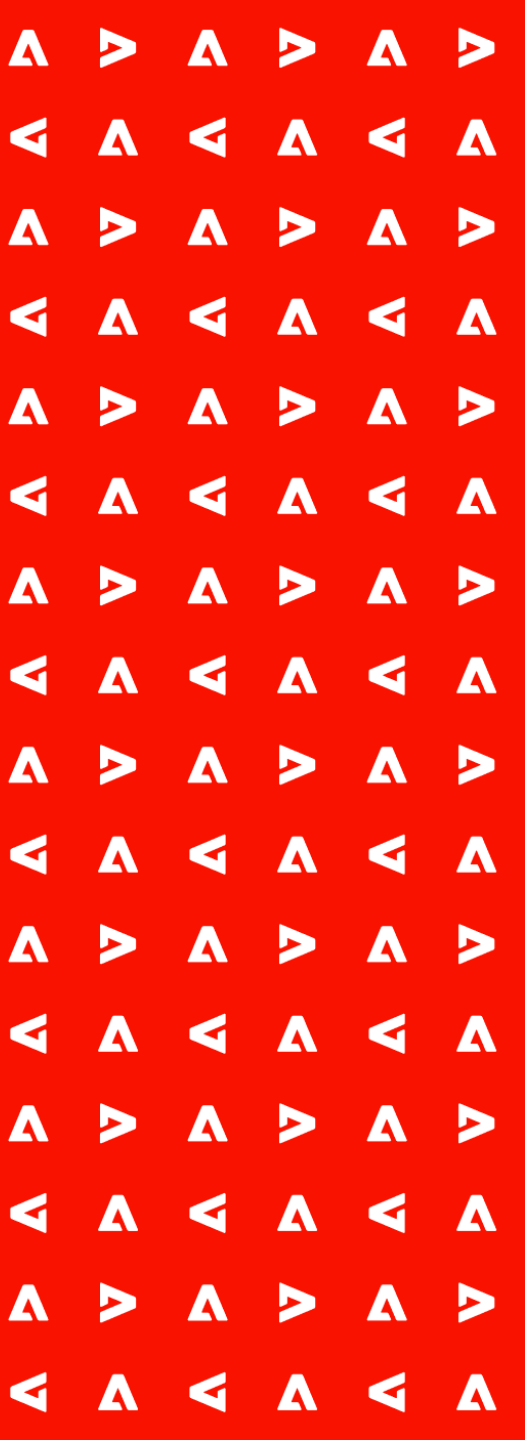
Layouts with Qt

Philippe Hermite | Software Engineer



Franck ARRECOT

Software engineer

Working at KDAB for 6 years, KDE contributor

Philippe HERMITE

Software engineer

Working at Allegorithmic/Adobe since 2016, maintaining Substance Painter

Introduction

Qt version: 5.12.x (LTS)

- Overall feedback:
  - Why use a layout?
  - Which layouts to use?
  - How to correctly align different layouts?
  - How to improve performance when displaying a lot of information?

Plan

- Using layouts to align your QWidgets
- Alignment between layouts
- Layout operations
- Manipulating a lot of QWidgets
- Debugging with GammaRay

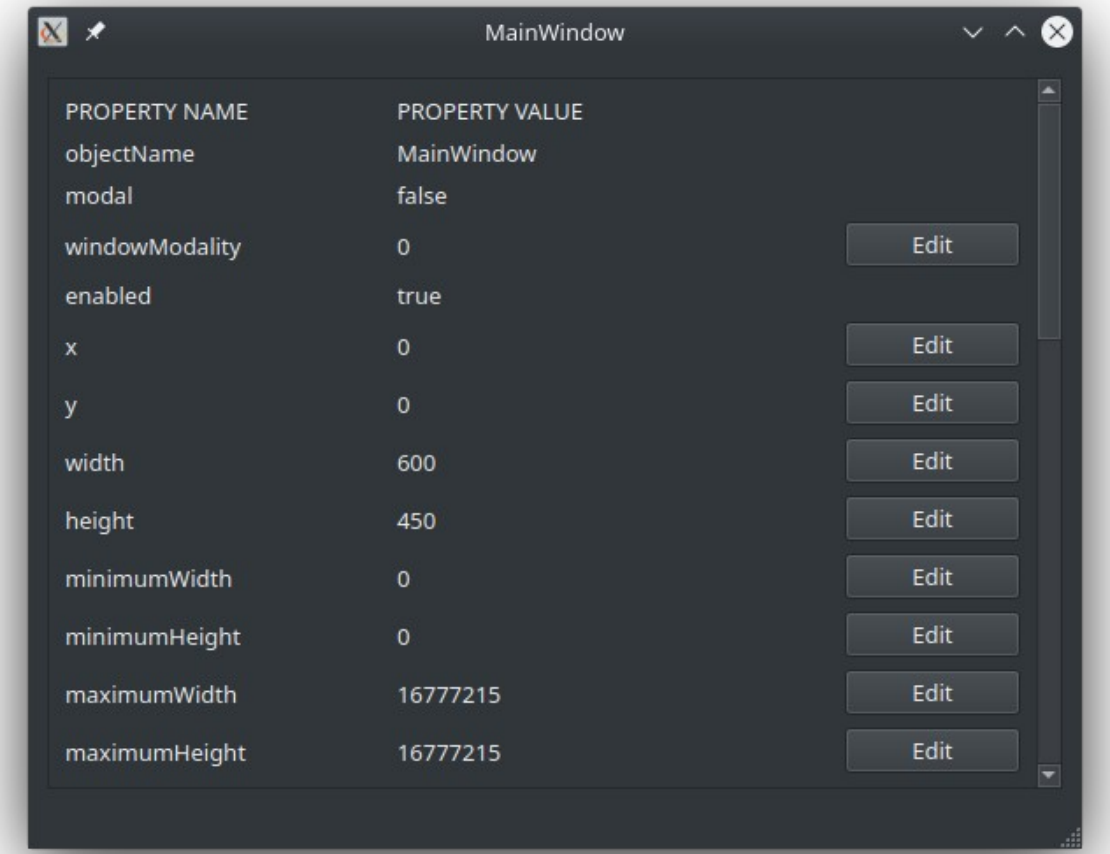
Using layouts to align your QWidgets

Using layouts to align your QWidgets

- Layout objectives:
  - Dynamic: the UI content should adapt its size
  - Structured: the UI content should be easily understood by the user
- QLayout provides dynamic resizing
- Alignment may be tricky depending on the subclass of QLayout

Project example

- Simple QMainWindow with ScrollArea
- A layout to display properties
- Each property in a row of the layout
  - Name of the property
  - Value of the property
  - Edit button to change the property



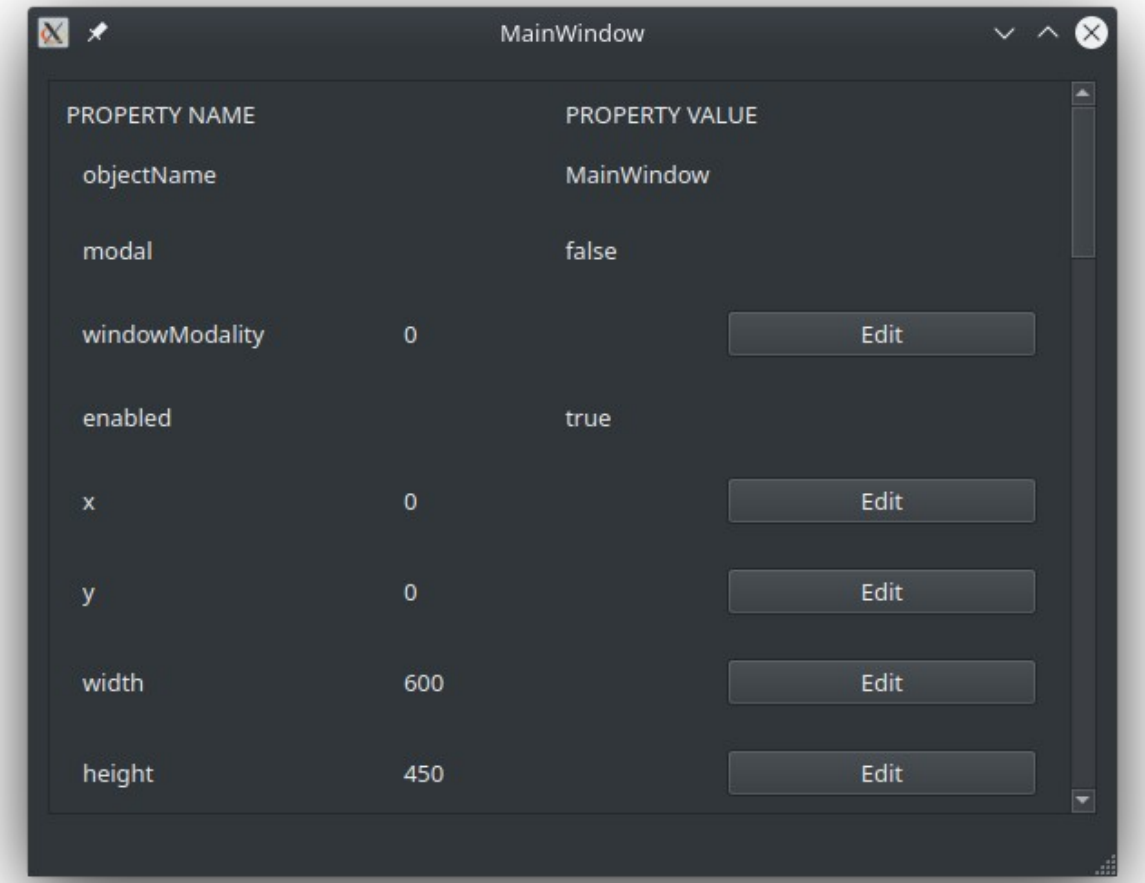
Using a QVBoxLayout

- Each property row is represented by a QHBoxLayout containing:
  - QLabel for the property name
  - QLabel for the property value
  - QPushButton to edit the property value if needed
- A QVBoxLayout stores the rows

```
200  //Add header [...]
201
202  // Parsing all properties
203  const auto moThis = this->metaObject();
204  for (int i = 0; i < moThis->propertyCount(); ++i) {
205      const auto property = moThis->property(i);
206      const QVariant value = property.read(this);
207
208      auto nameLabel = new QLabel(property.name());
209      auto valueLabel = labelForValue(value);
210
211      // Add QHBoxLayout
212      auto propertyWidget = new QWidget;
213      auto propertyLayout = new QHBoxLayout(propertyWidget);
214
215      // Adding labels to layout
216      propertyLayout->addWidget(nameLabel);
217      propertyLayout->addWidget(valueLabel);
218
219      // Edit value only if type int
220      if (value.type() == QVariant::Int) {
221          auto editButton = new QPushButton("Edit");
222          propertyLayout->addWidget(editButton);
223          //[...]
224      }
225
226      // Add property widget to the main vertical
227      contentLayout->addWidget(propertyWidget);
228  }
```

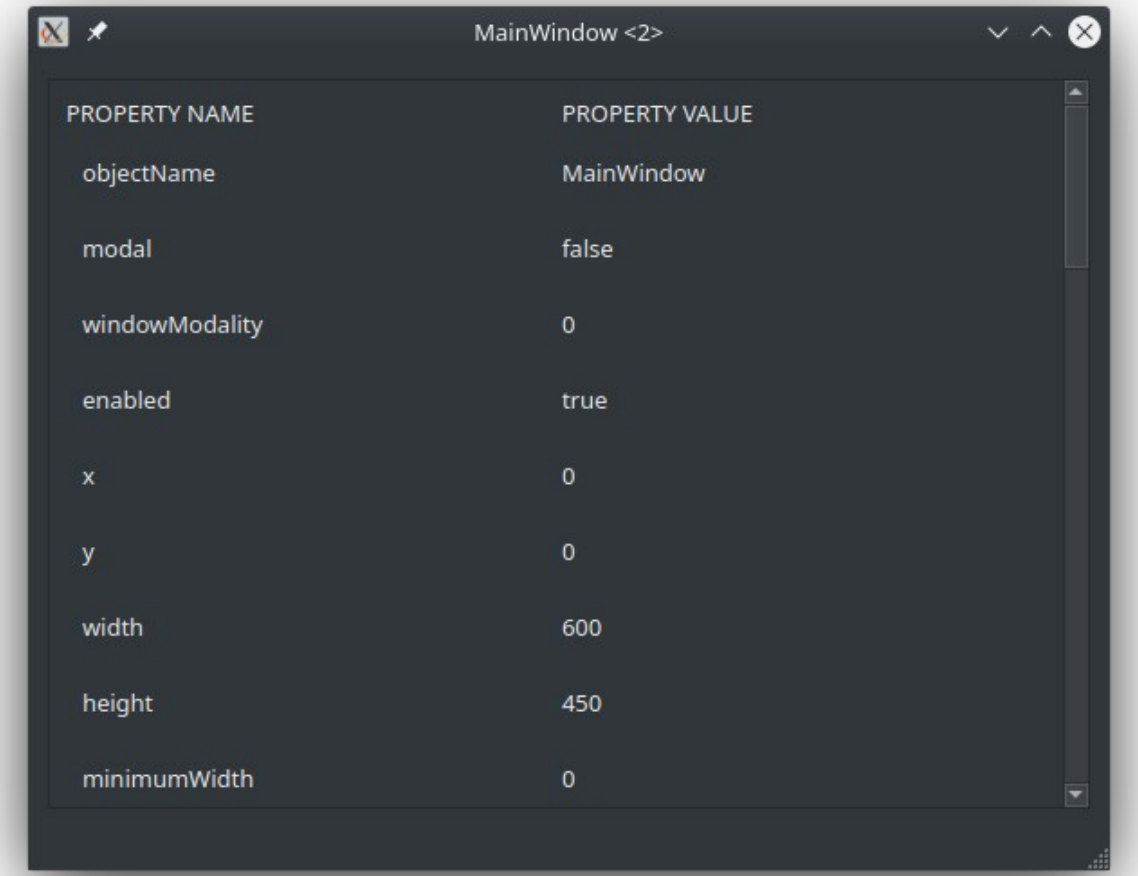
Using a QVBoxLayout

- Alignment issue:
  - Some rows contain 2 elements when the others have 3
  - By default QBoxLayout splits the width equally creating misalignment



Using a QVBoxLayout

- Removing the button solves the alignment



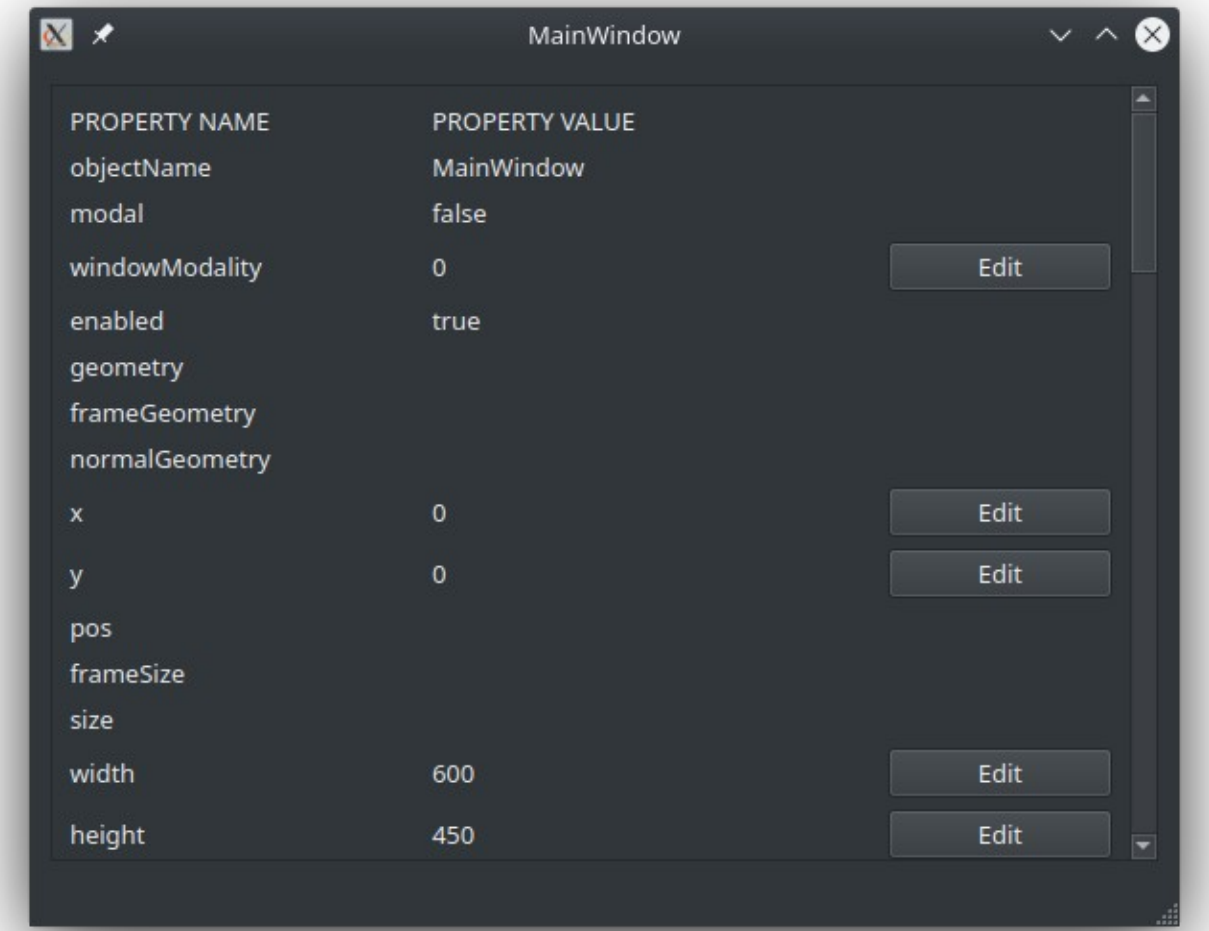
Using a QGridLayout

- QGridLayout instead of the QVBoxLayout
- For each property
  - Add the property name in the first column
  - Add the property value in the second column
  - Add the edit button in the third column

```
00    // Add Header
01    QGridLayout *contentGridLayout = new QGridLayout(scrollAreaContent);
02    contentGridLayout->addWidget(new QLabel("PROPERTY NAME"), 0, 0);
03    contentGridLayout->addWidget(new QLabel("PROPERTY VALUE"), 0, 1);
04
05    // Parsing all properties
06    const auto moThis = this->metaObject();
07    for (int i = 0; i < moThis->propertyCount(); ++i) {
08        const auto property = moThis->property(i);
09        const QVariant value = property.read(this);
10
11        // Add labels to layout
12        const int row = contentGridLayout->rowCount();
13        auto nameLabel = new QLabel(property.name());
14        auto valueLabel = labelForValue(value);
15        contentGridLayout->addWidget(nameLabel, row, 0);
16        contentGridLayout->addWidget(valueLabel, row, 1);
17
18        // Edit value only if type int
19        if (value.type() == QVariant::Int) {
20            auto editButton = new QPushButton("Edit");
21            contentGridLayout->addWidget(editButton, row, 2);
22            //[...]
23        }
24    }
```

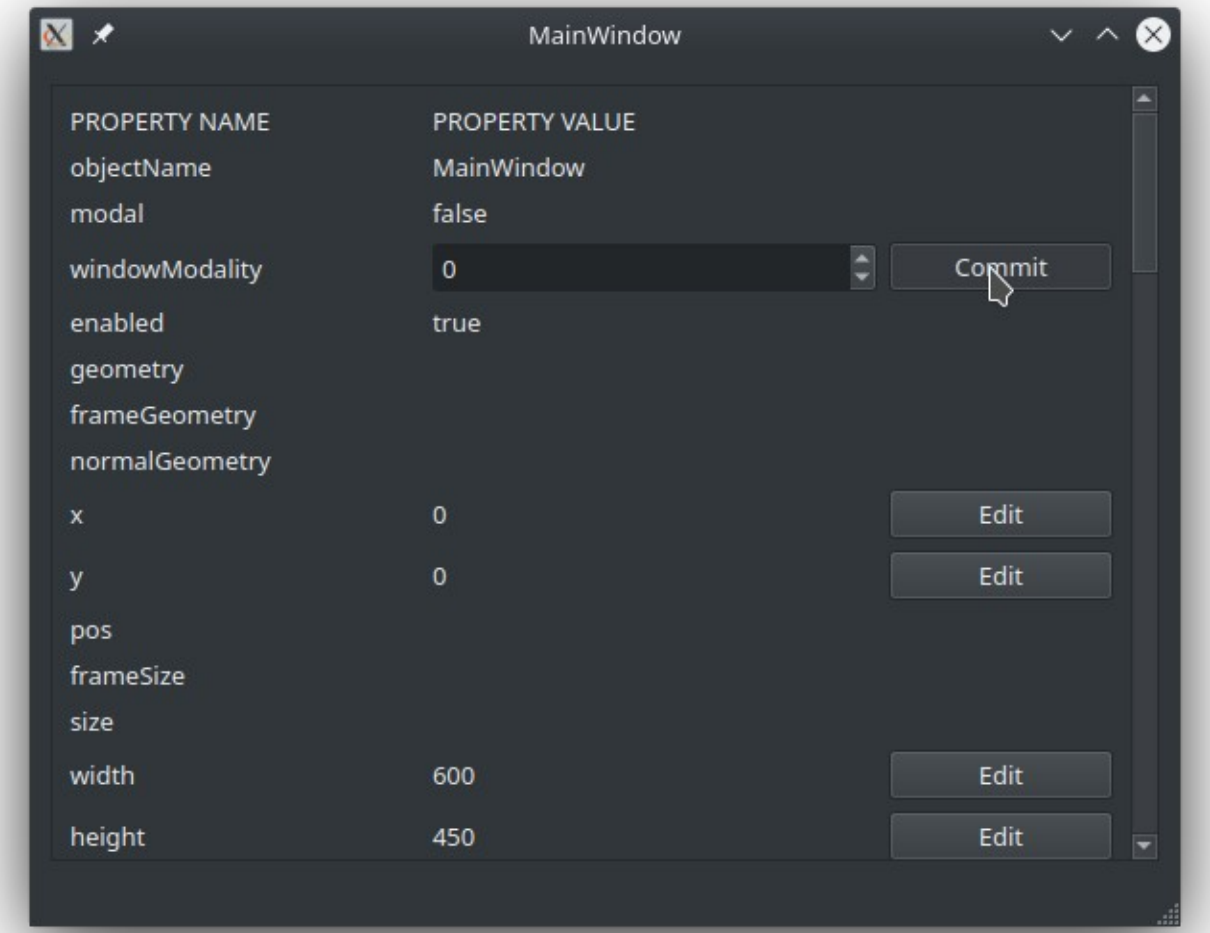
Using a QGridLayout

- Result looks good
- Less maintainable code
- What about the edit feature ?



Editing the property value

- When clicking on the edit button:
  - An editor replaces the value label
  - Clicking the edit button again commits the new value
  - The editor disappears and the value label reappears



Remember our QVBoxLayout...

- Each property row is represented by a QHBoxLayout containing:
  - QLabel for the property name
  - QLabel for the property value
  - QPushButton to edit the property value if needed
- A QVBoxLayout stores the rows

```
200   //Add header [...]
201
202   // Parsing all properties
203   const auto moThis = this->metaObject();
204   for (int i = 0; i < moThis->propertyCount(); ++i) {
205       const auto property = moThis->property(i);
206       const QVariant value = property.read(this);
207
208       auto nameLabel = new QLabel(property.name());
209       auto valueLabel = labelForValue(value);
210
211       // Add QHBoxLayout
212       auto propertyWidget = new QWidget;
213       auto propertyLayout = new QHBoxLayout(propertyWidget);
214
215       // Adding labels to layout
216       propertyLayout->addWidget(nameLabel);
217       propertyLayout->addWidget(valueLabel);
218
219       // Edit value only if type int
220       if (value.type() == QVariant::Int) {
221           auto editButton = new QPushButton("Edit");
222           propertyLayout->addWidget(editButton);
223           //[...]
224       }
225
226       // Add property widget to the main vertical
227       contentLayout->addWidget(propertyWidget);
228   }
```

Display the editor: QBoxLayout

- Value and Editor sharing same sublayout
  - Hide the value and show the editor on edition
  - Show the value and hide the editor on standard display
- QHBoxLayout and QVBoxLayout are equivalent

```
00  //[... Within the loop ...]
01  propertyLayout->addWidget(nameLabel);
02
03  // Add value label in a new sub layout
04  QVBoxLayout *valueAndEditorLayout = new QVBoxLayout;
05  auto valueLabel = labelForValue(value);
06  valueAndEditorLayout->addWidget(valueLabel);
07  propertyLayout->addLayout(valueAndEditorLayout);
08
09  // Add the editor with the value
10  if (value.type() == QVariant::Int) {
11      auto editor = new QSpinBox(this);
12      editor->setValue(value.toInt());
13      valueAndEditorLayout->addWidget(editor);
14      editor->setVisible(false);
15
16      auto editButton = new QPushButton("Edit");
17      propertyLayout->addWidget(editButton);
18      editButton->setCheckable(true);
19      editButton->setChecked(false);
20
21      QObject::connect(editButton, &QPushButton::toggled,
22      [=](bool checked) {
23          valueLabel->setVisible(!checked);
24          editor->setVisible(checked);
25      });
26
27      //[...]
```

Display the editor: QBoxLayout

- Flickering Issue
- Quick Video

Display the editor: QStackedLayout

- Using GridLayout
- Properties of the QStackedLayout:
  - Stack several QWidget
  - Show all the widgets simultaneously or individually
  - Size of the biggest widget
- QStackedLayout are essentially used for tabs management

```
00  // [... Within the loop ...]
01  const int row = contentGridLayout->rowCount();
02
03  // Add name label in the GridLayout
04  auto nameLabel = new QLabel(propertyName);
05  contentGridLayout->addWidget(nameLabel, row, 0);
06
07  // Add value label within a QStackedLayout
08  QStackedLayout *valueAndEditorLayout = new QStackedLayout;
09  valueAndEditorLayout->addWidget(valueLabel);
10  contentGridLayout->addLayout(valueAndEditorLayout, row, 1);
11
12  // Add the editor within QStackedLayout
13  if (value.type() == QVariant::Int) {
14      auto editor = new QSpinBox;
15      editor->setValue(value.toInt());
16      valueAndEditorLayout->addWidget(editor);
17
18      auto editButton = new QPushButton("Edit");
19      editButton->setCheckable(true);
20      contentGridLayout->addWidget(editButton, row, 2);
21
22      QObject::connect(editButton, &QPushButton::toggled,
23      [=](bool checked) {
24          if (checked)
25              valueAndEditorLayout->setCurrentWidget(editor);
26          else
27              valueAndEditorLayout->setCurrentWidget(valueLabel);
28      });
29
30      // [...]
```

Layouts to align widgets: Conclusion

To align correctly your UI:

- QGridLayout are often more reliable than QBoxLayout
- Don’t overuse QGridLayout since it is more complex to maintain

Hiding widgets with layouts can lead to flickering issues

- QStackedLayout can be used to avoid these issues
- Don’t overuse QStackedLayout for performance issues
- Using a stretch can reduce the flickering

Layouts to align widgets: Benchmarking

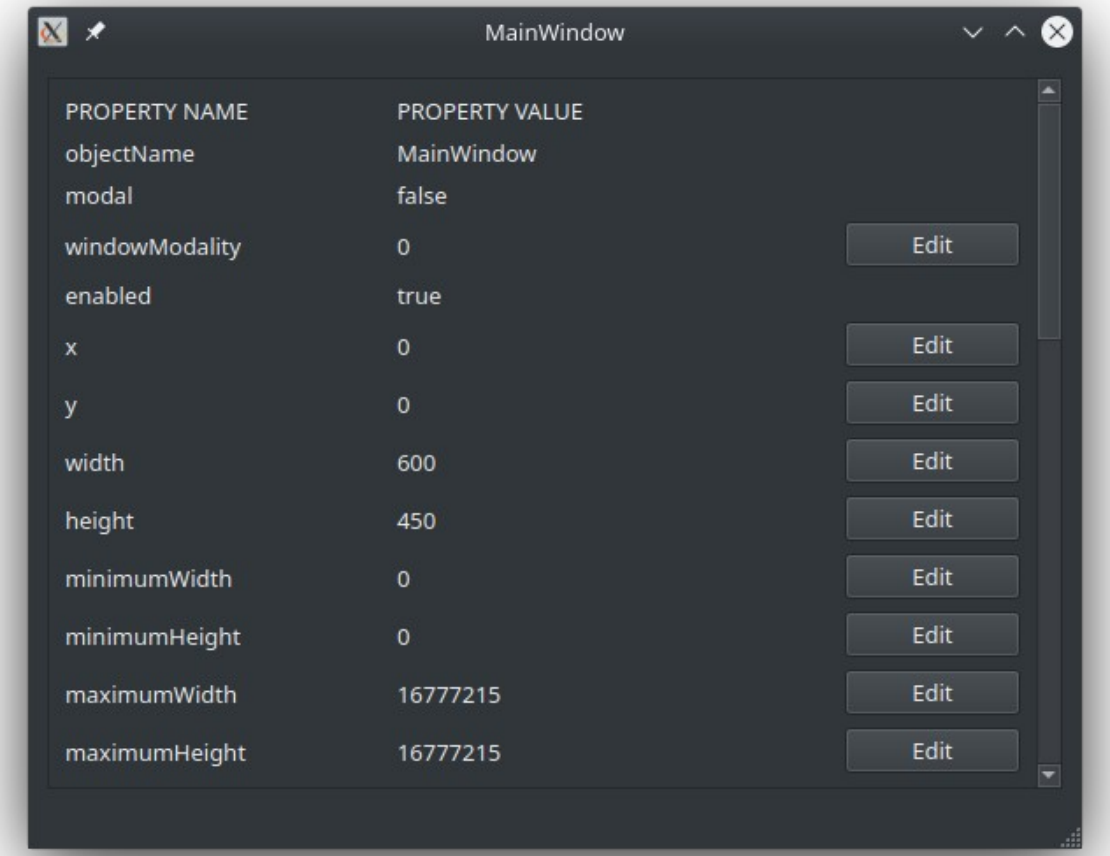
Using Qt5.12 Release with 20 runs to average out

10 rows	Without editor	Editor in QBoxLayout	Editor in QStackedLayout
QBoxLayouts	≈ 69 ms	≈ 73 ms	≈ 80 ms
QGridLayout	≈ 58 ms	≈ 66 ms	≈ 151 ms

100 rows	Without editor	Editor in QBoxLayout	Editor in QStackedLayout
QBoxLayouts	≈ 93 ms	≈ 96 ms	≈ 105 ms
QGridLayout	≈ 88 ms	≈ 91 ms	≈ 930 ms

Alignment between layouts

Project example



Alignment between layouts

- Layouts cannot share their alignments
- Example of a new feature:
  - With our previous example, we had a header at the top of the layout
  - Scrolling will hide the header
  - Separating the header from the QScrollArea allows us to always display it
  - How to correctly align the header layout with the content of the QScrollArea?

Using hard-coded values

- Most of the time, setting a default minimum size of a QWidget solves the issue

```
gridLayout->setColumnMinimumWidth(0, 200);
ui->propertyNameLabel->setMinimumWidth(200);
```

- In our example, it is difficult to define such a value
  - The content of the QGridLayout is generated dynamically
  - Alignment may be broken if the minimum size is not large enough
  - It may crop the display of the QWidget
- Hard-coded values should be easily retrieved and changed

Using hard-coded value after first resize

- The alignment can be fixed after the dynamic layout has determined its size

```
QTimer::singleShot(
    0,
    [this, gridLayout]()
    {ui->propertyNameLabel->setMinimumWidth(gridLayout->cellRect(0, 0).width());});
```

  - If no widgets are added or removed afterwards, this is safe
  - In the other case, this is not safe and alignment issues can appear during runtime
- Layouts do not send any signals to help us
  - Reset the value each time we add or remove a widget from the layout?
  - Reset the value each time we resize the layout?

Alignment between layouts: Conclusion

No ideal solution for this issue, sometimes hard-coded values may be preferable to avoid overly complicated code

Trying to determine at runtime the ideal size is not simple

- What can impact the size of a layout section?
- Can we check that the layout section has the right size?
- …

Layout operations

Layout operations

- Some layout operations can take a lot of time
- When inserting a new widget:
  - Avoid inserting too many widgets at the same time
  - If possible try to dispatch the insertion in several steps
- When removing a widget
  - Removing a lot of widgets at the same time can be a bottleneck
  - Disabling the layout before doing such operation is a good way to avoid performance issues

What about QFormLayout

- QFormLayout has a suitable api for our use case
  - More reliable than QGridLayout for code maintenance
- Hiding the content of a QFormLayout can lead to a spacing issue
  - Specific to QFormLayout

Manipulating a lot of QWidgets

Laying out without layouts

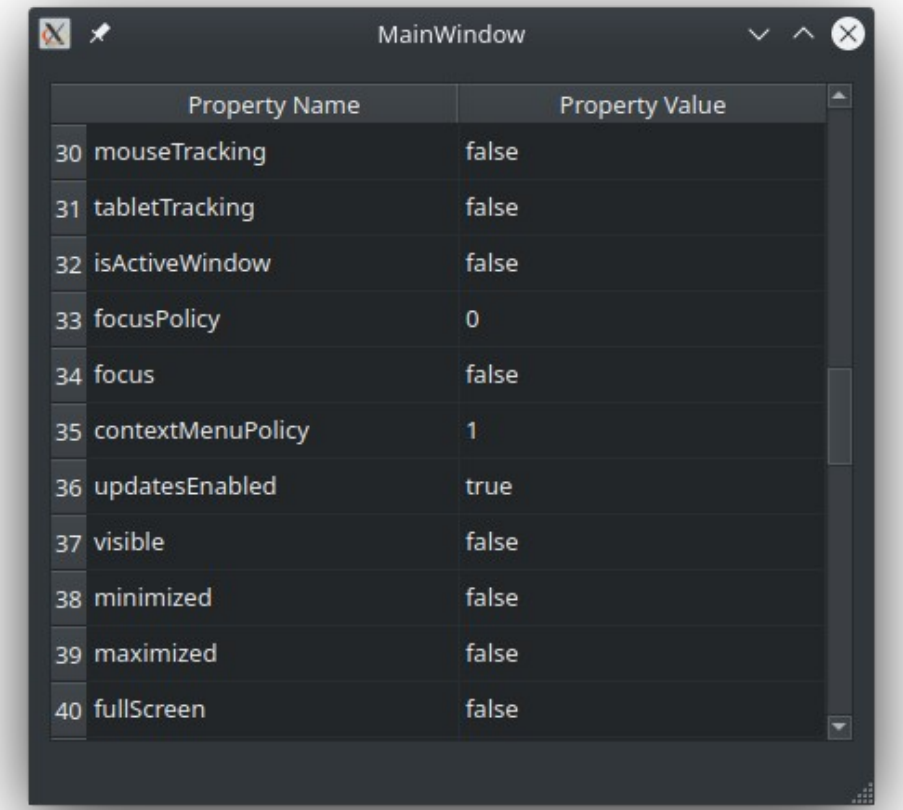
- Some QWidgets are designed specifically to display a lot of information instantly
- This is the case of QListView, QTreeView or QTableView
  - They use a QAbstractItemModel as an entry point
  - They manage their own layout depending on the model
  - They don’t create any widgets but draw each cells manually
- In our sample project, which of these views should we use?

Item views

- Creating your model
  - Built-in models: QStandardItemModel
  - Inherits from QAbstractItemModel or a more derived class
  - All models are compatible with all ItemViews
- In our case:
  - Each row have 2 columns
  - Rows don’t have any children
  - Let’s see QStandardItemModel + QTableView

Using a QTableView

- Proper alignment




```
QStandardItemModel *model = new QStandardItemModel(this);

// Fill model with each property in a standard item
const auto moThis = this->metaObject();
for (int i = 0; i < moThis->propertyCount(); ++i) {
    const auto property = moThis->property(i);
    const QVariant value = property.read(this);

    QStandardItem *propertyName = new QStandardItem(property.name());
    QStandardItem *propertyValue = new QStandardItem;
    propertyValue->setData(value, Qt::DisplayRole);
    propertyValue->setEditable(value.type() == QVariant::Int);

    model->appendRow({propertyName, propertyValue});
}

model->setHeaderData(0, Qt::Horizontal, "Property Name", Qt::DisplayRole);
model->setHeaderData(1, Qt::Horizontal, "Property Value", Qt::DisplayRole);

ui->tableView->setModel(model);
```

Using a QTableView

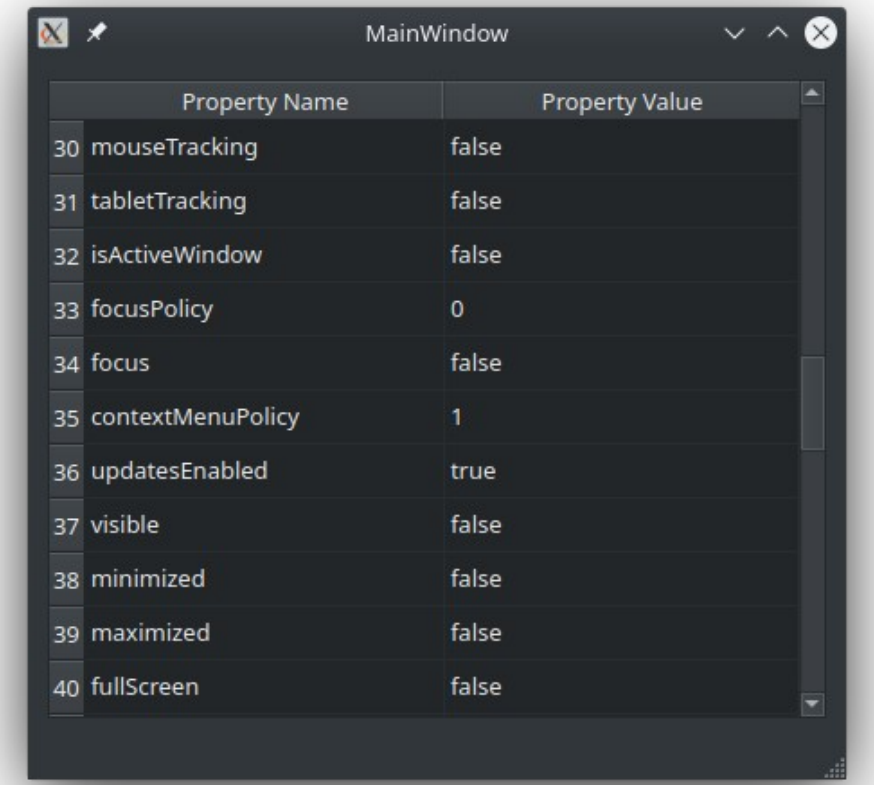
- One issue with QTableView:
  - Cells do not expand to take all the available space by default
  - Can be changed by using the QHeaderView api

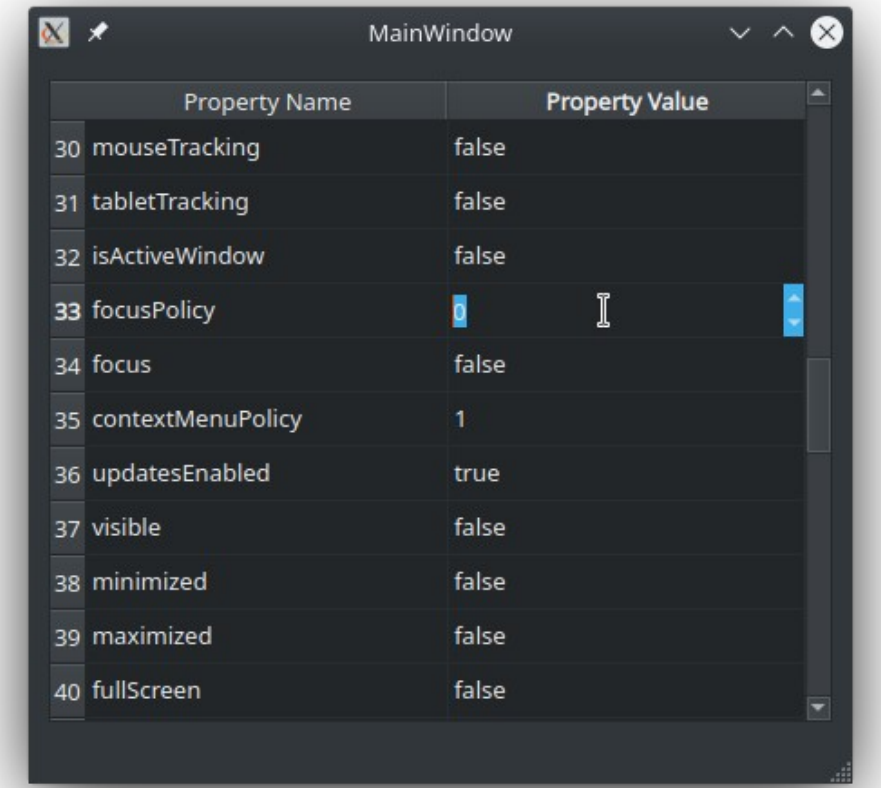
```
ui->modelView->horizontalHeader()->setStretchLastSection(true);
// or
ui->modelView->horizontalHeader()->setStretchLastSection(false);
ui->modelView->horizontalHeader()->setSectionResizeMode(1, QHeaderView::Stretch);
```

- Replacing QTableView by a QTreeView
  - QTreeView expands to take all the available space by default

Using a QTableView

- Display editor





Manipulating a lot of QWidgets: Benchmarking

Layout creation times:

	1 000 rows	10 000 rows	100 000 rows
QGridLayout	≈ 258 ms	≈ 1905 ms	≈ 18.5 s
QTableView	≈ 74 ms	≈ 85 ms	≈ 208 ms
QTreeView	≈ 53 ms	≈ 65 ms + 1 s (display)	≈ 184 ms + 6 s (display)

- Additional display on QTreeview can be nullified if uniformRowHeights is true

Manipulating a lot of QWidgets: Conclusion

Layouts are not the ideal solution when displaying a lot of QWidgets

- Performances can be a real issue
- Using item views solves that and keeps the proper alignment of your UI

But...Customizing an item view can be difficult :

- Creating a complex model may require some more code
- It is not easy to modify the behavior of these views


Adobe

Introducing Gammaray

Franck Arrecot

franck.arrecot@kdab.com

https://github.com/KDAB/GammaRay

The Qt, OpenGL and C++ experts

Thank you for your attention.