# QRunner documentation

The idea was born thinking how could be possible and feasible to create an application that was able to load and execute scripts (bash, perl, python, ...). This can make a testing process more automated if you think to prepare a bunch of scripts and let start all together to value the performance of an embedded device as can be a GNU/Linux gateway connected to you machine or simply to start a certain number of operations using scripts, think for example to compile the whole KDE and see the compiling errors/messages the task can produce. Using QRunner you can organize the scripts and run it and see the output in real time or later when you want.

## Ideas for the specification

The application has to let you load scipt file and execute them all together to get their output and to store it in a text file as log or in database (if needed). In a second release it could be possible to create some plugin to create a specific protocol to test well know boxes with a known specification.

Now it could/should be possible to (the implemented features have an * at the end):

1. create a script group*;
2. create more sub groups inside a group/subgroup*;
3. make possible to enable/disable the execution of a script group*;
4. make possible to enable/disable the execution of a single script that is part of a group/subgroup*;
5. move a script/group/subgroup into another group/subgroup as you want using drag & drop*;
6. load the scripts you keep in a directory of your system and don't go further with the subdirectories*;
7. set an environment for a script defining some environment variables*;
8. set parameters to pass through the script*;
9. define the script interpreter;
10. execute the script as a different user;
11. execute a certain number of times the scripts or the scripts group*;

12. save the virtual directories tree of the scripts within a project*;

13. load the virtual directories tree of the scripts created and saved within a previous project*;

14. load a virtual directory sub-tree starting from a specific point of the tree scripts.

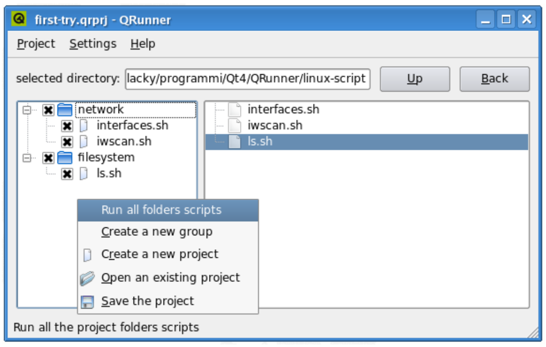
## A first preliminary analysis and possible use of the application

When I started writing the application I thought to keep it as simple as it could be from a end user point of view, than I focused on the use of the drag & drop so to let users to drag the scripts file directly in the script virtual directories.

The GUI is composed by four main "widgets":

1. a text box, not editable directly, that is the local file system directory from where you can get the scripts file;
2. a scripts virtual directory tree;
3. a file/directory tree related to the directory of the file system specified at the point 1;
4. a text area where you can read the output of the command executed by the scripts (stderr and stdout). You simply have to click on the script yu want to see it's output in real time when it's running.

Here a first screenshot directly from the application:



selected directory: lacky/programmi/Qt4/QRunner/linux-script

Run all the project folders scripts

***figure 1:*** *QRunner after it loaded a project and with the context menu active*

### *Menu*

The application commands are available from the menu placed on the top bar and from a context menu that you can get clicking on the right mouse button in the tree script area as you can see in the *figure 1*.

Three menu are available:

1. Project;
2. Edit;
3. Help.

**Project** menu has the project related items:

- New
- Open...
- Save
- Save as...
- Recent opened files
- Run all scripts directories
- Exit.

With this menu you can create a new project (choose *new* menu item), so the current will be removed with all its groups, subgroups and scripts files and if you didn't saved the project QRunner will ask you for it.

You can load a previous saved project (choose *open* menu item), or save the current one (choose *save* menu item) or rename the project file during the save (choose *save as...* menu item).

You can execute all the scripts directories in the project, of course the disabled groups/subgroups/scripts will not be executed. A group/subgroup/script is disabled if the checkbox on the left of the name is not checked. To disable a group means that the related branch, with all its subgroups and scripts, will not be executed.

With the **Edit** menu you can change the file system directory from where you can drag the scripts. At the QRunner start up it is the current directory form where you execute QRunner itself (in this case you can get the available script example); the absolute path is displayed in the text box in the top of the GUI and its content (files and directories) is displayed as a tree (see on the right of the GUI). You can change the directory from the *Edit* menu, but also, quickly, clicking on the text box as told (on the right of the *selected directory* as displayed in the *figure 1*).

## *The context Menu*

With the context menu, that you get clicking on the right button of the mouse when the mouse pointer is in the script directories area, you can also execute specific action related to the selected object type (group/subgroup or script or nothing) and than you can, for example, create, quickly, a new group/subgroup, remove it, execute only the corresponding sub-tree scripts.

Here an example of the context menu when you click on a group/subgroup widget:

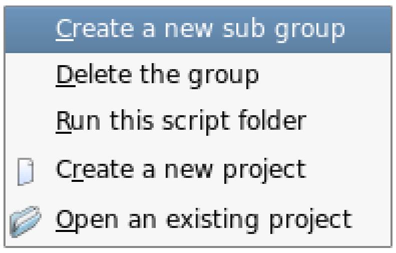


***figure 2:** the RMB on a group/subgroup widget*

When you click on a script, instead ("Save the project" appears if the project has been modified) you get:

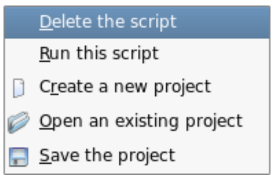


**figure 3:** the RMB on a script widget

In the next screenshot you can see the context menu when the project is empty, that is when QRunner has been just executed for the first time or when a new project (empty) has been created:

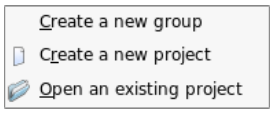


**figure 4:** the RMB at the start up (or when you got a new project)

## *Project format file to save*

When you save a QRunner project (extention .qrprj) this is saved as an XML file with a specific format. This file contains all the information needed to recreate the project with all its scripts settings and various settings.

A tipical example is:

```
<?xml version="1.0" ?>

<project version="1.0" >

 <group checked="true" name="network" >

  <subgroup checked="true" name="one" >

   <file path="/home/unruhe/QRunner/linux-script" checked="true" name="interfaces.sh" />

  </subgroup>
  <subgroup checked="true" name="two" >
   <file path="/home/unruhe/QRunner/linux-script" checked="true" name="iwscan.sh" />
  </subgroup>
 </group>
 <group checked="true" name="filesystem" >
  <file path="/home/unruhe/QRunner/linux-script" checked="true" name="ls.sh" />
   <environment>
    <env value="1" name="env1" />
    <env value="3" name="env2" />
    <env value="2" name="env3" />
   </environment>
 </group>
</project>
```

And it represents the scripts tree reported in the figure below:

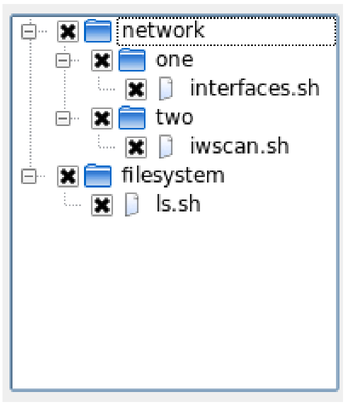


**figure 5:** a project tree

The XML project file structure starts with the **project** *tag* that has *version* attribute to specify the version of the XML file, so to execute a correct parsing. This *tag* has so much **group** *tags* for how many groups you have created. The **subgroup** *tag* is generic for all generic subgroup you have, in fact you can have a subgroup that is child of a group, or a subgroup that is child of another subgroup, and so on. Both the **group** *tag* and the **subgroup** *tag* use the attributes:

- checked;
- name.

***checked*** specifies if the item is enabled (*checked="true"*) or not (*checked="false"*) to execute the script or the scripts hold in the related branch tree.

The ***name*** attribute specifies a “widget” related alphanumeric identifier (the group/subgroup name).

The **file** *tag* specifies the script to execute and it has got four attributes:

- path;
- checked;
- name;
- times.

The *path* attribute is related to the *script file absolute path* inside the file system. The *checked* attribute specifies if the related script can be executed or not. The *name* attribute specifies the file name itself and the *times* attribute says how many times the script has to be executed, if this attribute is absent, than the script is executed just once.

Inside the **file** *tag* you can find the **environment** *tag* that have a list of **env** tags so to define all the wanted environment variables needed as the following:

```
<environment>
 <env value="1" name="env1" />
 <env value="3" name="env2" />
 <env value="2" name="env3" />
</environment>
```

That define the three environment variables:

1. *env1* with value 1;
2. *env2* with value 2;
3. *env3* with value 3.

## Test session

To start a test session (starting a lot of scripts together), you need to create all the groups and subgroups you want so you will have a script virtual directories tree (on the left part of the QRunner GUI). Once you created the structure of these virtual directories, you drag and drop the related script files you want to execute from the file system tree (see the right part of the GUI). Each script file is configurable using a series of parameters (you can define it for each different scripts). Of course you can execute also applications, not just single script. The important thing is that on GNU/Linux they have the executable attribute. On Windows they can be EXE file or .BAT file. In a future version of the QRunner could be possible using the QFreeDesktopMime class.

QRunner could filter file system files getting just scripts and executable file and using the correct interpreter (bash, perl, python, ...) thanks to the its MIME (PDF, JPG, perl script, ...). And of course you can't drag and drop the same script file in the same group/subgroup directory because you can't have more then one time the same file. It's enough to specify to execute it the number of times you want and have it just once in the virtual directory. The full parameters list to configure a script with QRunner is:

- the number of times the script has to be executed;
- the definable environment variables to execute the script;
- the input parameters for the script.

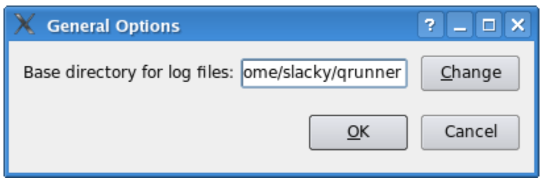
The ***the number of times the script has to be executed*** means that this will be executed more times, so as defined by the specified value.

The ***definable environment variables to execute the script*** can set a different operative environment for each script with the purpose to get different results for the same script if executed in a different environment. The environment is intended as a different set of values for the operative system environment variable.

The ***input parameters for the script*** let you give some input parameters to the script that it can use during its execution. Imagine parametric scripts, for general purpose, that, according to the assigned input parameters, you can have different results.

The script output data are written into a log file and are visible in a text area on the bottom of the GUI. This area will be shown when you click on the script you want to check: you will see to appear on the bottom of the GUI the text output/error the script has produced till when you clicked on it and you will see all the other output it will produce in the meanwhile. In future version you could save this information into a database too. When you want you can click on the right button of the mouse on the script name and open the related log file in text editor. The file is saved in the related virtual tree directories starting from the directory "qrunner" in the home user directory. So if you have the script ***list.sh*** under the sub group "case2" under the group "main" you will have the file into the directory ***$HOME/qrunner/main/case2/list.sh.log*** .

You can also change the base directory where the log files are saved. Simply go to the menu *Settings/Options*. You will get the dialog as in *figure* 6.



Base directory for log files: ome/slacky/qrunner

**figure 6:** setting the base directory for the log file

## Use case

Here reported the use case diagram.

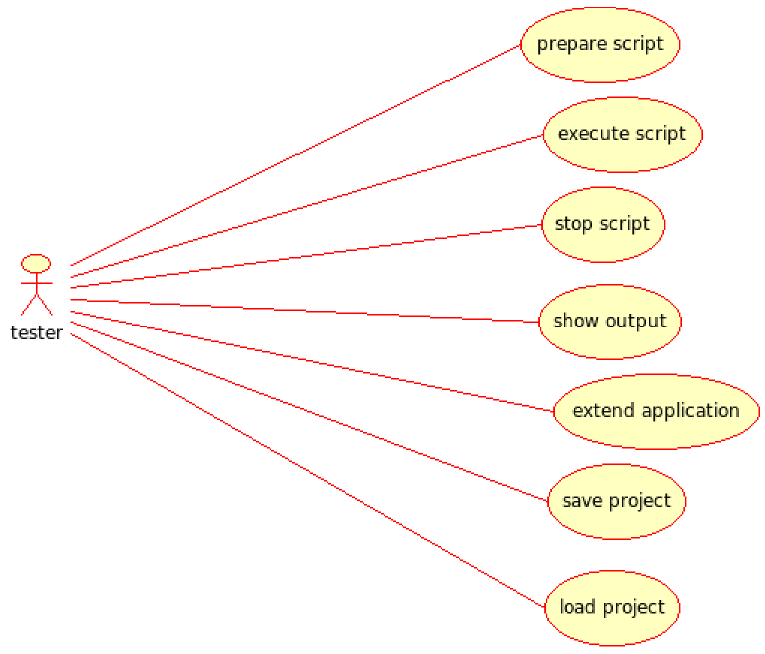


**Figure 7:** use case

The *actor* is represented by the person that has to execute tests. The following use case are available:

1. prepare script;
2. execute script;
3. stop script;
4. display output;
5. extend application;
6. save project;
7. load project.

***Pre-conditions***:

- Availability of the script file in a specific file system directory of the machine that execute the tester application.

***Prepare script***:

- create the "scripts group" directory;

- create the "sub groups scripts" directory;

- add the script file in the wanted sub directories with a drag and drop;

- repeat all the previous steps till when the project is complete.

***Execute script**:*

- enable/disable of the related file/group/sub group script to execute/don’t execute it;

- execute all the project enabled scripts or execute all the enabled scripts part of a script directory/sub directory;

- save all text output into a **log file**.

***Stop script**:*

- stop the script before they are naturally stopped.

***Show output**:*

- show the standard output and the standard error channel produced by the script execution selecting it.

***Save project**:*

- save on an XML file the scripts virtual directories/sub directories structure and the relative properties (enabled/disabled, number of time to execute a script, ...)

***Load project**:*

- load the script virtual directories/sub directories structure with the assignment of all related properties for the execution.

***Application extensions**:*

- possibility to extend/reduce the application features simply enabling/disabling its plugin (not yet implemented).