

Created by: Maxim Potekhin *potekhin@bnl.gov*

Release Notes

- Version 2.01 20180918 remove extraneous fragments and obsolete references
 - Version 1.05 20180307 setup is now available from the np04dqm directory
 - Version 1.04 20180301 added info about a working LARSoft example
 - Version 1.03 20180228 more verbose info on file permissions
 - Version 1.02 20180226 added information on EOS, directory locations etc
 - Version 1.01 20180120 first official version
-

Introduction

Blitz-p3s

If you want to cut to the chase and run the now-current payloads you can do the following

- obtain credentials from Nektarios Benekos (on the “need to know” basis only) for the production account **np04dqm**
- switch to the p3s *clients* directory `/afs/cern.ch/user/n/np04dqm/public/p3s/p3s/clients`
- run the *job.py* client as detailed below

Submitting a job can be done in this manner:

```
./job.py -j my_job_description.json -f my_input_file.root
```

To make this productive, you will need to look up the input file you want, and identify the job description that suits your needs. For the inputs, please examine the p3s input directory located at:

```
/eos/experiment/neutplatform/protodune/np04tier0/p3s/input
```

and select a file based on the run number or other parameters such as time stamp. As for job descriptions, there are production versions in located in the path `../inputs/larsoft/*` relative to the clients directory. So a realistic example in which jobs are submitted for the FEMB monitor, the 2D raw event display and the TPC monitor can look like this:

```
./job.py -v 2 -j ../inputs/larsoft/femb/fembcount_data.json -f np04_raw_run004429_0065_d18.  
./job.py -v 2 -j ../inputs/larsoft/evdisp/eventdisplay_data.json -f np04_raw_run004429_0065_d18.  
./job.py -v 2 -j ../inputs/larsoft/monitor/hitmonitor_data_main.json -f np04_raw_run004429_0065_d18.
```

The jobs should appear in the “jobs” section of the Web monitor (see one of the sections below for detail). Once completed, the jobs will post their results to the DQM server located at `p3s-content.cern.ch`.

About this document

This document is maintained in the “markdown” format and the source file is contained in the directory `p3s/documents/src`. To rebuild the PDF version as needed you can use the command (provided you have pandoc installed):

```
pandoc -s -o JOB.pdf JOB.md
```

The main purpose of this document is to explain how to set up and run the “job submission client”, with the purpose of executing protoDUNE software in a controlled and monitored way in CERN Tier-0. Details of the p3s design, organization and operation are omitted for sake of brevity, only essentials are presented. If anything does not work as expected, please consult the author of this document.

PLEASE READ THIS

When starting work with p3s it is crucial to keep in mind that your jobs will not run under your identity but under the **p3s** identity which currently corresponds to the user **np04dqm**. This is similar to a situation where your colleague wants to run your software. If this software, configuration files, I/O etc are located in your private directory tree i.e. do not have the right permissions (e.g. files are not readable for others) this will fail. So adjustments need to be made, just like in any kind of collaborative computing. For example, if the script or other kind of executable you intend to run is not both *readable and executable* by p3s (which is in your Linux group at CERN) your submission won’t work. So you need to make sure to use either the “public” sector of your AFS directory tree or an area in EOS readable by the group “np-comp” to host your script/executable. More information is provided in the text below.

Same consideration applies to storage of input and output data. If your script tries to create a file in an area which is write-accessible only to you it won’t work. Likewise, if your script tries to read a file which is only readable by a different user will also fail. However, making things work is fairly easy, for example one might follow these steps:

- create the script you wish to run in the `~/public` area of your AFS home directory at CERN
- make sure it’s “`chmod +x`”
- references to input and output files will work at CERN if they all are in the EOS storage area assigned to the “np-comp” group and you have permissions as described in the text below in detail.

One can also elect to run from the production account which means that most of these requirements are met automatically. There are certain disadvantages to that though.

Preparing to run

Group memberships

These instructions apply to the **lxplus** interactive Linux facility at CERN. To set up access to p3s on that platform one needs to follow a few simple steps as described below. To use p3s for the needs of the protoDUNE experiment it is strongly recommended that you ensure that your account

- is attributed to **np-comp** Linux group at CERN
- has membership in the following CERN e-groups:
 1. eos-experiment-cenf-np04-readers
 2. eos-experiment-cenf-np04-writers

This will allow you to access, move and delete if necessary the data produced by p3s. To get these memberships sorted out you may file a ticket on the CERN services pages and/or contact Nectarios Benekos and the author of this manual.

Getting access to the p3s client software

This can be accomplished in two ways:

- A. install (clone) from the p3s repo - recommended, as it allows you to play with the code
- B. use a pre-installed instance of software in the np04dqm account (in a folder available to the public). It's less flexible and you can't add your files in the same directory tree, but it's prefab so easier to start.

Both methods are documented below.

A. Install the p3s client software from the repo

At a minimum you need the p3s client script (which is written in Python) to submit job descriptions to p3s for execution. You will also benefit from looking at job templates stored as JSON files in the p3s repository. If not already done so, install p3s software at the location of your choice by *cloning the content from GitHub*. For the purposes of this writeup, you are assumed to be on an interactive node located at CERN such as lxplus. Run the command

```
git clone https://github.com/DUNE/p3s.git
```

This step is done in two cases only (so not often):

- you are just starting to use the system
- you are informed that there was an update and you should switch to a newer version

After you run “git clone” your current directory will contain a subdirectory **p3s**. Of immediate interest to you are the following folders within p3s:

- **p3s/clients** containing multiple client scripts with different functions; of particular interest to you right now is the script **job.py**
- **p3s/configuration** containing a few bash scripts which simplify setup for individual sites (such as CERN)
- **p3s/documents** with documentation (such as this writeup) in both Markdown

(md) and PDF format

- **p3s/inputs** and its subdirectories such as jobs/larsfot with job definition and wrapper script templates

B. Use the np04dqm installation (quite easy, really)

Access the following directory:

```
/afs/cern.ch/user/n/np04dqm/public/p3s/
```

In it, you will find same subdirectories as described above, so you can just use the client scripts and other files located there right away, provided that you set up the Python and Linux environment as described directly below.

Set up and verify the Python environment

The next step is also CERN-specific and its purpose is to set up the Python environment for p3s clients without having to install anything yourself. This needs to be done every time you have a fresh shell session which you plan to use for p3s interaction. It may be added to your log-in profile to save some typing line but can also be done manually. Either way, please activate the “Python virtual environment” by running this command:

```
source ~np04dqm/public/vp3s/bin/activate
```

To verify that this actually worked, please change the directory to **p3s/clients** which is mentioned above. Run the script:

```
./verifyImport.py
```

In the output you should see version of Python which is 3.5+, a couple of “OK” messages and finally the word “Success”. If anything is amiss, contact the author of this document.

Set up and verify the Linux environment

While you can specify the server URL and various other parameters the p3s clients need on the command line it is a lot more convenient to just set a few environment variables to be used by default. If running at CERN you could simply use the command (assuming you have the p3s directory as described above)

```
source p3s/configuration/lxvm_np04dqm.sh
```

Now, you can switch to the “clients” directory and try to run the command:

```
./summary.py -P
```

If it connects to the server successfully, it will print a few current stats for p3s.

Example of the output:

```
Domain: p3s-web:80, hostname p3s-web.cern.ch, uptime 22 days, 14:37:23.850000
Pilots: total 200, idle 200, running 0, stopped 0, T0 0
Jobs: total 22, defined 0, running 0, finished 6, pilotT0 16
Workflows 0
Datasets 3
```

If you see a failure, please contact the author of this document.

The Web monitor

The next step is to make sure that you can also see the Web pages served by p3s so you have monitoring functionality. Try to access **http://p3s-web.cern.ch** in your browser (if you are at CERN). Currently the server **p3s-web.cern.ch** is only accessible within the confines of the CERN firewall. If you want to access it from an external Linux machine, please use a ssh tunnel like in the command below

```
ssh -4 -L 8008:p3s-web.cern.ch:80 myCERNaccount@lxplus.cern.ch
```

... in which case pointing your browser to localhost:8008 will result in you seeing the p3s server (which is on port 80 at CERN). On Windows, you can use a ssh utility of your choice in a similar manner, provided it does have the tunneling functionality.

Submitting a job

An example of the job description

For general notes on how jobs are submitted in p3s please see the Appendix.

The purpose of the following dummy example (with rather arbitrary attributes, file names and variables) is to demonstrate how JSON is used to describe jobs in p3s. Let us assume that we created a file named “myjob.json” with the following contents:

```
[
  {
    "name":      "p3s_test",
    "timeout":   "100",
    "jobtype":   "print_date",
    "payload":   "/home/userXYZ/my_executable.sh",
    "env":       {"P3S_MODE": "COPY", "MYFILE": "/tmp/myfile.txt"},
    "priority":  "1",
    "state":     "defined"
  }
]
```

Note that this format corresponds to a *list* of objects i.e. such file can naturally contain a number of jobs; however having just one element in this list is absolutely fine.

For the job to be eligible for execution the “state” attribute needs to be set to “defined” as shown above. Other possible states will be discussed later.

The other two attributes that need to be set are the **payload** and **env**. They are explained in one of the sections below. The remaining attributes of the job are less relevant for initial testing so we’ll skip them for now.

N.B. *The script referenced in the “payload” attribute must be readable and executable by members of the same computing group as the pilot (in our case that’s “np-comp”) and if it’s located in AFS the relevant permissions come on top of that. The “public” directory in your AFS-based directory tree is a good choice to place your own scripts.*

The payload

The **payload** attribute of the job definition (such as in the example above) is the path of the script that will run. It is strongly recommended that this is a shell wrapper, and the *bash* shell is most commonly used. If the **env** attribute contains “P3S_MODE”: “COPY” then the script will be copied into a sandbox *by the pilot* at execution time. Otherwise an attempt will be made to execute it *in situ* which may or may not work depending on the permissions (including both AFS permissions if that’s what you are using, and also Linux flags which should be at least +rx).

The environment

The **env** attribute in the JSON snippet above defines the job environment in the Linux sense. It can be used to describe the complete environment required by your job so you can keep the configuration in one place. The author of the job has complete freedom in how the environment is used. *In particular, it can be used as a clean way to communicate to the running job the names of input, output and configuration files, path to directories etc.* This is typically implemented in the wrapper script itself. For example, in the dummy example above (within the “my_executable.sh” payload script) we may find:

```
foo -i $MYFILE
```

In this example, the binary executable *foo* will read input data from the file whose name is stored in the environment variable *MYFILE*. The name of this environment variable does not matter as long as it is consistent with what’s in the JSON file such as shown above.

Please note that the user has complete freedom as to how to factor the information between the **env** attribute JSON file and the script. There are few limits in designing job descriptions. For example, the “P3S_MODE” will need to be set to “COPY” in most cases, and the variable “P3S_INPUT_FILE” has a special meaning if you decided to use it (although you don’t have to) - if there is a corresponding variable set in your shell it will override the content of P3S_INPUT_FILE found in the JSON file. This is done to facilitate testing with a number of different input files without having to edit JSON each time. If you don’t want this behavior, simply unset the variable in your interactive shell. Note that you will achieve same effect (override of the input file name) by using the “-f” option of the job.py client if you so desire. So if there is “P3S_INPUT_FILE”:“foo” in the “env” attribute in JSON, and you run the job.py client with an additional option:

```
-f foobar
```

then at runtime your payload job will see the value “foobar” in the content of the P3S_INPUT_FILE variable instead of the original “foo”.

Running your first job

We will use a prefab example for your first run. Inspect the **p3s/inputs/jobs** directory of the repo that you cloned from GitHub. Find and examine the file **printEnv.json**.

```
[
  {
    "name": "printEnv",
```

```

        "timeout":    "100",
        "jobtype":    "printEnv",
        "payload":    "/bin/env",
        "priority":    "1",
        "state":       "defined",
        "env":         {
            "P3S_COMMENT": "we are not using a wrapper since we are just running the stand
        }
    }
]

```

Two things to note here:

- the payload is just the standard built-in “env” command which will print the environment to stdout. We are not using a wrapper in this example for simplicity’s sake.
- for same reason, we are not using the P3S_MODE environment variable as the executable does not need to be copied into a sandbox

Now we can submit this job to the server. Assuming the p3s client software is installed, and **we changed the current path to the “p3s/clients” directory**, let’s make sure we set the correct environment, so unless already done so use the command

```
source ../configuration/lxvm_np04dqm.sh
```

After that the following command can be used (once again from the “clients” directory)

```
./job.py -j ../inputs/jobs/printEnv.json
```

And that’s it. The path to the JSON file used with this client can be anything, it just has to be readable for your user identity. When looking at the monitoring pages of p3s this job will be marked with your userID on the system from which you submitted it e.g. if you work on lxplus this will be your lxplus userID. If you are at CERN, the URL where you can check your jobs will be:

```
http://p3s-web.cern.ch/monitor/jobs
```

Entries in the jobs table are clickable and will reveal more information if you access the link. Note that while in this example the execution time is negligible it may take a while for the state of the job in the monitor to be displayed as “finished” due to the way polling period and hearbeats are set in the pilot. Pay attention to the UUID of the job since this will allow you to locate the stdout and stderr of the job when it finishes, which will be contained in the directory

```
/eos/experiment/neutplatform/protodune/np04tier0/p3s/joblog
```

In this test case, the *.out* file will contain a printout of the environment found on the worker node. The *.err* file should be empty if everything worked well.

Since running a functioning wrapper script (as opposed to Linux built-in commands) is a more realistic use case note that it's entirely up to the author of the wrapper script to define the location of the output files other than stdout. Please see the "p3s/inputs" directory for examples of job descriptions which access different directories.

The **job** client script we use here and all clients in p3s suite support the "-h" command line option which prints an annotated list of all command line options. Take a look, it's helpful. If you need more verbose output, you can add "-v 2" to the command line (in most clients) to change the verbosity level.

Creating and editing your wrapper scripts

It is important to ensure that the wrapper you are testing or using is located in a storage area that's accessible to the members of the np-comp group (see note on group attribution in the beginning of this document). When working in AFS, this also requires that you use a directory under your "public" directory branch otherwise the file won't be readable by p3s regardless.

As a simple exercise, you may want to try the following:

- create and rename a copy of the JSON file used in the previous example
- edit the JSON file by replacing the reference to /bin/env with something like /afs/cern.ch/user/f/foo/public/myTest.sh
- put something simple in the bash script myTest.sh located in your public folder, like /bin/env or /bin/date
- submit this job to p3s by specifying the path to your modified JSON file on the command line for the "job.py" client
- watch the job execute and look for the output in the "joblog" directory as stated above

In case something is amiss please contact the author of this document. If all worked well, you may try to create a more realistic and complex job description. Your wrapper script is pretty free form so is you need to set up an environment or fetch a piece of input from somewhere, please do so. Look at the examples in the "inputs" directory.

"Test Wrapper"

There is a script which allows the user to test the setup of the JSON file and the payload working together by running everything on an interactive node such as lxplus as opposed to submitting it to lxbatch, which is often more convenient for basic debugging. Similar to the "job" client presented above, the test wrapper can be invoked from the p3s/clients directory as follows (assuming you have created a file "myOwnExample.json"):

```
./testwrapper.py -j myOwnExample.json
```

which is quite similar to normal submission. Like in the previous case, the “-h” option will output helpful information. For example:

- -p option allows to override the path of the payload script i.e. in principle you can run anything (i.e. any executable) using the same skeleton template
- -f and -F respectively overwrite the path defined by the P3S_INPUT_FILE and P3S_OUTPUT_FILE

Note on environment variables

With rare exceptions such as *P3S_MODE* and **P3S_INPUT_FILE* (which may change in future versions) there is no semantic importance to the exact names of the environment variables used in formulating your job. The only thing that matters is that the payload script contained correct references to the environment.

Location of the data and log files

Directories in EOS

For storage of data and log files p3s can use *any* storage area (e.g. a directory) which is read/write accessible to its pilots. This means the user id of the pilots running in p3s must be a member of requisite groups and/or have correct permission for the directory used for log and data storage in p3s. At CERN we are taking advantage of the distributed **EOS** storage facility that is accessible from both interactive and worker nodes. The top level directory for all sorts of p3s data has been defined as

```
/eos/experiment/neutplatform/protodune/np04tier0/p3s
```

Log files

The p3s configuration file mentioned above (such as *lxvm_np04dqm.sh*) is not used just by client scripts, but also by the server. As such, it contains other useful info such as pointers to directories to keep log files for HTCondor submission of pilot jobs, pilot logs and job logs. The latter will be of most interest for the end user. Consider the following example:

```
[np04dqm@lxplus056 src]$ env | grep P3
P3S_PILOTLOG=/eos/experiment/neutplatform/protodune/np04tier0/p3s/pilotlog
P3S_JOBLOG=/eos/experiment/neutplatform/protodune/np04tier0/p3s/joblog
```

```

P3S_SERVER=http://p3s-web:80/
P3S_CONDOR_ERROR=/afs/cern.ch/work/n/np04dqm/condor
P3S_PILOTS=100
P3S_CONDOR_LOG=/afs/cern.ch/work/n/np04dqm/condor
P3S_PILOT_MAXRUNTIME=90000
P3S_PILOT_T0=1000
P3S_VERBOSITY=0
P3S_SITE=lxvm
P3S_CONDOR_BASE=/afs/cern.ch/work/n/np04dqm/condor
P3S_DIRPATH=/eos/experiment/neutplatform/protodune/np04tier0/p3s
P3S_CONDOR_OUTPUT=/afs/cern.ch/work/n/np04dqm/condor

```

We are taking advantage of availability of distributed file systems at CERN with are visible from most nodes, i.e. all logs are kept in a single directory structure which can be browsed by the user (as opposed to local disks on a few dedicated nodes). The reason HTCondor logs are kept in AFS as opposed to EOS is a current CERN policy.

Moving on to “real” payloads

Software provisioning and the production account

At the time of writing the duneTPC software is provisioned to the CERN interactive and worker nodes via CVMFS for “most” releases, and is also available via AFS for a small number of select recent releases.

The protoDUNE-SP production account **np04dqm** is used to host standard job definitions etc, under the path `p3s/inputs/larsoft`. JSON files used to define LArSoft jobs are fairly standard in that they typically specify location of the FCL file, version of duneTPC etc.

For a p3s client to run properly in needs to have the correct Python environment. “Standard” job definitions have already the necessary commands baked in, so you don’t have to do anything. For example, you can immediately submit a prefab LArSoft test job (which simply dumps hits obtained from a prestaged ROOT file) by running the command:

```
./job.py -v 2 -j /afs/cern.ch/user/n/np04dqm/public/p3s/p3s/inputs/larsoft/test/hitdumpstest
```

In the command line above, “-v” stands for the verbosity level (which is 2 in this case) and the “-j” option allows the user to supply a job description in the JSON format, stored in a previously prepared JSON file.

If all works well, you will see a UUID of the job just submitted printed on screen in the shell you are running, and it will also be displayed in the p3s monitor at

p3s-web.cern.ch, the “jobs” section. Upon completion (e.g. when you refresh the monitor page and the state of the job switches from “running” to “finished”) you may examine the output directory

```
ls -ltr /eos/experiment/neutplatform/protodune/np04tier0/p3s/testoutput
```

You will see a number of files, and one of the latest will contain the UUID of your job in its name. The content of the file should be a list of the hits read from the input.

You can now copy `hitdumptest.json`, `hitdumptest.sh` and requisite FCL files to your public AFS directory (so they can be accessed by p3s) and start doing useful work of your own. Congratulations!

If you run a custom wrapper script which is not immediately derived from production and testing sample, you may need to set up the required Python environment by running the command:

```
source ~np04dqm/public/vp3s/bin/activate
```

Remote access to data

If you want to pull data produced by p3s to your local machine (including your laptop) it’s pretty easy provided that you have installed XRootD on that target machine. For example, to obtain a directory listing run a command similar to the following:

```
xrdls root://eospublic.cern.ch/ ls -ltr /eos/experiment/neutplatform/protodune/np04tier0/p3s/
```

To copy a file from CERN EOS to your machine use a command similar to this one:

```
xrdcp xroot://eospublic.cern.ch//eos/experiment/neutplatform/protodune/np04tier0/p3s/output/
```

Appendix

The mechanics of the job submission

Keep in mind that when you “submit a job” all you are doing is sending a record containing all the info necessary for running a particular executable, to the p3s database. The system (p3s) will then match this job with a live and available **pilot** occupying a batch slot in CERN Tier-0 facility and deploy the payload for execution in that batch slot (i.e. on one of the Worker Nodes at CERN). The pilot will monitor the state of the job under its management and send periodic “heartbeats” to the server to tell it that it’s still alive. Once the job completes, the pilot closes logs, optionally performs other “close out” functions and resumes querying the p3s server for the next job.

This is a typical case of a pilot-based framework with the following features:

- low latency between submission and the start of job execution since you are not waiting for a HTConfor or other queue; in some cases such as busy HT Condor queues speed gains can be quite substantial
- you don't have to run batch commands yourself and you don't have to manage HTCOnfor JDL files
- ease of automation (e.g. creating automatic scripts) since same tested job template can be used in automated submission
- easy to read tabulated view of all of your jobs in the p3s monitor which is a Web application
- the identity under which jobs are executed is not your identity but the production identity... This can be helpful or not helpful depending on situation, and path permissions need to be thought through. The NP04 group at CERN has access to various EOS and some AFS directories so this can be made to work