Qt Essentials - Model View 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

- Model/View Concept
- Showing Simple Data
- Proxy Models
- Custom Models

Overview

Using Model/View

- Introducing to the concepts of model-view
- Showing Data using standard item models
- Understand the limitations of standard item models
- How to interface your model with a data backend
- Understand what are proxy models and hot to use them

Custom Models

- Writing a simple read-only custom model.

Module: Model/View

- Model/View Concept
- Showing Simple Data
- Proxy Models
- Custom Models

Why Model/View?

- **Isolated domain-logic**
  - From input and presentation
- **Makes Components Independent**
  - For Development
  - For Testing
  - For Maintenance
- **Foster Component Reuse**
  - Reuse of Presentation Logic
  - Reuse of Domain Model

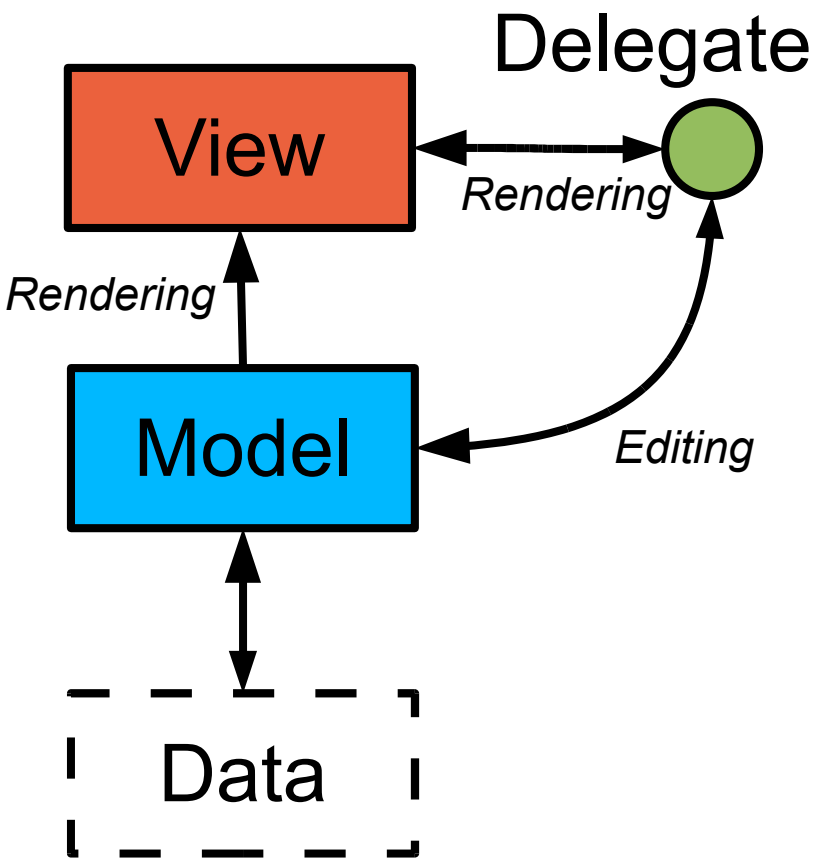
Model/View-Components

Model View Components

- **View**
  - Displays data structure
- **Delegate**
  - Renders single data
  - Supports editing data
- **Model**
  - Unified adapter to data

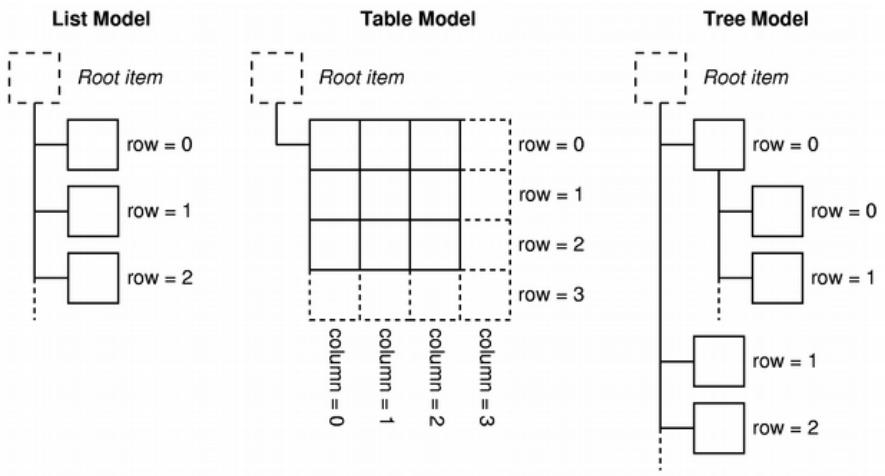
Model View Infrastructure

- ***Item***
  - Imaginary unit of data in model
- **Index**
  - Used to locate item in model
- Lets see how simple it can be
  - Demo modelview/ex-simple



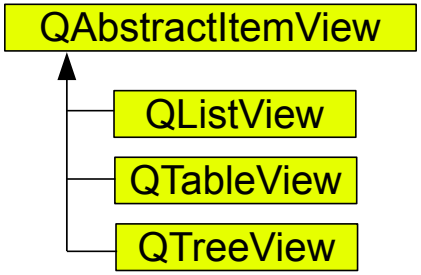
Model Structures

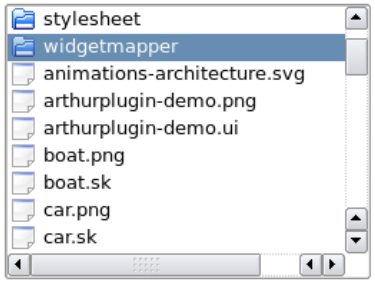
- **List** - One-Dimensional
  - Rows
- **Table** - Two-Dimensional
  - Rows
  - Columns
- **Tree** - Three-Dimensional
  - Rows
  - Columns
  - Parent/Child

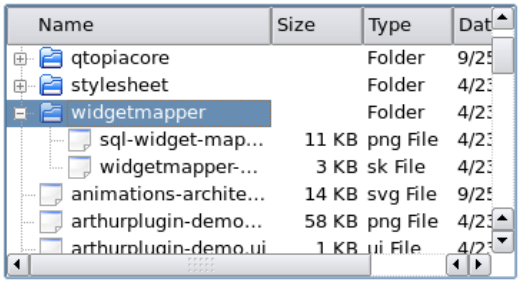


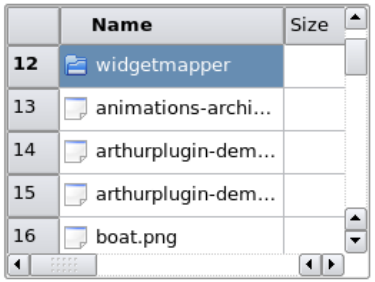
Display the Structure - View Classes

- **QAbstractItemView**
  - Abstract base class for all views
- **QListView**
  - List or icon view
- **QTreeView**
  - Items of data in hierarchical list
- **QTableView**
  - Item based row/column view









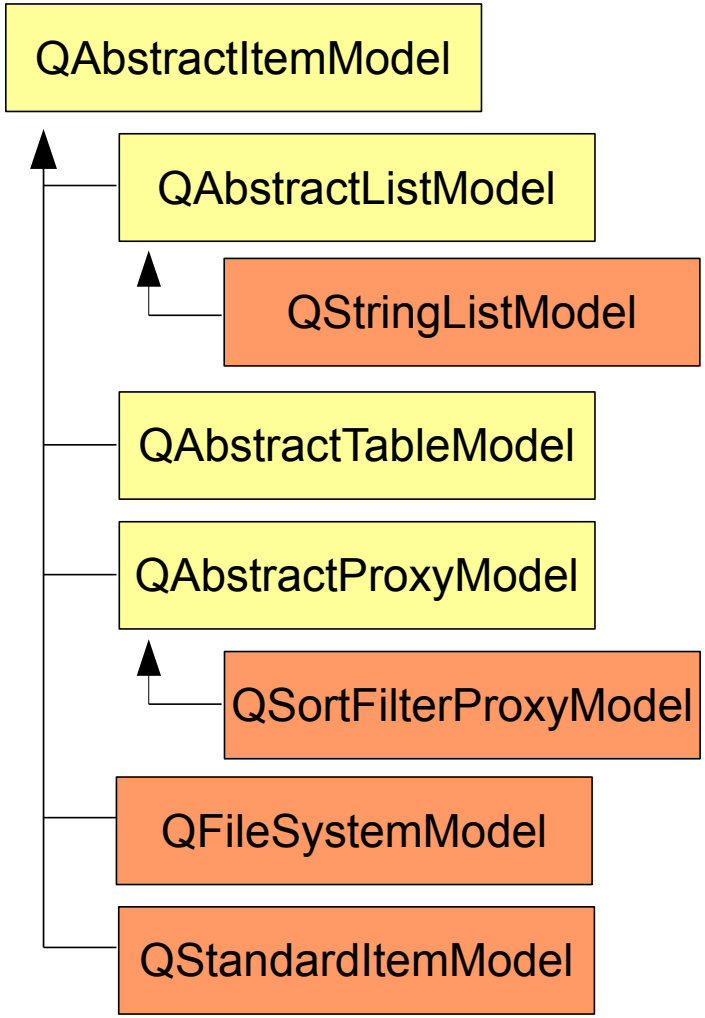
- **Other Views**
  - `QHeaderView` - Header for item views
  - `QColumnView` - A cascading list

See View Classes Documentation

Adapts the Data - Model Classes

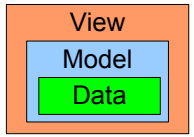
- **`QAbstractItemModel`**
  - Abstract interface of models
- **Abstract Item Models**
  - Implement to use
- **Ready-Made Models**
  - Convenient to use
- **Proxy Models**
  - Reorder/filter/sort your items

See Model Classes Documentation

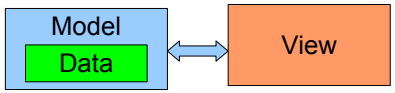


Data - Model - View Relationships

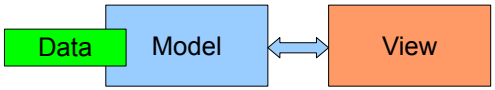
- **Item Widgets**
  - All combined
  - Model is your data



- **Standard Item Model**
  - Data+Model combined
  - View is separated
  - Model is your data



- **Custom Item Models**
  - Model is adapter to data
  - View is separated



Adressing Data - `QModelIndex`

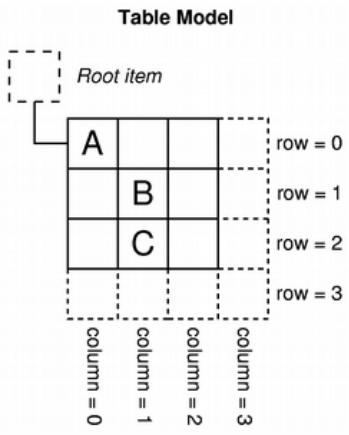
- Refers to item in model
- Contains all information to specify location
- Located in given row and column
  - May have a parent index

- **QModelIndex API**
  - `row()` - row index refers to
  - `column()` - column index refers to
  - `parent()` - parent of index
    - or `QModelIndex()` if no parent
  - `isValid()`
    - Valid index belongs to a model
    - Valid index has non-negative row and column numbers
  - `model()` - the model index refers to
  - `data( role )` - data for given role

QModelIndex in Table/Tree Structures

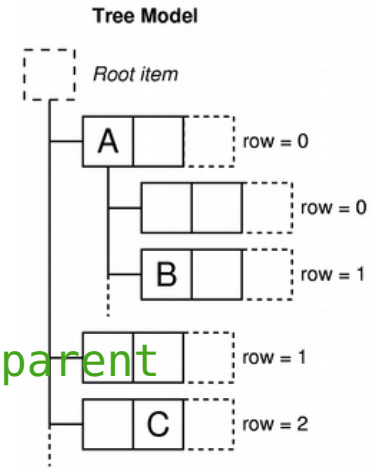
- **Rows and columns**
  - Item location in table model
  - Item has no parent (parent.isValid() == false)

```
indexA = model->index(0, 0, QModelIndex());
indexB = model->index(1, 1, QModelIndex());
indexC = model->index(2, 1, QModelIndex());
```



- **Parents, rows, and columns**
  - Item location in tree model

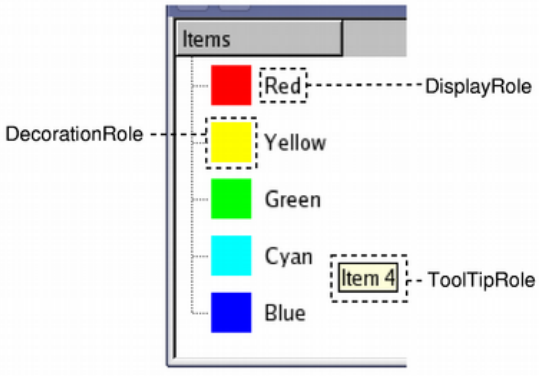
```
indexA = model->index(0, 0, QModelIndex());
indexC = model->index(2, 1, QModelIndex());
// asking for index with given row, column and parent
indexB = model->index(1, 0, indexA);
```



See Model Indexes Documentation

Item and Item Roles

- **Item performs various roles**
  - for other components (delegate, view, ...)
- **Supplies different data**
  - for different situations
- **Example:**
  - `Qt::DisplayRole` used displayed string in view



- **Asking for data**

```
QVariant value = model->data(index, role);
// Asking for display text
QString text = model->data(index, Qt::DisplayRole).toString()
```

- **Standard roles**
  - Defined by `Qt::ItemDataRole`
  - See enum Qt::ItemDataRole Documentation

Recap of Model/View Concept

- **Model Structures**
  - List, Table and Tree
- **Components**
  - Model - Adapter to Data
  - View - Displays Structure
  - Delegate - Paints Item
  - Index - Location in Model
- **Views**
  - `QListView`
  - `QTableView`
  - `QTreeView`
- **Models**
  - `QAbstractItemModel`
  - Other Abstract Models
  - Ready-Made Models
  - Proxy Models
- **Index**
  - `row(),column(),parent()`
  - `data( role )`
  - `model()`
- **Item Role**
  - `Qt::DisplayRole`
  - Standard Roles in `Qt::ItemDataRoles`

Things you may want to customize

Models - `QAbstractItemModel`

- If model is your data
  - `QStandardItemModel` or Item Widgets
- Otherwise
  - Adapt own data by subclassing a model

Delegate - `QAbstractItemDelegate`

- Standard Delegate sufficient
- Custom Delgate
  - To control display/edit of data

Views - `QAbstractItemView`

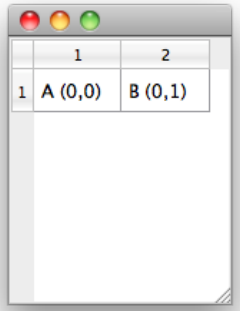
- Almost always be used as-is
- Exceptional to add new view

Module: Model/View

- Model/View Concept
- Showing Simple Data
- Proxy Models
- Custom Models

QStandardItemModel - Convenient Model

- QStandardItemModel
  - Classic item-based approach
  - Only practical for small sets of data




```
model = new QStandardItemModel(parent);
item = new QStandardItem("A (0,0)");
model->appendRow(item);
model->setItem(0, 1, new QStandardItem("B (0,1)"));
item->appendRow(new QStandardItem("C (0,0)"));
```

- *"C (0,0)" - Not visible. (table view is only 2-dimensional)*

- React on click

```
connect(m_view, SIGNAL(clicked(QModelIndex)) ...
```

  - In slot ...

```
QStandardItem *item = model->itemFromIndex(index);
```

See QStandardItemModel Documentation

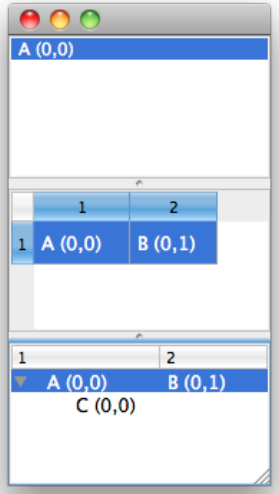
Sharing Models and Selection

```
// create a model
model = ... // with some items

// and several views
list = ...; table = ...; tree = ...;

// You can share model with views
list->setModel(model);
table->setModel(model);
tree->setModel(model);

// Even can share selection
list->setSelectionModel(tree->selectionModel());
table->setSelectionModel(tree->selectionModel());
```



Finishing Touch

- Customizing view headers

```
// set horizontal headers
model->setHorizontalHeaderItem(0, new QStandardItem("Column 1");
model->setHorizontalHeaderItem(1, new QStandardItem("Column 2");
// hide vertical headers on table
table->verticalHeader().hide();
```

- Customizing Items

```
item->setEditable(false); // disable edit
item->setCheckable(true); // checkbox on item
```

- Customize selection

```
// allow only to select single rows on table
table->setSelectionBehavior(QAbstractItemView::SelectRows);
table->setSelectionMode(QAbstractItemView::SingleSelection);
```

Demo modelview/ex-showdata

Selections - `QItemSelectionModel`

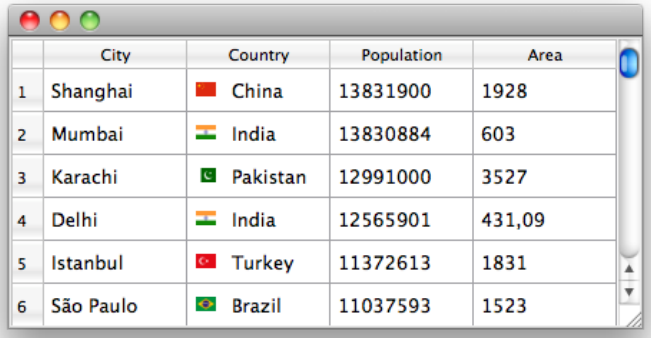
- Keeps track of selected items in view
- Not a `QAbstractItemModel`, just `QObject`
- `QItemSelectionModel API`
  - `currentIndex()`
  - *signal* `currentChanged(current, previous)`
  - `QItemSelection selection()`
    - List of selection ranges
  - `select( ... )`
  - *signal* `selectionChanged(selected, deselected)`

```
// selecting a range
selection = new QItemSelection(topLeft, bottomRight);
view->selectionModel()->select(selection);
```

Meet the City Engine

- Our Demo Model
  - 62 most populous cities of the world
  - Data in CSV file
- Data Columns
  - *City | Country | Population | Area | Flag*
- Implemented as data backend
  - Internal implementation is hidden
  - Code in `CityEngine` class

```
City;Country;Population;Area
Shanghai;China;13831900;1928
Mumbai;India;13830884;603;22
Karachi;Pakistan;12991000;35
Delhi;India;12565901;431.09;
Istanbul;Turkey;11372613;183
São Paulo;Brazil;11037593;15
Moscow;Russia;10508971;1081;
Seoul;South Korea;10464051;6
Beijing;China;10123000;1368.
Mexico City;Mexico;8841916;1
Tokyo;Japan;8795000;617;22px
Kinshasa;Democratic Republic
Jakarta;Indonesia;8489910;66
New York City;United States;
```



	City	Country	Population	Area
1	Shanghai	China	13831900	1928
2	Mumbai	India	13830884	603
3	Karachi	Pakistan	12991000	3527
4	Delhi	India	12565901	431,09
5	Istanbul	Turkey	11372613	1831
6	São Paulo	Brazil	11037593	1523

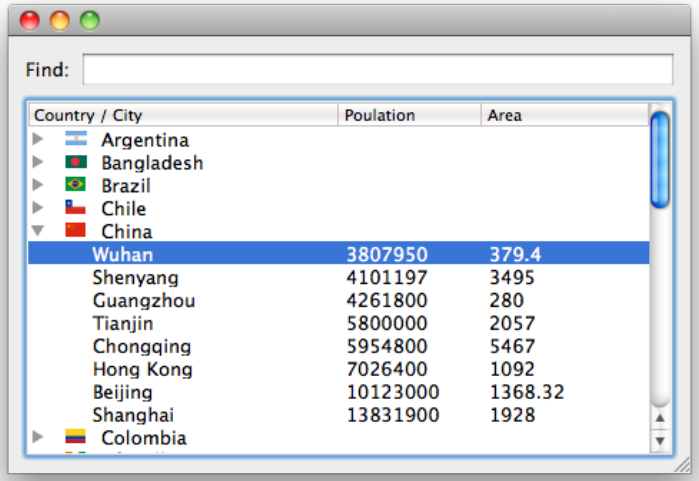
Our Backend CityEngine API

```
public CityEngine : public QObject {
  // returns all city names
  QStringList cities() const;
  // returns country by given city name
  QString country(const QString &cityName) const;
  // returns population by given city name
  int population(const QString &cityName) const;
  // returns city area by given city name
  qreal area(const QString &cityName) const;
  // returns country flag by given country name
  QIcon flag(const QString &countryName) const;
  // returns all countries
  QStringList countries() const;
  // returns city names filtered by country
  QStringList citiesByCountry(const QString& countryName) const;
};
```

- Demo /modelview/ex-standarditemmodel

Lab: Tree Model for CityEngine

- Implement `createTreeModel()` in `mainwindow.cpp`
- Display cities grouped by countries
- **Optional**
  - Provide a find for country field.
  - Found countries shall be selected



Lab modelview/lab-treecityengine

Module: Model/View

- Model/View Concept
- Showing Simple Data
- Proxy Models
- Custom Models

Proxy Model - `QSortFilterProxyModel`

- `QSortFilterProxyModel`
  - Transforms structure of source model
  - Maps indexes to new indexes

```
view = new QListView(parent);
// insert proxy model between model and view
proxy = new QSortFilterProxyModel(parent);
proxy->setSourceModel(model);
view->setModel(proxy);
```

Note: Need to load all data to sort or filter

Sorting/Filtering - `QSortFilterProxyModel`

- **Filter with Proxy Model**

```
// filter column 1 by "India"
proxy->setFilterWildcard("India");
proxy->setFilterKeyColumn(1);
```

- **Sorting with Proxy Model**

```
// sort column 0 ascending
view->setSortingEnabled(true);
proxy->sort(0, Qt::AscendingOrder);
```

- **Filter via `QLineEdit` signal**

```
connect(m_edit, SIGNAL(textChanged(QString)),
    proxy, SLOT(setFilterWildcard(QString)));
```

Demo modelview/ex-sortfiltertableview

Module: Model/View

- Model/View Concept
- Showing Simple Data
- Proxy Models
- Custom Models

Implementing a Model

- Variety of classes to choose from
  - **QAbstractListModel**
    - One dimensional list
  - **QAbstractTableModel**
    - Two-dimensional tables
  - **QAbstractItemModel**
    - Generic model class
  - **QStringListModel**
    - One-dimensional model
    - Works on string list
  - **QStandardItemModel**
    - Model that stores the data

- **Notice:** Need to subclass *abstract* models

Step 1: Read Only List Model

```
class MyModel: public QAbstractListModel {
public:
  // return row count for given parent
  int rowCount( const QModelIndex &parent) const;
  // return data, based on current index and requested role
  QVariant data( const QModelIndex &index,
                 int role = Qt::DisplayRole) const;
};
```

Demo modelview/ex-stringlistmodel

Step 2: Suppling Header Information

```
QVariant MyModel::headerData(int section,
                             Qt::Orientation orientation,
                             int role) const
{
  // return column or row header based on orientation
}
```

Demo modelview/ex-stringlistmodel-2

Step 3: Enabling Editing

```
// should contain Qt::ItemIsEditable
Qt::ItemFlags MyModel::flags(const QModelIndex &index) const
{
 return QAbstractListModel::flags() | Qt::ItemIsEditable;
}

// set role data for item at index to value
bool MyModel::setData( const QModelIndex & index,
                       const QVariant & value,
                       int role = Qt::EditRole)
{
  ... = value; // set data to your backend
  emit dataChanged(topLeft, bottomRight); // if successful
}
```

Demo modelview/ex-stringlistmodel-3

Step 4: Row Manipulation

```
// insert count rows into model before row
bool MyModel::insertRows(int row, int count, parent) {
   beginInsertRows(parent, first, last);
   // insert data into your backend
   endInsertRows();
}

// removes count rows from parent starting with row
bool MyModel::removeRows(int row, int count, parent) {
   beginRemoveRows(parent, first, last);
   // remove data from your backend
   endRemoveRows();
}
```

Demo modelview/ex-stringlistmodel-4

A Table Model

- 2-dimensional model (rows x columns)
- `int columnCount(parent)`
  - Enabling columns

```
int MyModel::columnCount ( parent ) const {
  // return number of columns for parent
}

QVariant MyModel::data( index, role ) const {
 // adapt to react on requested column from index
}
```

Demo modelview/ex-tablemodel

Lab: City Table Model

- Please implement a City Table Model
- Given:
  - The data in `CityEngine`
  - A main test function
- Your Task:
  - Adapt the data to a table model
- **Optional**
  - Make the model editable
  - Enable adding/removing cities

Lab modelview/lab-citymodel