use the REST API
https://docs.icecube.aq/WIPACrepo/iceprod/master/guide/restapi.html

1. Make a virtualenv and `pip install wipac-rest-tools`
2. Write a script like:

```
from rest_tools.client import SavedDeviceGrantAuth
api = SavedDeviceGrantAuth(
    address='https://iceprod2-api.icecube.wisc.edu',
    token_url='https://keycloak.icecube.wisc.edu/auth/realms/IceCube',
    filename='.iceprod-auth',
    client_id='iceprod-public'
)
# get a list of datasets
result = api.request_seq('GET', '/datasets', {'keys': 'dataset|jobs_submitted'})
for dataset_id, metadata in result.items():
    # do something with the dataset
```

Using the REST API - Example

```
#!/cvmfs/icecube.opensciencegrid.org/iceprod/v2.7.1/env-shell.sh python
from rest_tools.client import SavedDeviceGrantAuth
api = SavedDeviceGrantAuth(
    address='https://iceprod2-api.icecube.wisc.edu',
    token_url='https://keycloak.icecube.wisc.edu/auth/realms/IceCube',
    filename='.iceprod-auth',
    client_id='iceprod-public'
)

# get a list of datasets
result = api.request_seq('GET', '/datasets', {'keys': 'dataset|jobs_submitted'})
for dataset_id, metadata in result.items():
    print(metadata)

# for the last dataset, get a list of tasks
result = api.request_seq('GET', f'/datasets/{dataset_id}/tasks', {'keys': 'job_index|name|status'})
log_task_id = None
for task_id, metadata in result.items():
    print(metadata)
    if metadata['status'] == 'complete':
        log_task_id = task_id

# for the last completed task, get the last logs
result = api.request_seq('GET', f'/datasets/{dataset_id}/tasks/{log_task_id}/logs', {'group': True})
for log in result['logs']:
            if log['name'] == 'stdout':
            print(log['data'])
```

Processing Non-Sequential Files

Simulation files are easy to process, a set of files with sequential numbers

Data files or higher levels that add multiple datasets together present a problem: how to map the files to the jobs?

IceProd has a way to dynamically assign files to tasks, but we’ll need to use the API for this

Processing Non-Sequential Files

1. Load an IceProd environment somehow

   There are pre-existing installs in CVMFS:
   `/cvmfs/icecube.opensciencegrid.org/iceprod/v2.7.1/env-shell.sh`

   Or, `pip install iceprod`

2. Use the https://github.com/WIPACrepo/iceprod/blob/master/bin/basic_submit.py script with a list of input and output files

   This will create a dataset, jobs, and tasks, then map files onto tasks

Processing Non-Sequential Files

Script details:

Files are expected to be either full URLs or paths on the UW-Madison IceCube file system.

Script arguments will be passed as a string. They can use these built-in macros:

- $(input) = The input file list, space-separated
- $(output) = The output file
- $(dataset) = The dataset_id in numerical form
- $(job) = The job index within the dataset.

Processing Non-Sequential Files

An example submission:

```
./basic_submit.py --env_shell
'/cvmfs/icecube.opensciencegrid.org/py3-v4.3.0/icetray-env icetray/1.8.2'
my_script.py '--foo=bar $(input) $(output)' job_files.txt
```

This will execute `my_script.py` from the local directory, while in the icetray environment. If the first line of `job_files.txt` contains:

```
/data/user/XXX/gcdfile.i3.gz /data/user/XXX/infile_01.i3.gz /data/user/XXX/outfile_01.i3.gz
```

Then the first job will look like:

```
my_script.py --foo=bar gcdfile.i3.gz infile_01.i3.gz outfile_01.i3.gz
```

Done! You’re all experts now 😏