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 wander 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 runnning
sudo netstat -plunt |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/csitea/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 hypntotoad 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

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

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/htpasswd-generator/

3. BACKUP AND RESTORE PROJECTS DATA

You could easily add those commands to your crontab for scheduled execution - remember to add the absolution patch 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.

3.1. Load database connectivity configuration securely

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/funcs/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 differnt 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 "\dl"
```

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 pgsql scripts of the project database and than 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. Restore a database

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

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

3.5. 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 than 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. SHELL ACTIONS

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. NAMING CONVENTIONS

5.1. 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 ..

5.2. 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

6. 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.

6.1. Configure and use git ALWAYS by using ssh identities

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
```

6.2. 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.

6.3. 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
```

7. KNOWN ISSUES AND WORKAROUNDS

7.1. Morbo is stuck

7.1.1. Symptoms

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

```
# 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/csitea/qto/qto.0.4.9.dev.ysg] $
```

7.1.2. Probable root cause

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

```
# 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/csitea/qto/qto.0.4.9.dev.ysg] $
```

7.1.3. Known solution and workaround

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

```
ps -ef | grep -i perl

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