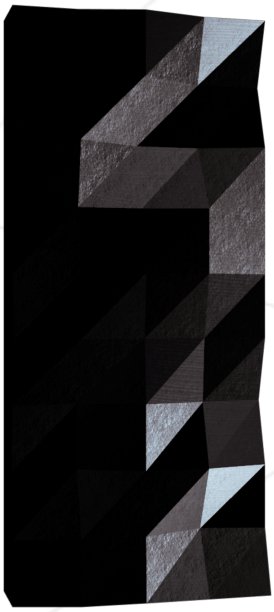


Model/view framework

Item models, views, and delegates

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

Summary: Many applications need to show data from a database, network repository or file. Qt model/view framework allows several views to share the same model data efficiently. The model may be the actual storage of the data or just a wrapper to the data, stored persistently somewhere else. Views are customised using delegates.

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 <u>must not</u> 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: Item views

Implement a trivial file browser. Use a ready-made `QAbstractItemModel` subclass, which provides a class model of a local file system. Show the model data in a tree widget. Start browsing from your home folder, but do not hard-code the home folder path. The browser must work in any desktop platform.

Chapter III

Assignment 01: Standard item model

Implement a program, showing car names and modes in a table view. When the user clicks on an item in the view, the car image is shown in the label. Use the three source code files in `res/classes/01_standarditem_model` in your project. `CarLabel` extends `QLabel` so that the label widget is hidden, when clicked. The only file you would need to change is `main.cpp`.

The `main()` function contains a list of car names and models.

- Add car names and models to two separate columns in a standard item model. Avoid all hard-coded magic numbers. You must use `setData()` and `data()` functions to set and get model data. Other member functions of `QAbstractItemModel` can be used additionally.

- Set column names in the model.

- Implement functionality to show the car image in `CarLabel`, when an item is clicked. Use a suitable signal. Add images to your project as resources from the `res/car_images` folder. Get the car file name from the model. Use a pixmap to show the image in the label.

Chapter IV

Assignment 02: Sorting and filtering

You are provided with three source code files in the `res` folder. The files implement a simple model/view program as shown in Fig. IV.

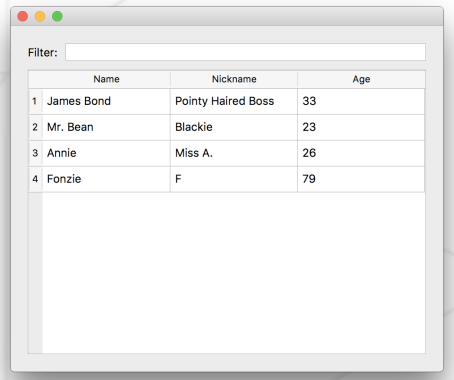


Figure IV.1: Proxy model

Your task is to sort and filter model data using a sort filter proxy model.

- Standard item model supports item sorting. The table view allows the user to choose a column for sorting. Change the table view to enable column-specific sorting.

- Add a sort filter proxy model between the standard item model and the table view. Use the sort filter proxy model to do the sorting.

- Use the line edit widget in the template code to allow user to input filter strings. Filter out rows, which do not contain the filter string. Apply filtering to the column, which is enabled for sorting.

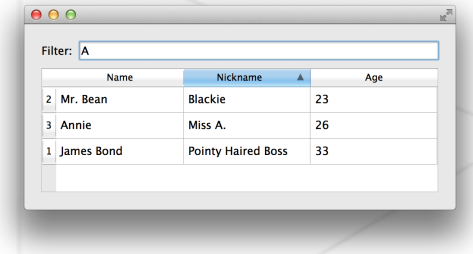
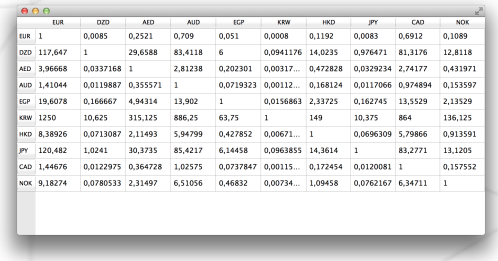


Figure IV.2: Proxy model with a column filter

Chapter V

Assignment 03: Custom models

Implement a program, which shows ten currencies and corresponding conversion rates in a 10 x 10 table. The `main.cpp` file is provided to you in the `res` folder. All currencies in a row are converted based on the currency in the first column. So, one EUR is 0.0085 DZD and one DZD is 0.0085 EUR as shown in Fig. V.



	EUR	DZD	AED	AUD	EGP	KRW	HKD	JPY	CAD	NOK
EUR	1	0,0085	0,2521	0,709	0,051	0,0008	0,1192	0,0083	0,6912	0,1089
DZD	117,647	1	29,6588	83,4118	6	0,0941176	14,0235	0,976471	81,3176	12,8118
AED	3,96668	0,0337168	1	2,81238	0,202301	0,00317...	0,472828	0,0329234	2,74177	0,431971
AUD	1,41044	0,0119887	0,355571	1	0,0719323	0,00112...	0,168124	0,0117066	0,974894	0,153597
EGP	19,6078	0,166667	4,94314	13,902	1	0,0156863	2,33725	0,162745	13,5529	2,13529
KRW	1250	10,625	315,125	886,25	63,75	1	149	10,375	864	136,125
HKD	8,38926	0,0713087	2,11493	5,94799	0,427852	0,00671...	1	0,0696309	5,79866	0,913591
JPY	120,482	1,0241	30,3735	85,4217	6,14458	0,0963855	14,3614	1	83,2771	13,1205
CAD	1,44676	0,0122975	0,364728	1,02575	0,0737847	0,00115...	0,172454	0,0120081	1	0,157552
NOK	9,18274	0,0780533	2,31497	6,51056	0,46832	0,00734...	1,09458	0,0762167	6,34711	1

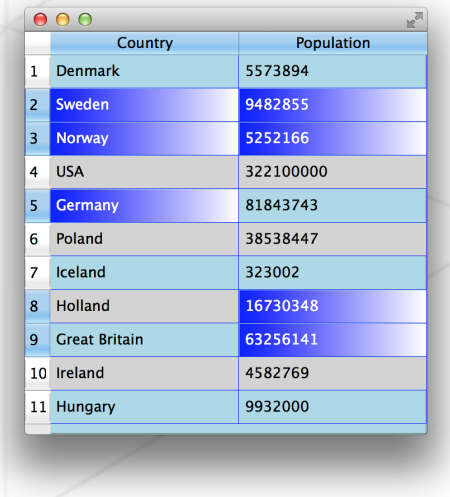
Figure V.1: Currencies read from the custom model

- Ten currencies in the `main.cpp` are stored in the model data and the user must be able to change the currency values. Extra score is given, if new currencies can be added to the model without replacing any existing ones.
- Subclass `QAbstractTableModel`.
- Only the first row should be editable. The user can set the each currency rate based on EUR.
- Include the currency names in the header data.

Chapter VI

Assignment 04: View customisation

Implement a custom `QTableView` subclass, which has some custom decoration and supports item deletion. The `main.cpp` has been given in the `res` folder. Your task is to create, implement, and add a custom view. The application screen shot is shown below.



	Country	Population
1	Denmark	5573894
2	Sweden	9482855
3	Norway	5252166
4	USA	322100000
5	Germany	81843743
6	Poland	38538447
7	Iceland	323002
8	Holland	16730348
9	Great Britain	63256141
10	Ireland	4582769
11	Hungary	9932000

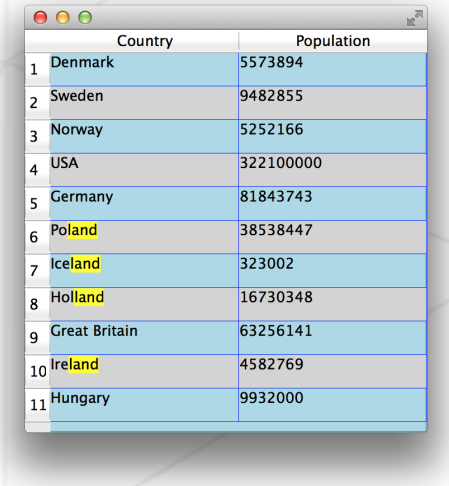
Figure VI.1: Custom view

- Implement simple decoration. Use a custom grid line, background, and alternate background colours.
- Selections should be emphasised using a linear gradient colour.
- Implement functionality to remove items from the model based on the user's selections. The deletion takes place, when the user presses the 'D* key. All selection modes must be supported, also empty selections. If either (or both) of the two columns is selected, the corresponding row must be removed from the model.

Chapter VII

Assignment 05: Custom delegate

Use a custom delegate to implement search functionality to the custom view, which you implemented in the previous assignment.



	Country	Population
1	Denmark	5573894
2	Sweden	9482855
3	Norway	5252166
4	USA	322100000
5	Germany	81843743
6	Poland	38538447
7	Iceland	323002
8	Holland	16730348
9	Great Britain	63256141
10	Ireland	4582769
11	Hungary	9932000

Figure VII.1: Custom delegate used to highlight search string

- Search is started, when a user presses the 'F' key.
- Use `QInputDialog` to ask the user for a search string.
- Store the search string into the model using a custom role.
- Subclass `QItemDelegate` and reimplement the `paint` function. Paint the text data from the model and a rectangle, highlighting the searched text. `QFontMetrics` provides useful functions.