

MAINTENANCE GUIDE

1. GUIDING PRINCIPLE'S

1.1. Personal responsibility

2. RUN-TIMES STATES CHANGES

2.1. Check the status of the postgresql

2.2. Stop the postgresql

2.3. Start the postgresql

2.4. Check the port on which it is listening

2.5. Check the postgres status

2.6. Application Layer run-state management

2.6.1. Start the application layer

2.6.2. Stop the application layer

2.6.3. Restart OS network service

2.7. Security related operations

2.7.1. Add, modify and delete new users to the application

2.7.2. Regular users visibility

3. BACKUP AND RESTORE PROJECTS DATA

3.1. Load database connectivity configuration securely

3.2. Backup a database

3.3. Backup a database table

3.4. Backup only the database tables insert data

3.5. Restore a database

3.6. Alter table and reload the db meta data to the application layer

3.6.1. Restore a database from full database backup

3.6.2. Restore a database from db inserts file

3.7. Restore a database table

4. OTHER SHELL ACTIONS

4.1. Run increase-date action

4.2. Load xls sheet to db a doc table

5. BULK ISSUES MANAGEMENT

5.1. Opening a monthly period

5.1. Closing a monthly period

5.2. Publish the release issues

5.3. Dirs naming conventions

6. NAMING CONVENTIONS

6.1. Naming conventions for directories

6.2. Root Dirs naming conventions

7. SOURCE CODE MANAGEMENT

7.1. Configure and use git ALWAYS by using ssh identities

7.2. Aim for e2e traceability

7.3. Restart the application layer

8. KNOWN ISSUES AND WORKAROUNDS

8.1. Morbo is stuck

8.1.1. Symptoms

8.1.2. Probable root cause

8.1.3. Known solution and workaround

MAINTENANCE GUIDE

1. GUIDING PRINCIPLE'S

This section might seem too philosophical for a start, yet all the development in the qto has ATTEMPTED to follow the principles described bellow. If you skip this section now you might later on wonder many times why something works and it is implemented as it is ... and not "the right way".

Of course you are free to not follow these principles, the less you follow them the smaller the possibility to pull features from your instance(s) - you could even use the existing functionality to create a totally different fork with different name and start developing your own toll with name X - the authors give you the means to do that with this tool ... , but if you want to use and contribute to THIS tool than you better help defined those leading principles and follow them.

1.1. Personal responsibility

Any given instance of the qto should have ONE and only ONE person which is responsible at the end for the functioning of THE instance - so think carefully before attempting to take ownership of an instance. The author(s) of the code are not responsible for the operation, bugs or whatever happens to a new instance. As a responsible owner of an instance you could create, share and assign issues to the authors of the source code, yet there is no Service Level Agreement, not even promise to help.

2. RUN-TIMES STATES CHANGES

2.1. Check the status of the postgresql

To check the status of the postgresql issue:

```
sudo /etc/init.d/postgresql status
```

2.2. Stop the postgresql

To stop the postgresql issues:

```
sudo /etc/init.d/postgresql stop
```

2.3. Start the postgresql

To start the postgresql issues:

```
sudo /etc/init.d/postgresql start
```

2.4. Check the port on which it is listening

To check the port on which it is listening issue:

```
sudo netstat -tulntp | grep -i postgres
# tcp        0      0 127.0.0.1:5432      0.0.0.0:*
LISTEN      8095/postgres
```

2.5. Check the postgres status

Check the postgres status.
Check the port to which the Postgres is running with this command:

```
sudo /etc/init.d/postgresql status

# restart if needed
sudo /etc/init.d/postgresql restart

# check on which ports it is running
sudo netstat -plnt |grep postgres
```

2.6. Application Layer run-state management

Remember to cd to the product instance dir you are going to work on for example:
`cd /vagrant/opt/csiteda/qto/qto.0.6.5.dev.ysg`
All the examples bellow are assuming you've done that in advance.

2.6.1. Start the application layer

To start the application layer in development mode use the morbo command (debug output will be shown) , to start it in production mode use the hypnotoad pattern

```
# start hypnotoad ( does the stop as well )
bash src/bash/qto/qto.sh -a mojo-hypnotoad-start

# start morbo
bash src/bash/qto/qto.sh -a mojo-morbo-start
```

2.6.2. Stop the application layer

To stop the application layer in development mode use the morbo command (debug output will be shown) , to start it in production mode use the hypnotoad pattern. Note that the morbo command does not stop any running morbo on OTHER product instance dir, but the hypnotoad does stop all - aka hypnotoad as the binary of production must be running only on 1 and only one instance on a host.

```
# only stop hypno
bash src/bash/qto/qto.sh -a mojo-hypnotoad-stop

# only stop morbo
bash src/bash/qto/qto.sh -a mojo-morbo-stop
```

2.6.3. Restart OS network service

Sometimes you might just need to restart the whole network service on Ubuntu:

```
sudo /etc/init.d/networking restart
# or
bash src/bash/qto/qto.sh -a restart-network # this one
restarts the postgres as well
```

2.7. Security related operations

There are 2 security modes of operations in qto:
- none authenticative one (no login , all can be changed by anyone)
- native authentication mode - the user credentials are stored per db

Add, modify and delete new users to the application

2.7.1.

You as the owner of the instance you are running must be aware that the requests to register to the instances you are operating will come via e-mail. Simply add, update and delete users in the users table and sent the password with prompt to edit it to the new user.

2.7.2. Regular users visibility

Use the following http password generator:
<http://www.htaccesstools.com/httpasswd-generator/>

3. BACKUP AND RESTORE PROJECTS DATA

You could easily add those commands to your crontab for scheduled execution - remember to add the absolute path of the qto.sh entry script. Anything you perform as shell action in qto could be applied not only to the current product instance dir, but also any other project instance dir, which has a directory structure, which is compatible with the current qto product.

Load database connectivity configuration securely

3.1.

Qto provides you with the means and tools to work on tens of databases, yet one at the time. Thus once you open a shell to run the tools you must have the connectivity to the database you want to work on.

```
source lib/bash/fncs/export-json-section-vars.sh
# optionally use a different project than the current
product instance dir
export PROJ_INSTANCE_DIR=/hos/opt/kone/kone-qto

# optionally use a different configuration file for this proj
instance dir
export PROJ_CONF_FILE=/hos/opt/org/org-
qto/cnf/env/tst.env.json

# load the env vars from this project
doExportJsonSectionVars $PROJ_CONF_FILE '.env.db'

# set the psql with the correct credentials valid ONLY for
this proj
alias psql="PGPASSWORD=${postgres_db_useradmin_pw:-} psql -v
-t -X -w -U ${postgres_db_useradmin:-} --port
$postgres_db_port --host $postgres_db_host"

# now you can run any psql
psql -d my_db -c "\d1"
```

3.2. Backup a database

You backup a database (all the objects, roles and data) with the following one-liner.

```
# obs you must have the shell vars pre-loaded !!! Note dev,
tst or prd instances !
# clear; doParseCnfEnvVars cnf/qto.prd.host-name.cnf
bash src/bash/qto/qto.sh -a backup-postgres-db
```

3.3. Backup a database table

You backup a database table with the following one-liner. Noe

You restore a database by first running the pgsq scripts of the project database and then restoring the insert data

```
# obs you have to have the shell vars pre-loaded !!!
# clear; doParseCnfEnvVars <<path-to-cnf-file>>
bash src/bash/qto/qto.sh -a backup-postgres-table -t
my_table
```

3.4. Backup only the database tables insert data

```
# obs you have to have the shell vars pre-loaded !!!
bash src/bash/qto/qto.sh -a backup-postgres-db-inserts
```

3.5. Restore a database

```
psql -d $postgres_db_name < \
dat/mix/sql/pgsql/dbdumps/dev_qto/dev_qto.20180813_202202.in
srt.dmp.sql
```

3.6. Alter table and reload the db meta data to the application layer

The qto does load the db meta-data it needs for the application layer during startup and ALWAYS when loading one of the following tables - meta_tables, meta_columns, items_doc. Thus each time you alter a table which is used by the list pages you must manually reload one of those tables afterwards, which is much less invasive than full application restart by the way, not to mention any non-dynamic UI applications which WILL require coding work to adapt the UI to the new database structure ...

```
psql -d tst_qto -c "alter table monthly_issues_202001 add
column tags varchar(10) defaultl 'tag'";
psql -d tst_qto -c "alter table monthly_issues_202001 drop
column tags"
```

3.6.1. Restore a database from full database backup

You restore a database by basically creating first the empty database and applying the dump file to that database. Practice this several times in dev before going to tst !!!

```
# if you DO HAVE THE DB - probably not a bad idea to first
backup it !!!
# bash src/bash/qto/qto.sh -a backup-postgres-db
# psql -d postgres -c "CREATE DATABASE dev_qto;"
# drop it if it was there ...
# psql -d postgres -c "DROP DATABASE dev_qto;"

# create the db
psql -d postgres -c "CREATE DATABASE dev_qto;"

psql -d dev_qto < dat/mix/2020/2020-01/2020-01-
11/sql/tst_qto/tst_qto.20200111_195947.full.dmp.sql
```

3.6.2. Restore a database from db inserts file

This type of restore assumes you are 100% sure that the schema you took the db inserts backup from is the same as the schema you are applying the db inserts to, which should be the case when you are restoring data from the same qto version db. In qto we do not believe in migrations, which is a totally different discussion, but if you do not have the same schema you WILL HAVE errors and you ARE basically on your own, because you ended-up here by basically NOT taking regularly backups and not applying regular updates and that is YOUR fault and not the fault of the product instance owner you are getting the qto source from

```
# if you DO HAVE THE DB - probably not a bad idea to first
backup it !!!
# bash src/bash/qto/qto.sh -a backup-postgres-db
# psql -d postgres -c "CREATE DATABASE dev_qto;"
# drop it if it was there ...
# psql -d postgres -c "DROP DATABASE dev_qto;"

bash src/bash/qto/qto.sh -a run-qto-db-ddl

psql -d $postgres_db_name < \
dat/mix/sql/pgsql/dbdumps/dev_qto/dev_qto.20180813_202202.in
srt.dmp.sql
```

3.7. Restore a database table

You restore a database table by first running the pgsql scripts of the project database or ONLY for that table and then restoring the insert data from the table insert file.

```
# re-apply the table ddl
psql -d $postgres_db_name < src/sql/pgsql/dev_qto/13.create-
table-requirements.sql
```

4. OTHER SHELL ACTIONS

We are not going to lie ... Basically there are a lot of automated shell actions - aka small bash functions which are wrappers around tasks, which usually if you would be doing maintenance and operations on a different application would be your own scripts or automations or simply cheat sheet MIGHTY oneliners. In the spirit of qto - those ARE also part of the source code base of the application, that is shareable to all the other users ... Thus should you figure out a new and/or different and/or even better way of doing something in even other run-times than bash, perl, python or whatever everything has been used in this project start by reading the [devops_guide_doc-0](#) and creating a Merge Request via GitHub !

```
while read -r f ; do echo `echo $(basename $f)|sed -e
's/.func.sh//g'`; done < <(find "src/bash/qto/funcs" -type f
-name '*.sh'|sort) | less
```

4.1. Run increase-date action

You track the issues of your projects by storing them into xls files in "daily" proj_txt dirs. Each time the day changes by running the increase-date action you will be able to clone the data of the previous date and start working on the current date.

```
bash src/bash/qto/qto.sh -a increase-date
```

4.2. Load xls sheet to db a doc table

To load xls issues to db and from db to txt files

```

export do_truncate_tables=0 ;
clear ; perl src/perl/qto/script/qto.pl --do xls-to-db --
tables requirements_doc

# check that the rows where inserted
psql -d dev_qto -c "SELECT * FROM requirements_doc;"

```

5. BULK ISSUES MANAGEMENT

By bulk issues management herewith is meant the bulk handling via sql to the issues items via the psql binary, which IS basically just moving data from table to table provided they have the same set of columns.

5.1. Opening a monthly period

Each monthly period is basically a table with the naming convention `monthly_issues_YYYYMM` for example the monthly period for the year 2020 January will be the `monthly_issues_202001`.

To create the table you need to basically copy the `monthly_issues` table to the one with the period at the end and replace the `monthly_issues` with the `monthly_issues_<<yyyymm>>` string and run the table as follows.

```

# copy the table
cp -v src/sql/pgsql/qto/tables/03.create-table-
monthly_issues.sql \ src/sql/pgsql/qto/tables/2020/2020-
01/create-table-monthly_issues_202001.sql

# source the env vars loading func
source lib/bash/funcs/export-json-section-vars.sh

# load the env vars
doExportJsonSectionVars cnf/env/dev.env.json '.env.db'
psql -d dev_qto < src/sql/pgsql/qto/tables/2020/2020-
01/create-table-monthly_issues_202001.sql

# generate the sql for moving the data between the two
tables
bash src/tpl/psql-code-generator/psql-code-generator.sh
tst_qto monthly_issues monthly_issues_202001

# copy paste the generated code as follows to "save" it into
the $sql_code bash variable
IFS='' read -r -d '' sql_code <<"EOF_CODE"
    INSERT INTO monthly_issues_202001
( guid , id , type , category , status , prio , name ,
description , owner , update_time )
    SELECT guid , id , type , category , status , prio ,
name , description , owner , update_time
    FROM monthly_issues
    ON CONFLICT (id) DO UPDATE SET
    guid = excluded.guid, id = excluded.id, type =
excluded.type, category = excluded.category, status =
excluded.status, prio = excluded.prio, name = exclu
ded.name, description = excluded.description, owner =
excluded.owner, update_time = excluded.update_time;
EOF_CODE

# run the sql code by pointing to the bash variable
psql -d tst_qto -c "$sql_code"

```

5.1. Closing a monthly period

Closing a monthly period would simply mean to ensure that all the done issues will be moved to the release_issues table

```
bash src/tpl/psql-code-generator/psql-code-generator.sh
dev_qto monthly_issues_202001 release_issues
IFS='' read -r -d '' sql_code <<"EOF_CODE"
    INSERT INTO release_issues
        ( guid , id , type , category , status , prio , name ,
description , owner , update_time )
    SELECT guid , id , type , category , status , prio ,
name , description , owner , update_time
    FROM monthly_issues_202001
    ON CONFLICT (id) DO UPDATE SET
        guid = excluded.guid, id = excluded.id, type =
excluded.type, category = excluded.category, status =
excluded.status, prio = excluded.prio, name = excluded.name,
description = excluded.description, owner = excluded.owner,
update_time = excluded.update_time;
EOF_CODE
psql -d tst_qto -c "$sql_code"
psql -d tst_qto -c "delete from release_issues where status
<> '09-done';"
```

5.2. Publish the release issues

You publish the release issues by moving all the issues with '09-done' status to the release_issues table

5.3. Dirs naming conventions

The dir structure should be logical and a person navigating to a dir should almost understand what is to be find in thre by its name ..

6. NAMING CONVENTIONS

6.1. Naming conventions for directories

The dir structure should be logical and a person navigating to a dir should almost understand what is to be find in thre by its name ..

6.2. Root Dirs naming conventions

The root dirs and named as follows:
bin - contains the produced binaries for the project
cnf - for the configuration
dat - for the data of the app
lib - for any external libraries used
src - for the source code of the actual projects and subprojects

7. SOURCE CODE MANAGEMENT

The qto is a derivative of the wrapp tool - this means that development and deployment process must be integrated into a single pipeline.

Configure and use git ALWAYS by using ssh identities

7.1.

You probably have access to different corporate and public git repositories. Use your personal ssh identity file you use in GitHub to push to the qto project. The following code snippet demonstrates how you could preserve your existing git configurations (even on corporate / intra boxes) , but use ALWAYS the personal identity to push to the qto...


```
# create the company identity file
ssh-keygen -t rsa -b 4096 -C "first.last@corp.com"

# save private key to ~/.ssh/id_rsa.corp,
cat ~/.ssh/id_rsa.corp.pub

# copy paste this string into your corp web ui security ssh
keys

# create your private identify file
ssh-keygen -t rsa -b 4096 -C "me@gmail.com"
# save private key to ~/.ssh/id_rsa.me, note the public key
~/.ssh/id_rsa.me.pub
cat ~/.ssh/id_rsa.me.pub # copy paste this one into your
githubs, private keys

# set alias for the git command to avoid overtyping ...
alias git='GIT_SSH_COMMAND="ssh -i ~/.ssh/id_rsa.ysg " git'

# clone a repo
git clone git@git.in.corp.com:corp/project.git

export git_msg="my commit msg with my corporate identity,
explicitly provide author"
git add --all ; git commit -m "$git_msg" --author "MeFirst
MeLast <first.last@corp.com>"
git push
# and verify
clear ; git log --pretty --format='%h %ae %<(15)%an ::: %s
```

7.2. Aim for e2e traceability

Aim for traceability between user-stories, requirements, features and functionalities

Once the issues are defined and you start working on your own branch which is named by the issue-id aim to map one on one each test in your code with each listed requirement in qto.

7.3. Restart the application layer

Well just chain the both commands.

```
# start does actually re-start always
bash src/bash/qto/qto.sh -a mojo-morbo-start
```

8. KNOWN ISSUES AND WORKAROUNDS

8.1. Morbo is stuck

8.1.1. Symptoms

This one occurs quite often , especially when the application layer is restarted, but the server not

This one occurs quite often , especially when the application layer is restarted, but the server not

List the running perl processes which run the morbo and kill the instances

```
# the error msg is
[INFO ] 2018.09.14-10:23:14 EEST [qto][@host-name] [4426]
running action :: mojo-morbo-start:doMojoMorboStart
(Not all processes could be identified, non-owned process
info
will not be shown, you would have to be root to see it
all.)
tcp      0      0 0.0.0.0:3001        0.0.0.0:*
LISTEN   6034/qto
tcp      0      0 0.0.0.0:3002        0.0.0.0:*
LISTEN   7626/qto
Can't create listen socket: Address already in use at
/usr/local/share/perl/5.26.0/Mojo/IOLoop.pm line 130.
[INFO ] 2018.09.14-10:23:16 EEST [qto][@host-name] [4426]
STOP FOR qto RUN with:
[INFO ] 2018.09.14-10:23:16 EEST [qto][@host-name] [4426]
STOP FOR qto RUN: 0 0 # = STOP MAIN = qto
qto-dev ysg@host-name [Fri Sep 14 10:23:16]
[/vagrant/opt/csited/qto/qto.0.4.9.dev.ysg] $
```

8.1.2. Probable root cause

```
# the error msg is
[INFO ] 2018.09.14-10:23:14 EEST [qto][@host-name] [4426]
running action :: mojo-morbo-start:doMojoMorboStart
(Not all processes could be identified, non-owned process
info
will not be shown, you would have to be root to see it
all.)
tcp      0      0 0.0.0.0:3001        0.0.0.0:*
LISTEN   6034/qto
tcp      0      0 0.0.0.0:3002        0.0.0.0:*
LISTEN   7626/qto
Can't create listen socket: Address already in use at
/usr/local/share/perl/5.26.0/Mojo/IOLoop.pm line 130.
[INFO ] 2018.09.14-10:23:16 EEST [qto][@host-name] [4426]
STOP FOR qto RUN with:
[INFO ] 2018.09.14-10:23:16 EEST [qto][@host-name] [4426]
STOP FOR qto RUN: 0 0 # = STOP MAIN = qto
qto-dev ysg@host-name [Fri Sep 14 10:23:16]
[/vagrant/opt/csited/qto/qto.0.4.9.dev.ysg] $
```

8.1.3. Known solution and workaround

```
ps -ef | grep -i perl

# be carefull what to kill
kill -9 <<proc-I-know-is-the-one-to-kill>>
```

