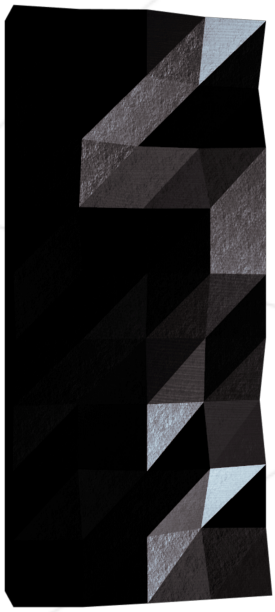

Model/view framework in QML

Models, views, and delegates in QML

Qt training training@qt.io
42 Pedago Staff pedago@42.fr

Summary: *QML provides a model/view framework similar to C++. In addition to C++ models, it is possible to use QML models.*

Contents

Chapter I

General instructions

Unless explicitely specified, the following rules will apply every day of this Piscine.

- This subject is the one and only trustable source. Don't trust any rumor.
- This subject can be updated up to one hour before the turn-in deadline.
- The assignments in a subject must be done in the given order. Later assignments won't be rated unless all the previous ones are perfectly executed.
- Be careful about the access rights of your files and folders.
- You must follow the `turn-in process` for each assignment. The url of your `GIT` repository for this day is available on your intranet.
- Your assignments will be evaluated by your Piscine peers.
- In addition to your peers evaluation, a program called the "Moulinette" will also evaluate your assignments. Fully automated, The Moulinette is tough and unforgiving in its evaluations. As a consequence, it is impossible to bargain your grade with it. Uphold the highest level of rigor to avoid unpleasant surprises.
- You must not leave in your turn-in repository any file other than the ones explicitly requested by the assignments.
- You have a question? Ask your left neighbor. Otherwise, try your luck with your right neighbor.
- Every technical answer you might need is available in the `mans` or in the Qt Documentation, which is available both in `QtCreator` IDE and on http://doc.qt.io.
- Remember to use the Piscine forum of your intranet and also Slack!
- You must read the examples thoroughly. They can reveal requirements that are not obvious in the assignment's description.
- Use the latest Qt version. Qt version 5.10 or newer is recommended.

- Many Qt classes are available as standard C++ classes. Qt classes should be preferred to standard classes, as you are supposed to learn Qt.
- Basic Qt coding style should be used, so read `http://wiki.qt.io/Qt_Coding_Style` before writing any assignments.
- Deprecated macros, functions or classes must not be used.
- Any build system, such as `qmake`, `cmake` or Qt Build System, can be used in the assignments. Instructions are based on `qmake`.
- Any editor or IDE, supporting Qt, can be used. However, `QtCreator` usage is strongly recommended.
- By Thor, by Odin! Use your brain!!!

Chapter II

Assignment 00: Phone book

Implement a phone book using the QML `ListView`.

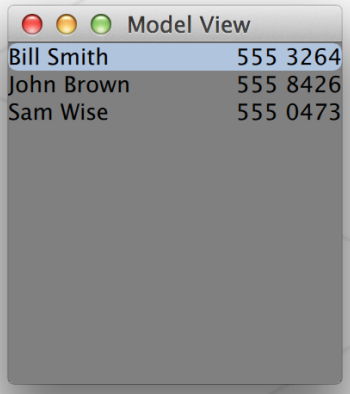

Figure II.1: Simple phone book

- Implement the phone book model and list view delegate in their own files.
- Use a QML `ListModel` with a couple of `ListElement` children. The list elements should contain name and phone number roles.
- The delegate should contain two `Text` items. The name role is aligned to the left and the phone number role to the right.
- Add a list view to the window. Use a highlight component as well. It should be a rectangle with a small radius value. Make sure arrow keys can be used to change the current item.

Chapter III

Assignment 01: Efficient model manipulation

Add 100,000 random contacts to the phone book model of the previous assignment. The addition must not freeze the UI creation significantly, so you must do it in a background thread.

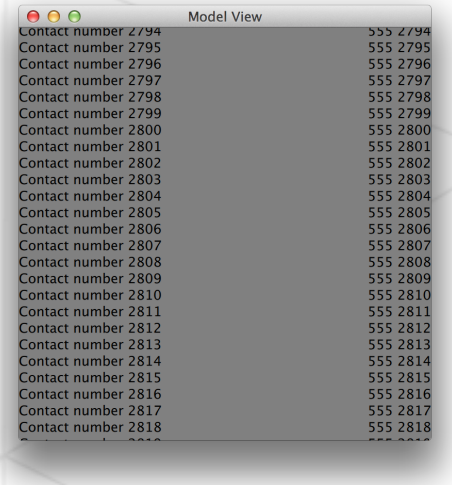

Figure III.1: Simple phone book with 100,000 + contacts

Chapter IV

Assignment 02: RSS reader

Use `XmlListModel` to read news from the RSS feed. The news are read from http://feeds.bbci.co.uk/news/rss.xml. The interesting items are `title` and `link`, which are strings. The absolute XQuery path is `//item`.

Write a delegate, which shows the title and a link. If the delegate is pressed, show the actual piece of news using `WebEngineView` as shown below.

Figure IV.1: News reader

Chapter V

Assignment 03: List view decoration

Decorate a list view with header and footer, which may be simple rectangles as shown below.

Define a list model with some animals and animal groups. Use list view sections to group animals into different sections. Implement also a highlight rectangle. Add a simple text item at the bottom of the window, which shows the current list view item.

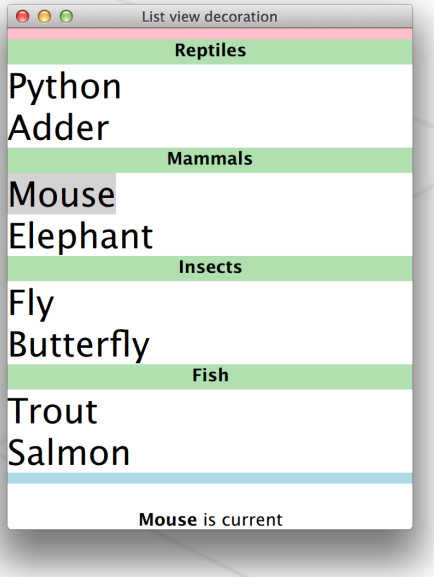

Figure V.1: Decorated list view

Chapter VI

Assignment 04: Path view decoration

Create an application, showing text items in `PathView`. Define any shape for the path. For example V-shape is rather straightforward with `PathLine` objects. Add some ten items to your model and use a delegate to draw texts.

Change the item opacity and scale value, based on the item position on the path. Set the values to 1, e.g. on the horizontal or bottom part of the path as shown in the figure VI. When the item approaches the top left or right corner, its opacity and scale change to 0.5, for example.

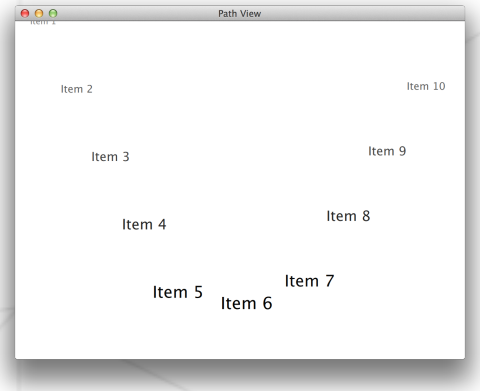

Figure VI.1: Decorated path view

Chapter VII

Assignment 05: Views

Implement a QML UI, which splits the window equally between a table view and a grid view as shown below. Both views share the same model, which contains two text roles: a name and a radio button check state. Implement delegates for the views, using `RadioDelegate` Qt Quick Controls. Create the model elements dynamically after the window has been created. Some 100 items would be enough.

Store the radio button states. When the user scrolls items up and down, the radio button state is maintained, although the button delegates are dynamically created. Simple solution, like caching the delegates, will not give you any score. You need to use both Qt Quick Controls 1 and 2 in this exercise, as the former does not have a `RadioDelegate` type and the latter does not have `TableView` control.

Radio buttons must not paint outside the table cell. Obviously, clipping delegates is not allowed, as it will have negative effect on performance.

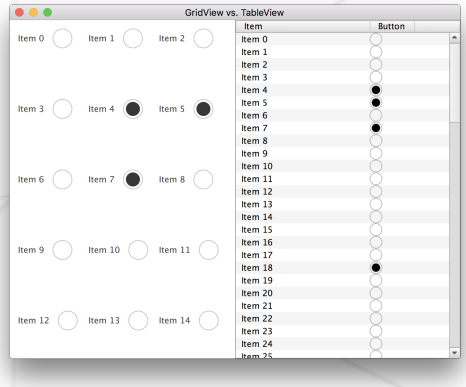

Figure VII.1: Grid view and table view

Chapter VIII

Assignment 06: View decoration

Extend Assignment 04 by customising the delegate. Change the delegate so that it contains a grid of 10,000 black rectangles of size 10x10. Use lazy loading to create or destroy the grid of elements, when the delegate is clicked. As a result, 10,000 rectangles in the delegate should not affect the startup time or item animations significantly.

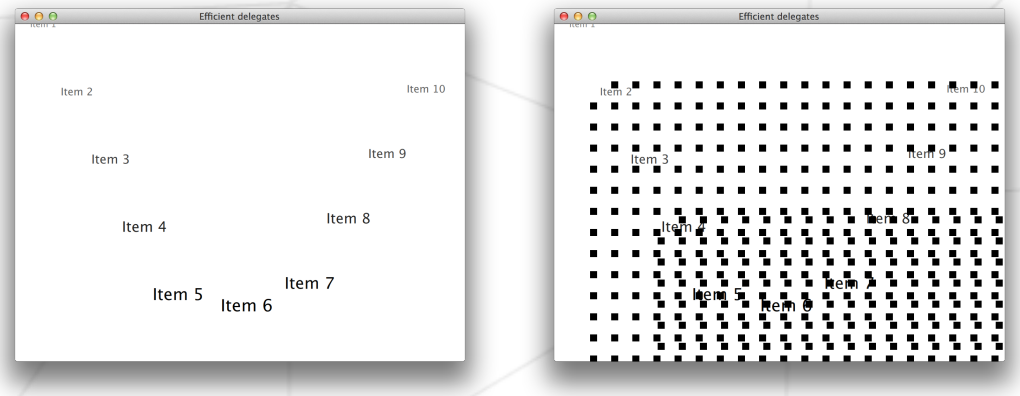

Figure VIII.1: Path view with 10,000 rectangles