Qt Essentials - Model View 2 Module

Qt Essentials - Training Course

Produced by Nokia, Qt Development Frameworks

Material based on Qt 4.7, created on December 15, 2010

http://qt.nokia.com

Module: Model/View II

- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Overview

Custom Model/View

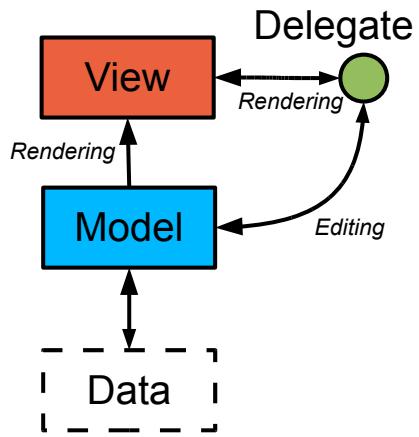
- Editable Models
- Custom Delegates
- Using Data Widget Mapper
- Custom Proxy Models
- Drag and Drop

Module: Model/View II

- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

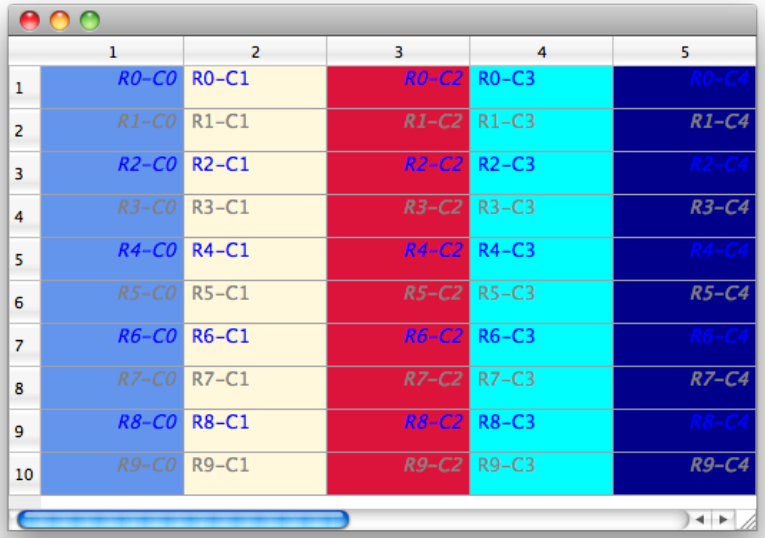
Item Delegates

- `QAbstractItemDelegate` subclasses
  - Control appearance of items in views
  - Provide edit and display mechanisms
- `QItemDelegate`, `QStyledItemDelegate`
  - Default delegates
  - Suitable in most cases
  - Model needs to provide appropriate data
- When to go for Custom Delegates?
  - More control over appearance of items



Item Appearance

Data table shown
has no custom delegate



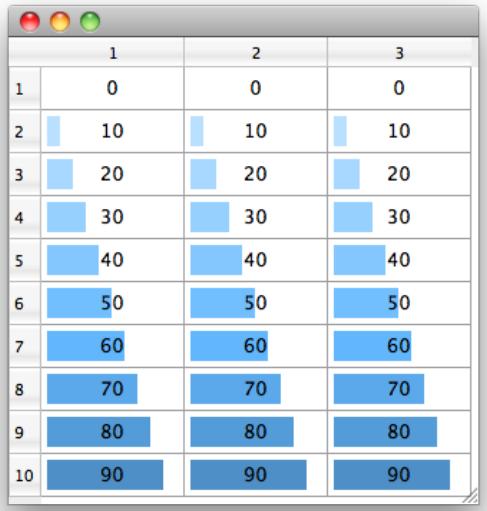
	1	2	3	4	5
1	R0-C0	R0-C1	R0-C2	R0-C3	R0-C4
2	R1-C0	R1-C1	R1-C2	R1-C3	R1-C4
3	R2-C0	R2-C1	R2-C2	R2-C3	R2-C4
4	R3-C0	R3-C1	R3-C2	R3-C3	R3-C4
5	R4-C0	R4-C1	R4-C2	R4-C3	R4-C4
6	R5-C0	R5-C1	R5-C2	R5-C3	R5-C4
7	R6-C0	R6-C1	R6-C2	R6-C3	R6-C4
8	R7-C0	R7-C1	R7-C2	R7-C3	R7-C4
9	R8-C0	R8-C1	R8-C2	R8-C3	R8-C4
10	R9-C0	R9-C1	R9-C2	R9-C3	R9-C4

- No need for custom delegate!
- Use `Qt::ItemRole` to customize appearance

Delegate from QAbstractItemDelegate

```
class BarGraphDelegate : public QAbstractItemDelegate {
public:
  void paint(QPainter *painter,
             const QStyleOptionViewItem &option,
             const QModelIndex &index) const;
  QSize sizeHint(const QStyleOptionViewItem &option,
                 const QModelIndex &index) const;
};
```

See QAbstractItemDelegate Documentation



Integer Value - Bar Graph Delegate

```
void BarGraphDelegate::paint(painter, option, index) const {
  if(index.data(Qt::EditRole).userType() == QVariant::Int) {
    int value = index.data(Qt::EditRole).toInt();
    // prepare rect with a width proportional to value
    QRect rect(option.rect.adjusted(4,4,-4,-4));
    rect.setWidth(rect.width()*value/MAX_VALUE);
    // draw the value bar
    painter->fillRect(rect, QColor("steelblue").lighter(value));
    painter->drawText(option.rect, index.data().toString());
  }
}

QSize BarGraphDelegate::sizeHint(option, index) const {
    Q_UNUSED(index)
    return QSize(MIN_BAR_WIDTH, option.fontMetrics.height());
}
```

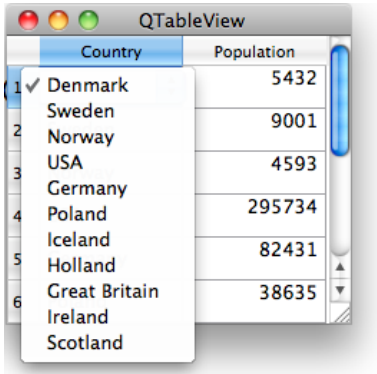
Demo modelview2/ex-bargraphdelegate

Module: Model/View II

- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

An Editor Delegate

- Provides QComboBox
  - for editing a series of values




```
class CountryDelegate : public QItemDelegate
{
public:
  // returns editor for editing data
  QWidget *createEditor( parent, option, index ) const;
  // sets data from model to editor
  void setEditorData( editor, index ) const;
  // sets data from editor to model
  void setModelData( editor, model, index ) const;
  // updates geometry of editor for index
  void updateEditorGeometry( editor, option, index ) const;
};
```

Providing an editor

- Create editor by index

```
QWidget *CountryDelegate::createEditor( ... ) const {
  QComboBox *editor = new QComboBox(parent);
  editor->addItems( m_countries );
  return editor;
}
```

- Set data to editor

```
void CountryDelegate::setEditorData( ... ) const {
  QComboBox* combo = static_cast<QComboBox*>( editor );
  QString country = index.data().toString();
  int idx = m_countries.indexOf( country );
  combo->setCurrentIndex( idx );
}
```

Submitting data to the model

- When user finished editing
  - view asks delegate to store data into model

```
void CountryDelegate::setModelData(editor, model, index) const {
  QComboBox* combo = static_cast<QComboBox*>( editor );
  model->setData( index, combo->currentText() );
}
```

- If editor has finished editing

```
// copy edtitors data to model
emit commitData( editor );
// close/destroy editor
emit closeEditor( editor, hint );
// hint: indicates action performed next to editing
```

Updating the editor's geometry

- Delegate manages editor's geometry
- View provides geometry information
  - `QStyleOptionViewItem`

```
void CountryDelegate::updateEditorGeometry( ... ) const {
  // don't allow to get smaller than editors sizeHint()
  QSize size = option.rect.size().expandedTo(editor->sizeHint());
  QRect rect(QPoint(0,0), size);
  rect.moveCenter(option.rect.center());
  editor->setGeometry( rect );
}
```

Demo modelview2/ex-editordelegate

- Case of multi-index editor
  - Position editor in relation to indexes

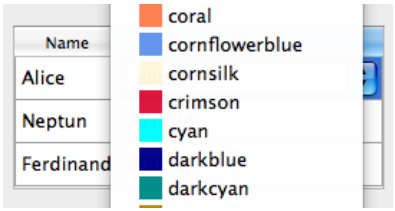
Setting Delegates on Views

- `view->setItemDelegate( ... )`
- `view->setItemDelegateForColumn( ... )`
- `view->setItemDelegateForRow(... )`

Type Based Delegates

- Our color editor widget

```
class ColorListEditor : public QComboBox {
 ...
};
```



- Registering editor for type QVariant::Color

```
QItemEditorFactory *factory = new QItemEditorFactory;
QItemEditorCreatorBase *creator =
        new QStandardItemEditorCreator<ColorListEditor>();
// registers item editor creator given type of data
factory->registerEditor(QVariant::Color, creator);
// new and existing delegates will use new factory
QItemEditorFactory::setDefaultFactory(factory);
```

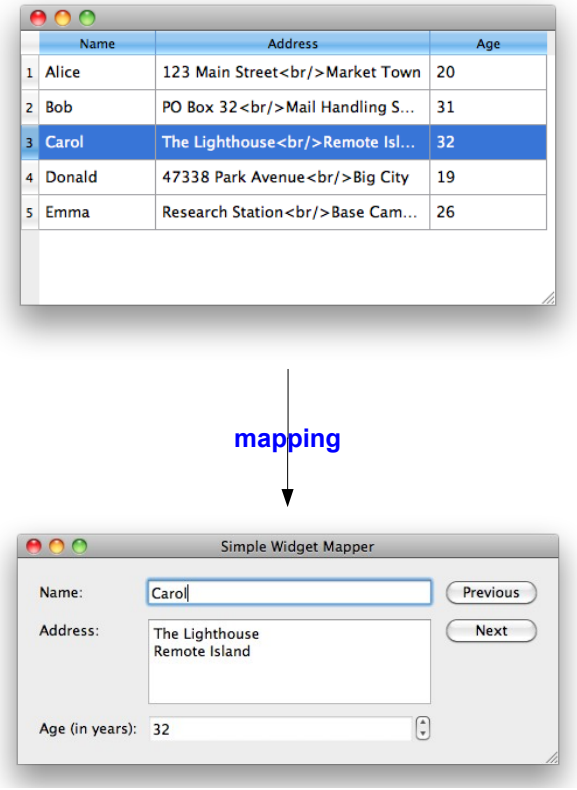
Demo $QTDIR/examples/itemviews/coloreditorfactory

Module: Model/View II

- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Data Widget Mapper- `QDataWidgetMapper`

- Maps model sections to widgets
- Widgets updated, when current index changes
- Orientation
  - Horizontal => Data Columns
  - Vertical => Data Rows



Using QDataWidgetMapper

- Mapping Setup

```
mapper = new QDataWidgetMapper(this);
mapper->setOrientation(Qt::Horizontal);
mapper->setModel(model);
// mapper->addMapping( widget, model-section)
mapper->addMapping(nameEdit, 0);
mapper->addMapping(addressEdit, 1);
mapper->addMapping(ageSpinBox, 2);
// populate widgets with 1st row
mapper->toFirst();
```

- Track Navigation

```
connect(nextButton, SIGNAL(clicked()),
          mapper, SLOT(toNext()));
connect(previousButton, SIGNAL(clicked()),
          mapper, SLOT(toPrevious()));
```

Demo $QTDIR/examples/itemviews/simplewidgetmapper

Mapped Property - The USER Property

```
class QLineEdit : public QWidget
{
  ...
  Q_PROPERTY(QString text
             READ text WRITE setText NOTIFY textChanged
             USER true) // USER property
  ...
};
```

- USER indicates property is user-editable property
- Only one USER property per class
- Used to transfer data between the model and the widget

```
addMapping(lineEdit, 0); // uses "text" user property
addMapping(lineEdit, 0, "inputMask"); // uses named property
```

Demo $QTDIR/examples/itemviews/combowidgetmapper

Module: Model/View II

- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Drag and Drop for Views

- Enable the View

```
// enable item dragging
view->setDragEnabled(true);
// allow to drop internal or external items
view->setAcceptDrops(true);
// show where dragged item will be dropped
view->setDropIndicatorShown(true);
```

- Model has to provide support for drag and drop operations

```
Qt::DropActions MyModel::supportedDropActions() const
{
  return Qt::CopyAction | Qt::MoveAction;
}
```

- Model needs to support actions
  - For example `Qt::MoveAction`
    - implement `MyModel::removeRows( ... )`

Drag and Drop with `QStandardItemModel`

- Setup of Model
  - Model is ready by default
  - `model->mimeTypes()`
    - "application/x-qabstractitemmodeldatalist"
    - "application/x-qstandarditemmodeldatalist"
  - `model->supportedDragActions()`
    - `QDropEvent::Copy | QDropEvent::Move`
  - `model->supportedDropActions()`
    - `QDropEvent::Copy | QDropEvent::Move`
- Setup of Item

```
item = new QStandardItem("Drag and Droppable Item");
// drag by default copies item
item->setDragEnabled(true);
// drop mean adding dragged item as child
item->setDropEnabled(true);
```

Demo modelview2/ex-dndlist

Drag and Drop on `QAbstractItemModel`

```
class MyModel : public QAbstractItemModel {
public:
  // actions supported by the data in this model
  Qt::DropActions supportedDropActions() const;
  // for supported index return Qt::ItemIs(Drag|Drop)Enabled
  Qt::ItemFlags flags(const QModelIndex &index) const;
  // returns list of MIME types that are supported
  QStringList QAbstractItemModel::mimeTypes() const;
  // returns object with serialized data in mime formats
  QMimeData *mimeData(const QModelIndexList &indexes) const;
  // true if data and action can be handled, otherwise false
  bool dropMimeData(const QMimeData *data, Qt::DropAction action,
          int row, int column, const QModelIndex &parent);
};
```

Demo modelview2/ex-drag-and-drop

Module: Model/View II

- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

A Custom Tree Model in 5 Steps

1. Read-Only Model
2. Editable Model
3. Insert-Remove Model
4. Lazy Model
5. Drag and Drop Model

A (Simple) Node Structure

```
class Node {
public:
  Node(const QString& aText="No Data", Node *aParent=0);
  ~Node();
  QVariant data() const;
public:
  QString text;
  Node *parent;
  QList<Node*> children;
};
```

Demo modelview2/ex-treemodel (node.h)

Read-Only Model

```
class ReadOnlyModel : public QAbstractItemModel {
public:
  ...
  QModelIndex index( row, column, parent ) const;
  QModelIndex parent child ) const;

  int rowCount( parent ) const;
  int columnCount( parent ) const;
  QVariant data( index, role) const;

protected: // important helper methods
  QModelIndex indexForNode(Node *node) const;
  Node* nodeForIndex(const QModelIndex &index) const;
  int rowForNode(Node *node) const;
};
```

Demo modelview2/ex-treemodel (readonlymodel.h)

Editable Model

```
class EditableModel : public ReadOnlyModel {
public:
  ...
  bool setData( index, value, role );
  Qt::ItemFlags flags( index ) const;
};
```

Demo modelview2/ex-treemodel (editablemodel.h)

Insert Remove Model

```
class InsertRemoveModel : public EditableModel {
public:
    ...
    void insertNode(Node *parentNode, int pos, Node *node);
    void removeNode(Node *node);
    void removeAllNodes();
};
```

Demo modelview2/ex-treemodel (insertremovemodel.h)

Lazy Model

```
class LazyModel : public ReadOnlyModel {
public:
  ...
  bool hasChildren( parent ) const;
  bool canFetchMore( parent ) const;
  void fetchMore( parent );
};
```

Demo modelview2/ex-treemodel (lazymodel.h)

DnD Model

```
class DndModel : public InsertRemoveModel {
public:
  ...
  Qt::ItemFlags flags( index ) const;
  Qt::DropActions supportedDragActions() const;
  Qt::DropActions supportedDropActions() const;
  QStringList mimeTypes() const;
  QMimeData *mimeData( indexes ) const;

  bool dropMimeData(data, dropAction, row, column, parent);
  bool removeRows(row, count, parent);
  bool insertRows(row, count, parent);
};
```

Demo modelview2/ex-treemodel (dndmodel.h)