QCPump

Release 0.1

Randle Taylor

Aug 01, 2022

CONTENTS:

CHAPTER

ONE

RELEASE NOTES

1.1 v0.3.14

- Fix issue with making test level comments optional on MPC pumps

1.2 v0.3.13

- Made comment uploaded to QATrack+ optional (defaults to off because the comment prevents auto review)

1.3 v0.3.12

- Fixed bug related to logging of certificate paths
- Added a "Fast Search" option to MPC pumps. This option will restrict search for Results.csv files to subdirectories called MPCChecks. (defaults to on)
- Adjusted auth headers to make it possible for QCPump to talk to RadMachine
- For FFF beams, added Beam Shape Constancy results from the DQA3_TREND table for DQA3 v1.06 SQL Server DBs. There are 9 new results sent to the server:

  ```
  bsc{1,2,3}_{beam}
  bsc{1,2,3}_baseline_{beam}
  bsc{1,2,3}_diff_{beam}

  e.g.

  bsc2_6_fff
  bsc2_baseline_6_fff
  bsc_diff_6_fff
  ```

  It is believed that bsc2 is the value shown in the DQA3 software so you should configure a test in QATrack+/RadMachine with slugs like bsc2_6_fff or bsc2_10_fff
- Partial work around for a suspected race condition occuring in the DQA3 database software. The race condition causes an incorrect data_key to be written to the dqa3_trend table, and hence data was being sent to the wrong unit. The workaround implmented here results in the dosimetry data being sent to the correct unit, but the temperature, pressure, comment, signature, and device serial number will still be incorrect if this occurs again. This is the result of a defect in the DQA3 software and should be exceedingly rare.

1.4 v0.3.11

- Fixed bug with base64 encoding of files for QATrackGenericBinaryFileUploader pump
- MPC pumps should now handle 2.5X and HDTSE beams
- MPC pumps should handle directory names like NDS-WKS-SN1234-2015-01-01-00-00-00-0001-10x-Beam
- QATrack+ validation requests receiving a 307 Temporary Redirect response will retry their request. This is an attempt (possibly in vain) to work arround network monitoring software which may temporarily return 307s.

1.5 v0.3.10

- Added DQA SQL Server database pumps

1.6 v0.3.9

- Fix issue with missing Unit names for MPC
- Fix issue with "Â" appearing in some files
- Fixed broken link to QATrack docs

1.7 v0.3.8

- Allow disabling certificate patching by putting a file called nopatch.txt in C:ProgramDataQATrack Projectqcpump
- fix to retain unit choices when connection to QATrack+ fails for DQA3 pumps
- Attempt to fix SSL certificate errors in some networks.

1.8 v0.3.6

- Resolves an issue with unknown null urls being cached leading to pumps needing to be restarted in order to "re-discover" the URL to upload data to.

1.9 v0.3.5

- Ensure ssl certificate files are found / installed by pyinstaller
- Switch to using site_config_dir instead of user_config_dir for storing QCPump configs and logs so that multiple users can use same configuration. It is recommended that you "Install for All Users" when installing QCPump.
- Instructions for starting QCPump automatically have been added.

1.10 v0.3.4

- retracted

1.11 v0.3.3

- Fix stdio redirect
- Fix logging window history
- Add more json decoding errors

1.12 v0.3.2

- Handle non-missing test 400 Bad Requests when uploading to QATrack+

1.13 v0.3.1

- The File Mover pump types can now be set to *Copy* mode to have source files copied to the destination directory rather than moved
- The Python package *python-certifi-win32* has been added so that requests can use the Windows Certificate Store for SSL verification rather than using its bundled certificate chain. This should resolve some issues users were having with firewalls & network monitoring software.
- Added a missing permission for the DQA3 QCPump Firebird user

1.14 v0.3.0

1.14.1 General Features & Fixes

- A bug has been fixed where only the last subsection of a *Multiple* configuration section was being validated.
- Pumps which are marked as *Inactive* will not run validation code until they are re-activated. This eliminiates un-necessary network calls and other validation work which, in addition to being more efficient overall, makes debugging QCPump itself simpler when multiple pumps are configured.
- New MPC pump type for uploading MPC results. See *Varian MPC Pumps*.
- New generic pump types for uploading Text & Binary Files to QATrack+ have been implemented. See *QATrack+ Generic File Uploads*.
- A *DISPLAY_NAME* attribute has been added to Pump Types to aid with grouping together similar pump types when adding new pumps.
- Warning level debug messages were being logged as errors. This has been fixed.
- A new *PUMP_ON_STARTUP* (see *QCPump Settings*) setting has been added to allow pumping to begin immediately after QCPump is launched. This allows you to place QCPump in a startup folder and have it launched & start pumping when your computer is restarted.

1.14.2 DQA3 Pump Type Changes

- The *DATEADD* for calculating a *work_completed* value in Firebird DQA3 queries has been eliminated in order to allow the query to work with Firebird versions < 2.1. *work_completed* is now just calculated in Python code instead.
- The template for looking up Test Lists for beams now defaults to:

```
Daily QA3 Results: {{ beam_name }}
```

  where *beam_name* is is the DQA3 test name (e.g. '6MeV', '6MV WDG', '6MV EDW 60 Weekly', '20 MeV DQA3 Daily'). This allows QCPump to handle a wider variety of beam types/configurations.
- More context variables are available when generating your test list name. In most cases you should only need to use *beam_name*, however other variables are available should you need them. See the *DQA3 Test List Name docs*.
- New *Multiple Beam Per Test List* DQA3 pumps have been added which will group results from multiple measurements together based on the results being recorded in a short window of time. There are two disadvantages to using the Multiple Beams Per Test List:
  1. If you have many beams configured this will result in long test lists which can impact performance when uploading data, or reviewing data in QATrack+.
  2. If you perform a measurement twice (e.g. take 2 6X measurements), only the 2nd result will be included.
- QATrack+ Unit names will now be displayed along with their Site in order to disambiguate units with the same name
- DQA3 machine names will now be shown with their Room name to disambiguate machines using the same tree names.

CHAPTER

TWO

INSTALLING QCPUMP

2.1 Installing with the Windows Installer

On Windows platforms please download the QCPump Installer. Download and run the installer following the prompts. You may choose to install for the current user only, or all users on this system.

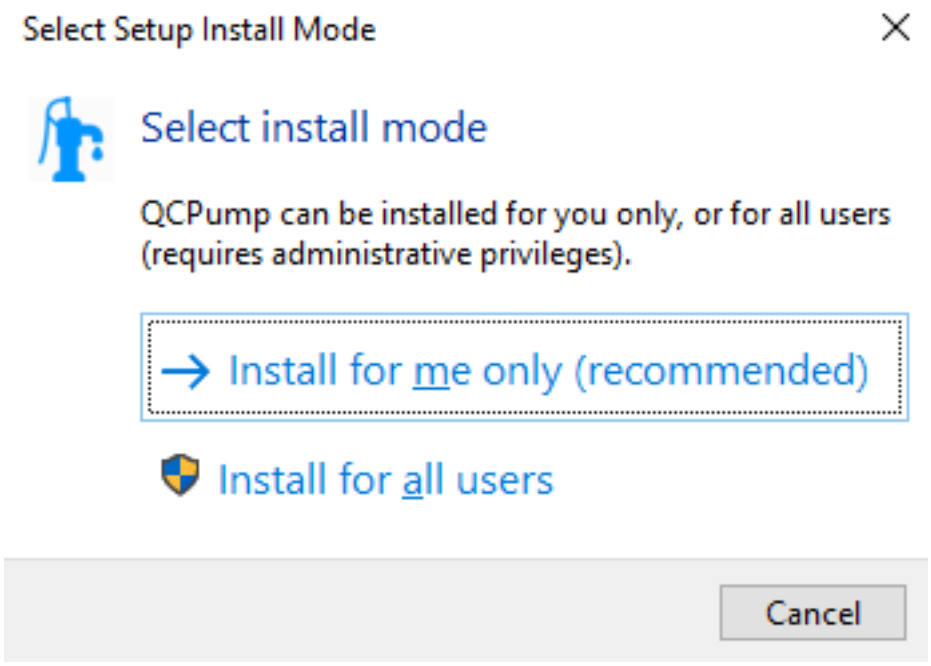


Fig. 1: QCPump: Choose whether to install for all users or just the current user.

Installing for All Users requires administrator privileges while installing for only the current user should be possible regardless of user type. Please note that either way you install, the pump configuration will be shared across all users regardless of whether you install for the current user or all users.

2.2 After Installing

Now that QCPump is installed, go to the *Configuring New Pumps* page to start configuring some *Pumps*. After that you may want to see the following section on starting QCPump automatically.

2.3 Starting QCPump Automatically

If you want want to have QCPump start up automatically when a user logins you can do so by adding QCPump your startup folder.

First create a shortcut to QCPump on the desktop by right clicking on the desktop and selecting "New -> Shortcut":

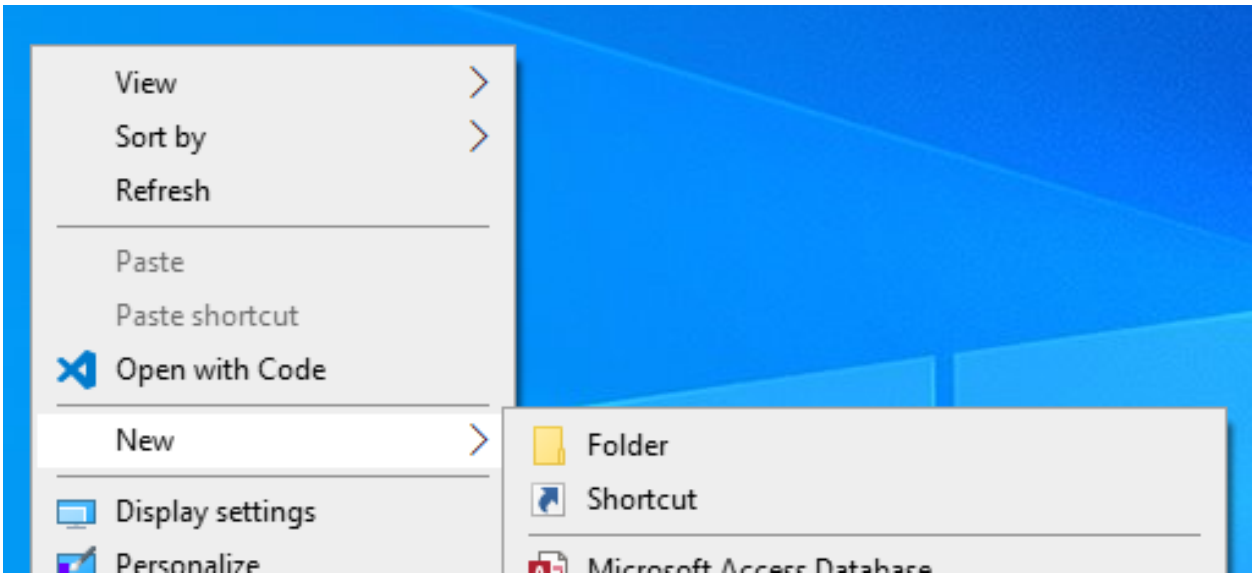


Fig. 2: QCPump: Create a desktop shortcut for QCPump

and select the QCPump application (usually located at *C:\Users\YOURUSERNAME\AppData\Local\Programs\QCPump\qcpump.exe* if you installed for a single user, or *C:\Program Files (x86)\QCPump\qcpump.exe* if you installed for all users).

and then finish the dialog.

Next we need to move the shortcut to the appropriate startup folder.

To startup for just the current user open the Run dialog (Win Key + R), type *shell:startup* and click OK.

Or to startup for all users open the Run dialog (Win Key + R), type *shell:common startup* and click OK.

Now drag and drop the desktop link into the startup folder:

Finally launch QCPump and ensure that "Run Pumps On Launch" is checked.

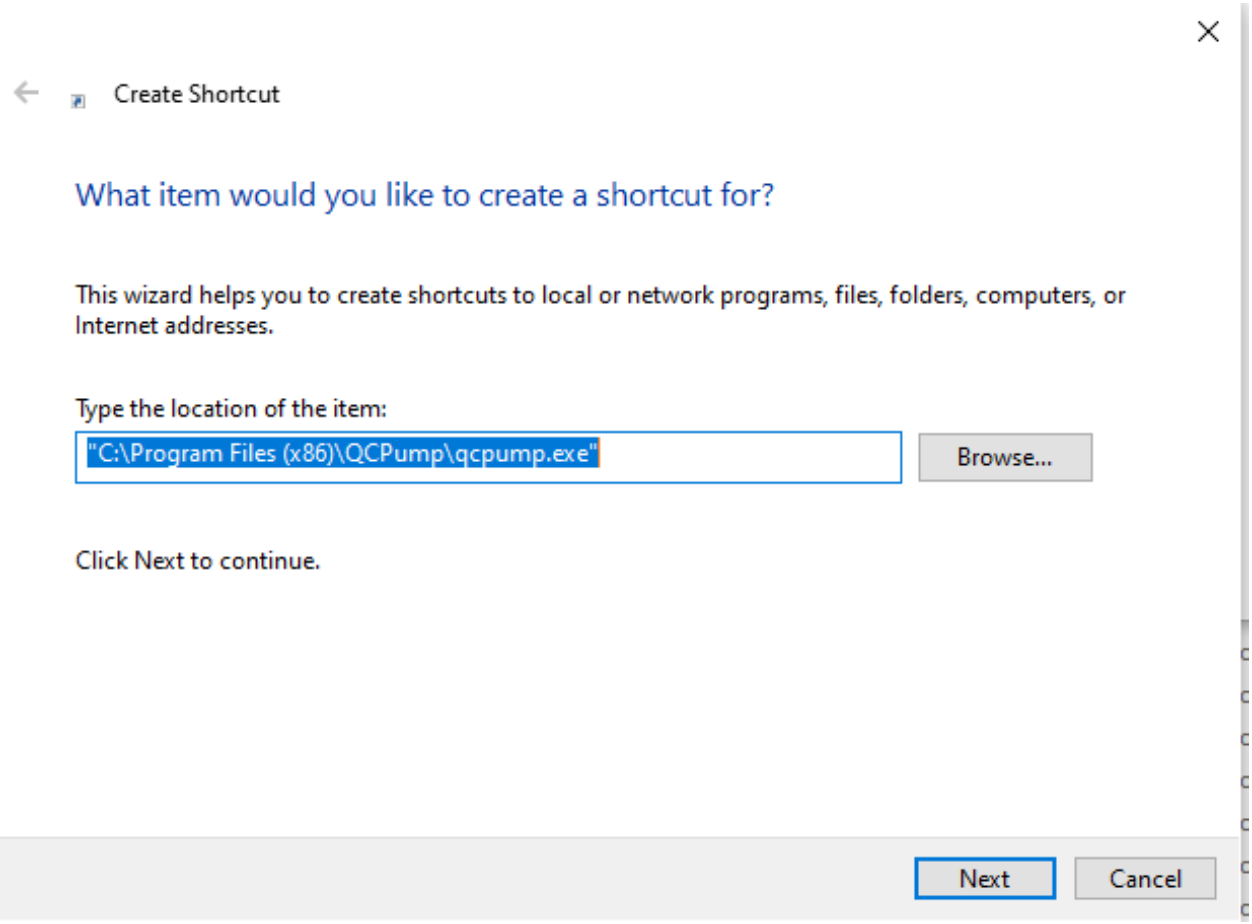


Fig. 3: Select the QCPump application to link to

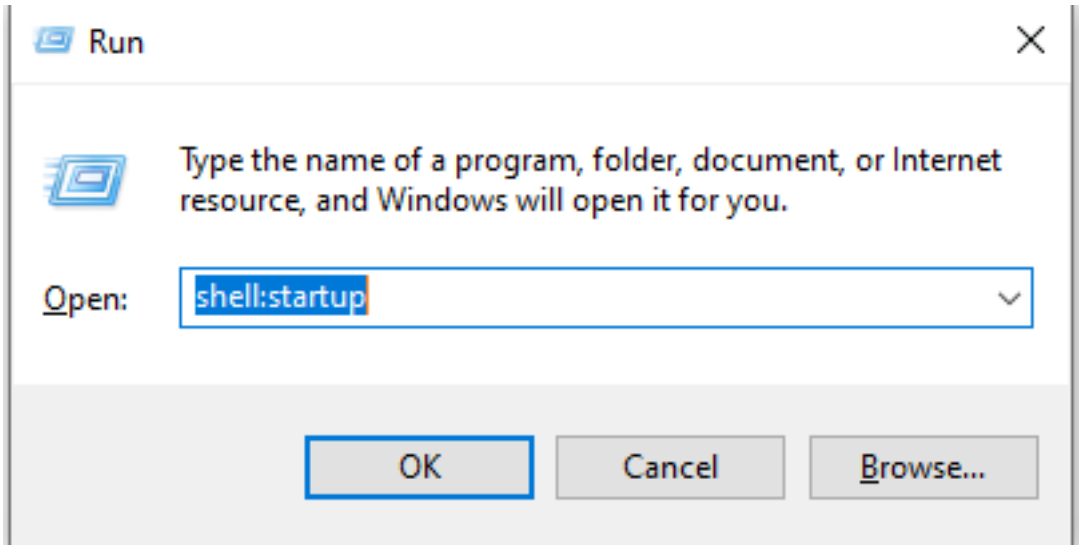


Fig. 4: Open Startup folder

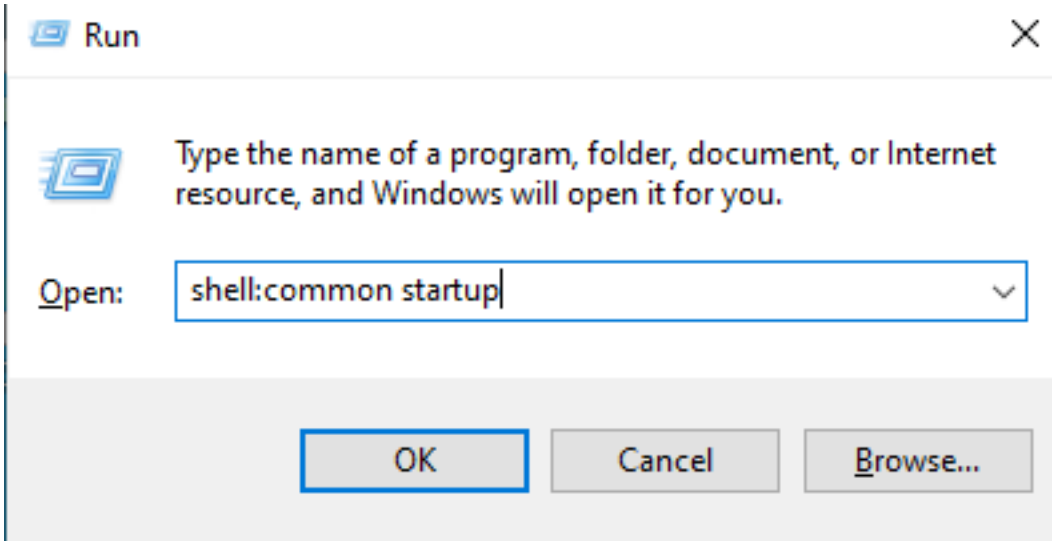


Fig. 5: Open Common Startup folder

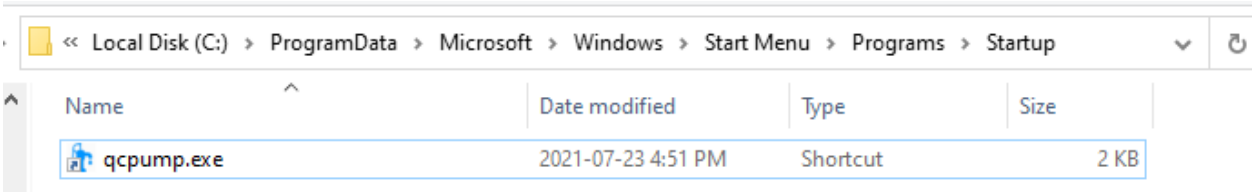


Fig. 6: Startup Shortcut

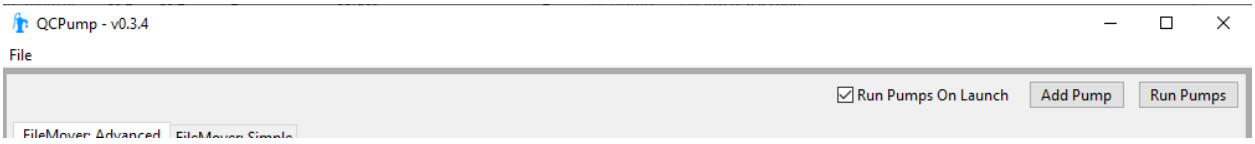


Fig. 7: Tell QCPump to automatically start running pumps on launch

2.4 Obtaining and running from source

Note: You will need both Python (version 3.7-3.9) and Git installed on your computer to run QCPump from source.

In order to obtain the source code for QCPump install Git and then clone the QCPump repository:

```
git clone https://github.com/qatrackplus/qcpump.git
```

If you are on Windows, visit https://www.lfd.uci.edu/~gohlke/pythonlibs/ and download DukPy for your particular version of Python. Now create a new venv to install the QCPump requirements:

```
cd qcpump

# create your venv
python -m venv env

# activate venv on Windows and install requirements
env\Scripts\Activate.ps1
pip install C:\path\to\dukpy-0.2.3-cp39-cp39-win_amd64.whl
pip install -r requirements\base.txt

# activate venv on *nix
source env/bin/activate
# replace 18.04 with your Ubuntu version
pip install -U -f https://extras.wxpython.org/wxPython4/extras/linux/gtk3/ubuntu-18.
↪04 wxPython
pip install -r requirements/base.txt
```

and then to run the program:

```
python launch_qcpump.py
```

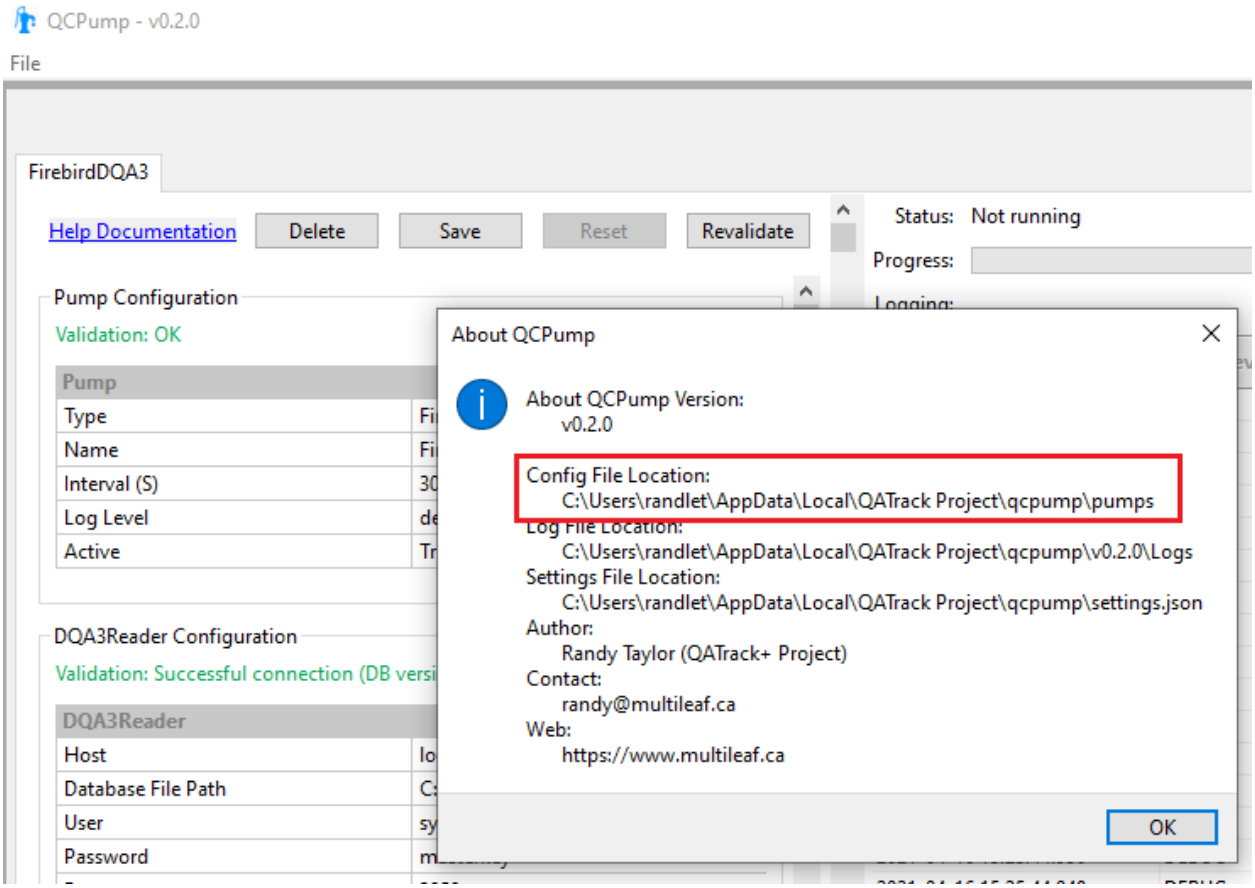
Now you can proceed to the *Configuring New Pumps* page to start configuring some *Pumps*.

CHAPTER

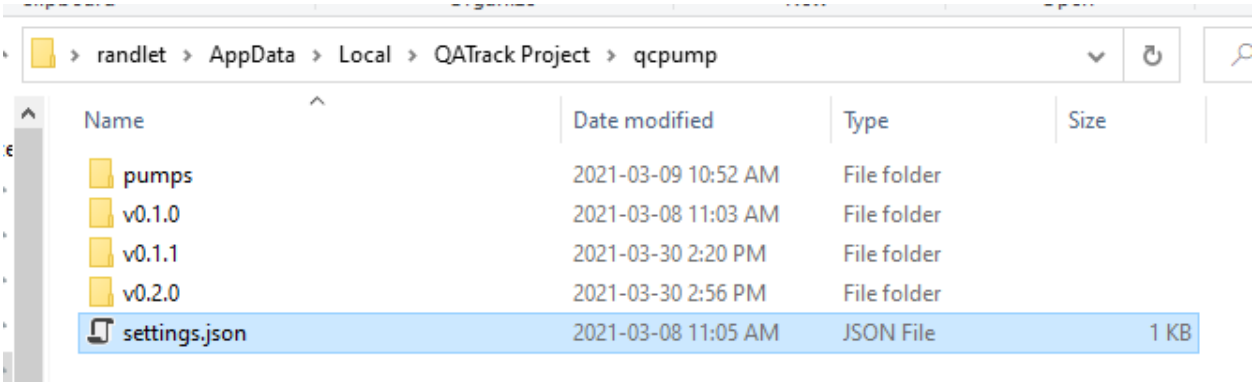
THREE

QCPUMP SETTINGS

You can find the location of your QCPump settings file location file by launching QCPump and going to File->About:



You can edit the settings.json file with any text editor:



3.1 Available Settings

You may add one or more of the following settings to the settings.json file to override the default values:

BROWSER_USER_AGENT (string) The User-Agent to use when making outgoing web requests. (Default "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.19582")

DB_CONNECT_TIMEOUT (integer) Timeout in seconds for database connections where available. (Default 30)

DEBUG (*true*, *false*) Currently only used to redirect std input / output. Must be set to *true* if you want to use an interactive debugger while developing QCPump. (Default: *false*)

LOG_LEVEL (*debug, info, warning, error, critical*) Choose the QCPump application logging level (individual pumps are not affected by this). One of. (Default: *info*)

LOG_TO_CONSOLE (*true*, *false*): Should logs be written to console as well as log files?

PUMP_DIRECTORIES (list of file paths or null) Set to list of other directories to include user defined pump types from. See *Pump Type Development*.

PUMP_ON_STARTUP (*true*, *false*) Should QCPump immediately start pumping when it is launched. This is useful for e.g. adding QCPump to a startup folder so it launches when a machine is rebooted and starts pumping immediately. (Default *false*)

CHAPTER

FOUR

OPERATING QCPUMP

4.1 Running your Pumps

After you've configured and *saved at least one Pump* you can run all active pumps by clicking the *Run Pumps* toggle button at the top of the QCPump window.

4.1.1 Running pumps automatically on launch

If you want your Pumps to start pumping automatically when they launch, set the *PUMP_ON_STARTUP* to *True*. This allows you to have QCPump launch automatically at Windows Startup and start pumping without user intervention.

4.2 Stopping your Pumps

At any time you may stop the current pumps by clicking the *Stop Pumps* toggle button at the top of the QCPump window.

4.3 Log Files, Config Files, and Settings Files

In order to see where QCPump is storing your log files, config files and QCPump setting files use the *About* menu option in the *File* menu:

Each *Pump* you configure will have its own configuration directory which contains a *config.json* JSON document that represents the current configuration of your *Pump*.

Each pump will have its own log file placed in the main QCPump log directory. These log files can provide important information when trying to debug any issues you are having with QCPump.

4.3.1 QCPump Settings

QCPump has a limited set of settings you can configure by editing the settings.json JSON document found in the QCPump config directory. Settings available are:

DEBUG Useful for developers only. Set to *"DEBUG": true* to prevent redirecting stderr/stdout to file.

LOG_LEVEL Controls the logging level for the QCPump application itself (not the pumps which have their own log level settings). Set to *"LOG_LEVEL": "debug"* to get more detailed log info. Other options include *info, warning, error, critical*.

DB_CONNECTION_TIMEOUT Timeout for database connections where available. Default value is 3s.

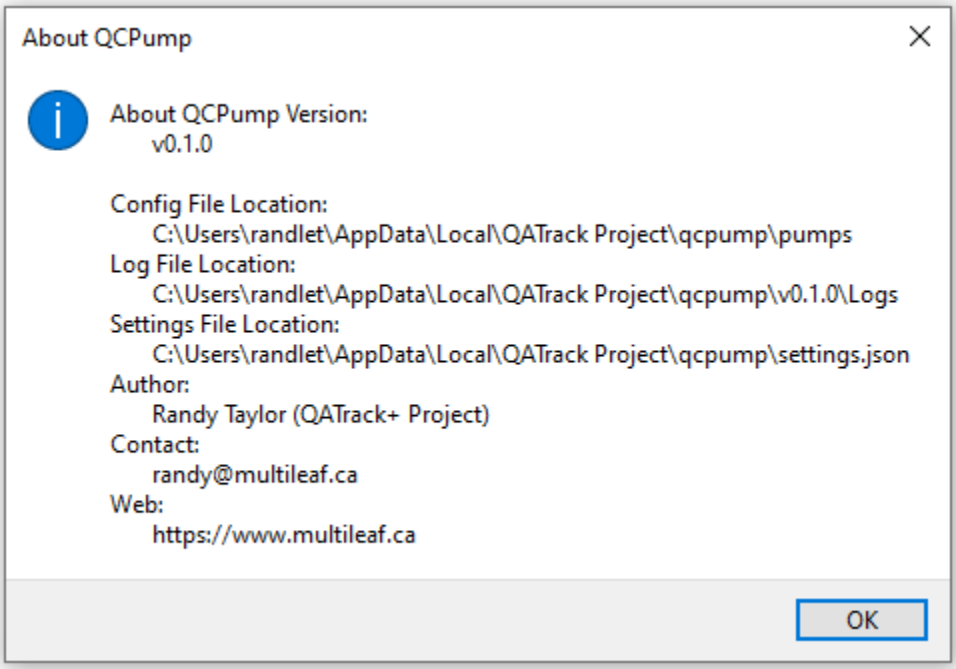


Fig. 1: QCPump About Page

PUMP_DIRECTORIES A list of directories to look for *custom Pump Types* you are using. This should be an empty list (*"PUMP_DIRECTORIES": []*) unless you are using custom *Pump Types* not included with QCPump.

A sample settings.json document might look like:

```
{
    "LOG_LEVEL": "info",
    "DEBUG": False,
    "PUMP_DIRECTORIES": ["C:/Users/yourusername/pumps/"],
    "DB_CONNECT_TIMEOUT": 3
}
```

CHAPTER

FIVE

QCPUMP BUILT IN PUMP TYPES

See below for documentation relevant to specific pump types.

5.1 Configuring New Pumps

When you first launch QCPump you will be met with a more or less blank slate.

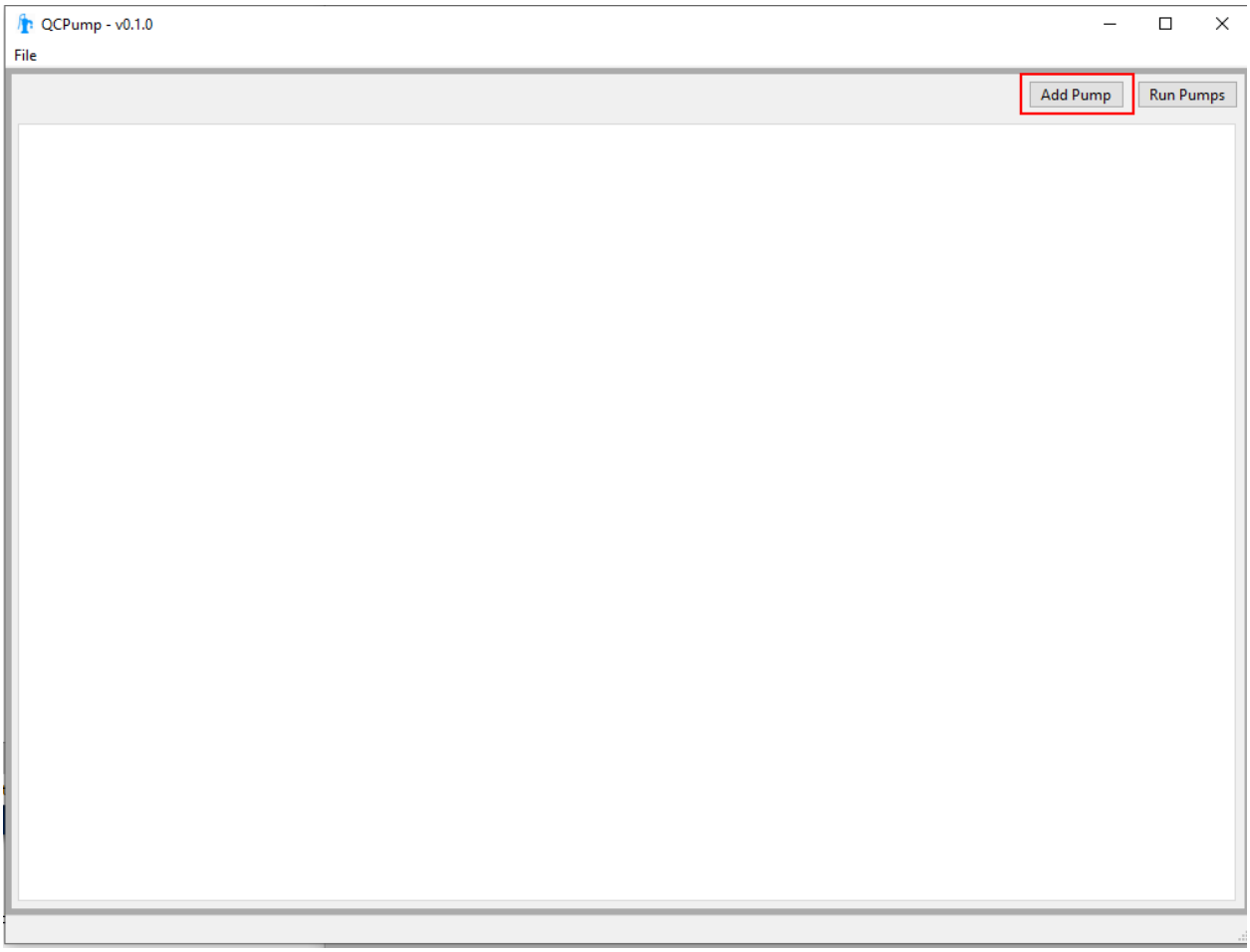


Fig. 1: QCPump with no pumps installed

Click the *Add Pump* button at the top right. Select the *Pump Type* you want to create, give the *Pump* a meaningful name and click *OK*.

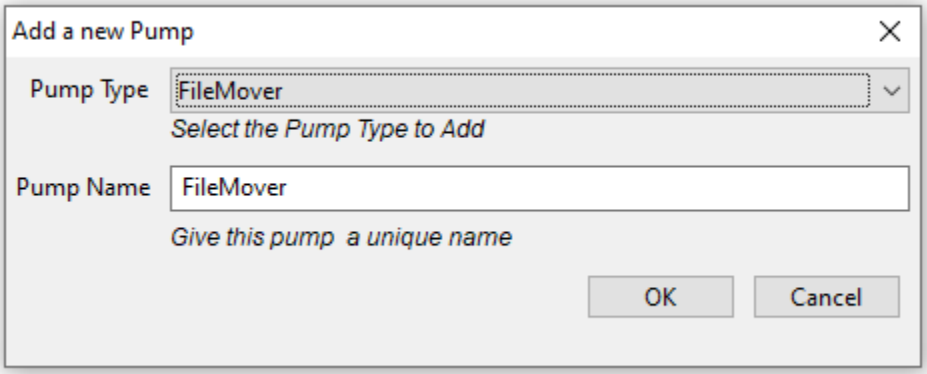


Fig. 2: QCPump with no pumps installed

After adding you click *OK* you will see a tab for your new *Pump* displayed:

On the left of the tab are the configuration options for the pump and on the right is a logging window which is used to display messages from your pump.

The configuration panel is separated into different configuration sections. All pumps have a common set of configuration options included under the *Pump Configuration* heading which control things like whether this pump is active and how often the pump should run.

5.1.1 Common Pump Configuration Options

Every *Pump* you configure has the following options available in the *Pump Configuration* section:

Type Displays the *Pump Type* of the current *Pump* (not editable)

Name Displays the name of the current *Pump* (not editable)

Interval (s) How often the pump should run in seconds.

Log Level A dropdown allowing you set the verbosity of logging for this Pump. Set to *debug* to get the maximum level of verbosity. Logging messages will be shown both in the status panel as well as written to log files. The location of the log files can be seen by accessing the *About* menu option in the *File* menu.

Active Select whether this pump will be activated when you click the *Run Pumps* button.

Pump Type specific options are covered in their own pages:

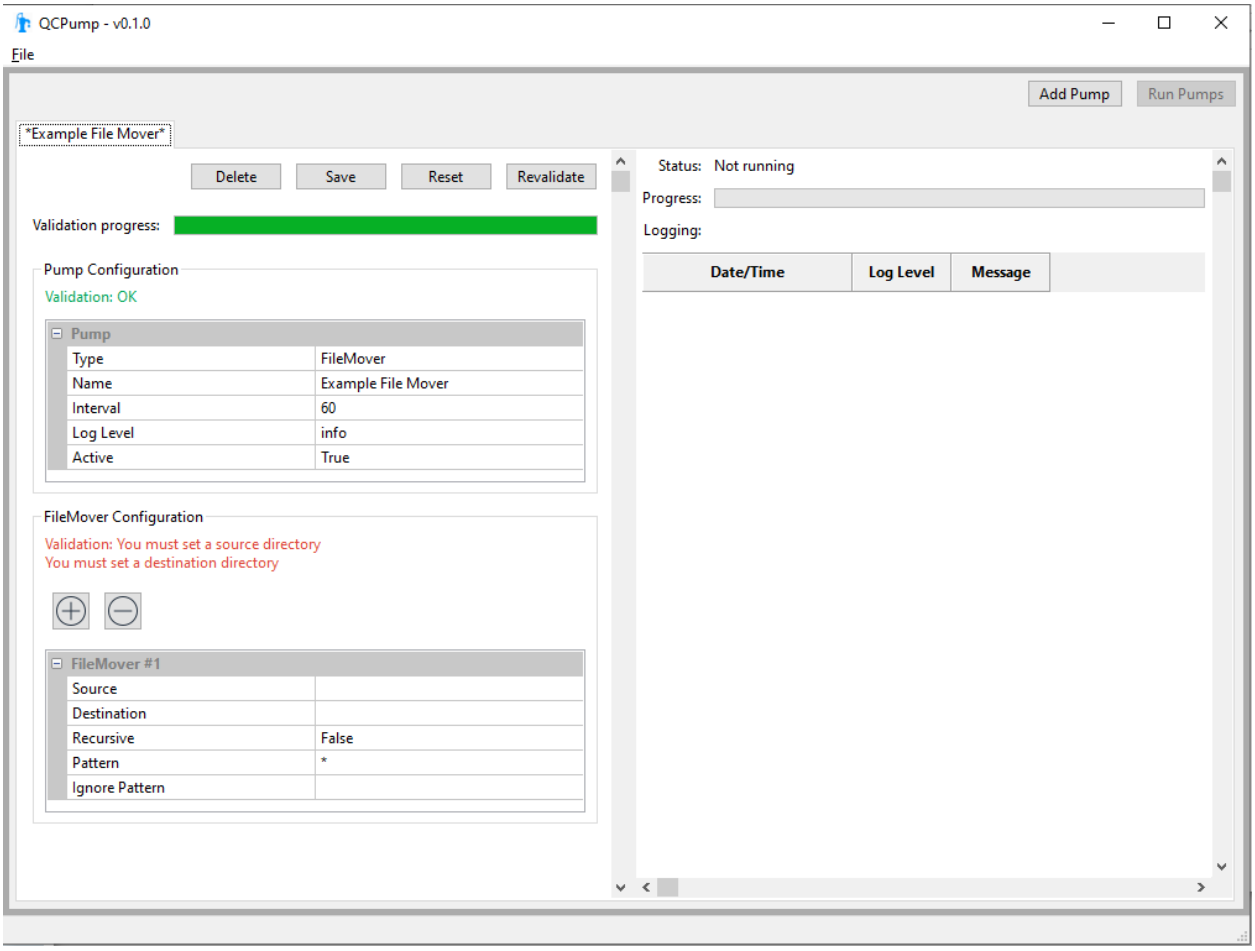


Fig. 3: A new pump added

FileMover Pumps

The two FileMover pumps are primarily meant to serve as simple examples of how to write your own pump types. There is both a *Simple File Mover* which simply moves all files from one directory to another, and a slightly more complex *File Mover* which allows you to filter based on file patterns, and iterate through subdirectories.

Simple File Mover

The *Simple File Mover Pump* is a very simple pump that simply moves files from one or more source directories to one or more destination directories periodically. It will not recurse through subdirectories

> **Warning:** Files in your Destination directory may be overwritten by new files if they have the same name!

Configuration options

Define one or more file movers using these options:

Source The directory to move files from

Destination: The directory to move files to

Mode: Choose whether files should be moved, or copied

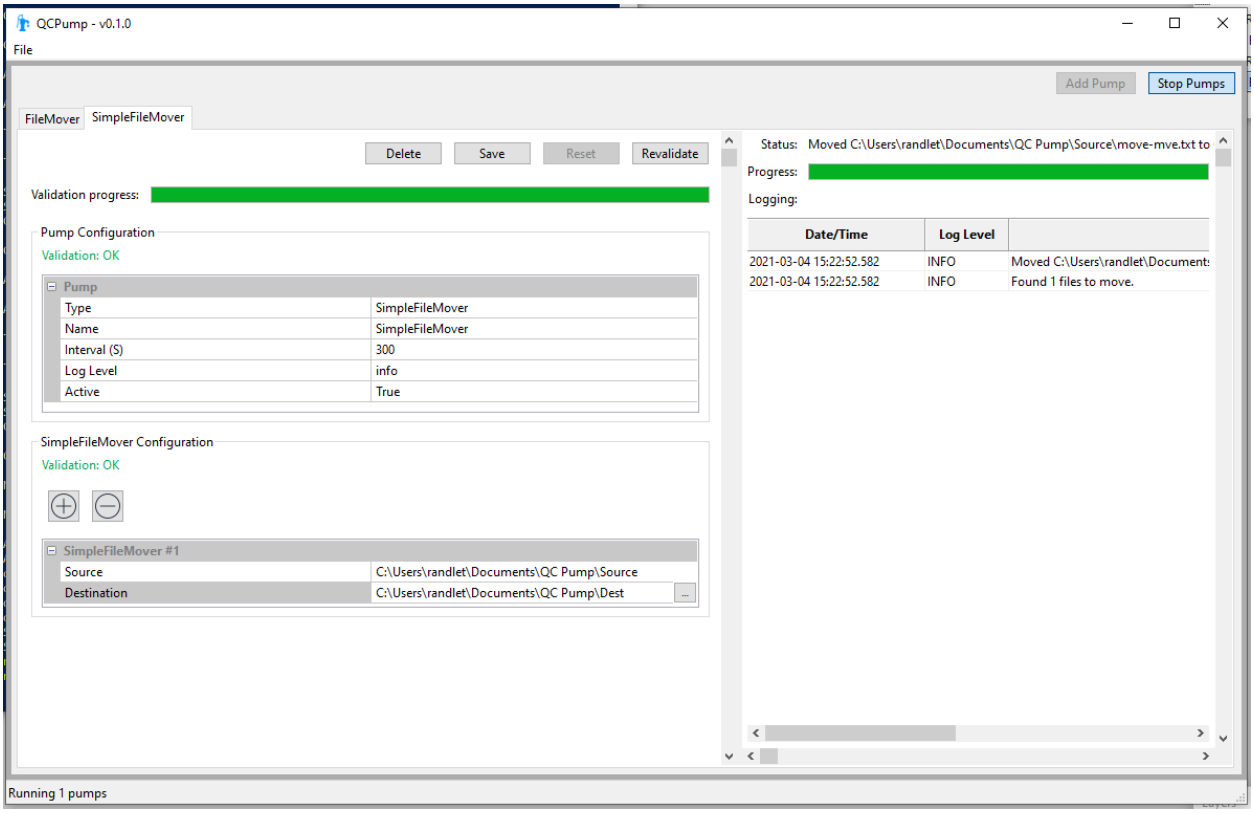


Fig. 4: QCPump Simple File Mover

File Mover

The *File Mover Pump* is a pump that moves files from one or more source directories to one or more destination directories periodically. This pump differs from the *Simple File Mover* by adding options allowing you to perform *glob* file matching and filtering, as well as optionally recursing through subdirectories.

> **Warning:** Files in your Destination directory may be overwritten by new files if they have the same name!

Configuration options

Define one or more file movers using these options:

Source The directory to move files from

Destination The directory to move files to

Mode: Choose whether files should be moved, or copied

Recursive Set to *True* to have the file mover look through all subdirectories of the *Source* directory.

Pattern Set to a Glob style pattern to specify what files to include. Only files with a matching filename will be moved.

Ignore Pattern Set to a Glob style pattern to specify what files to exclude from the matched files. The *Ignore Pattern* is applied after the matching by *Pattern* is done and therefor acts as a filter on matched files. For example to move all .dcm files *except* for those that start with *foo-* set *Pattern* = **.dcm* and *Ignore Pattern* = *foo-*.

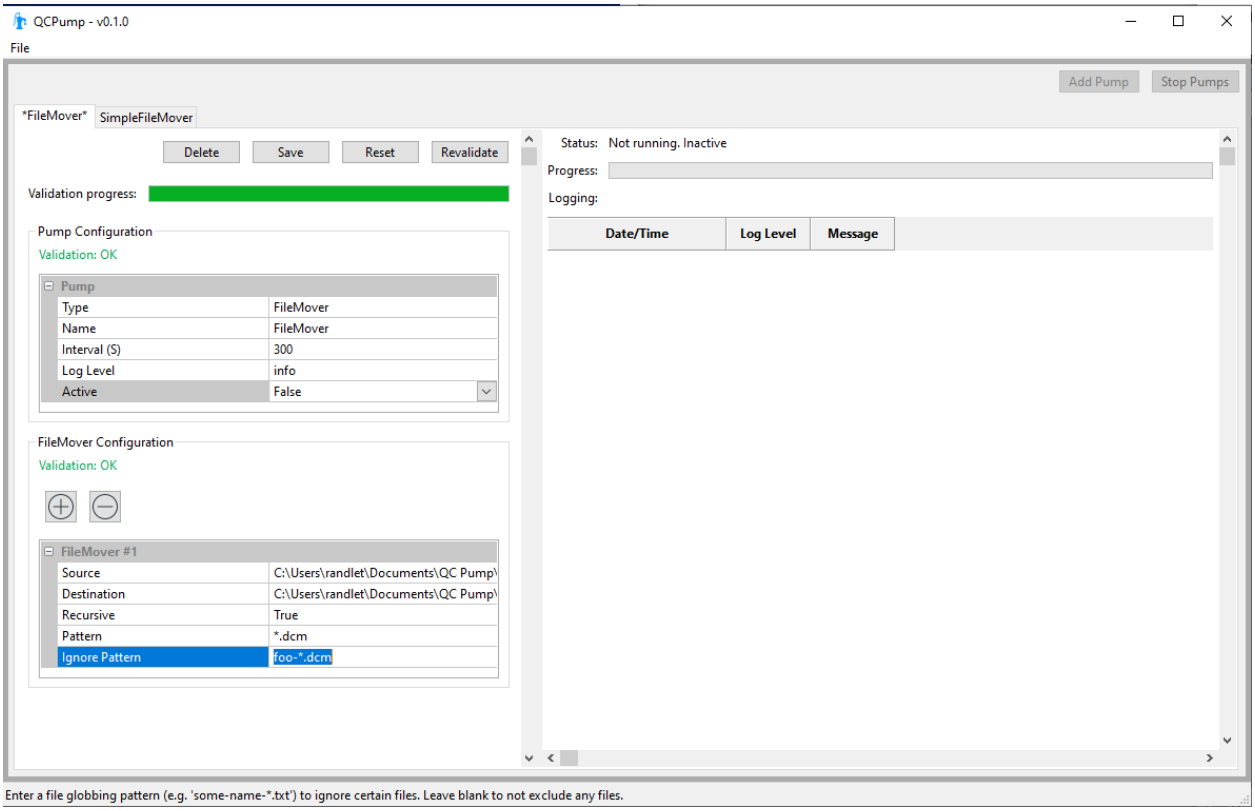


Fig. 5: QCPump File Mover

Daily QA3 Pumps: One Beam Per Test List

This page refers to Pumps which upload results to a different test list for each beam. For example, if you have the following 10 beams configured on a single unit: 6X, 6FFF, 10X, 10FFF, 6X EDW60, 6E, 9E, 12E, 16E, 20E, then you will need 10 different test lists assigned to your unit in QATrack+ to record all of your DQA3 results. If you prefer to combine all of your results in a single test list, please see: *Daily QA3 Pumps: Multiple Beams Per Test List*.

Note: There are two disadvantages to using the Multiple Beams Per Test List:

1. If you have many beams configured this will result in very long test lists which can impact performance when uploading data, or reviewing data in QATrack+. 2. If you perform a measurement twice (e.g. take 2 6X measurements), only the 2nd result will be included.

QCPump currently has the ability to retrieve data from the following Daily QA3 data sources:

- DQA3 Firebird Database version 01.03
- DQA3 Firebird Database version 01.04
- DQA3 SQL Server Database version 01.06
- DQA3 data from Atlas 1.5

Note: The DQA3 pumps are tested on QATrack+ v3.1.X QCPump is not compatible with QATrack v0.3.X

Contents

Configuring QATrack+ for DQA3 Data

In order to upload DQA3 data to QATrack+ you need to do a bit of setup work in QATrack+ first.

Create an API Token

In order to upload your data to QATrack+ via the API you will require an API token. See the QATrack+ documentation for how to create an API token. You may wish to create a dedicated user in QATrack+ just for use with QCPump. The user will only need a single permission in order to upload data: *qa* | *test list instance* | *Can add test list instance*.

Configure Test Lists

Note: In order to simplify the creation of these test lists, there is a script included with QATrack+ v3.1.0+ to generate either a Test Pack or to create a TestList directly in your database. To run the script activate your virtualenv, changed to the QATrack+ root directory and then run

```
# create a 6X test list in the db (replace 6X with your beam type)
python manage.py runscript create_dqa3_testlist --script-args db 6X

# or create a 9E test pack (replace 9E with your beam type)
python manage.py runscript create_dqa3_testlist --script-args testpack 9E
```

QCPump requires QATrack+ to have a Test List configured for each beam type you want to upload results for. For example, if on your linacs you use 6X, 6FFF, 10X, 10FFF, 15X, 6E, 9E, 12E, 16E, 20E beams you will need 10 total test lists for DQA3 results. The Test List must have a specific set of attributes:

Test List Name

The simplest method to have QCPump find the correct test list to upload data to is to give the test list a name which contains the name of the DQA3 Test (e.g. "6 MV", or "6X Daily", or "6 MV EDW60") By default QCPump uses a test list name like:

Daily QA3 Results: {{ beam_name }}

where QCPump will replace *{{ beam_name }}* with the name of the DQA3 test beam (e.g. "Daily QA3 Results: 6X") before searching QATrack+ for the test list to perform. You may also customize your test list name template with other variables which include:

beam_name *dqa3_template.tree_name* from a Firebird database (e.g. "6 MV", "6MeV", "6MV WDG", or *MachineTemplate.MachineTestName* from an Atlas database. e.g. ("6 MV DQA3 Daily", "6MV EDW60 Weekly", "20 MeV DQA3 Daily")

energy The beam energy: 6, 9, 10, 15, 18 etc

beam_type One of "Photon", "FFF", or "Electron"

wedge_type Empty for non wedge beams, otherwise "EDW" or "Static"

wedge_angle Empty for non wedge beams, otherwise 30, 45, 60 etc

wedge_orient *dqa3_template.wedgeorient* for FBD databases, or *MachineTemplate.WedgeOrient* for Atlas databases.

device Device serial number

machine_name *dqa3_machine.tree_name* for FBD databases, or *Machine.MachineName* for Atlas databases.

room_name *room.tree_name* for FBD databases, or *Machine.RoomNumber* for Atlas databases.

In order to record your data in QATrack+ you will need to add tests to your Test list with one or more of the following macro names:

data_key: String data_key is a key from the DQA3 database used by QCPump and QATrack+ to ensure duplicate entries are not uploaded

signature: String signature is used to record the username of who completed the measurement

temperature: Simple numerical Temperature measured by the DQA3 device

pressure: Simple numerical The pressure measured by the DQA3 device

dose: Simple Numerical The dose measured by the DQA3 Device

dose_baseline: Simple Numerical Baseline dose value used

dose_diff: Simple Numerical Difference between measured dose and baseline

axsym: Simple Numerical Axial symmetry value

axsym_baseline: Simple Numerical Axial symmetry baseline value

axsym_diff: Simple Numerical Difference between measured axial symmetry and baseline

trsym: Simple Numerical Transverse symmetry value

trsym_baseline: Simple Numerical Transverse symmetry baseline value

trsym_diff: Simple Numerical Difference between measured transverse symmetry and baseline

qaflat: Simple Numerical Flatness value

qaflat_baseline: Simple Numerical Flatness baseline value

qaflat_diff: Simple Numerical Difference between measured flatness and baseline

energy: Simple Numerical Measured energy value

energy_baseline: Simple Numerical Energy baseline value (always 0)

energy_diff: Simple Numerical Difference between measured and baseline energy

xsize: Simple Numerical Measured width of profile in x direction

xsize_baseline: Simple Numerical Baseline width of profile in x direction

xsize_diff: Simple Numerical Difference bewteen measured and baseline width of profile in x direction

ysize: Simple Numerical Measured width of profile in y direction

ysize_baseline: Simple Numerical Baseline width of profile in y direction

ysize_diff: Simple Numerical Difference bewteen measured and baseline width of profile in y direction

xshift: Simple Numerical Measured shift of center of profile in x direction

xshift_baseline: Simple Numerical Baseline shift of center of profile in x direction

xshift_diff: Simple Numerical Difference between measured and baseline shift of center of profile in x direction

yshift: Simple Numerical Measured shift of center of profile in y direction

yshift_baseline: Simple Numerical Baseline shift of center of profile in y direction

yshift_diff: Simple Numerical Difference between measured and baseline shift of center of profile in y direction

Here is what a sample test list might look like:

Assign Test Lists to Units

Once you have created these Test Lists in QATrack+ you need to assign them to units you want to record DQA3 data for.

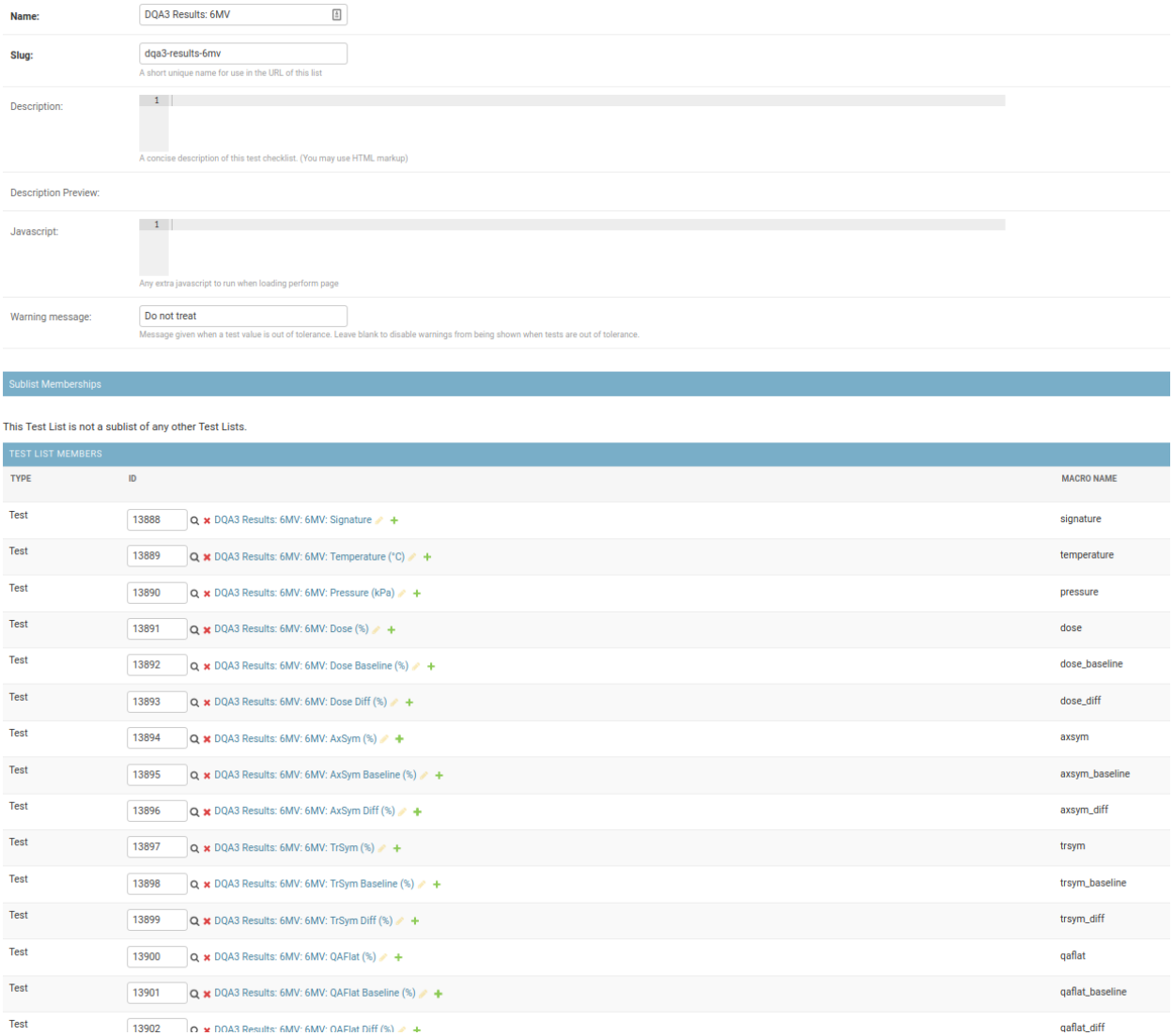


Fig. 6: A test list for recording 6MV results

DQA3 Common Configuration Options

Most of the configuration options are the same for the two DQA3 *Pump Types*. Those settings are outlined here and the DQA3 database connection specific options are described below.

QATrack+ API

Api Url Enter the root api url for the QATrack+ instance you want to upload data to. For Example http://yourqatrackserver/api

Auth Token Enter an authorization token for the QATrack+ instance you want to upload data to

Throttle Enter the minimum interval between data uploads (i.e. a value of 1 will allow 1 record per second to be uploded)

Verify SSL Set to False if you want to bypass SSL certificate checks (e.g. if your QATrack+ instance is using a self signed certificate)

Http Proxy QCPump will try to autodetect your current proxy settings. However if you want to manually provide a proxy url you may do so. Proxy authentication url e.g. http://10.10.1.10:3128 or socks5://user:pass@host:port

Https Proxy QCPump will try to autodetect your current proxy settings. However if you want to manually provide a proxy url you may do so.Proxy authentication url e.g. https://10.10.1.10:3128 or socks5://user:pass@host:port

Test List (depends on QATrack+ API)

Name Enter a template for searching QATrack+ for the name of the Test List you want to upload data to. The default is :

Daily QA3 Results: {{ energy }}{{ beam_type }}{{ wedge_type }}{{ wedge_angle }}

In the template *{{ energy }}* will be replaced by the DQA3 beam energy (e.g. 6, 10, 15) and *{{ beam_type }}* will be replaced by the DQA3 beam type (e.g. X, E, FFF). This template would result in QCPump trying to find a Test List called e.g. "Daily QA3 Results: 6X".

Unit (depends on QATrack+ API and DQA3Reader configs)

These config options are used to map DQA3 machine names to QATrack+ Unit names.

Dqa3 Name Select the DQA3 machine name to map

Unit Name Select the QATrack+ Unit name to map the DQA3 name to

DQA3: Firebird Individual Beams Pump Type

Config options specific to Firebird DQA3 databases (01.03.00.00 & 01.04.00.00).

DQA3Reader

Host Enter the host name of the Firebird database server you want to connect to

Database Enter the path to the database file you want to connect to on the server. For example C:UsersYourUserNamedatabasesSncdata.fdb

User Enter the username you want to use to connect to the database with

Password Enter the password you want to use to connect to the database with

Port Enter the port number that the Firebird Database server is listening on

Driver Select the database driver you want to use. Use firebirdsql unless you have a good reason not to.

History Days Enter the number of prior days you want to look for data to import. If you are importing historical data you may want to temporarily set this to a large number of days (i.e. to get the last years worth of data set History days to 365) but normally a small number of days should be used to minimize the number of records fetched.

Creating a Read-Only User for QCPump

While it is not required, you may wish to create a read only user for QCPump to connect to your database with. You may either use the Firebird tools *gsec* and *isql* to create the user or a third party tool like FlameRobin which is a great option for managing users and databases.

Using gsec to create a new user

On the server where your Firebird database is located, open a CMD prompt and enter the following command to create a user with the username *qcpump* and password *qcpump*:

```
# for firebird 1.5
C:\Program Files (x86)\Firebird\Firebird_1_5\bin\gsec.exe" -user sysdba -password␣
↪masterkey -database "localhost:C:\Program Files (x86)\Firebird\Firebird_1_5\
↪security.fdb

# for firebird 2.5
C:\Program Files (x86)\Firebird\Firebird_2_5\bin\gsec.exe" -user sysdba -password␣
↪masterkey -database "localhost:C:\Program Files (x86)\Firebird\Firebird_1_5\
↪security2.fdb


GSEC> add qcpump -pw qcpump
GSEC> q
```

Next you can grant your user select rights using isql. Open isql specifying your username and password on the command line:

```
# for firebird 1.5
"C:\Program Files (x86)\Firebird\Firebird_1_5\bin\isql.exe" -user sysdba -password␣
↪masterkey

# for firebird 2.5
"C:\Program Files (x86)\Firebird\Firebird_2_5\bin\isql.exe" -user sysdba -password␣
↪masterkey
```

and connect to your database:

```
CONNECT "localhost:C:\Path\To\Your\Database\Sncdata.fdb";
```

(*note, you may need to replace `localhost` with your actual server host name*) then grant your user select rights on the tables required:

```
GRANT SELECT ON atlas_master to USER qcpump;
GRANT SELECT ON dqa3_machine to USER qcpump;
GRANT SELECT ON dqa3_trend to USER qcpump;
GRANT SELECT ON dqa3_data to USER qcpump;
GRANT SELECT ON device to USER qcpump;
GRANT SELECT ON dqa3_calibration to USER qcpump;
GRANT SELECT ON dqa3_template to USER qcpump;
GRANT SELECT ON dqa3_machine to USER qcpump;
GRANT SELECT ON room to USER qcpump;
quit;
```

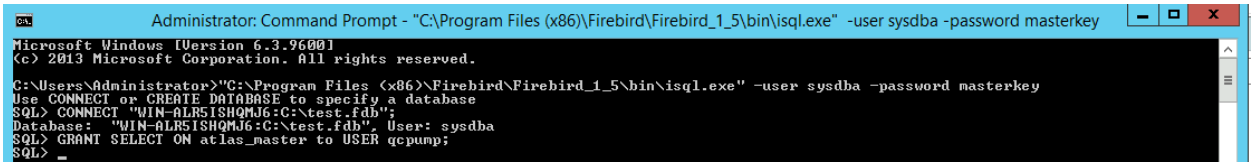


Fig. 7: Grant qcpump user rights

You should now be able to use the username *qcpump* and password *qcpump* for the *User* and *Password* settings described above.

DQA3: SQL Server Individual Beams DQA3 Pump Type

Config options specific to DQA3 SQL Server databases.

DQA3Reader

Host Enter the host name of the SQL Server database server you want to connect to

Database Enter the name of the database you want to connect to on the server. For example 'atlas'

User Enter the username you want to use to connect to the database with

Password Enter the password you want to use to connect to the database with

Port Enter the port number that the SQL Server database server is listening on

Driver Select the database driver you want to use. On Windows you will typically want to use the *ODBC Driver 17 for SQL Server* driver (ensure you have this driver installed on the computer running QCPump!). On Linux you will likely want to use one of the TDS drivers.

History Days Enter the number of prior days you want to look for data to import. If you are importing historical data you may want to temporarily set this to a large number of days (i.e. to get the last years worth of data set History days to 365) but normally a small number of days should be used to minimize the number of records fetched.

DQA3: Atlas Individual Beams DQA3 Pump Type

Config options specific to Atlas DQA3 databases (SQLServer).

DQA3Reader

Host Enter the host name of the SQL Server database server you want to connect to

Database Enter the name of the database you want to connect to on the server. For example 'atlas'

User Enter the username you want to use to connect to the database with

Password Enter the password you want to use to connect to the database with

Port Enter the port number that the SQL Server database server is listening on

Driver Select the database driver you want to use. On Windows you will typically want to use the *ODBC Driver 17 for SQL Server* driver (ensure you have this driver installed on the computer running QCPump!). On Linux you will likely want to use one of the TDS drivers.

History Days Enter the number of prior days you want to look for data to import. If you are importing historical data you may want to temporarily set this to a large number of days (i.e. to get the last years worth of data set History days to 365) but normally a small number of days should be used to minimize the number of records fetched.

Varian MPC Pumps

Contents

Note: The DQA3 pumps are tested on QATrack+ v3.1. QCPump is not compatible with QATrack v0.3.X

Varian's automated Machine Performance Check (MPC) stores its results in CSV files on disk which makes it easy to review those results using third party programs (thanks Varian!).

How The MPC Pump Works

The MPC Pump works by periodically searching the *va_transer\TDS* directory for any *Results.csv* files and then grouping the data from the csv files together and uploading that data to QATrack+. The data from the *Results.csv* files are combined based on the following three factors:

1. **The machine they were performed on.** QCPump uses the serial number from the MPCChecks directories to match with a QATrack+ Unit. For example, for an MPC directory like:

```
NDS-WKS-SN1234-2020-11-06-09-59-21-0009-GeometryCheckTemplate6xMVkV
```

QCPump considers the serial number to be *1234*. In order to upload results to QATrack+, QCPump must be able to find a QATrack+ unit with its serial number set to *1234*.

2. **The type of MPC check that was run.** QCPump will retrieve data from the following MPC directory types:

   a. **GeometryCheckTemplate6xMVkV** e.g. NDS-WKS-SN####-YYYY-MM-DD-HH-MM-SS-####-GeometryCheckTemplate6xMVkV\Results.csv

   b. **BeamCheckTemplate{energy}{beam_type}** e.g. NDS-WKS-SN####-YYYY-MM-DD-HH-MM-SS-####-BeamCheckTemplate6x\Results.csv

   c. **GeometryCheckTemplate6xMVkVEnhancedCouch** e.g. NDS-WKS-SN####-YYYY-MM-DD-HH-MM-SS-####-GeometryCheckTemplate6xMVkVEnhancedCouch\Results.csv

   d. **EnhancedMLCCheckTemplate6x** e.g. NDS-WKS-SN####-YYYY-MM-DD-HH-MM-SS-####-EnhancedMLCCheckTemplate6x\Results.csv

   results from a. and b. are grouped together into a single set of results to upload to QATrack+, while "enhanced" results from c. and d. are uploaded on their own.

3. **The date & time the results files were generated.** For the GeometryCheckTemplate & BeamCheckTemplate check types, QCPump will combine results from a period of N minutes into a single data set.

For example, given the following list of directories and Results.csv files with a time grouping window of 20 minutes:

```
NDS-WKS-SN5678-2020-06-25-07-10-30-0000-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv
NDS-WKS-SN5678-2020-06-25-07-20-30-0000-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv
NDS-WKS-SN6789-2020-09-30-09-15-17-0011-EnhancedMLCCheckTemplate6x\Results.csv
NDS-WKS-SN1234-2020-11-06-09-59-21-0009-GeometryCheckTemplate6xMVkV\Results.csv
NDS-WKS-SN1234-2020-11-06-09-59-21-0000-BeamCheckTemplate6xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-09-59-21-0000-BeamCheckTemplate10x\Results.csv
NDS-WKS-SN1234-2020-11-06-09-59-21-0000-BeamCheckTemplate10xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-10-01-21-0000-BeamCheckTemplate18x\Results.csv
NDS-WKS-SN1234-2020-11-06-10-01-21-0000-BeamCheckTemplate6e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-01-21-0000-BeamCheckTemplate9e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-02-21-0000-BeamCheckTemplate12e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-02-21-0000-BeamCheckTemplate16e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-02-21-0000-BeamCheckTemplate20e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-03-21-0009-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv
NDS-WKS-SN1234-2020-11-06-11-59-21-0009-GeometryCheckTemplate6xMVkV\Results.csv
NDS-WKS-SN1234-2020-11-06-11-59-21-0000-BeamCheckTemplate6xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-11-59-21-0000-BeamCheckTemplate10x\Results.csv
NDS-WKS-SN1234-2020-11-06-11-59-21-0000-BeamCheckTemplate10xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-12-01-21-0000-BeamCheckTemplate18x\Results.csv
NDS-WKS-SN1234-2020-11-06-12-01-21-0000-BeamCheckTemplate6e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-01-21-0000-BeamCheckTemplate9e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-02-21-0000-BeamCheckTemplate12e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-02-21-0000-BeamCheckTemplate16e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-02-21-0000-BeamCheckTemplate20e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-03-21-0009-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv
```

the results would be grouped as:

```
# Group 1) SN 5678, 2020-06-25 07:10, Enhanced Couch Checks
NDS-WKS-SN5678-2020-06-25-07-10-30-0000-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv
```

(continues on next page)

(continued from previous page)

```
# Group 2) SN 5678, 2020-06-25 07:20, Enhanced Couch Checks
# Even though these results were performed within 20min of Group 1),
# they are not grouped together because they are an "Enhanced" check type.
NDS-WKS-SN5678-2020-06-25-07-20-30-0000-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv

# Group 3) SN 6789, 2020-09-30 09:15, Enhanced MLC Checks
NDS-WKS-SN6789-2020-09-30-09-15-17-0011-EnhancedMLCCheckTemplate6x\Results.csv

# Group 4) SN 6789, 2020-11-06 09:59, Beam and Geometry Checks
# These are grouped together because they are not "enhanced" check types and
# all occurred within 20min of each other
NDS-WKS-SN1234-2020-11-06-09-59-21-0009-GeometryCheckTemplate6xMVkV\Results.csv
NDS-WKS-SN1234-2020-11-06-09-59-21-0000-BeamCheckTemplate6xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-09-59-21-0000-BeamCheckTemplate10x\Results.csv
NDS-WKS-SN1234-2020-11-06-09-59-21-0000-BeamCheckTemplate10xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-10-01-21-0000-BeamCheckTemplate18x\Results.csv
NDS-WKS-SN1234-2020-11-06-10-01-21-0000-BeamCheckTemplate6e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-01-21-0000-BeamCheckTemplate9e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-02-21-0000-BeamCheckTemplate12e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-02-21-0000-BeamCheckTemplate16e\Results.csv
NDS-WKS-SN1234-2020-11-06-10-02-21-0000-BeamCheckTemplate20e\Results.csv


# Group 5) SN 1234, 2020-11-06 03:21, Enhanced Couch Checks
NDS-WKS-SN1234-2020-11-06-10-03-21-0009-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv

# Group 6) SN 1234, 2020-11-06 11:59, Beam and Geometry Checks
# These are grouped together because they are not "enhanced" check types and
# all occurred within 20min of each other. They are not grouped with Group 4)
# results because they occurred at least 20 min after the last result from Group 4
NDS-WKS-SN1234-2020-11-06-11-59-21-0009-GeometryCheckTemplate6xMVkV\Results.csv
NDS-WKS-SN1234-2020-11-06-11-59-21-0000-BeamCheckTemplate6xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-11-59-21-0000-BeamCheckTemplate10x\Results.csv
NDS-WKS-SN1234-2020-11-06-11-59-21-0000-BeamCheckTemplate10xFFF\Results.csv
NDS-WKS-SN1234-2020-11-06-12-01-21-0000-BeamCheckTemplate18x\Results.csv
NDS-WKS-SN1234-2020-11-06-12-01-21-0000-BeamCheckTemplate6e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-01-21-0000-BeamCheckTemplate9e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-02-21-0000-BeamCheckTemplate12e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-02-21-0000-BeamCheckTemplate16e\Results.csv
NDS-WKS-SN1234-2020-11-06-12-02-21-0000-BeamCheckTemplate20e\Results.csv

# Group 7) SN 1234, 2020-11-06 12:03, Enhanced Couch Checks
NDS-WKS-SN1234-2020-11-06-12-03-21-0009-GeometryCheckTemplate6xMVkVEnhancedCouch\
↪Results.csv
```

Configuring QATrack+ for MPC Data

In order to upload MPC data to QATrack+ you need to do a bit of setup work in QATrack+ first.

Create an API Token

In order to upload your data to QATrack+ via the API you will require an API token. See the QATrack+ documentation for how to create an API token. You may wish to create a dedicated user in QATrack+ just for use with QCPump. The user will only need a single permission in order to upload data: *qa* | *test list instance* | *Can add test list instance*.

Configure Your Unit's Serial Numbers

In order to determine which unit to upload MPC results to, QCPump queries the QATrack+ API to look for a unit with a serial number matching the MPC directory it finds results in. For example if QCPump finds a Results.csv file in a directory like "NDS-WKS-SN**1234**-2020-11-06-12-03-21-0009-GeometryCheckTemplate6xMVkVEnhancedCouch" then it will look for a unit configured in QATrack+ with a serial number of 1234. If QCPump can't find a unit with a matching serial number, the MPC results from that directory will be ignored.

Configure Test Lists

You will need 1 or more test lists to record the MPC data uploaded by QCPump.

1. A test list named "MPC: Beam and Geometry Checks" to record data from *GeometryCheckTemplate6xMVkV* & *BeamCheckTemplate{energy}{beam_type}* MPC results.
2. A test list named "MPC: Enhanced Couch Checks" to record data from *GeometryCheckTemplate6xMVkVEnhancedCouch* results files.
3. A test list named "MPC: Enhanced MLC Checks" to record data from *EnhancedMLCCheckTemplate6x* results files.

These test lists should have tests with macro names corresponding to the name of results in the Results.csv files. The Results.csv files have names like the following:

```
IsoCenterGroup/IsoCenterSize [mm]
IsoCenterGroup/IsoCenterMVOffset [mm]
IsoCenterGroup/IsoCenterKVOffset [mm]
BeamGroup/BeamOutputChange [%]
BeamGroup/BeamUniformityChange [%]
BeamGroup/BeamCenterShift [mm]
CollimationGroup/MLCGroup/MLCMaxOffsetA [mm]
CollimationGroup/MLCGroup/MLCMaxOffsetB [mm]
CollimationGroup/MLCGroup/MLCMeanOffsetA [mm]
GantryGroup/GantryAbsolute [°], -0.09, 0.3, Pass
GantryGroup/GantryRelative [°], 0.11, 0.3, Pass
```

and QCPump transforms these names into valid QATrack+ macro names using the following rules:

1. All slashes (/) and spaces are converted into underscores (_)
2. Unit replacements are made as follows:
   - *[mm]* is replaced with *mm*,
   - *[%]* is replaced with *per*,

- *[°]* is replaced with *deg*

3. The beam energy/type is appended to the macro name
4. The macro name is lowercased.

Examples of this substitution from a Results.csv file in a directory called *NDS-WKS-SN1234-2020-12-01-01-00-00-0009-GeometryCheckTemplate6xMVkV* are:

```
IsoCenterGroup/IsoCenterSize [mm] --> isocentergroup_isocentresize_mm_6x
BeamGroup/BeamOutputChange [%] -->  beamgroup_beamoutputchange_per_6x
GantryGroup/GantryAbsolute [°] --> gantrygroup_gantryabsolute_deg_6x
```

Warning: Currently results for individual MLC leaves are not included. Any result which starts with any of these 4 strings:

```
CollimationGroup/MLCGroup/MLCLeavesA/MLCLeaf
CollimationGroup/MLCGroup/MLCLeavesB/MLCLeaf
CollimationGroup/MLCBacklashGroup/MLCBacklashLeavesA/MLCBacklashLeaf
CollimationGroup/MLCBacklashGroup/MLCBacklashLeavesB/MLCBacklashLeaf
```

will not be included in the api payload.

As a further example, the following Results.csv file found in a directory with the name *NDS-WKS-SN1234-2020-12-01-01-00-00-0009-GeometryCheckTemplate6xMVkV*

```
Name [Unit], Value, Threshold, Evaluation Result
IsoCenterGroup/IsoCenterSize [mm], 0.3, 0.5, Pass
IsoCenterGroup/IsoCenterMVOffset [mm], 0.26, 0.5, Pass
IsoCenterGroup/IsoCenterKVOffset [mm], 0.25, 0.5, Pass
BeamGroup/BeamOutputChange [%], -0.17, 2, Pass
BeamGroup/BeamUniformityChange [%], 0.31, 2, Pass
BeamGroup/BeamCenterShift [mm], 0.07, 0.5, Pass
CollimationGroup/MLCGroup/MLCMaxOffsetA [mm], 0.33, 1, Pass
CollimationGroup/MLCGroup/MLCMaxOffsetB [mm], 0.38, 1, Pass
CollimationGroup/MLCGroup/MLCMeanOffsetA [mm], 0.24, 1, Pass
...
GantryGroup/GantryAbsolute [°], -0.09, 0.3, Pass
```

would result in an API payload like this:

```
{
    "unit_test_collection": "https://qatrack.example.com/api/qa/unittestcollections/
↪1234/",
    "work_started": "2020-12-01 01:00",
    "work_completed": "2020-12-01 01:01",
    "user_key": "1234-2020-12-01-01-01",
    "day": 0,
    "tests": {
        isocentergroup_isocentersize_mm_6x: {"value": 0.3},
        isocentergroup_isocentermvoffset_mm_6x: {"value": 0.26},
        # ...
        beamgroup_beamoutputchange_per_6x: {"value": -0.17},
        # ... and so on
        gantrygroup_gantryabsolute_deg: {"value": -0.09}
        # ... and so on
    }
}
```

Therefore you will need to configure *Simple Numerical* tests for your test lists with these macro names (or a subset of them). The names of the tests can be anything you like, but naming your test the same as the names in the Results.csv file might be a good idea. So a Test List might look like:

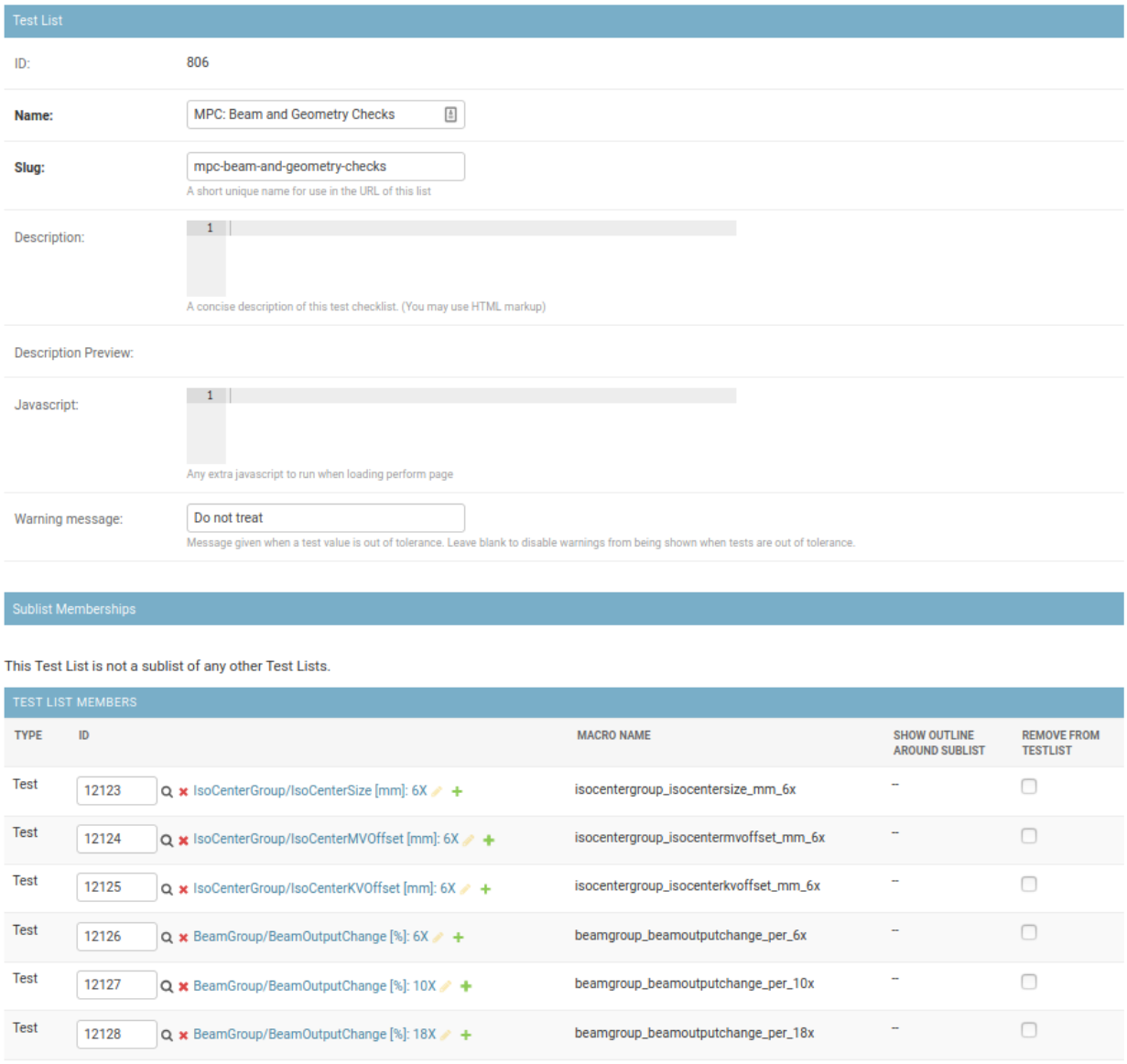


Fig. 8: MPC Beam And Geometry Example Test List

Note: If QCPump detects that not all tests results for a given test list are included when it tries to upload results. It will automatically skip those results and attempt to upload the data again. This allows QCPump to upload partial result sets when e.g. you only run a single beam in MPC but your test list is configured to receive results from multiple beam types.

Assign Test Lists to Units

Once you have created these Test Lists in QATrack+ you need to assign them to units you want to record DQA3 data for.

MPC QCPump Configuration Options

MPC

TDS Directory The "TDS directory" where MPC results are stored. Examples may be I:\TDS or \\\YOURSERVER\VA_Transer\TDS

Fast Search Restricts the search for Results.csv files to MPCChecks subdirectories. MPCChecks is the official Varian directory name and unless you have MPCResults in folders named something other than "MPCChecks", it is recommended you leave this setting on for performance reasons.

Days of history The number of prior days you want to look for data to import. This should generall be 1 unless you are doing an initial import of historical results

Results group time interval (min) Enter the time interval (in minutes) for which results should be grouped together. That is to say, for Beam & Geometry checks how large of a time window should be used to consider MPC results part of the same session. This value should be a little bit longer than the typical time it takes you to run all your morning MPC checks.

Wait for results (min) Wait this many minutes for more results to be written to disk before uploading grouped results. In order to ensure all results from an MPC session, are written to disk, QCPump will wait this many minutes after the most recent Results.csv file it finds for a given machine before uploading results to QATrack+.

QATrack+ API

Api Url Enter the root api url for the QATrack+ instance you want to upload data to. For Example http://yourqatrackserver/api

Auth Token Enter an authorization token for the QATrack+ instance you want to upload data to

Throttle Enter the minimum interval between data uploads (i.e. a value of 1 will allow 1 record per second to be uploded)

Verify SSL Set to False if you want to bypass SSL certificate checks (e.g. if your QATrack+ instance is using a self signed certificate)

Http Proxy QCPump will try to autodetect your current proxy settings. However if you want to manually provide a proxy url you may do so. Proxy authentication url e.g. http://10.10.1.10:3128 or socks5://user:pass@host:port

Https Proxy QCPump will try to autodetect your current proxy settings. However if you want to manually provide a proxy url you may do so.Proxy authentication url e.g. https://10.10.1.10:3128 or socks5://user:pass@host:port

Test List (depends on QATrack+ API)

Name Enter a template for searching QATrack+ for the name of the Test List you want to upload data to. The default is :

MPC: {{ check_type}}

In the template *{{ check_type }}* will be replaced by either:

- Beam and Geometry Checks
- Enhanced Couch Checks
- Enhanced MLC Checks

depending on the results being uploaded

Configuration Validation

Any time you edit a configuration value your *Pump* will be re-validated. This may include actions such as ensuring a file system directory exists, or sending an HTTP request to check that a QATrack+ API endpoint is reachable.

Note: Currently the *Pump* tab is disabled during validation to prevent multiple validations from running simultaneously.

After validation, a message will be shown indicating whether each configuration setting is valid or not.

You can force a revalidation (say if some external factor was reconfigured / updated) by pressing the *Revalidate* button.

Configuration Dependencies

Some *Pump Types* will have configuration sections which depend on other configuration sections being validated successfully before they can be completed. For example, selecting a Unit configured in QATrack+ will not be possible until QCPump has verified it can talk to your QATrack+ API.

When a dependency of a configuration section has not been met, the dependent section will be disabled and a message will be shown to inform you of which sections need to be completed before the dependent section can be verified.

Multiple Configuration Subsections

Some configuration sections allow you to use multiple repeated subsections. An example of this would be the configuration used by the DQA3 Pump Types to map DQA3 Machines to QATrack+ Units:

To add new configuration subsections click the + button, and to remove a subsection, highlight one of its configuration options and click the -.

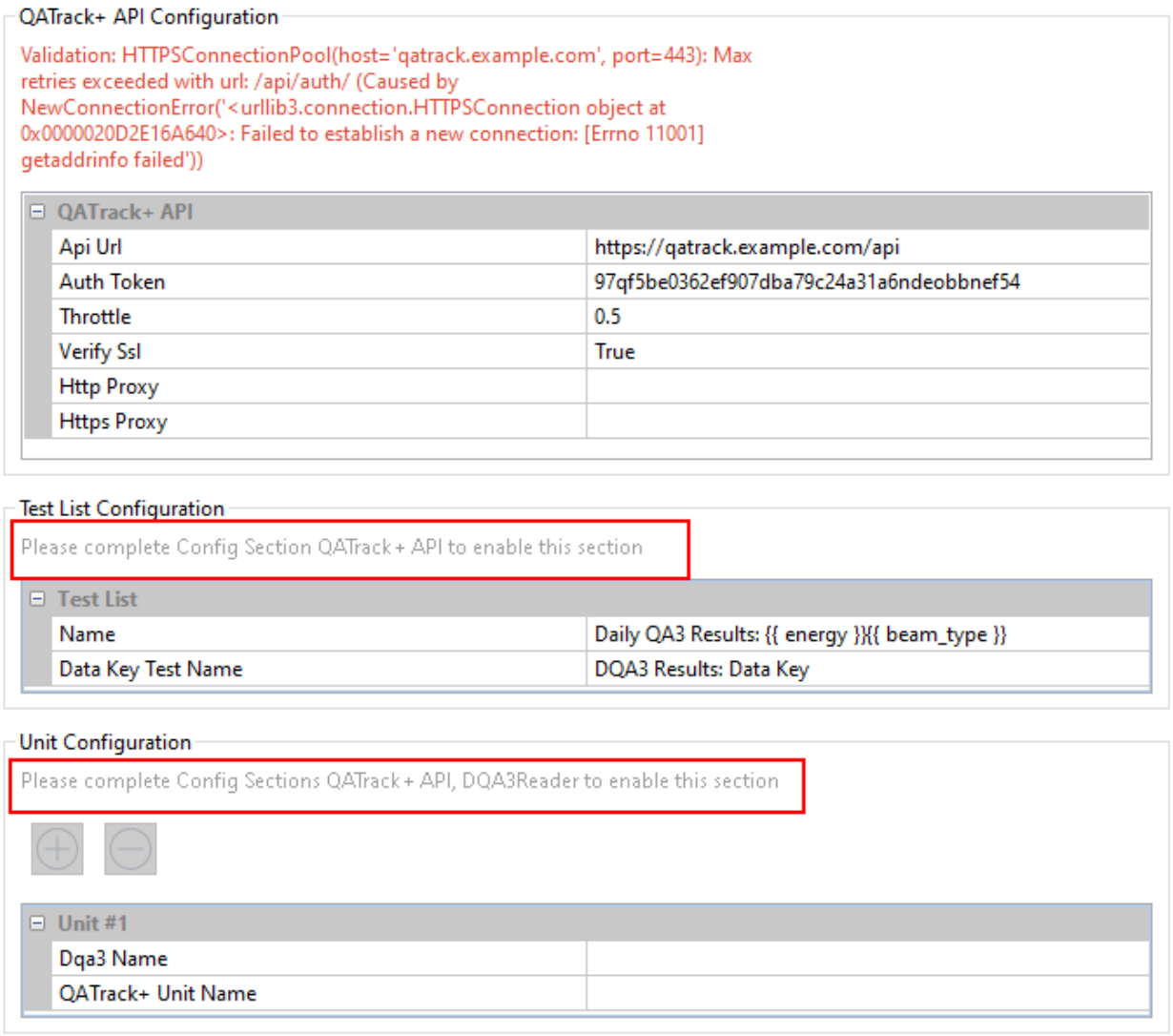


Fig. 9: QCPump config with unmet dependencies

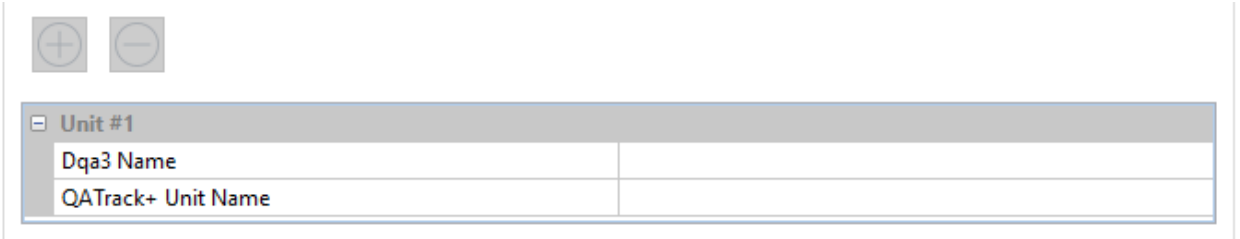


Fig. 10: Multiple config subsections

5.1.2 Saving your Pumps

Before you can run your pumps, they all must be saved (pumps which haven't been saved will be marked with asterisks surrounding their pump tabs name). Once you have configured and validated your *Pump* click the *Save* button at the top of its config section. The location of the configuration files for QCPump can be found by accessing the *About* menu option in the *File* menu.

5.1.3 Reseting a Configuration

If you want to reset your Pump config, click the *Reset* button at the top of its config. This will reset the pumps config to the most recently saved version of the pumps configuration.

5.1.4 Deleting a Pump

If you want to delete a Pump config, click the *Delete* button at the top of its config. This will permanently delete this pump and remove its configuration file from disk.

5.2 Daily QA3 Pumps: Multiple Beams Per Test List

This page refers to Pumps which group together results from a single unit recorded during a short time period to a single test list. For example, if you have the following 10 beams configured on a single unit: 6X, 6FFF, 10X, 10FFF, 6X EDW60, 6E, 9E, 12E, 16E, 20E, and perform measurements for all these beams in a 10 minute period, then QCPump will combine all of these results and upload to QATrack+ to perform a single test list. If you prefer to use a single test list per beam type to record your results, please see: *Daily QA3 Pumps: One Beam Per Test List*.

Note: There are two disadvantages to using the Multiple Beams Per Test List:

1. If you have many beams configured this will result in very long test lists which can impact performance when uploading data, or reviewing data in QATrack+.
2. If you perform a measurement twice (e.g. take 2 6X measurements), only the 2nd result will be included.

QCPump currently has the ability to retrieve data from the following Daily QA3 data sources:

- DQA3 Firebird Database version 01.03
- DQA3 Firebird Database version 01.04
- DQA3 SQL Server Database version 01.06
- DQA3 data from Atlas 1.5

Note: The DQA3 pumps are tested on QATrack+ v3.1.X QCPump is not compatible with QATrack v0.3.X

Contents

- *DQA3: Firebird Multiple Beams Per Test List Pump Type*
- *DQA3: SQL Server Multiple Beams Per Test List Pump Type*
- *DQA3: Atlas Multiple Beams Per Test List Pump Type*

5.2.1 Configuring QATrack+ for DQA3 Data

In order to upload DQA3 data to QATrack+ you need to do a bit of setup work in QATrack+ first.

Create an API Token

In order to upload your data to QATrack+ via the API you will require an API token. See the QATrack+ documentation for how to create an API token. You may wish to create a dedicated user in QATrack+ just for use with QCPump. The user will only need a single permission in order to upload data: *qa* | *test list instance* | *Can add test list instance*.

Configure Test Lists

Note: In order to simplify the creation of these test lists, there is a script included with QATrack+ v3.1.0+ to generate either a Test Pack or to create a TestList directly in your database. To run the script activate your virtualenv, changed to the QATrack+ root directory and then run

```
# create a test list in the db for photons
python manage.py runscript create_grouped_dqa3_testlist --script-args db "Daily QA3␣
↪Results: Photons" 6X 6FFF 10X 10FFF "6X EDW60"

# or create a test pack
python manage.py runscript create_grouped_dqa3_testlist --script-args testpack "Daily␣
↪QA3 Results: Electrons" 6E 9E 12E 16E 20E
```

QCPump requires QATrack+ to have a single Test List per unit configured to record all your Daily QA3 Results. The Test List must have a specific set of attributes:

Test List Name Enter the name of the test list you configured in QATrack+ to record your DQA3 results e.g.:

> Daily QA3 Results: Photons

When QCPump gathers data to post to QATrack+, it will convert the DQA3 test name, into a QATrack+ macro name for each test result. For example if the DQA3 Test/Beam is named "6 MV EDW60" then QCPump would generate test results for macros named *dose_6_mv_edw60*, *dose_baseline_6_mv_edw60*, *dose_diff_6_mv_edw60* and so on. You will need to create tests with at least one of the following macro names (where {beam_name} is suitably replaced according to which beam it is for):

data_key_{beam_name}: String data_key is a key from the DQA3 database used by QCPump and QATrack+ to ensure duplicate entries are not uploaded

signature_{beam_name}: String signature is used to record the username of who completed the measurement

temperature_{beam_name}: Simple numerical Temperature measured by the DQA3 device

pressure_{beam_name}: Simple numerical The pressure measured by the DQA3 device

dose_{beam_name}: Simple Numerical The dose measured by the DQA3 Device

dose_baseline_{beam_name}: Simple Numerical Baseline dose value used

dose_diff_{beam_name}: Simple Numerical Difference between measured dose and baseline

axsym_{beam_name}: Simple Numerical Axial symmetry value

axsym_baseline_{beam_name}: Simple Numerical Axial symmetry baseline value

axsym_diff_{beam_name}: Simple Numerical Difference between measured axial symmetry and baseline

trsym_{beam_name}: Simple Numerical Transverse symmetry value

trsym_baseline_{beam_name}: Simple Numerical Transverse symmetry baseline value

trsym_diff_{beam_name}: Simple Numerical Difference between measured transverse symmetry and baseline

qaflat_{beam_name}: Simple Numerical Flatness value

qaflat_baseline_{beam_name}: Simple Numerical Flatness baseline value

qaflat_diff_{beam_name}: Simple Numerical Difference between measured flatness and baseline

energy_{beam_name}: Simple Numerical Measured energy value

energy_baseline_{beam_name}: Simple Numerical Energy baseline value (always 0)

energy_diff_{beam_name}: Simple Numerical Difference between measured and baseline energy

xsize_{beam_name}: Simple Numerical Measured width of profile in x direction

xsize_baseline_{beam_name}: Simple Numerical Baseline width of profile in x direction

xsize_diff_{beam_name}: Simple Numerical Difference bewteen measured and baseline width of profile in x direction

ysize_{beam_name}: Simple Numerical Measured width of profile in y direction

ysize_baseline_{beam_name}: Simple Numerical Baseline width of profile in y direction

ysize_diff_{beam_name}: Simple Numerical Difference bewteen measured and baseline width of profile in y direction

xshift_{beam_name}: Simple Numerical Measured shift of center of profile in x direction

xshift_baseline_{beam_name}: Simple Numerical Baseline shift of center of profile in x direction

xshift_diff_{beam_name}: Simple Numerical Difference between measured and baseline shift of center of profile in x direction

yshift_{beam_name}: Simple Numerical Measured shift of center of profile in y direction

yshift_baseline_{beam_name}: Simple Numerical Baseline shift of center of profile in y direction

yshift_diff_{beam_name}: Simple Numerical Difference between measured and baseline shift of center of profile in y direction

Here is an example of what a test list configured with a sublist per beam might look like:

and the sublist for recording the 6MV results:

Fig. 11: Parent test list for recording DQA3 results

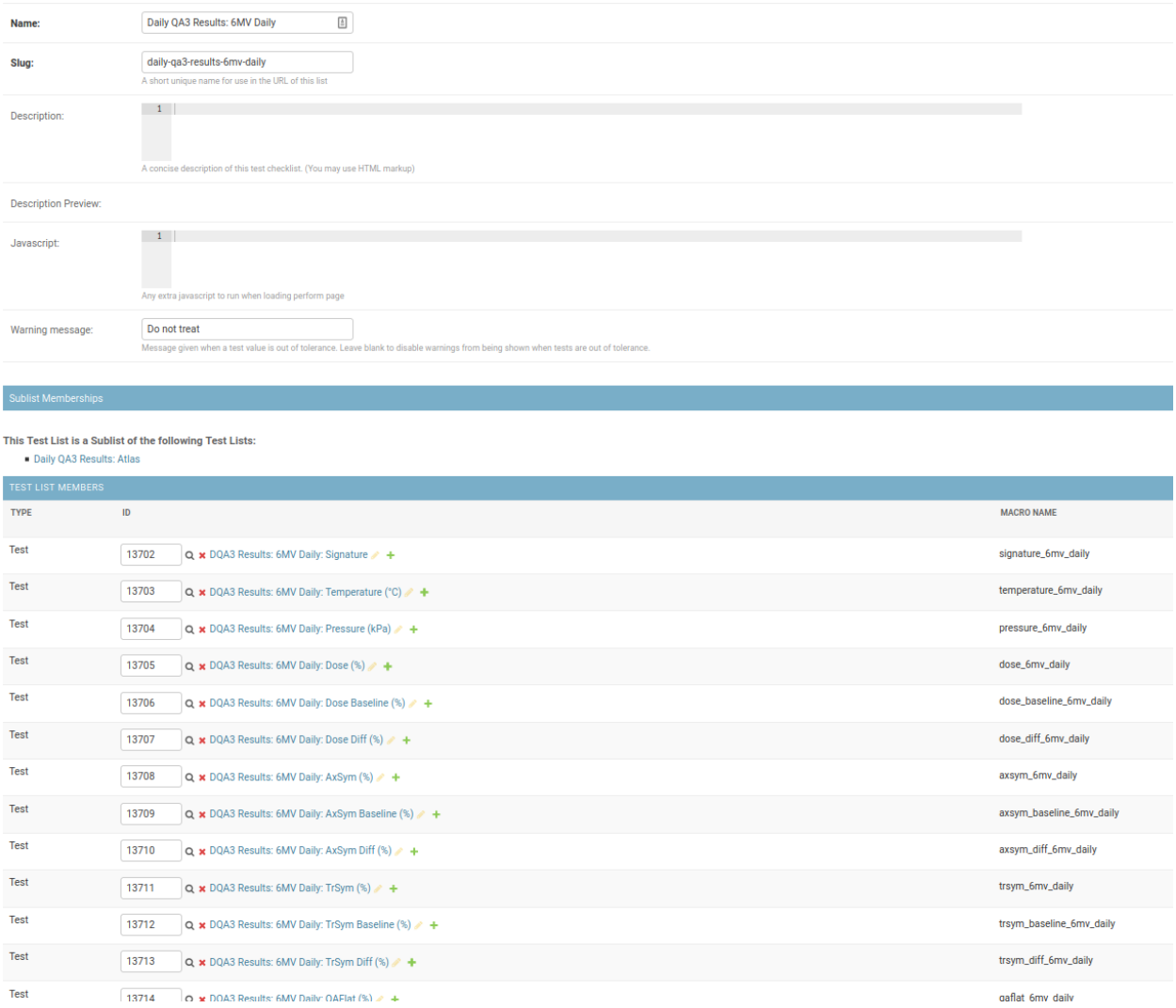


Fig. 12: Child test list for recording 6MV DQA3 results

Assign Test Lists to Units

Once you have created these Test Lists in QATrack+ you need to assign them to units you want to record DQA3 data for.

5.2.2 DQA3 Common Configuration Options

Most of the configuration options are the same for the two DQA3 *Pump Types*. Those settings are outlined here and the DQA3 database connection specific options are described below.

QATrack+ API

Api Url Enter the root api url for the QATrack+ instance you want to upload data to. For Example http://yourqatrackserver/api

Auth Token Enter an authorization token for the QATrack+ instance you want to upload data to

Throttle Enter the minimum interval between data uploads (i.e. a value of 1 will allow 1 record per second to be uploded)

Verify SSL Set to False if you want to bypass SSL certificate checks (e.g. if your QATrack+ instance is using a self signed certificate)

Http Proxy QCPump will try to autodetect your current proxy settings. However if you want to manually provide a proxy url you may do so. Proxy authentication url e.g. http://10.10.1.10:3128 or socks5://user:pass@host:port

Https Proxy QCPump will try to autodetect your current proxy settings. However if you want to manually provide a proxy url you may do so.Proxy authentication url e.g. https://10.10.1.10:3128 or socks5://user:pass@host:port

Test List (depends on QATrack+ API)

Name Enter a template for searching QATrack+ for the name of the Test List you want to upload data to. The default is :

Daily QA3 Results: {{ energy }}{{ beam_type }}

In the template *{{ energy }}* will be replaced by the DQA3 beam energy (e.g. 6, 10, 15) and *{{ beam_type }}* will be replaced by the DQA3 beam type (e.g. X, E, FFF). This template would result in QCPump trying to find a Test List called e.g. "Daily QA3 Results: 6X".

Unit (depends on QATrack+ API and DQA3Reader configs)

These config options are used to map DQA3 machine names to QATrack+ Unit names.

Dqa3 Name Select the DQA3 machine name to map

Unit Name Select the QATrack+ Unit name to map the DQA3 name to

5.2.3 DQA3: Firebird Multiple Beams Per Test List Pump Type

Config options specific to Firebird DQA3 databases (01.03.00.00 & 01.04.00.00).

DQA3Reader

Host Enter the host name of the Firebird database server you want to connect to

Database Enter the path to the database file you want to connect to on the server. For example C:UsersYourUserNamedatabasesSncdata.fdb

User Enter the username you want to use to connect to the database with

Password Enter the password you want to use to connect to the database with

Port Enter the port number that the Firebird Database server is listening on

Driver Select the database driver you want to use. Use firebirdsql unless you have a good reason not to.

History Days Enter the number of prior days you want to look for data to import. If you are importing historical data you may want to temporarily set this to a large number of days (i.e. to get the last years worth of data set History days to 365) but normally a small number of days should be used to minimize the number of records fetched.

Results group time interval (min) Enter the time interval (in minutes) for which results should be grouped together. That is to say, for Beam & Geometry checks how large of a time window should be used to consider MPC results part of the same session. This value should be a little bit longer than the typical time it takes you to run all

Wait for results (min) Wait this many minutes for more results to be written to disk before uploading grouped results. In order to ensure all results from an MPC session, are written to disk, QCPump will wait this many minutes after the most recent Results.csv file it finds for a given machine before uploading results to QATrack+.

Beam Types Select which beam types (Photon, Electron, All) you want to include for this pump. It is a good idea to create separate pumps for electrons and photons and corresponding test lists in QATrack+ for recording photon & electron results seperately. The Photon option will include FFF & wedged beams.

Creating a Read-Only User for QCPump

While it is not required, you may wish to create a read only user for QCPump to connect to your database with. You may either use the Firebird tools *gsec* and *isql* to create the user or a third party tool like FlameRobin which is a great option for managing users and databases.

Using gsec to create a new user

On the server where your Firebird database is located, open a CMD prompt and enter the following command to create a user with the username *qcpump* and password *qcpump*:

```
# for firebird 1.5
C:\Program Files (x86)\Firebird\Firebird_1_5\bin\gsec.exe" -user sysdba -password␣
↪masterkey -database "localhost:C:\Program Files (x86)\Firebird\Firebird_1_5\
↪security.fdb

# for firebird 2.5
C:\Program Files (x86)\Firebird\Firebird_2_5\bin\gsec.exe" -user sysdba -password␣
↪masterkey -database "localhost:C:\Program Files (x86)\Firebird\Firebird_1_5\
↪security2.fdb


GSEC> add qcpump -pw qcpump
GSEC> q
```

Next you can grant your user select rights using isql. Open isql specifying your username and password on the command line:

```
# for firebird 1.5
"C:\Program Files (x86)\Firebird\Firebird_1_5\bin\isql.exe" -user sysdba -password␣
↪masterkey

# for firebird 2.5
"C:\Program Files (x86)\Firebird\Firebird_2_5\bin\isql.exe" -user sysdba -password␣
↪masterkey
```

and connect to your database:

```
CONNECT "localhost:C:\Path\To\Your\Database\Sncdata.fdb";
```

(*note, you may need to replace `localhost` with your actual server host name*) then grant your user select rights on the tables required:

```
GRANT SELECT ON atlas_master to USER qcpump;
GRANT SELECT ON dqa3_machine to USER qcpump;
GRANT SELECT ON dqa3_trend to USER qcpump;
GRANT SELECT ON dqa3_data to USER qcpump;
GRANT SELECT ON device to USER qcpump;
GRANT SELECT ON dqa3_calibration to USER qcpump;
GRANT SELECT ON dqa3_template to USER qcpump;
GRANT SELECT ON dqa3_machine to USER qcpump;
GRANT SELECT ON room to USER qcpump;
quit;
```

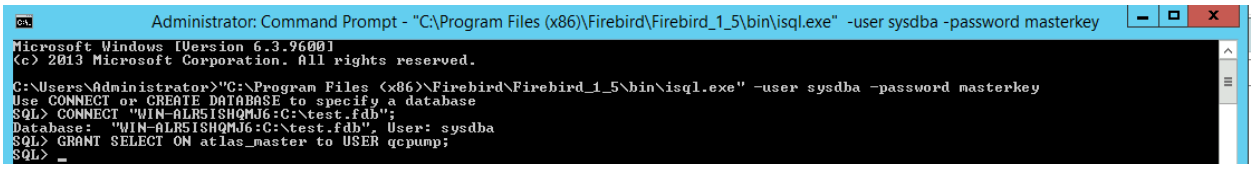


Fig. 13: Grant qcpump user rights

You should now be able to use the username *qcpump* and password *qcpump* for the *User* and *Password* settings described above.

5.2.4 DQA3: SQL Server Multiple Beams Per Test List Pump Type

Config options specific to SQL Server DQA3 databases.

DQA3Reader

Host Enter the host name of the SQL Server database server you want to connect to

Database Enter the name of the database you want to connect to on the server. For example 'atlas'

User Enter the username you want to use to connect to the database with

Password Enter the password you want to use to connect to the database with

Port Enter the port number that the SQL Server database server is listening on

Driver Select the database driver you want to use. On Windows you will typically want to use the *ODBC Driver 17 for SQL Server* driver (ensure you have this driver installed on the computer running QCPump!). On Linux you will likely want to use one of the TDS drivers.

History Days Enter the number of prior days you want to look for data to import. If you are importing historical data you may want to temporarily set this to a large number of days (i.e. to get the last years worth of data set History days to 365) but normally a small number of days should be used to minimize the number of records fetched.

Results group time interval (min) Enter the time interval (in minutes) for which results should be grouped together. That is to say, for Beam & Geometry checks how large of a time window should be used to consider MPC results part of the same session. This value should be a little bit longer than the typical time it takes you to run all

Wait for results (min) Wait this many minutes for more results to be written to disk before uploading grouped results. In order to ensure all results from an MPC session, are written to disk, QCPump will wait this many minutes after the most recent Results.csv file it finds for a given machine before uploading results to QATrack+.

Beam Types Select which beam types (Photon, Electron, All) you want to include for this pump. It is a good idea to create separate pumps for electrons and photons and corresponding test lists in QATrack+ for recording photon & electron results seperately. The Photon option will include FFF & wedged beams.

5.2.5 DQA3: Atlas Multiple Beams Per Test List Pump Type

Config options specific to Atlas DQA3 databases (SQLServer).

DQA3Reader

Host Enter the host name of the SQL Server database server you want to connect to

Database Enter the name of the database you want to connect to on the server. For example 'atlas'

User Enter the username you want to use to connect to the database with

Password Enter the password you want to use to connect to the database with

Port Enter the port number that the SQL Server database server is listening on

Driver Select the database driver you want to use. On Windows you will typically want to use the *ODBC Driver 17 for SQL Server* driver (ensure you have this driver installed on the computer running QCPump!). On Linux you will likely want to use one of the TDS drivers.

History Days Enter the number of prior days you want to look for data to import. If you are importing historical data you may want to temporarily set this to a large number of days (i.e. to get the last years worth of data set History days to 365) but normally a small number of days should be used to minimize the number of records fetched.

Results group time interval (min) Enter the time interval (in minutes) for which results should be grouped together. That is to say, for Beam & Geometry checks how large of a time window should be used to consider MPC results part of the same session. This value should be a little bit longer than the typical time it takes you to run all

Wait for results (min) Wait this many minutes for more results to be written to disk before uploading grouped results. In order to ensure all results from an MPC session, are written to disk, QCPump will wait this many minutes after the most recent Results.csv file it finds for a given machine before uploading results to QATrack+.

Beam Types Select which beam types (Photon, Electron, All) you want to include for this pump. It is a good idea to create separate pumps for electrons and photons and corresponding test lists in QATrack+ for recording photon & electron results seperately. The Photon option will include FFF & wedged beams.

5.3 QATrack+ Generic File Uploads

QCPump currently has two pumps for uploading text or binary files to QATrack+. Both pumps operate by watching a directory for files, uploading them to a QATrack+ test list, and optionally, moving the file to a new directory after the file is processed.

5.3.1 Configuration options

Todo: Add common config options

> **Warning:** Files in your Destination directory may be overwritten by new files if they have the same name!

QATrack+ File Upload: Generic Text File

For uploading text files (e.g. csv, Profiler exports, etc)

QATrack+ File Upload: Generic Binary File

Todo: Text File upload docs

CHAPTER

SIX

DEVELOPMENT NOTES

6.1 Runing tests

The tests can be run by running:

```
py.test
```

in the root qcpump directory.

6.2 Release Checklist

- [ ] Tests all passing
- [ ] docs/release_notes.rst updated
- [ ] Version updated in:
  - [ ] setup.py
  - [ ] qcpump/settings.py
  - [ ] qcpump-installer.iss
  - [ ] docs/install.rst (link to installer)
- **[ ] Exe built (*pyinstaller qcpump.spec*)**
  - [ ] New data files added to qcpump.spec::data_files
  - [ ] New dependencies added to qcpump.spec::hidden_import
- **[ ] Installer built**
  - [ ] Installer comitted to repository
- [ ] Installer tested
- [ ] Release tagged *git tag -a vX.X.X -m vX.X.X*
- [ ] Push to master with tags *git push origin master –tags*

6.3 Building an exe on Windows

In order to create an executable version of QCPump on Windows you need to install pyinstaller:

```
pip install pyinstaller
```

Then run:

```
pyinstaller qcpump.spec
```

6.4 Building the Installer

- Install Inno Setup Compiler
- Build the exe as described above
- Open qcpump-installer.iss in Inno
- Update MyAppVersion
- Build Menu -> Compile (or Ctrl+F9)
- Test that the installer works and QCPump launches after installation
- Add the installer to the repository and push.

CHAPTER

SEVEN

PUMP TYPE DEVELOPMENT

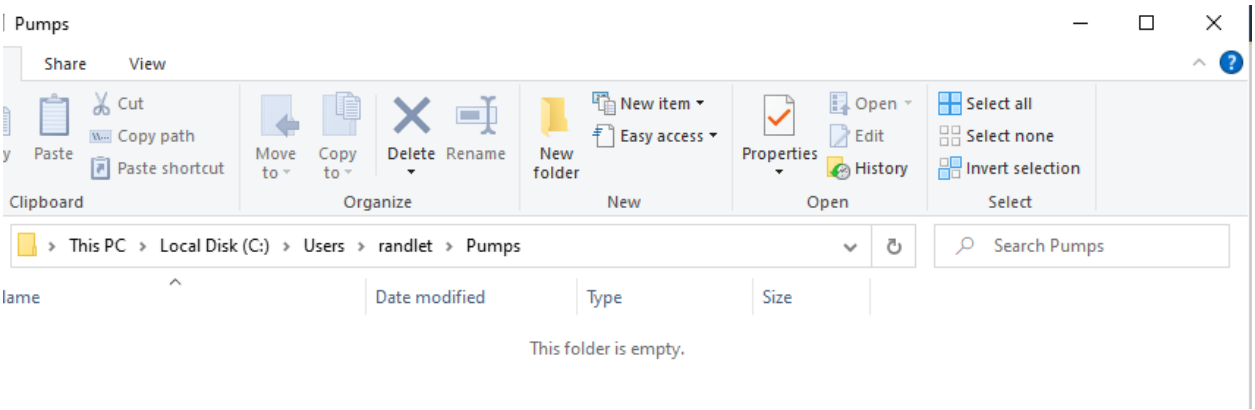
7.1 Tutorial

7.1.1 Install QCPump

Follow the *installation instructions* and either install the precompiled executable version of QCPump using the Windows installer, or install QCPump from source

7.1.2 Create a PumpType directory

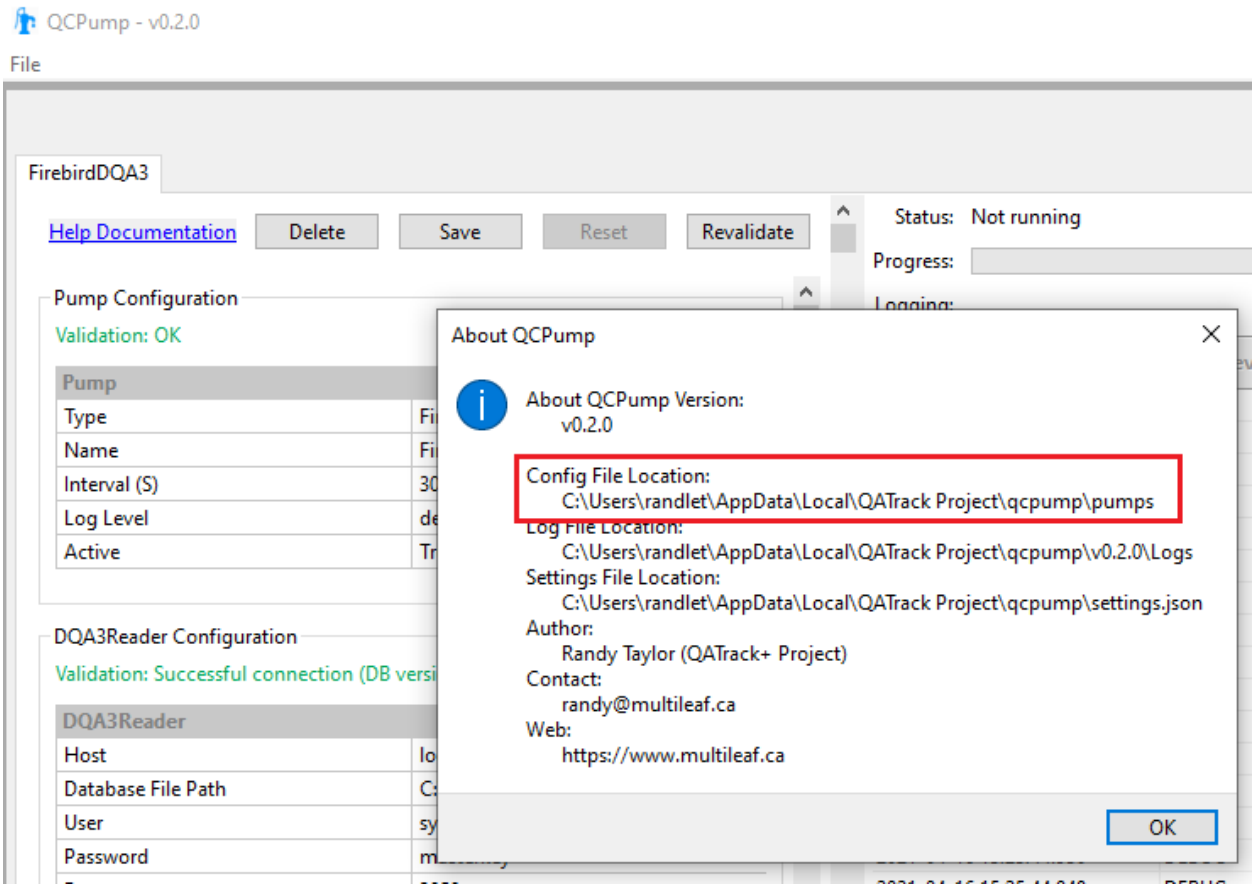
Create a directory somewhere to contain your pump types. I am going to use a directory in my home directory called *Pumps*:

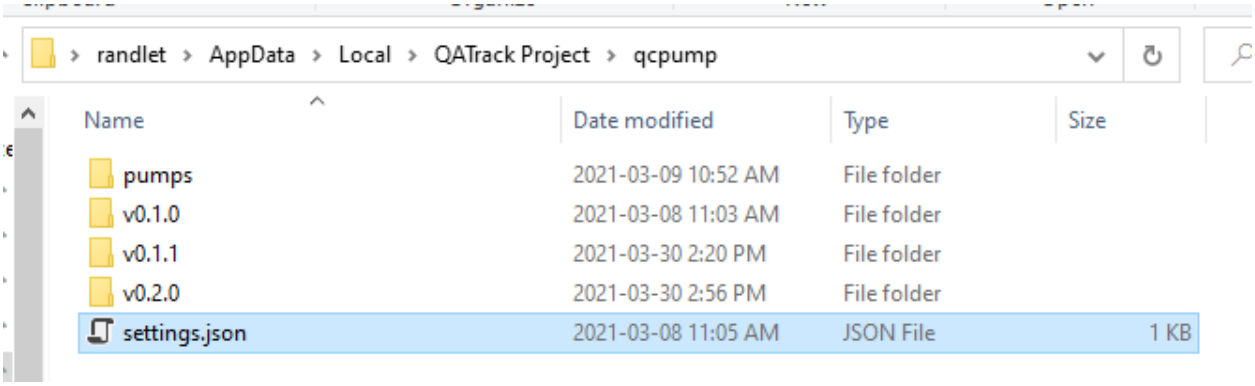


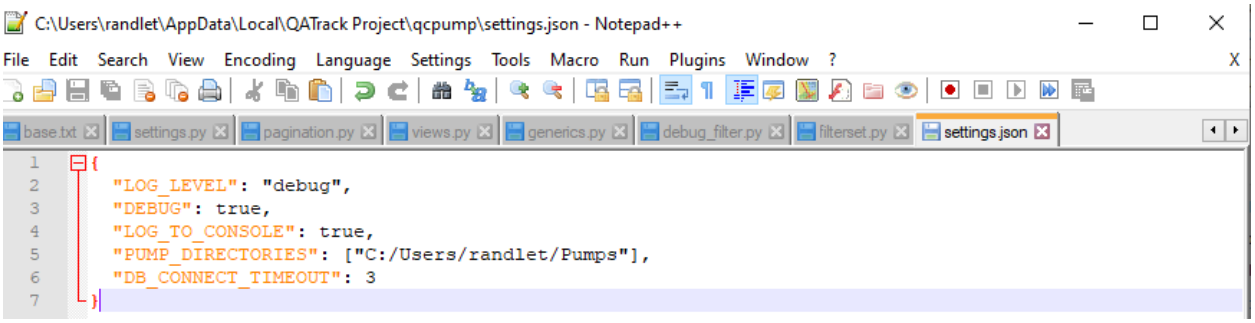
Now, find your QCPump settings file location file by launching QCPump and going to File->About:

Navigate to that directory and edit the settings.json file (you can just use notepad.exe or whatever your favourite editor is):

adding your new PumpType directory to the *PUMP_DIRECTORIES* list (use forward slashes / instead of back slashes \ !):


QCPump - v0.2.0
File
FirebirdDQA3
Help Documentation
Delete
Save
Reset
Revalidate
Status: Not running
Progress:
Pump Configuration
Validation: OK
Pump
Type
Name
Interval (S)
Log Level
Active
DQA3Reader Configuration
Validation: Successful connection (DB versi
DQA3Reader
Host
Database File Path
User
Password
About QCPump
About QCPump Version:
v0.2.0
Config File Location:
C:\Users\randlet\AppData\Local\QATrack Project\qcpump\pumps
Log File Location:
C:\Users\randlet\AppData\Local\QATrack Project\qcpump\v0.2.0\Logs
Settings File Location:
C:\Users\randlet\AppData\Local\QATrack Project\qcpump\settings.json
Author:
Randy Taylor (QATrack+ Project)
Contact:
randy@multileaf.ca
Web:
https://www.multileaf.ca
OK


randlet › AppData › Local › QATrack Project › qcpump
Name
Date modified
Type
Size
pumps
2021-03-09 10:52 AM
File folder
v0.1.0
2021-03-08 11:03 AM
File folder
v0.1.1
2021-03-30 2:20 PM
File folder
v0.2.0
2021-03-30 2:56 PM
File folder
settings.json
2021-03-08 11:05 AM
JSON File
1 KB


C:\Users\randlet\AppData\Local\QATrack Project\qcpump\settings.json - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
base.txt
settings.py
pagination.py
views.py
generics.py
debug_filter.py
filterset.py
settings.json
{
"LOG_LEVEL": "debug",
"DEBUG": true,
"LOG_TO_CONSOLE": true,
"PUMP_DIRECTORIES": ["C:/Users/randlet/Pumps"],
"DB_CONNECT_TIMEOUT": 3
}

7.2 Developing your own Pump Types

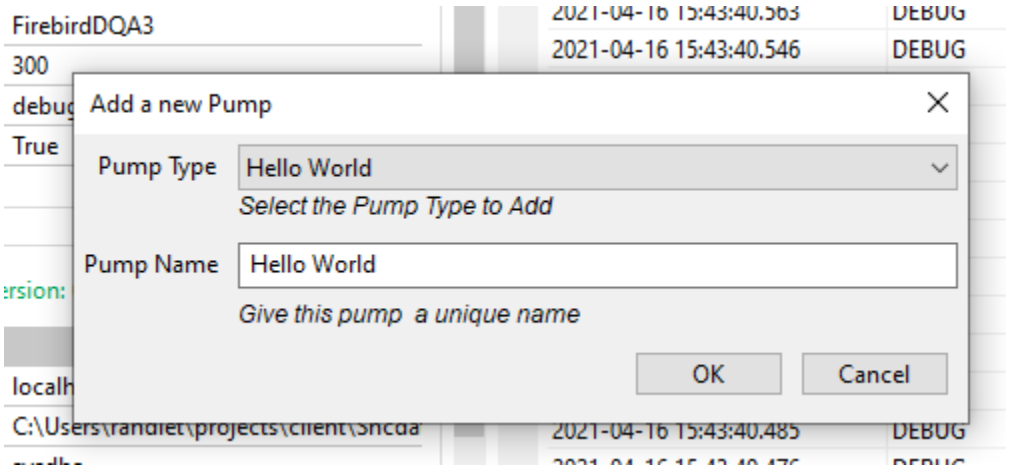
To create your own PumpType you will need to create a subclass of *qcpump.pumps.base.BasePump* and, at a minimum, implement a *pump* method.

First create a subdirectory in your pumps subdirectory called *HelloWorld* and in that directory create a file called *hello_world.py* with the following content:

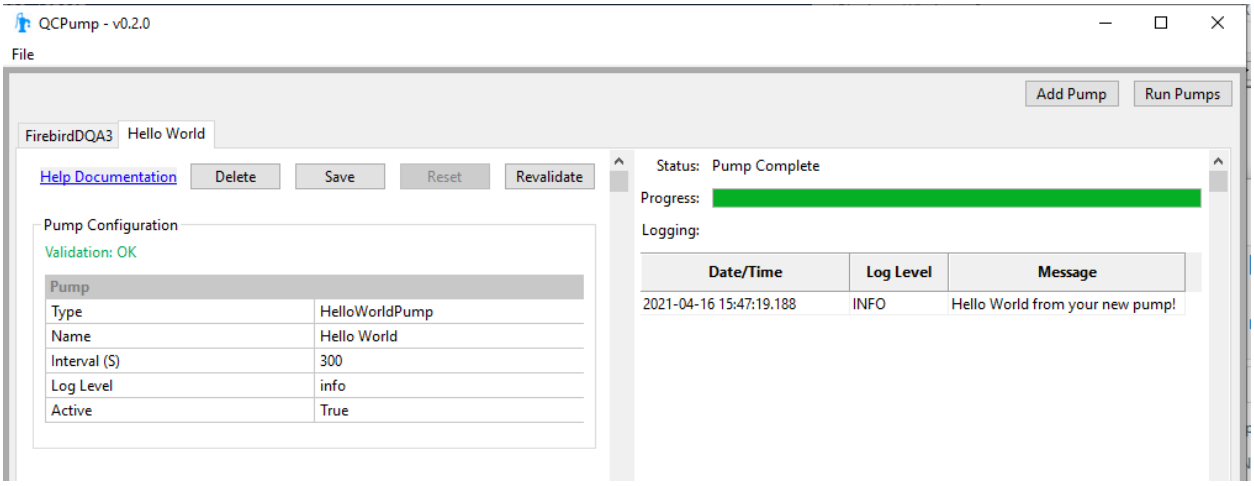
```
from qcpump.pumps.base import BasePump


class HelloWorldPump(BasePump):

    DISPLAY_NAME = "Hello World"

    def pump(self):
        self.log_info("Hello World from your new pump!")
```

This is the simplest possible pump; all it does is log a debug message every time QCPump calls its *pump* method.

Now launch QCPump and click add a new Hello World pump:



On the *Hello World* pump page, click *Save* then *Run Pumps* (note if you have other pumps configured you may want to deactivate them first!). You should see a message logged to the status pane of your Hellow World pump:



That's it! You've just written your first Pump Type. It doesn't do much, but should get you started on your journey.

7.3 Adding Configuration Options To Your Pump

Let's extend our Hello World Pump to log a message that can be customized by our users.

Modify your hello_world.py file so it has the following code:

```
from qcpump.pumps.base import BasePump, STRING

class HelloWorldPump(BasePump):

    DISPLAY_NAME = "Hello World"

    CONFIG = [
        {
            'name': 'Hello World',
            'fields': [
                {
                    'name': 'message',
                    'type': STRING,
                    'required': True,
                    'help': "Enter the message you want to see logged",
                    'default': "A default message",
                },
            ],
        },
    ]

    def pump(self):
        message = self.get_config_value("Hello World", "message")
        self.log_info(f"Hello World says: {message}")
```

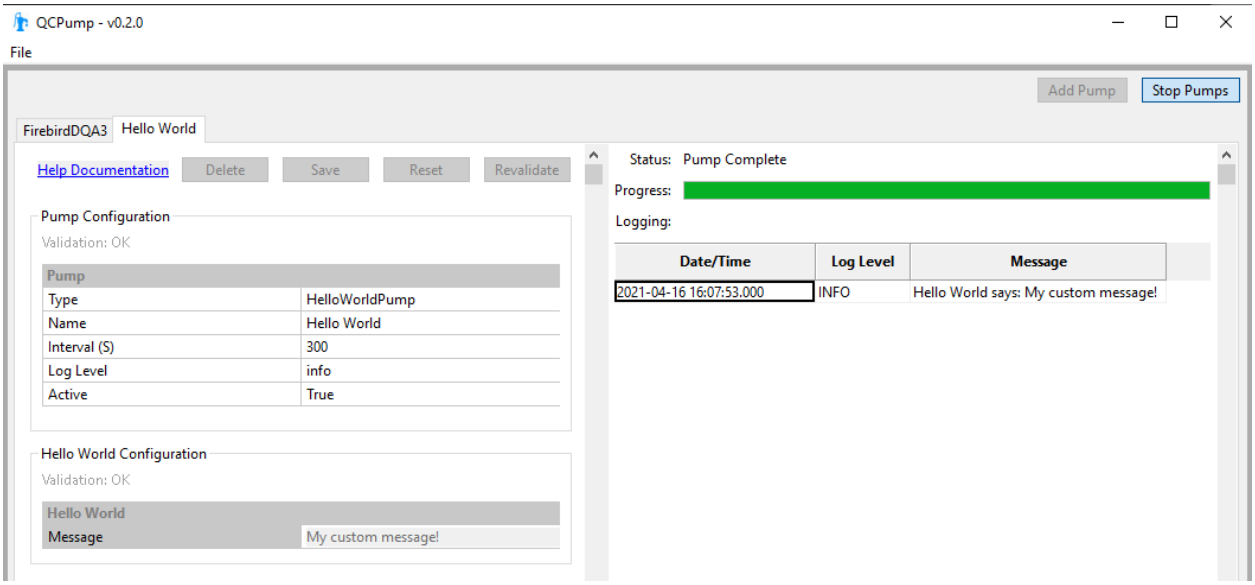
By adding the *CONFIG* option, we are telling QCPump we want our users to be able to configure an option called *message* in a configuration section called *Hello World* that will have a string type (see *Config Options* for more option types). If you relaunch QCPump, you should now see your new configuration option. Change the default message, click *Save*, and run the pump again and you should see it log your new custom message:

7.4 Adding Validation To Your Pump

QCPump allows you to do some data validation to ensure things are configured correctly. Modify your hello_world.py to add a validation method for our message option:

```
from qcpump.pumps.base import BasePump, STRING

class HelloWorldPump(BasePump):

    DISPLAY_NAME = "Hello World"

    CONFIG = [
        {
            'name': 'Hello World',
            'validation': 'validate_message',
            'fields': [
```

(continues on next page)



(continued from previous page)

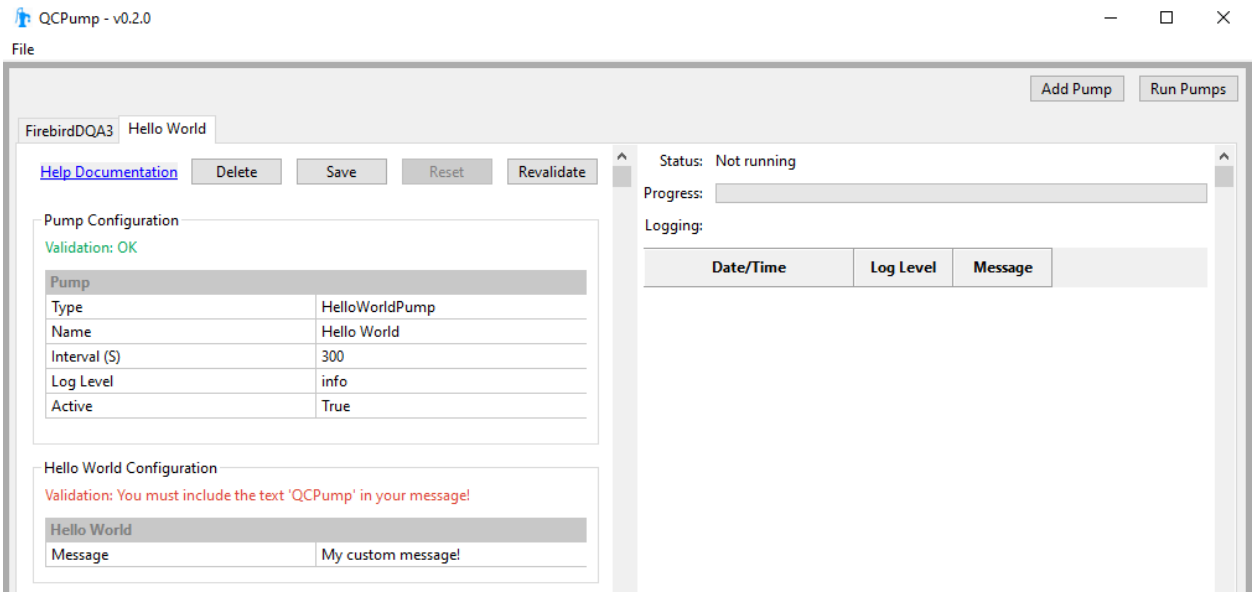
```
                {
                    'name': 'message',
                    'type': STRING,
                    'required': True,
                    'help': "Enter the message you want to see logged",
                    'default': "A default message",
                },
            ],
        },
    ]

    def pump(self):
        message = self.get_config_value("Hello World", "message")
        self.log_info(f"Hello World says: {message}")

    def validate_message(self, values):
        message = values['message']

        if "QCPump" not in message:
            valid = False
            message = "You must include the text 'QCPump' in your message!"
        else:
            valid = True
            message = "Thank you for including 'QCPump' in your message!"

        return valid, message
```

Here we have added a *validate_message* method that ensures our users have the text 'QCPump' in their message. The *values* variable is a dictionary with keys made up by the names of our configuration options from this section, and values made up of the current value the user has configured. Since our *Hello World* configuration section only has a single option, *values* is just a dictionary with a single key:

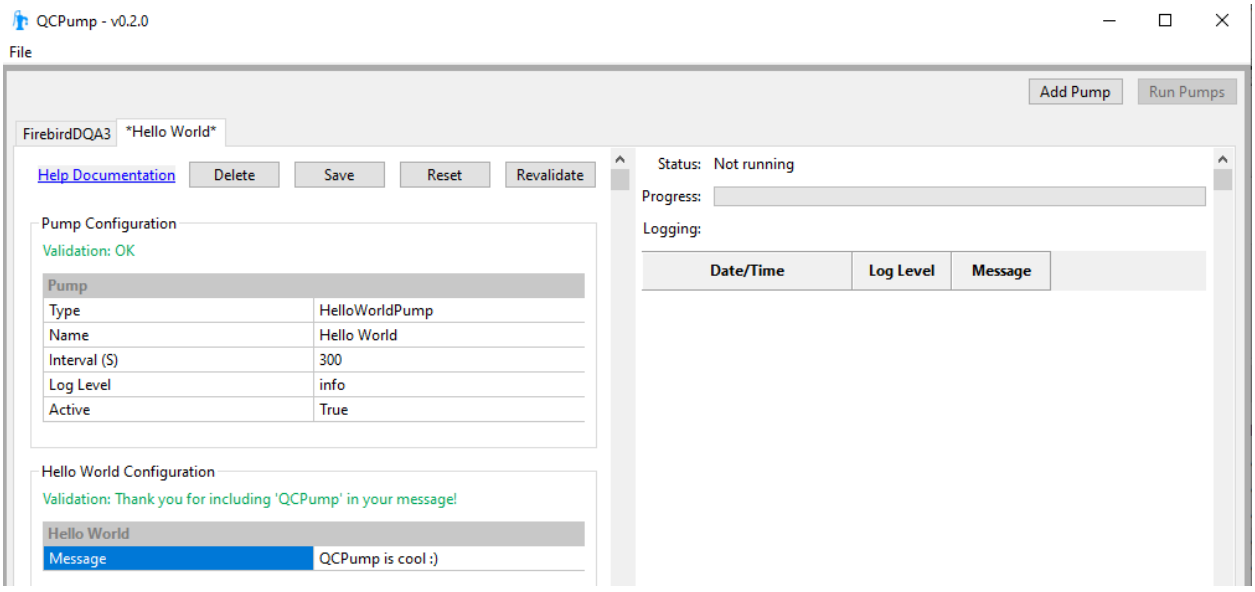
```
values == {
    'message': "Some configured message",
}
```

Validation methods must return an iterable of length two (e.g. a two tuple, or a list of two items) where the first item is True/False representing whether the configuration section is valid or not, and the second item is a string validation message which will be displayed to the user.

If you launch QCPump again and enter a value without the string 'QCPump' you should see your Pump will be marked as invalid:



Adding the text 'QCPump' will validate the pump:



For more information on writing QCPumps please see the information below.

7.5 Config Options

7.5.1 Option Types

You can add options of type:

STRING Short free form text

BOOLEAN A True/False drop down

INT An integer value

UINT = 'uint' A unsigned/positive integer value

FLOAT A decimal (floating point) value

MULTCHOICE A dropdown to select one option

DIRECTORY A path to a directory

7.5.2 Setting choices for Multiple Choice options

Todo: Choices Docs

7.5.3 Validation

Todo: Validation docs

7.6 Dependencies

Todo: Dependency Docs

7.7 QATrack+ Mixins

7.7.1 QATrackAPIMixin

Todo: QATrackAPIMixin docs

7.7.2 QATrackFetchAndPost

Todo: QATrackFetchAndPost docs

7.7.3 QATrackFetchAndPostTextFile

Todo: QATrackFetchAndPostTextFile docs

7.7.4 QATrackFetchAndPostBinarFile

Todo: QATrackFetchAndPostBinaryFile docs

CHAPTER

EIGHT

QC PUMP OVERVIEW

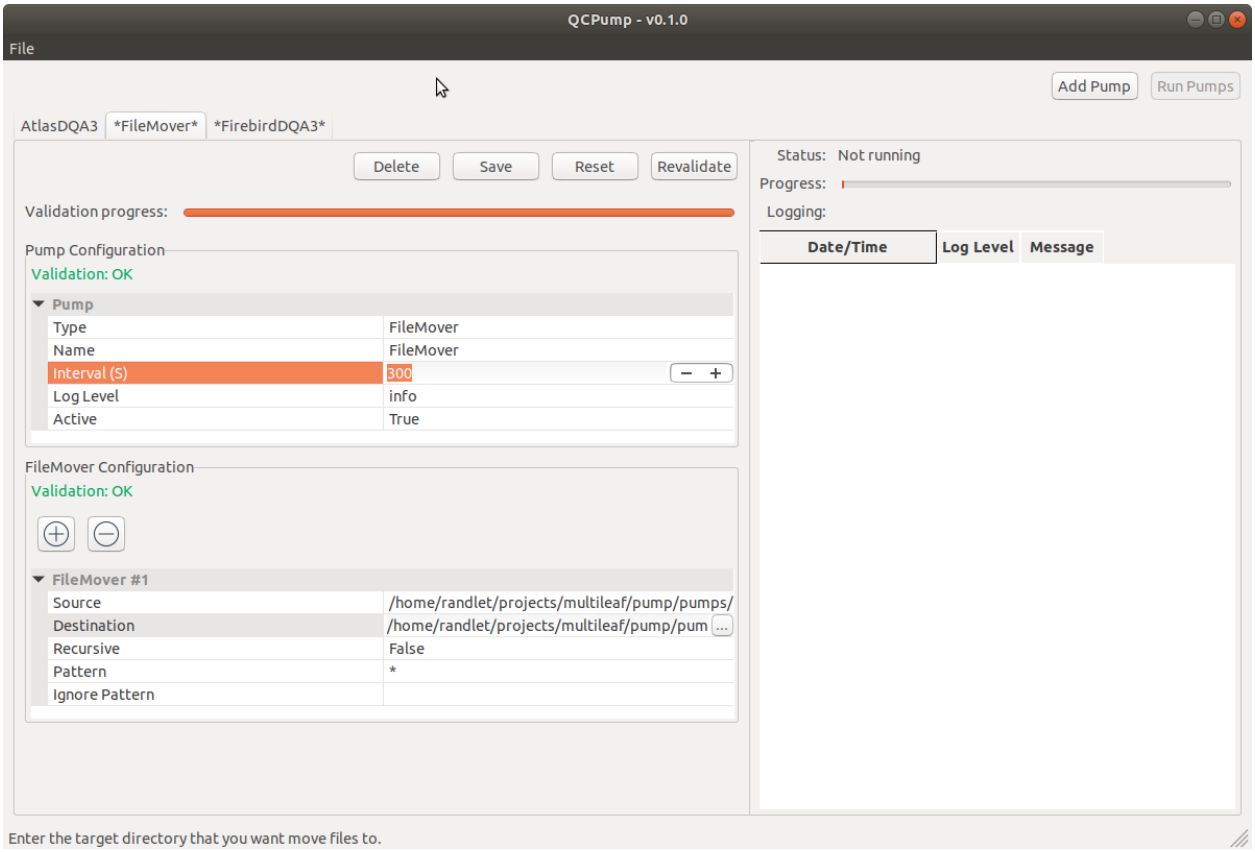


Fig. 1: QCPump Interface

QCPump is an extensible utility for extracting data from various sources (databases, filesystem etc) and moving it to other locations.

QCPump allows you to define one or more *Pumps* of various types (known as *Pump Types* e.g. *File Mover* and *Firebird DQA3*) to retrieve data from databases, filesystems, and other data sources. Typically these *Pumps* run at regular interval to look for new data, and when that data is present they extract it, potentially transform it in some way, and then upload it to QATrack+, move it somewhere else on disk, leave it alone, or take some other action.

QCPump currently includes the following Pump Types:

- *Simple File Mover & File Mover* which serve as examples for writing your pump types.

- *FirebirdDQA3 & AtlasDQA3* which can be used for retrieving Daily QA3 data from Firebird & Atlas (SQLServer) databases.

QC Pump was built primarily as a tool for uploading data to QATrack+, but there is nothing preventing you from using it for other tasks you want to perform periodically. For example, you could *write a Pump Type of your own* that ran once a day to generate a backup file of a database. If you think your *Pump Type* would be valuable to others please contribute it back to QCPump!

8.1 QC Pump License

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search