How to use and write templates

INTRODUCTION

Templates is a new functionality available from Qlucore Omics Explorer version 3.5 which enables you to create scripts to control the software. This addition of flexibility opens a whole new world of possibilities, some examples are:

- Configure your preferred work-flows for easy access and cross-organizational consistency
- Run the same analysis many times with only one key press.
- Integrate Qlucore Omics Explorer (QOE) in existing workflows and expand them.
- Write Templates to enable users in your organization to fetch data from internal databases.
- Generate data sets and variable lists from external sources and open them in Qlucore Omics Explorer without any need for the user to go through several import and configuration steps.
- Configure your favorite analysis setup for a quick start. After execution of a template the user can select to continue the analysis in Qlucore exactly as if all steps had been done through the normal user interface up to this point,

THE BASICS

- Templates are written and executed in Python and commands are communicated to QOE.
- The program is shipped with several example Templates.
- A Template can have a user interface, meaning that the Template can ask the user for input. A typical example is to ask the user for which annotation to use for a statistical test or which q-value cut-off to use.
- Python is automatically installed when QOE is installed and the Template functionality is directly available to the user. Version 3.5 of Qlucore Omics Explorer ships with version 3.7.1 of Python.

USING A TEMPLATE

A Template can be started in one of the following ways:

- Via the Template Browser. Select the Template to run and press Execute.
- By using the "Execute Template..." option in the File menu and selecting the template (file) to run
- By dragging and dropping a .qctemplate file onto Qlucore Omics Explorer

In fact a Template is a zip file with the extension changed to .qctemplate and a well-defined content. More information about the content is provided in the section Writing Templates. This knowledge is not required to use templates.

The Template Browser shows all Templates that are available in the configured Templates paths. Two paths can be set for this purpose, and these paths are searched recursively for files with the .qctemplate extension. These paths are set in the Templates Preferences tab in the Preferences dialog that is reached from the File menu.

WRITING TEMPLATES

This chapter is divided into three sections and it is assumed that the reader has basic knowledge of programming:

- The Template Content, i.e. what is required for a template to work
- The Description file
- The Script, i.e. how to write a suite of commands that can be executed

Template contents

Templates for Qlucore Omics Explorer have the extension .qctemplate. These files are in fact .zip archives. A .qctemplate file can be renamed with the extension .zip and uncompressed. The archive contains the following (all paths are relative to the root of the archive):

- A description file named **"description.txt"**
- A Python script named **"main.py"**
- A folder named "Images" (Optional)
- Any other files that the template may need, such as data sets, variable lists etc (Optional)

The **description file named "description.txt"** contains the configuration of the template needed to display the descriptions in the Template Browser, as well as the definition of the initial GUI and other configurations needed to run the template.

The **Python script named "main.py"** contains the executable code for the template, written in Python. This script needs to import the Qlucore API to gain access to the Templates features. For more information, refer to the section below – The actual script.

The Images folder contains any images used in the Templates GUI. These images shall be in the .png file format. The Images folder is optional.

As the .qctemplate is a .zip archive, it may contain any number of other files that the Template may use. This could be additional .py scripts, scripts for other scripting languages that can be invoked by Python, .gedata files, text files with variable lists etcetera.

The Description file

Templates are configured by editing their description file. The description file is always called "description.txt" and shall be placed inside the Template archive. It contains configurations of the following:

- Templates API version
- Name of the Template
- A short description

- A long description (May be multiple lines)
- Whether the template requires a data set
- Initial GUI image
- Initial GUI text
- Initial GUI widgets

Each property in the description file is described with a tag. A tag is a fixed string, enclosed in square brackets, followed by one or more lines of text. The tags may appear in any order in the description file. All tags but the name tag are optional.

Supported tags

[QCTEMPLATE_VERSION] Disregarded by the software at the moment.

[TEMPLATE_NAME] Used by the Template Browser.

[SHORT_DESCRIPTION] Used by the Template Browser.

[LONG_DESCRIPTION] Used by the Template Browser.

[REQUIRES_DATA_SET] If set to "yes" a data set must be present for the template to execute.

[REMOVE_OK_BUTTON] If set to "yes" the "OK" button in the Templates GUI Dialog is removed. Has no effect when the template is executed in the Template Browser.

[DIALOG_IMAGE] The image file name to show in the initial GUI. Assumed to exist in the Template's Images folder.

[DIALOG_TEXT] The text to be displayed in the initial GUI.

[DIALOG_WIDGETS] Widgets to be shown in the initial GUI.

Template GUI widget descriptors

The [DIALOG_WIDGETS] tag is followed by a list of widgets to show. These are described with the following syntax:

type|name|option1|option2|...|optionN

The types that are available are:

- selectAnnotation : A Drop-down that lets the user select a sample annotation from the active data set.
    - option1 : Widget description used in GUI.
- selectAnnotationGroup : A Drop-down that lets the user select a sample annotation group from the active data set.
    - option1 : Widget description used in GUI.
    - option2 : Name of associated selectAnnotation widget.
- selectTestDirection : A Drop-down that lets the user select a statistics test direction for directed statistical tests.
    - option1 : Widget description used in GUI.

- pqSpinner : A spin-box that lets the user select a floating point value between 0 and 1
  - option1 : Widget description used in GUI.
- pushButton : A button
  - option1 : Widget description used in GUI.
  - option1 : Text written on button in GUI.

The widget description, if not left empty, is written next to the widget when it is displayed. The name of a widget is used by Python script code to access the widget's data.

Here follows a simple example of a description file that uses several of the tags.

```
[QCTEMPLATE_VERSION]
1.0
[TEMPLATE_NAME]
t-test
[SHORT_DESCRIPTION]
Performs a t-test and displays a heatmap and a box plot.
[LONG_DESCRIPTION]
Performs a t-test on the active data set based on the selected sample annotation. The results are displayed in two plots, one heat map and one box plot. A list of the discriminating variables is generated. The variables added to the list is based on the selected q-value (False Discovery Rate) cut-off. The list is named after the used annotation. The plots are tiled.
[REQUIRES_DATA_SET]
yes
[DIALOG_IMAGE]
qlucore_ttest.png
[DIALOG_TEXT]
Welcome to the t-test assistant template. Please select the variable annotation for which to do the test, the group for which the test is to be done and the direction of the test below.
[DIALOG_WIDGETS]
selectAnnotation|annotationDropdown|Select annotation.
selectAnnotationGroup|groupDropdown|Select group.|annotationDropdown
selectTestDirection|directionDropdown|Select test direction.
pqSpinner|qValueSpinner|Select q-value cut-off.
```

The Script

The main Python script, which is used as an entry point by the Templates execution mechanism is the Python script main.py, which shall be present in the .qctemplate file.

As a first example, we will take a look at the canonical 'Hello, World!' program as a Python script for Templates. We must first of all import the Qlucore api:

```
import qlucore
```

The next thing we need to do is to create a dialog box:

```
GUI = qlucore.templatesGUI
GUI.dialogTitle = "Sample Template Script"
GUI.dialogText = "Hello, World!"
```

This code goes into our main.py file. We then need a description.txt file as well. We use a minimal description.txt here, that only contains this:

```
[QCTEMPLATE_VERSION]
1.0
[TEMPLATE_NAME]
Hello World
[SHORT_DESCRIPTION]
The mandatory "Hello, World!" example.
[LONG_DESCRIPTION]
Hello, World! is a concept developed by Brian Kernighan and Dennis
Ritchie.
```

When we have the two files, the Template is compressed as described earlier in this document.

Plots, Cores and Datasets

The API for controlling Qlucore Omics Explorer in Python relies heavily on the concept of properties. To control a feature, you very often use assignment syntax, for example like this:

```
p2.order.variables = qlucore.VariableOrder.HIERARCHICAL_CLUSTERING
```

In the Python API, we work with two types of properties:

- Write-only properties or "Setter Properties"
- Read-only properties, "Getter Properties"

These are used to control various aspects of the QOE data and calculations model, and to read values from it.

Three central concepts exist in the API:

- **Data sets** are represented by the class `qlucore.DataSet` in the API. A data set holds all data and annotations that are loaded from the same source.
- **Cores**. A Core has a relationship to a data set by being a view of it. Each Core has an independent set of Statistics parameters, for example.
- **Plots.** A Plot is what you see on the screen, and it is a concretization of a Core. Several plots can share a Core, in which case they are *synchronized plots*. Two plots that draw the same data set, but which are not synchronized have independent Core objects.

Typically, when you work with Templates, you need to access the Plot, Core and DataSet objects for various entities. This always starts by getting a plot. There are several ways of getting hold of a Plot object, but the simplest way is to ask for the active plot:

```
myPlotObject1 = qlucore.plotUtils.activePlot
```

Another way to get hold of a plot is to load a data set from a file:

```
myPlotObject2 = qlucore.dataSetIO.openExampleFile("Acute
Lymphoblastic Leukemia")
myPlotObject3 = qlucore.dataSetIO.openFile("C:/tmp/somefile.gedata")
```

A Plot object can also be obtained by copying another Plot object:

```
myPlotObject4 = myPlotObject1.clone()
myPlotObject5 = myPlotObject1.cloneSynchronized()
```

From a Plot object, we can get its' Core, and from a Core, we can get a DataSet object:

```
core1 = myPlotObject1.core
dataSet1 = core1.dataSet
```

Most operations done in Qlucore Omics Explorer can be accomplished by operating on Core, DataSet and Plot objects. Here are a few examples:

Getting the active plot, closing all other plots, then creating a new plot synchronized with the active plot:

```
p1 = qlucore.plotUtils.activePlot
p1.closeAllButThis()
p2 = p1.clone()
p3 = p2.cloneSynchronized()
qlucore.omicsExplorerGUI.tileWindows()
```

Loading an example file and changing the plot type to a heat map:

```
p1 = qlucore.dataSetIO.openExampleFile("Qlucore Test Data Set")
p1.type = qlucore.PlotType.HEAT
```

Setting up a statistical two-group test for a Core:

```
qlucore.omicsExplorerGUI.openStatisticsWindow()
qlucore.omicsExplorerGUI.activateStatisticsGeneralTab()
g5.statistics.testType = qlucore.TestType.TWO_GROUP
g5.statistics.annotation = "Treatment"
g5.statistics.annotationValue = "Placebo"
```

```
g5.statistics.testDirection = qlucore.TestDirection.LESS
g5.statistics.q = 0.87
```

Template Dialog Boxes

The GUI will always be shown as a dialog box if it was invoked from Python code. To show GUI elements from Python code, the class TemplatesGUI is used via the templatesGUI object (Note the casing!):

```
qlucore.templatesGUI.dialogTitle = "Execution finished"
qlucore.templatesGUI.dialogText = "The test has now finished
running."
qlucore.templatesGUI.dialogImage = "example.png"
qlucore.templatesGUI.addButton("button0", "Repeat", "Repeat the
test.")
qlucore.templatesGUI.showTemplateWindow()
```

Interacting with the user interface from Python

The Python code can perform two basic interactions with GUI elements. It can get values from widgets, and it can react to a button that was pushed. Here is an example of gathering values from widgets:

```
# Get the widget data.
dialog = qlucore.templatesGUI

annotationName = dialog.getWidgetAnnotationName("annotationDropdown")
valueName = dialog.getWidgetValueName("groupDropdown")
testDirection = dialog.getWidgetTestDirection("directionDropdown")
qValue = dialog.getWidgetDoubleValue("qValueSpinner")
```

Here is an example of code reacting to buttons being pushed:

```
theButton = qlucore.templatesGUI.lastClickedButton
if theButton == "button0":
qlucore.templatesGUI.dialogTitle = "Kitteh piccy"
qlucore.templatesGUI.dialogImage = "kitteh.png"
qlucore.templatesGUI.dialogText = "I can haz cheezburger...?"
qlucore.templatesGUI.showTemplateWindow()
if theButton == "button1":
qlucore.templatesGUI.dialogTitle = "Doggeh piccy"
qlucore.templatesGUI.dialogImage = "doggeh.png"
qlucore.templatesGUI.dialogText = "Doggeh be zleepy!!!"
qlucore.templatesGUI.showTemplateWindow()
```

Completeness

The entire functionality of Qlucore Omics Explorer has not yet been introduced into the Templates API. For a full list of available functionality please refer to the Templates API Reference Manual.

Advanced

The error handling in Templates is very conservative and strict. Basically, if a command fails, because it was given bad parameters, or the software is in a state where the command cannot be processed, the execution of the Template will terminate, and you will be given an error message on the screen.

There is one exception to this rule: If an error occurs before the Template has started running, which may be the case when there are syntax errors in the Python script, you will get an error message "Early syntax error." To debug such errors, it is suggested to use a Python interpreter or debugger to figure out what is wrong with your script.

The Python execution environment shipped with Qlucore Omics Explorer (3.5) is version 3.7.1 with the pyzmq library version 17.1.2 installed. The only other non-standard add-on to Python is the two files qcutil.py and qlucore.py, located in Python's Lib folder.

It is possible to run your own installation of Python if you want to, however, you must comply with the following requirements:

- You need to have pyzmq installed in Python, recommended version is 17.1.2
- You must copy the files qlucore.py and qcutil.py (located in the Lib folder of the Python installation shipped with QlucoreOmics Explorer) to the Lib folder of your Python installation

To specify that you want to run your own version of Python, go to the File Menu, select Preferences and navigate to the Templates tab. There, you can point to your own Python installation.

DISCLAIMER

The contents of this document are subject to revision without notice due to continuous progress in methodology, design, and manufacturing.

Qlucore shall have no liability for any error or damages of any kind resulting from the use of this document.

Qlucore Omics Explorer is only intended for research purposes.