Qt Developer Days 2013

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

Qt Developer Days 2013

Driver terminals for
Harvesters and tractors

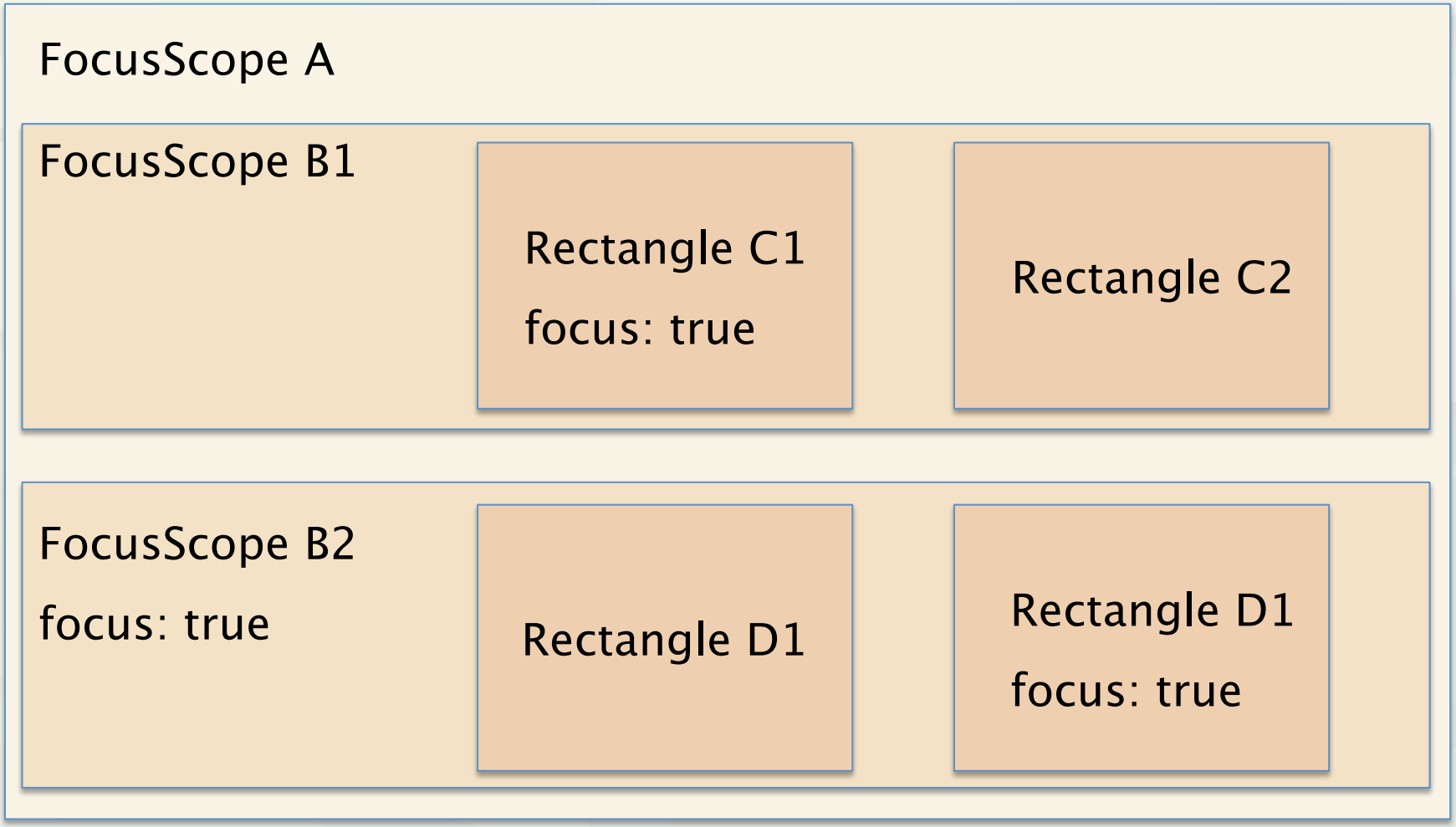
Active Focus

- 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

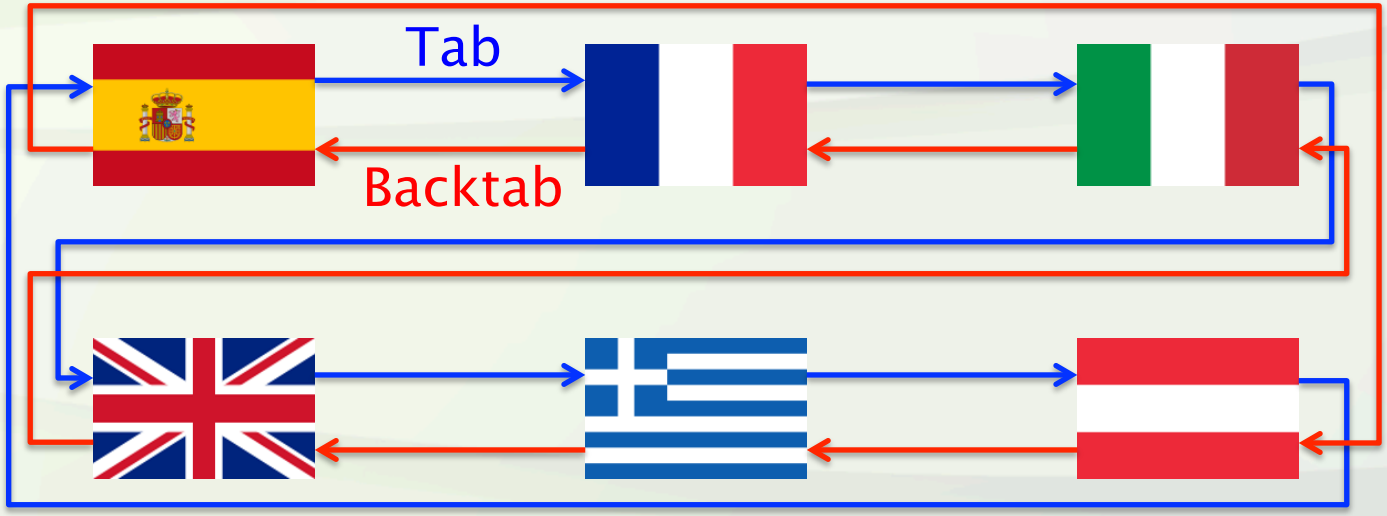
Focus Scopes

- 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?

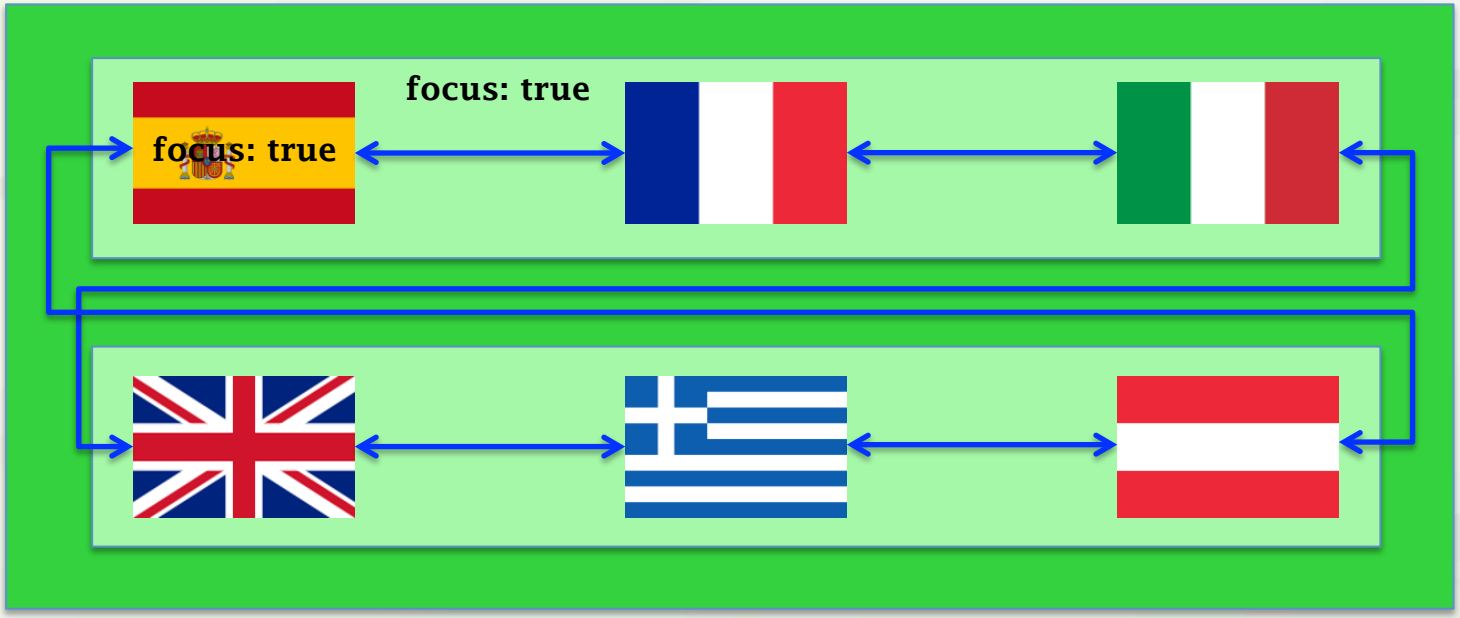


Recap: KeyNavigation Attached Property



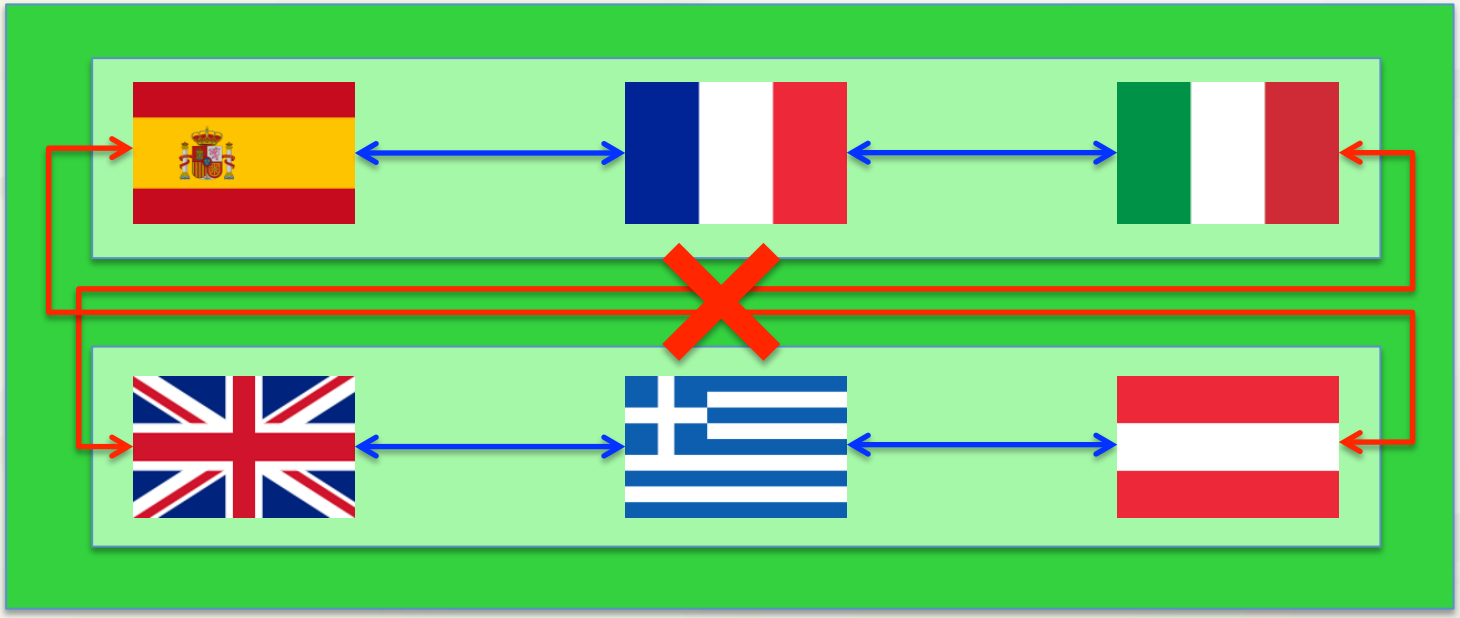

```
FlagButton {
    id: france
    KeyNavigation.backtab: spain
    KeyNavigation.tab: italy
```

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

Crossing Focus Scopes with KeyNavigation (3)

- Solution:

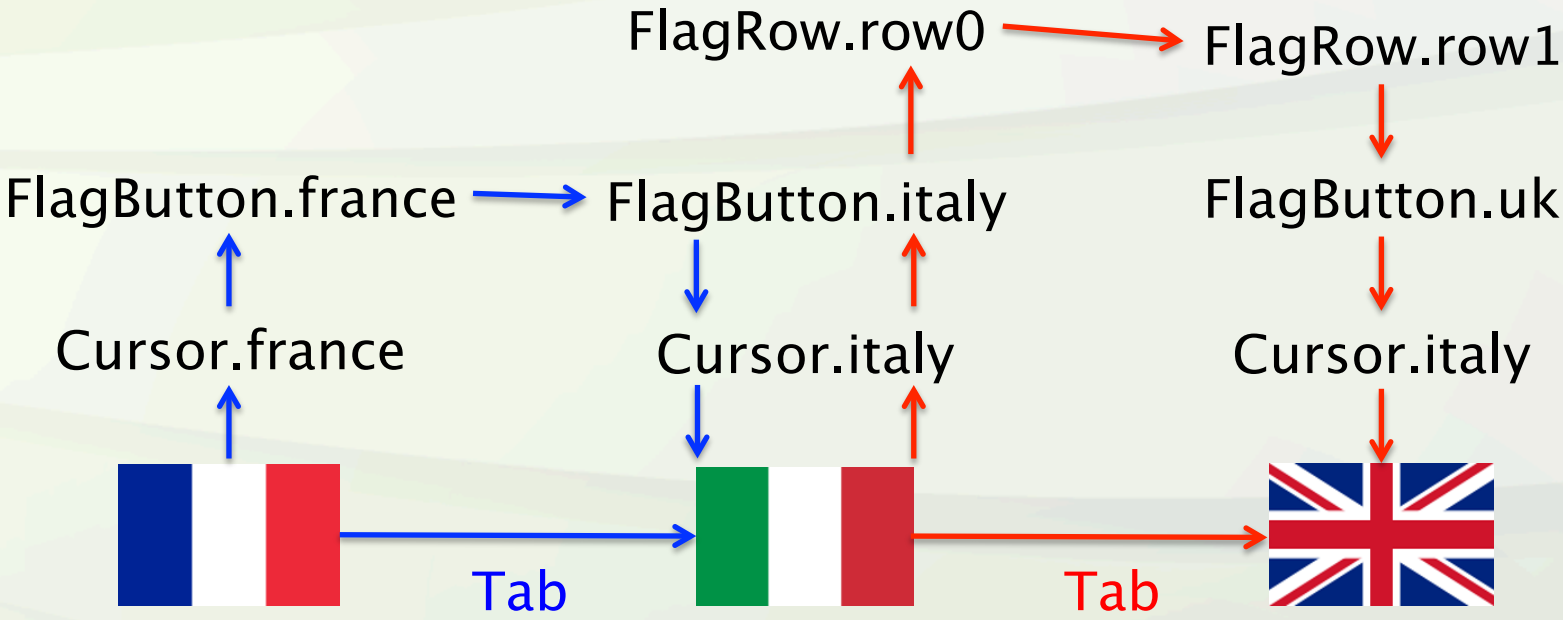
```
FlagButton {
    id: italy
    KeyNavigation.backtab: france
    KeyNavigation.tab: uk
    Keys.onTabPressed: uk.forceActiveFocus()
```

- 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/backtabDow

Introducing New Attached Property KeyNav

- KeyNav
  - tabUp : Item
  - backtabUp: Item
  - tabDown: Item
  - backtabDown: Item
- Attached properties ≈ multipe inheritance
  - Save us from declaring four properties in each QML component
- Example use in middle FlagButton

```
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

Also add “pressed” State to states property

Move out of if-clause to stop default key handling of ListView (Up and Down)

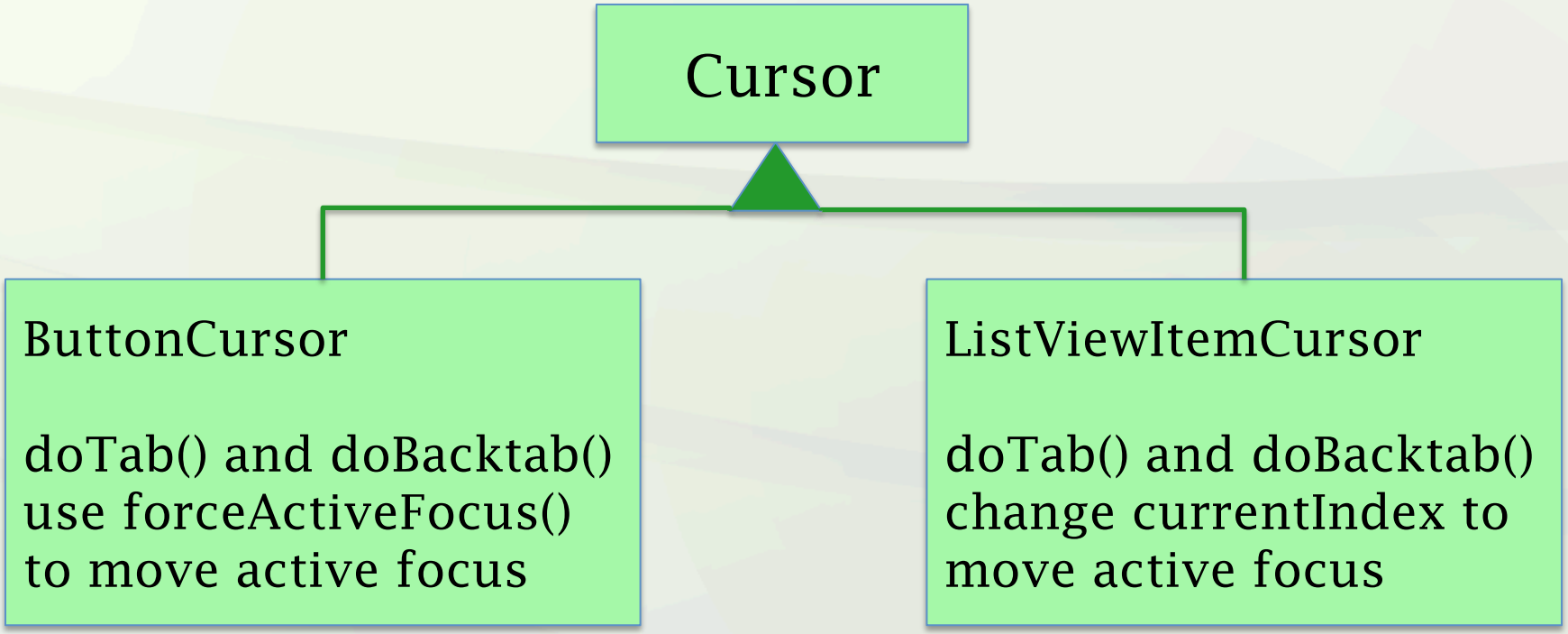
Forward in Cursor instance of FlagButton: onReleased: root.release()

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



Key Navigation in ListViews (3)

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

In BaseListView:

```
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

Adding Mouse Handling to Cursor Components


```
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
    }
}
```

Active focus on item pressed, no dereferencing of tab chain needed

Mouse press different for buttons and list view items

Do not execute “release” when item lost focus, e.g., when error dialog opened

Adding Mouse Handling to Cursor Components (2)

In ButtonCursor:

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

In ListViewItemCursor:

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

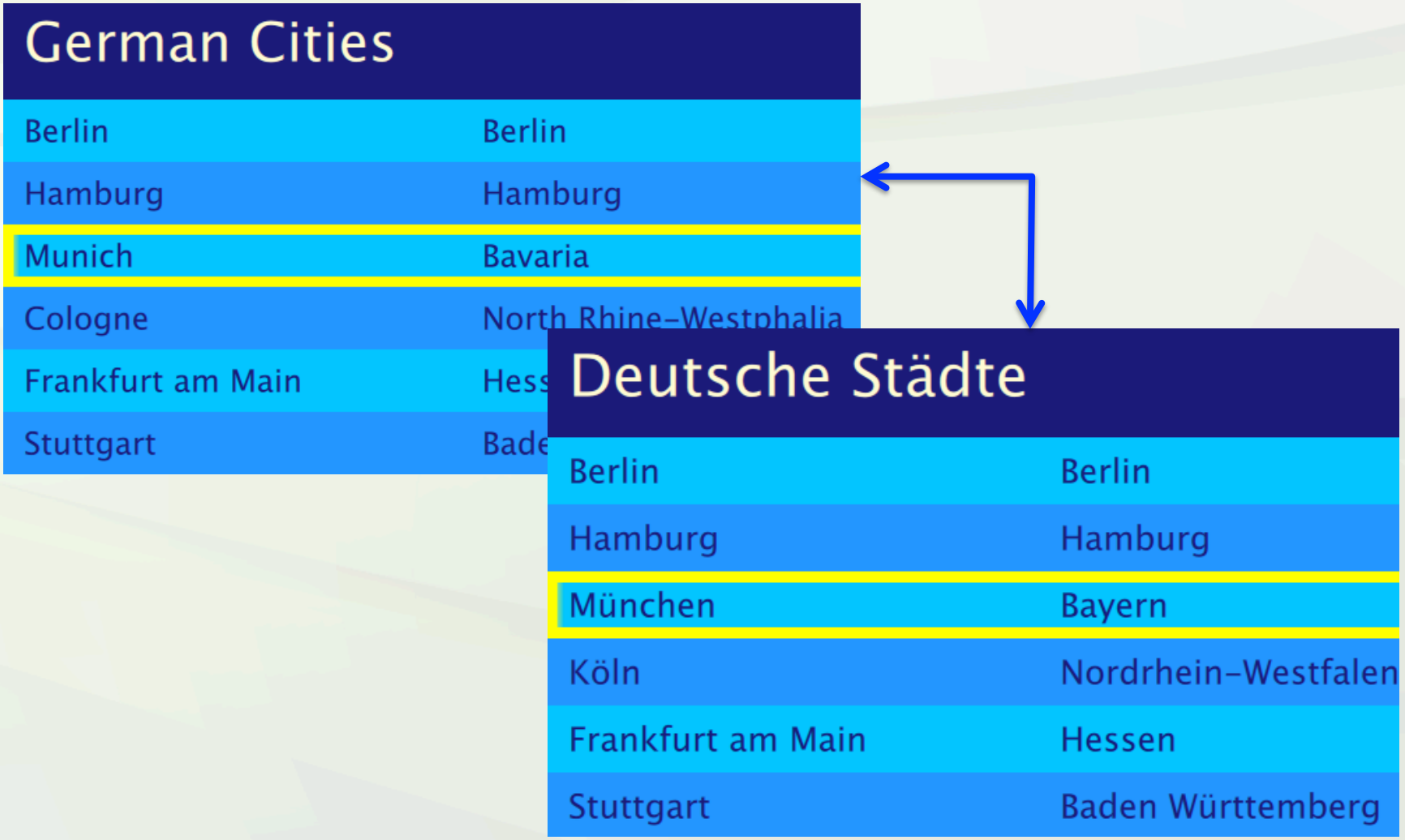
index provided by delegate in ListView

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



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:

```
connect(TranslationManager::instance(),
        SIGNAL(languageChanged(QString)),
        this,
        SLOT(retranslate(QString)));
```

Retranslating Qt/C++ Models (2)


```
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 title property has changed

QML calls title(), which returns tr(rawTitle())

Notify QML ListView that all its items have changed and need reloading

Delegate retranslation, as model is “view” on database

Retranslating Qt/C++ Models (3)


```
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

Reevaluating qsTr on Language Change (2)

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

Reevaluating qsTr on Language Change (3)

On instance of QQuickView:

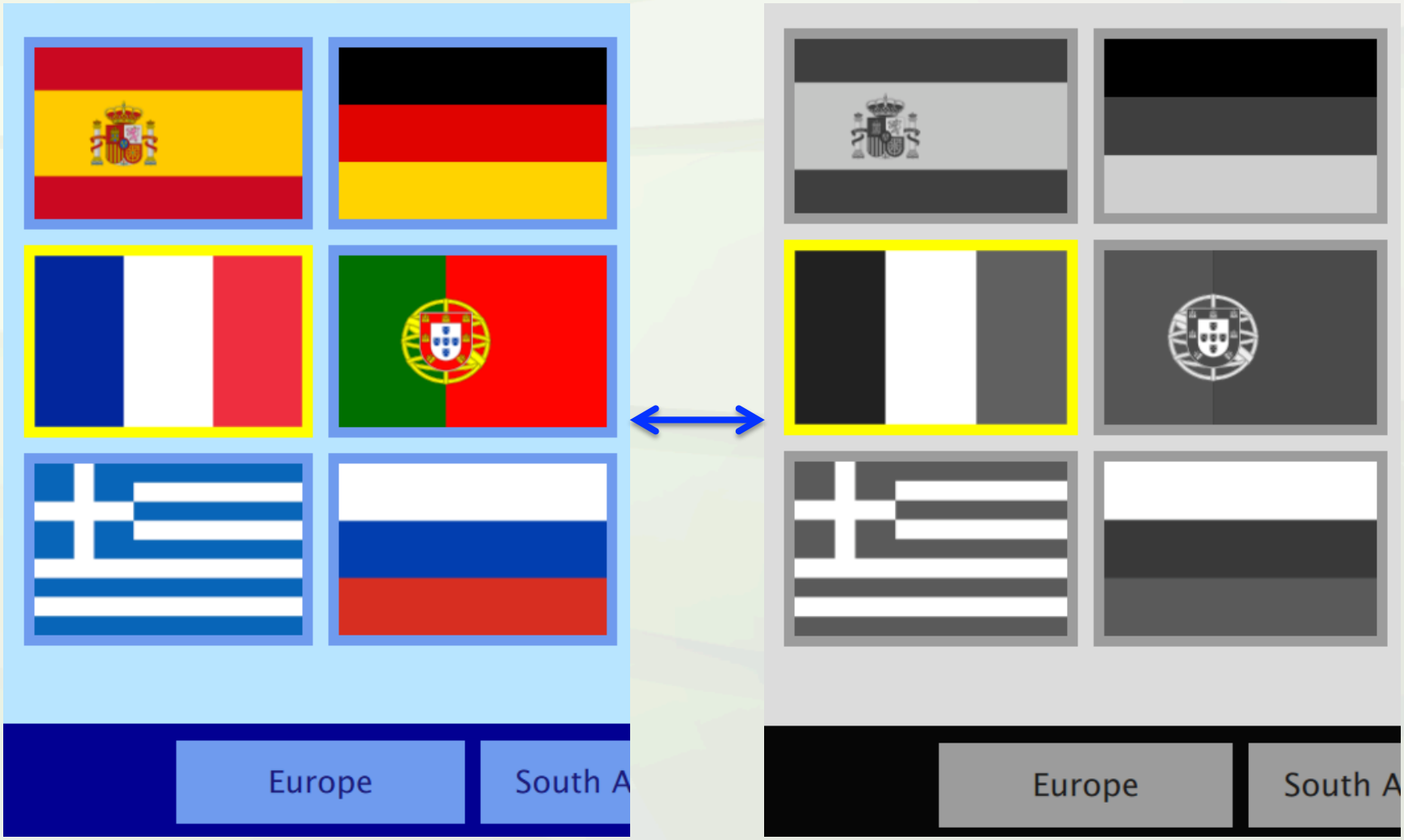
```
view->rootContext()->setContextProperty(
    “g_tr”, TranslationManager::instance());
```

Makes pointer to TranslationManager globally available in QML under name g_tr

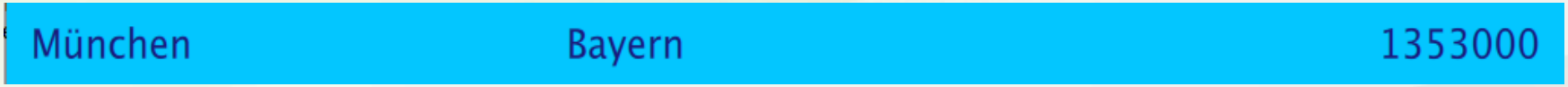
Contents

- Key Navigation
- Dynamic Language Change
- Themes

Dynamic Theme Change



Theming QML Code




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

  Row {
    Text {
      text: city.name
      color: “#191970”
```

Unthemed

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

  Row {
    Text {
      text: city.name
      color: g_theme.listViewItem.
          textColor
```

Themed

Implementing the Themes


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

München

```
QtObject {
    property QtObject listViewItem : QtObject {
        property color backgroundColor: “#A5A5A5”
        property color backgroundColorAlt: “#818181”
        property color textColor: “#1E1E1E”
    }
```

München

Changing Themes

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")
    }
}
```

Global variable accessible from everywhere in QML

Set theme on start-up

QQuickView forwards signal themeChanged(QString theme)

The End

Qt Developer Days 2013

Thank you!