Ensuring Efficiency in Factory and Production Planning

Building complex QML-GUIs with QtQuick2
Content

- Introduction to IPO.Plan’s software
- Data model concept
- Worker Gantt - View
- Material Zone Editor - View
- Rack Planogram - View
- Live implementing
Introduction to IPO.Plan’s software

- Located in Leonberg (services) and Ulm (research and development department)
- 60 employees approximately
- Our process experts provide manifold services, supported by our homemade expert tools
- Main customers are Daimler AG, Siemens AG, Eisenmann GmbH & Co. KG, Liebherr GmbH
Introduction to IPO.Plan’s software

- Software products made by IPO.Plan:
  - Purely software
  - Software + Hardware
Data model concept
MVC-Concept

- Model
  - C++
  - BasicModels
  - Methods
  - Signal/Slots
  - Invokables

- ViewController
  - C++
  - SortFilterModel
  - RepresentationModel
  - CollectionModel
  - Properties
  - Signal/Slots
  - Invokables

- View
  - QML
  - Possibility to implement Model-Behavior
  - Set needed context properties
Data model concept

Model

- Using only one custom-defined data role ("objectRole")
- Providing instance references directly to QML-Code by this role
- You have to inherit the "QAbstractListModel"

→ Detailed description in attachments
Data model concept
SortFilterModel

Example
- Ascending index order
- Filter indices < 4

Model<T>

SourceModel

SortFilterModel<T>

Example

- Ascending index order
- Filter indices < 4
Data model concept

RepresentationModel

Example
- Use a type R to represent a single source instance

Model<T>
- T4
- T3
- T1
- T2

SourceModel

RepresentationModel<R>
- R4
- R3
- R1
- R2

Keep in mind that instances of R may already exist or be allocated by the RepresentationModel.
Data model concept

RepresentationModel

Example 2
- Use a type $S$ to represent multiple source instances

```
Model<T>  
T_4  
T_3  
T_1  
T_2  

SourceModel

RepresentationModel<S>  
S_{34}  
S_{12}  
```
Data model concept
CollectionModel

Example
- Collect by a random criteria

Model<T>

T₁
T₂
T₃
T₄

SourceModel₁

Model<U>

U₁
U₂
U₃
U₄

SourceModel₂

CollectionModel<R>

U₁
R₁
U₂
R₂
U₃
R₃
U₄
R₄

T and U have to have the mutual super class R
You can combine the mentioned models to create manifold behavior

Just queue them in a model chain

Later: RackPlanogram in IPO.Log
Worker Gantt - View

Worker
Use a SortFilterModel for Products

ProductLine
Use a ListModel of Activities and ActivityGroups

Activity

ActivityGroup
The hierarchy of the Worker Gantt - View

WorkerGantt

 Worker

 ProductLine

 Activities and ActivityGroups
Material Zone Editor - View

**Area**
Contains all components in a simpleListModel

**ComponentOnArea**

**ComponentOnAreaEdit**

**ComponentOnAreaGroup**
Uses a SortFilterModel of selected Components
Planogram

PlanogramArea
Uses a SortFilterModel to get all areas with selected components

PlanogramRackComponent

PlanogramRackInfoGrid

PlanogramInfoGrid
The hierarchy of the Rack Planogram - View

Planogram
  \- PlanogramArea
    \- PlanogramRackComponent
      \- PlanogramRackInfoGrid
    \- PlanogramInfoGrid
  PlanogramBoxGroupComponent
    \- PlanogramBoxGroupInfoGrid
    \- PlanogramInfoGrid
We are hiring!

If you...
- ...are interested in our stuff
- ...are of course firm in Qt, QML
- ...are firm in C++, C#, Java or another language
- ...want to work in a young, creative and flexible team
- ...search for challenges
- ...looking for virtual reality, simulation and 3D graphic

Maybe you can find the right job for you at IPO.Plan

→ You are also welcome to our stand here at QtDeveloperDays
THANK YOU 😊
IPO.Plan GmbH

- Martin Lang
  E-Mail: martin.lang@ipoplan.de

- Office Ulm.
  Grünhofgasse 3 | D-71229 Ulm
  Fon: +49.7152.70010.82 | Fax: +49.7152.70010.99
Your contact person

IPO.Plan GmbH

- Dipl.-Ing. Dennis Effmert
  E-Mail: dennis.effmert@ipoplan.de

- Office Ulm.
  Grünhofgasse 3 | D-71229 Ulm
  Fon: +49.7152.70010.82 | Fax: +49.7152.70010.99
class DataListModel : public QAbstractListModel
{
    Q_OBJECT
public:
...
enum Roles {ObjectRole = Qt::UserRole + 1};
void append(QObject *newObject);
QHash<int, QByteArray> roleNames() const;
QVariant data(const QModelIndex &index, int role) const;
...
protected:
    QList<QObject*> m_objects;
}
... void DataListModel::append(QObject *object)
{
    beginInsertRows(QModelIndex(), m_objects.count(), m_objects.count());

    m_objects.append(object);
    // To observe the QObject::destroyed() signal
    if (m_tracking)
        trackObject(object);

    endInsertRows();
}
...
DataListModel::roleNames() const
{
    QHash<int, QByteArray> roles;
    roles[DataListModel::ObjectRole] = "object";
    return roles;
}
...

...
Attachment: Data model concept

DataModel.cxx

```cpp
...
QVariant DataListModel::data(const QModelIndex &index, int role) const
{
    if (index.row() < 0 || index.row() >= m_objects.size())
        return QVariant();

    switch (role)
    {
    case ObjectRole:
        return QVariant::fromValue(m_objects.at(index.row()));
    }

    return QVariant();
}
...
```