Practical QML

Burkhard Stubert
Chief Engineer, Embedded Use
www.embeddeduse.com
Contents

- Key Navigation
- Dynamic Language Change
- Themes
Key Navigation in Cars

Navigation clusters for controlling in-vehicle infotainment systems
Key Navigation in Harvesters

Driver terminals for Harvesters and tractors
Active Focus

- QML item needs **active focus** to receive key events
- Only single item has active focus
- Property `Item.activeFocus` (read-only)
  - True if item has active focus
- Function `Item.forceActiveFocus()`
  - Forces item to have active focus
- Property `Item.focus`
  - Requests active focus when set to true
Focus Scopes

✧ Component FocusScope
  - Controls which child item gets active focus
  - Needed for introducing new components with key handling

✧ When FocusScope receives active focus:
  - Last item to request focus gains active focus
  - When last item is FocusScope, active focus is forwarded to FocusScope
Who gains active focus?

FocusScope A

FocusScope B1

Rectangle C1
focus: true

Rectangle C2

FocusScope B2
focus: true

Rectangle D1

Rectangle D1
focus: true
FlagButton {
  id: france
  KeyNavigation.backtab: spain
  KeyNavigation.tab: italy
}
Crossing FocusScopes with KeyNavigation

- Enclose flag rows with FocusScope as preliminary for FlagRow component
- What happens when crossing to other flag row?
Crossing FocusScopes with KeyNavigation (2)

- KeyNavigation stops when crossing to other FocusScope
- Reason: FocusScope changes focus instead of activeFocus
Solution:

FlagButton {
  id: italy
  KeyNavigation.backtab: france
  KeyNavigation.tab: uk
  Keys.onTabPressed: uk.forceActiveFocus()
}

KeyNavigation not suited for components

- Reason: top item of component always a FocusScope
- KeyNavigation forces monolithic code
Introducing a Generic Cursor Component

✧ Forces guiding the solution
  - Write code for state machine, visual items, key and mouse handling only once
  - Use only one way to move active focus: forceActiveFocus()
  - Tab and backtab chains must take component structures into account
Moving Active Focus in Item Hierarchy

KeyNavigation structure needs four properties: tabUp/tabDown and backtabUp/backtabDown
Introducing New Attached Property KeyNav

✧ KeyNav
  - `tabUp : Item`
  - `tabDown: Item`
  - `backtabUp: Item`
  - `backtabDown: Item`

✧ Attached properties ≈ multiple inheritance
  - Save us from declaring four properties in each QML component

✧ Example use in middle FlagButton

```qml
FlagButton {
    id: flag1
    KeyNav.backtabUp: flag0.KeyNav.backtabDown
    KeyNav.tabUp: flag2.KeyNav.tabDown
}
```
Handling the Return Key in Cursor

```
signal released()

Keys.onPressed: {
    if (event.key === Qt.Key_Return) {
        root.state = "pressed"
        event.accepted = true
    }
}

Keys.onReleased: {
    if (event.key === Qt.Key_Return) {
        root.state = "focused"
        root.released()
        event.accepted = true
    }
}
```

- Make key and mouse handling look the same for clients
- Move out of if-clause to stop default key handling of ListView (Up and Down)
- Also add "pressed" State to states property
- Forward in Cursor instance of FlagButton:
  `onReleased: root.release()`
Key Navigation in ListViews

 Forces guiding the solution

- ListView item has no way to find out previous and next item
  - Cannot use forceActiveFocus()

- Changing currentIndex changes focus
  - Reimplement doTab() and doBacktab() for Cursor

- Special cases for moving the active focus into the ListView with Tab and Backtab
  - Implement doTab() and doBacktab() for ListView
Key Navigation in ListViews (2)

- Extract `doTab()` and `doBacktab()` from `Cursor` into `ButtonCursor` and `ListViewItemCursor`.

  - `ButtonCursor`:
    - `doTab()` and `doBacktab()` use `forceActiveFocus()` to move active focus.

  - `ListViewItemCursor`:
    - `doTab()` and `doBacktab()` change `currentIndex` to move active focus.
Key Navigation in ListViews (3)

✧ Every ListView inherits from BaseListView
✧ BaseListView provides tabbing and backtabbing into list view

In BaseListView:

```javascript
function doTab() {
    root.positionViewAtIndex(0, ListView.Beginning)
    root.currentIndex = 0
    root.forceActiveFocus()
}
```

- Ensure that first item will be visible
- Request focus for first item
- Forces active focus on ListView, which passes it to first item
MouseArea {
  anchors.fill: parent
  onPressed: {
    root.doMousePress()
    root.state = "pressed"
    mouse.accepted = true
  }
  onReleased: {
    if (root.activeFocus) {
      root.state = "focused"
      root.released()
    }
    mouse.accepted = true
  }
}
Adding Mouse Handling to Cursor Components (2)

In ButtonCursor:

```javascript
function doMousePress() {
    root.forceActiveFocus()
}
```

In ListViewItemCursor:

```javascript
function doMousePress() {
    delegateRoot.ListView.view.currentIndex = index
    delegateRoot.ListView.view.view.forceActiveFocus()
}
```

For the case when the flag row has active focus and the user clicks in list view. Avoids multiple cursors.
Contents

- Key Navigation
- Dynamic Language Change
- Themes
# Dynamic Language Change

## German Cities

<table>
<thead>
<tr>
<th>City</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>Berlin</td>
</tr>
<tr>
<td>Hamburg</td>
<td>Hamburg</td>
</tr>
<tr>
<td>Munich</td>
<td>Bavaria</td>
</tr>
<tr>
<td>Cologne</td>
<td>North Rhine-Westphalia</td>
</tr>
<tr>
<td>Frankfurt am Main</td>
<td>Hessen</td>
</tr>
<tr>
<td>Stuttgart</td>
<td>Baden Württemberg</td>
</tr>
</tbody>
</table>

## Deutsche Städte

<table>
<thead>
<tr>
<th>City</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>Berlin</td>
</tr>
<tr>
<td>Hamburg</td>
<td>Hamburg</td>
</tr>
<tr>
<td>München</td>
<td>Bayern</td>
</tr>
<tr>
<td>Köln</td>
<td>Nordrhein-Westfalen</td>
</tr>
<tr>
<td>Frankfurt am Main</td>
<td>Hessen</td>
</tr>
<tr>
<td>Stuttgart</td>
<td>Baden Württemberg</td>
</tr>
</tbody>
</table>
Dynamic Language Change for QWidgets

- `QCoreApplication::installTranslator()` sends `LanguageChange` event to application object
- `QApplication::event()` posts `LanguageChange` event to every top-level widget (QWidget*)
- `QWidget::event()` calls `changeEvent()` on the widget and sends `LanguageChange` event to all its children
  - `changeEvent()` is called on every widget in the widget tree rooted at a top-level widget
Problems in QML

- Not a single QWidget in QML applications
  - Not even QQuickView derives from QWidget
- QApplication not used in QML applications
  - Note: QApplication derives from QGuiApplication

Need to rebuild LanguageChange infrastructure in QML
Dynamic Language Change in QML

- TranslationManager emits signal languageChanged()
- Qt/C++ classes (e.g., list models) connect signal with their retranslate() slot
- Every qsTr() call in QML must be reevaluated when signal emitted
Changing the Language

- TranslationManager::setLanguage(language)
  - Load translation file for language in QTranslator
  - Remove old translator from application
  - Install new translator in application
  - emit languageChanged(language)
- Call setLanguage() before main view of application is created
- Call setLanguage() when user changes language
Retranslating Qt/C++ Models

- Equivalent to reimplementing `changeEvent()` and calling `retranslateUi()`
- In constructor of model class:
  ```cpp
class ModelClass {
private:
  TranslationManager *tm;
public:
  ModelClass() {
    connect(tm, SIGNAL(languageChanged(QString)), this, SLOT(retranslate(QString)));
  }
};
```
```cpp
void BiggestCitiesModel::retranslate(const QString &language)
{
    emit titleChange();
    CityDatabase::instance()->retranslate(language);
    emit dataChanged(index(0), index(m_cities.count() - 1));
}
```

- Notify QML code that the title property has changed.
  - QML calls `title()`, which returns `tr(rawTitle())`.
- Delegate retranslation, as the model is a "view" on the database.
- Notify QML ListView that all its items have changed and need reloading.
const char *CityDatabase::m_strings[][2] = {
    { QT_TR_NOOP("Munich"), QT_TR_NOOP("Bavaria") }, ...

void CityDatabase::retranslate(const QString &language) {
    if (m_currentLanguage != language) {
        for (int i = 0; i < m_cities.count(); ++i) {
            m_cities[i]->setName(tr(m_strings[i][0]));
            ...
        }
        m_currentLanguage = language;
    }
}

Guard against multiple “views” (e.g., German cities, British cities) requesting retranslation to same language

Reset visible members (e.g., city name, state) with new translation of raw string
Reevaluating qsTr on Language Change

Text {
    text: qsTr("City:") + g_tr.languageChanged
    ...
}

Use Property Binding:
- Whenever g_tr.languageChanged changes, text must be reevaluated:
- qsTr() is called and returns translation for new language
In TranslationManager:

Q_PROPERTY(QString languageChanged
 READ emptyString
 NOTIFY languageChanged)

QString emptyString() const {
 return "";
}

Emitting this signal forces QML to call emptyString(), the READ method of languageChanged property

Empty string can be appended to translated string without changing anything
On instance of QQuickView:

```cpp
view->rootContext()->setContextProperty("g_tr", TranslationManager::instance());
```

Makes pointer to TranslationManager globally available in QML under name g_tr.
Contents

✧ Key Navigation
✧ Dynamic Language Change
➢ Themes
Dynamic Theme Change
Theming QML Code

Rectangles:

Unthemed

```
Rectangle {
    color: index % 2 === 0 ? 
        "#1E90FF" :
        "#00BFFF"
}
```

Themed

```
Rectangle {
    color: index % 2 === 0 ? 
        g_theme.listViewItem.
        backgroundColor : 
        g_theme.listViewItem.
        backgroundColorAlt
    
    g_theme.listViewItem.
    textColor
}
```

Rows:

```
Row {
    Text {
        text: city.name
        color: "#191970"
    }
}
```

```
Row {
    Text {
        text: city.name
        color: g_theme.listViewItem.
        textColor
    }
}
```
Implementing the Themes

QtObject {
    property QtObject listViewItem : QtObject {
        property color backgroundColor: “#1E90FF”
        property color backgroundColorAlt: “#00BFFF”
        property color textColor: “#191970”
    }
}

QtObject {
    property QtObject listViewItem : QtObject {
        property color backgroundColor: “#A5A5A5”
        property color backgroundColorAlt: “#818181”
        property color textColor: “#1E1E1E”
    }
}
In top-level QML item (main.qml)
property alias g_theme: loader.item
Loader { id: loader }
Component.onCompleted: {
    loader.source = Qt.resolveUrl("BlueTheme.qml")
}
Connections {
    target: g_viewer
    onThemeChanged: {
        loader.source = Qt.resolvedUrl(theme + "Theme.qml")
    }
}
Thank you!