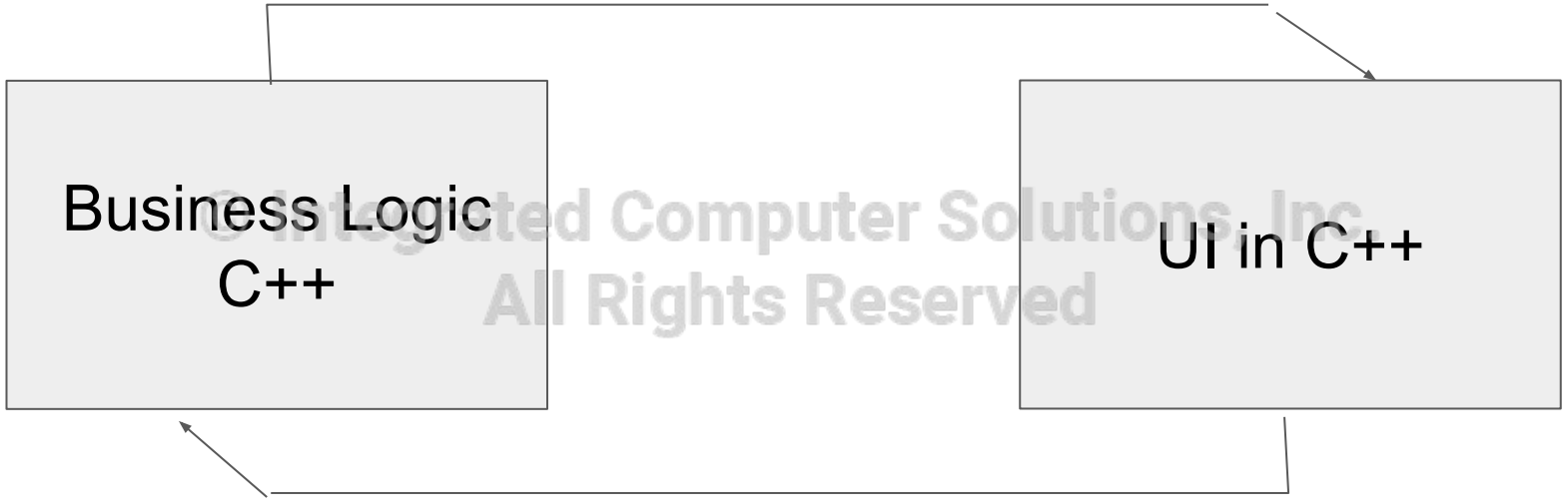
Qt For Beginners Training

Part 5

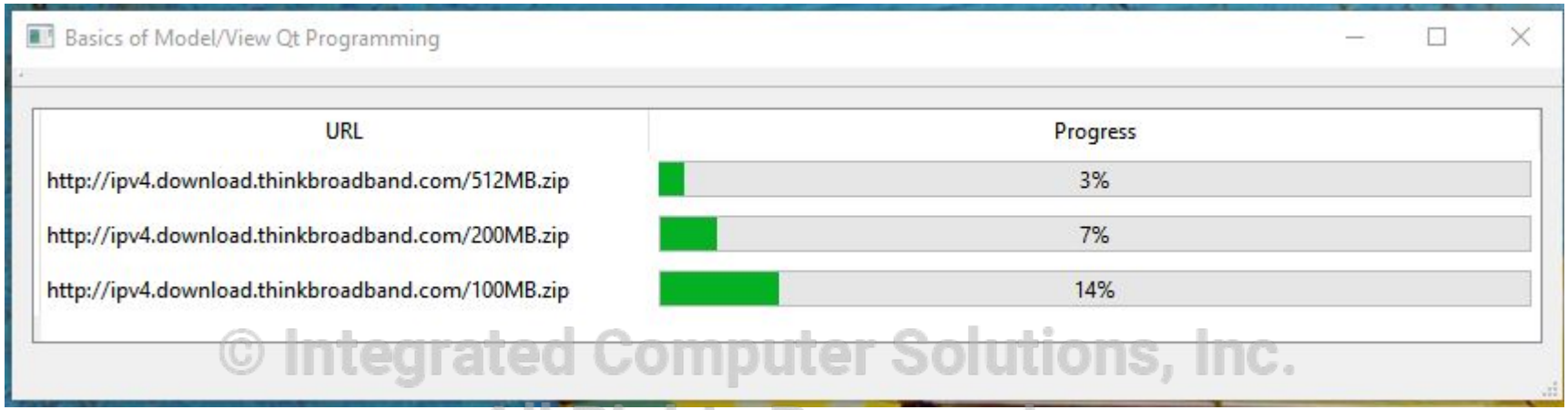
Business Logic/UI Paradigm

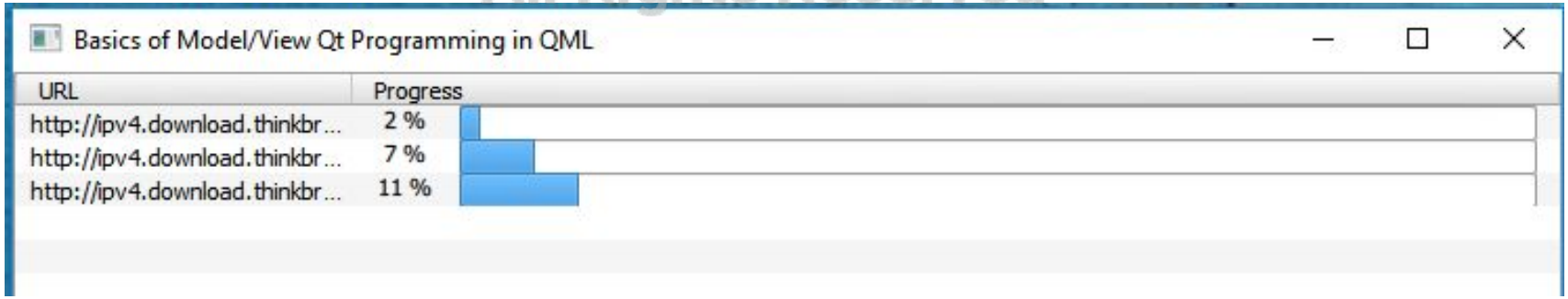
-



An Example

- An Example using Widgets



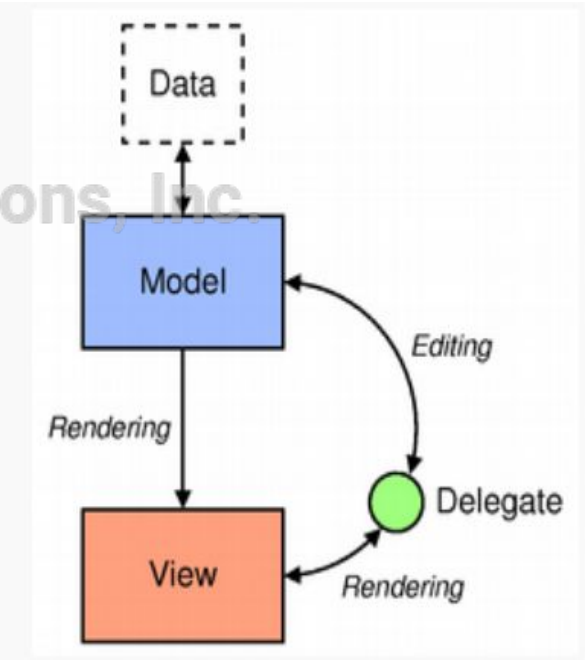


- Same Example using QML

The Model

- The model communicates to the view a source of data (e.g. text, icons, font)
- All item models are based on the ***QAbstractItemModel*** class
- All models inherit from ***QAbstractItemModel***

- ***QAbstractItemModel***
  - If structure of model is a tree
- ***QAbstractListModel***
  - If structure of model is a one-dimensional list
- ***QAbstractTableModel***
  - If structure of model is a table



The Data Function

- All abstract models must re-implement this function:

QVariant QAbstractItemModel::data(const QModelIndex &index, int role) const

```
QVariant MyModel::data(const QModelIndex &index, int role) const
{
   if (role == Qt::DisplayRole)
   {
      return QString("Row%1, Column%2")
                  .arg(index.row() + 1)
                  .arg(index.column() +1);
   }
   return QVariant();
}
```

What is QModelIndex?

- The ***QModelIndex*** class is used to locate data in a data model

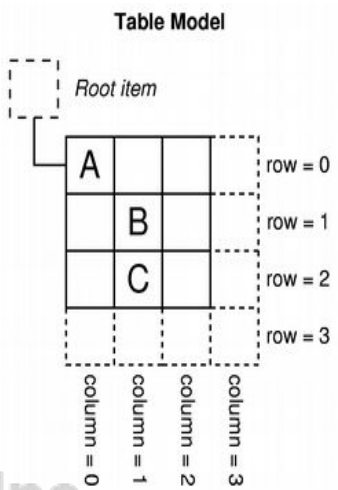
- Temporary model indexes are provided by the ***QModelIndex*** class, and persistent model indexes are provided by the ***QPersistentModelIndex***

- ***int QModelIndex::row() const***
- ***int QModelIndex::column() const***
- ***QModelIndex QModelIndex::parent() const***

QModelIndex in Table/Tree Structures

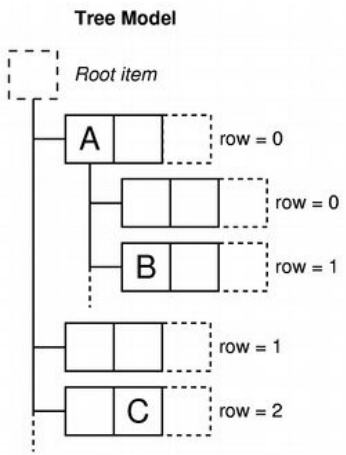
- **Rows and columns**
  - Item location in table model
  - Item has no parent (parent.isValid() == false)

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

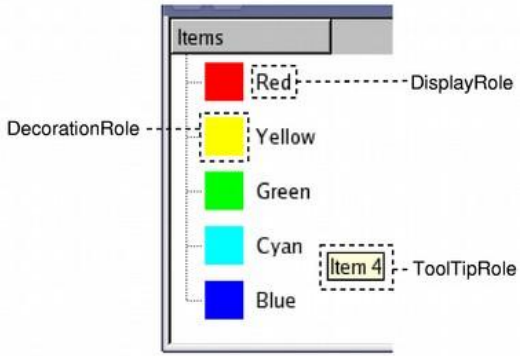


- **Parents, rows, and columns**
  - Item location in tree model

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



And What about 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

When Data Changes

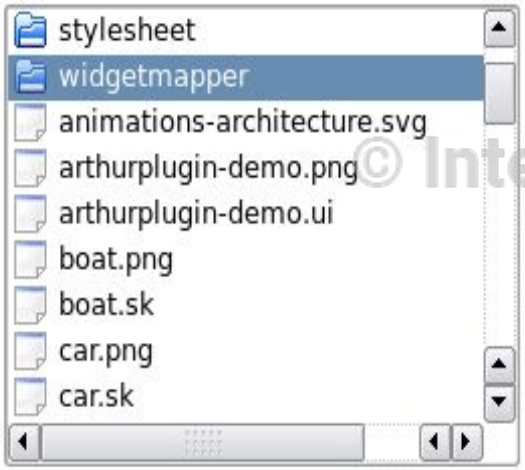
```
void DownloadModel::onUpdateProgress(qint64, qint64)
{
   Downloads* download = qobject_cast<Downloads*>(sender());
   if( download != Q_NULLPTR )
   {
      int row = this->downloads.indexOf(download);
      QModelIndex index = this->index(row,1);
      emit dataChanged(index, index); // refresh download column
   }
}
```

When Data is Inserted/Removed

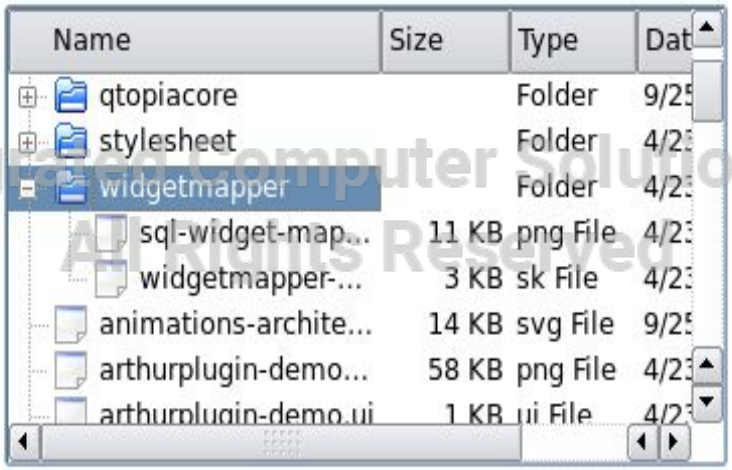
```
/ insert count rows into model before row
 bool MyModel::insertRows(int row, int count, parent) {
   beginInsertRows(parent, first, last);
   // insert data into your backend
   // ... endInsertRows();
}

// 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();
}
```

The View Classes in Widgets

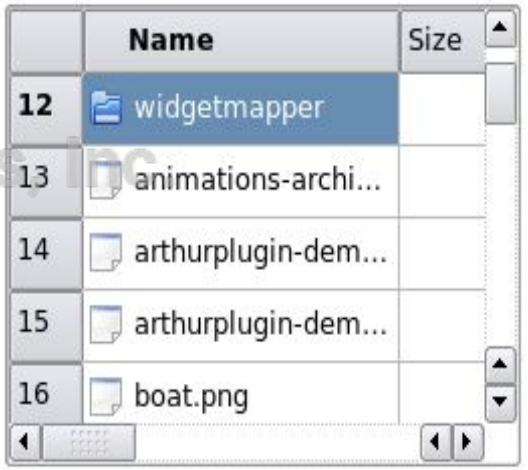
QListView



QTreeView

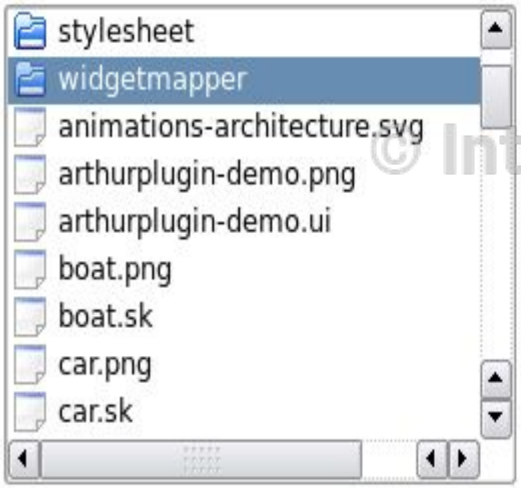


QTableView

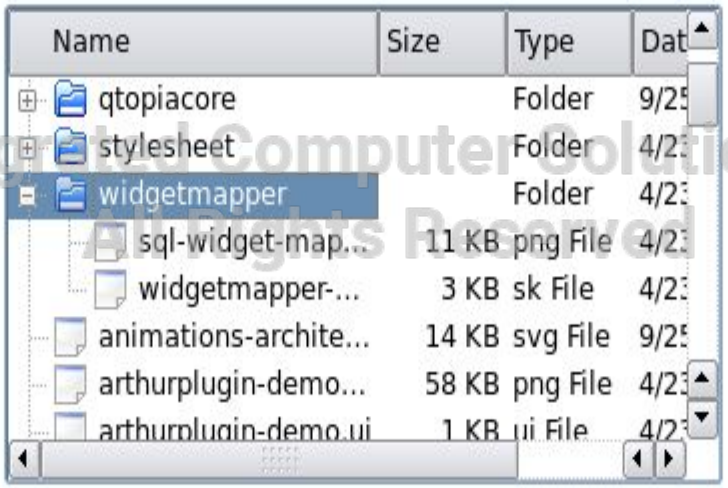


The View Classes in QML

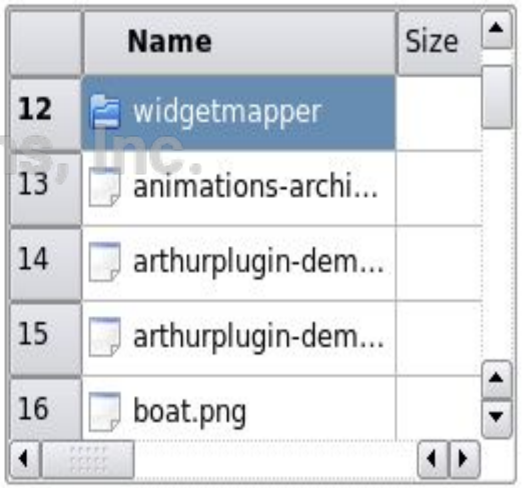
ListView



TreeView



TableView



How it looks in C++ Code

```
model = new DownloadModel(this);
ui->tableView->setModel(model);

ProgressBarDelegate* delegate = new ProgressBarDelegate(this);
ui->tableView->setItemDelegateForColumn(1, delegate);
```

A Word on the Delegate

```
class BarGraphDelegate : public QAbstractItemDelegate {

public:

  void paint(QPainter *painter,

             const QStyleOptionViewItem &option,

             const QModelIndex &index) const;


};

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
```

	1	2	3	4
1	0	12/31/13	0	0
2	10	12/21/13	10.1	10
3	20	12/11/13	20.2	20
4	30	12/1/13	30.3	30
5	40	11/21/13	40.4	40
6	50	11/11/13	50.5	50
7	60	11/1/13	60.6	60
8	70	10/22/13	70.7	70
9	80	10/12/13	80.8	80
10	90	10/2/13	90.9	90

Further Reading

- **Model/View Programming**

  http://doc.qt.io/qt-5/model-view-programming.html

- **Model/View Programming Tutorial**

  http://doc.qt.io/qt-5/modelview.html

Thank you!

Be sure to check out www.ics.com/training for additional resources