Virtual Keyboards for Qt applications

Tobias König, Senior Software Engineer at KDAB

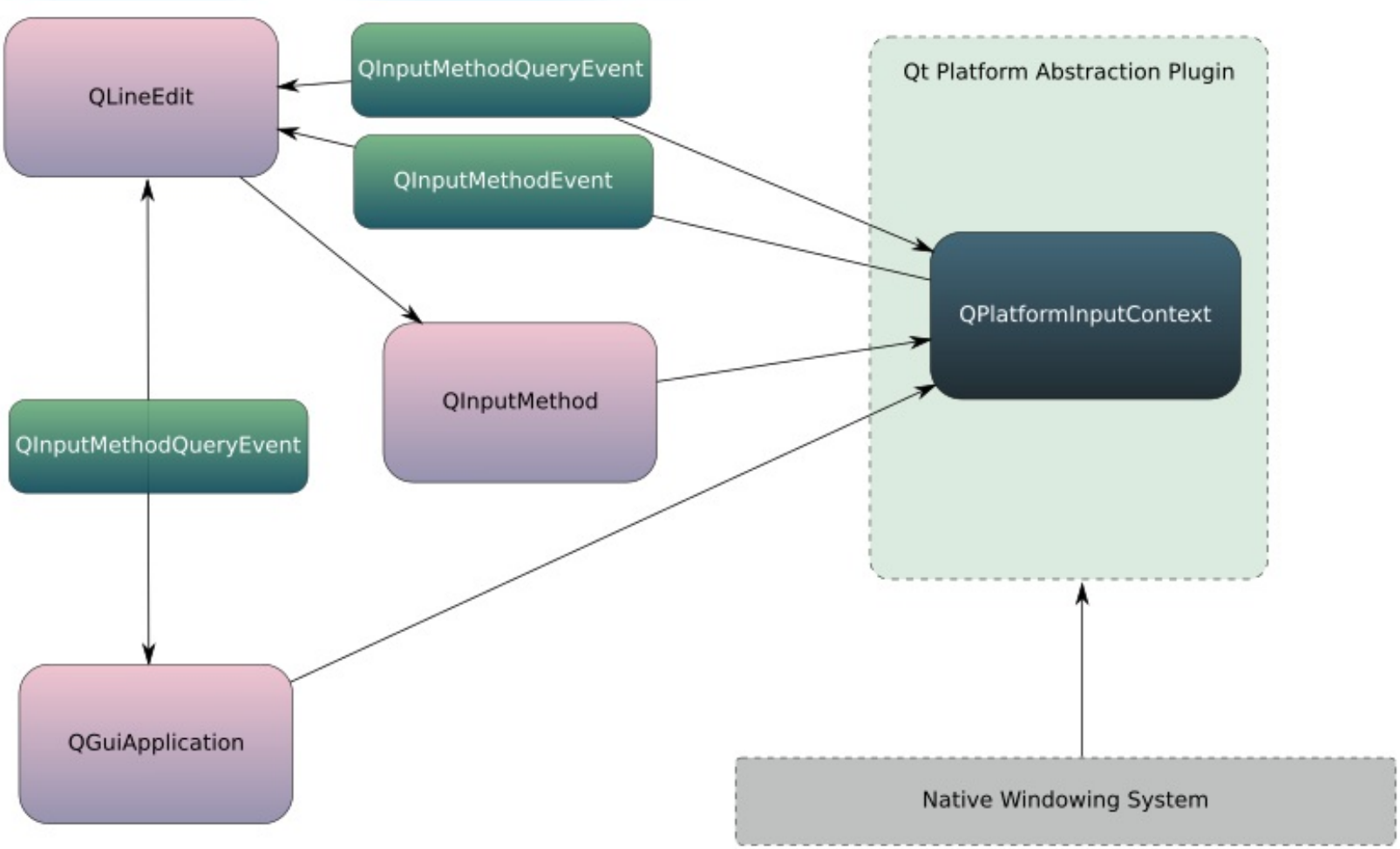
Qt WORLD SUMMIT 2015

- **Input Method API in Qt**
- Virtual Keyboards in Qt
- Use Qt input method API in Qt applications

Virtual Keyboards based on Input Method API

- **QInputMethod**
  - Access virtual keyboard from application
- **QPlatformInputContext**
  - Reimplemented by virtual keyboard
- **QInputMethodQueryEvent**
  - Send information from application to virtual keyboard
- **QInputMethodEvent** / **QKeyEvent**
  - Send input events from virtual keyboard to application

Input Methods in Qt - Overview



- Input Method API in Qt
- **Virtual Keyboards in Qt**
- Use Qt input method API in Qt applications

QPlatformInputContext

- Virtual keyboard side of API
- Part of Qt Platform Abstraction (QPA)
- Two kinds of QPlatformInputContext for virtual keyboards
  - Native provided by platform
  - Custom via QPlatformInputContextFactory

Native Virtual Keyboards

- Uses the virtual keyboard provided by the system
- Supported QPA platforms:
  - android
  - ios
  - wayland
  - windows

- QPlatformInputContextFactory
- Supported QPA platforms:
  - eglfs
  - xcb
  - wayland
  - windows

QPlatformInputContextFactory

- Plugin is defined via QT_IM_MODULE
- Input context creation harmonized in Qt 5.6
  - null: default platform context
  - empty: no context
  - set: set one, if it exists and is valid (otherwise no context)

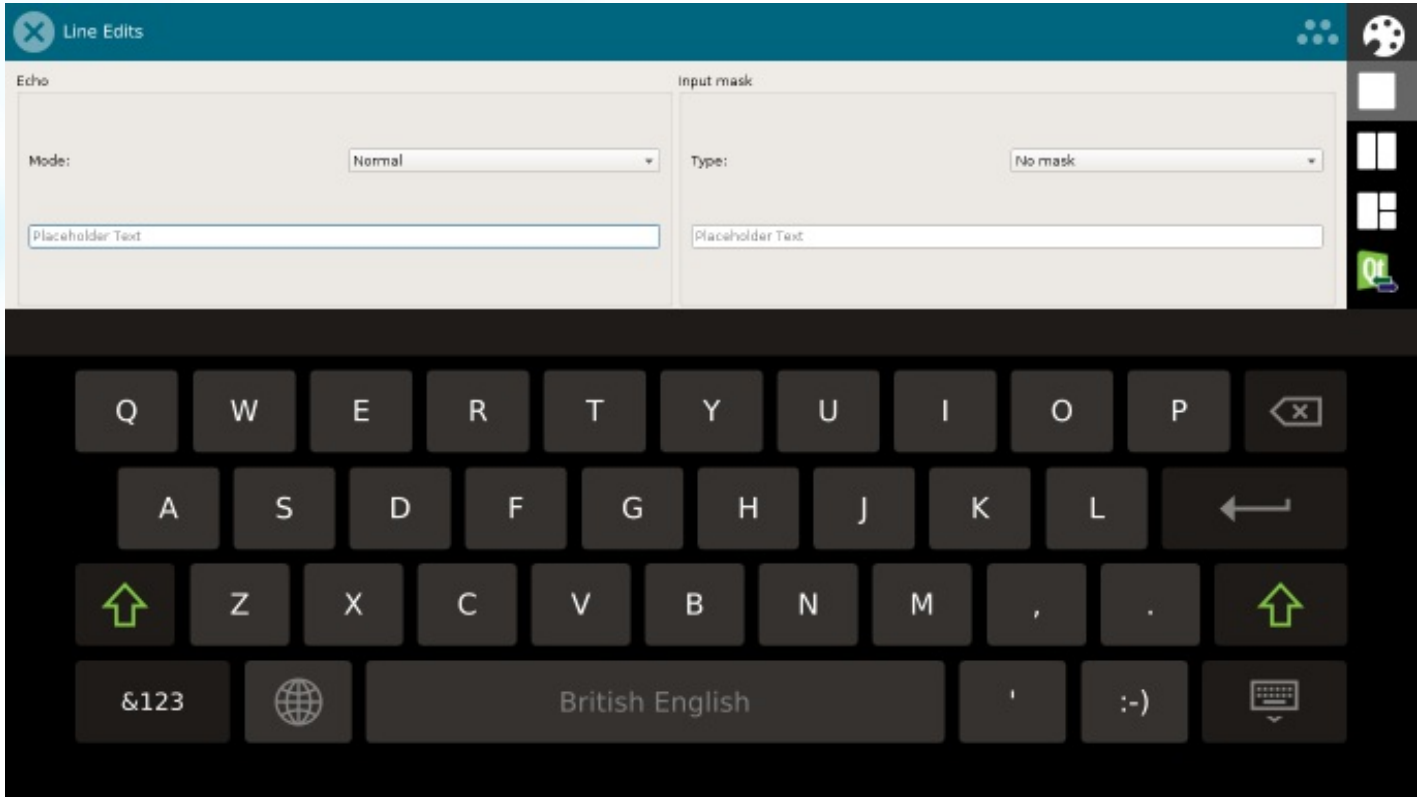
- The default platform context uses the keyboard provided by the compositor via the "text" protocol
- When using a QtWayland compositor and the patch from https://codereview.qt-project.org/#/c/113975/ the default context forwards the Qt Input Method API from the application to the compositor so that one can just use any QPlatformInputContext on compositor side

Embedding keyboard in application

- Some virtual keyboards (like Qt Virtual Keyboard) allow embedding in an application
- Especially useful for platforms without multiple window management like eglfs
- With previously mentioned patch it can be used in a QtWayland compositor to embed the Qt Virtual Keyboard in the compositor

```
1  import QtQuick.Enterprise.VirtualKeyboard 1.3
2
3  Item {
4      id: desktop
5
6      Keyboard {
7          id: keyboard
8
9          anchors.bottom: parent.bottom
10         anchors.left: parent.left
11         anchors.right: parent.right
12
13         visible: Qt.inputMethod.visible
14     }
15 }
```

Embedding keyboard in QtWayland compositor



Custom keyboard example

- Depending on the requirement it might make sense to create an own custom keyboard
- We will show a really simple keyboard with just a "Hello World" button for embedding in a QML application

First we need to provide a QPlatformInputContext

```
1  #include <QtGui/qpa/qplatforminputcontext.h>
2
3  class QtWSKeyboardInputContext : public QPlatformInputContext
4  {
5      Q_OBJECT
6
7  public:
8      QtWSKeyboardInputContext();
9
10     bool isValid() const override;
11
12     void showInputPanel() override;
13     void hideInputPanel() override;
14     bool isInputPanelVisible() const override;
15
16     void setFocusObject(QObject *object) override;
17
18     Q_INVOKABLE void send(const QString &text);
19
20 private:
21     bool m_inputPanelVisible;
22 };
```

Implementation of the isValid() and showInputPanel() methods:

```
1  bool QtWSKeyboardInputContext::isValid() const
2  {
3      return true;
4  }
5
6  void QtWSKeyboardInputContext::showInputPanel()
7  {
8      if (m_inputPanelVisible)
9          return;
10
11     m_inputPanelVisible = true;
12     emitInputPanelVisibleChanged();
13 }
```

For sending input text to the application we use the QInputMethodEvent

```
1  void QtWSKeyboardInputContext::send(const QString &text)
2  {
3      if (!QGuiApplication::focusObject())
4          return;
5
6      QInputMethodEvent event;
7      event.setCommitString(text);
8
9      QCoreApplication::sendEvent(QGuiApplication::focusObject(), &event);
10 }
```

Use a QPlatformInputContextPlugin to create the QPlatformInputContext

```
#include <QtGui/qpa/qplatforminputcontextplugin_p.h>

class QtWSKeyboard : public QPlatformInputContextPlugin
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID QPlatformInputContextFactoryInterface_iid FILE "qtwskeyboard.json")

public:
    QPlatformInputContext *create(const QString &key, const QStringList &paramList) override;
};

QPlatformInputContext *QtWSKeyboard::create(const QString &key, const QStringList &)
{
    if (QString::compare(key, QLatin1String("qtwskeyboard"), Qt::CaseInsensitive) != 0)
        return nullptr;

    qmlRegisterSingletonType<QtWSKeyboardInputContext>("com.kdab.KeyboardExample", 1, 0,
                                                       "InputContext", returnInputContext);

    context = new QtWSKeyboardInputContext;
    return context;
}
```

Custom keyboard example

Implement the keyboard itself in QML in our case just a button which sends "Hello World"

```
1  import QtQuick 2.0
2
3  import com.kdab.KeyboardExample 1.0
4
5  Rectangle {
6      color: "lightsteelblue"
7
8      height: keyLabel.implicitHeight + 20
9
10     Text {
11         id: keyLabel
12         anchors.fill: parent
13         text: "Hello World"
14     }
15
16     MouseArea {
17         anchors.fill: parent
18         onClicked: InputContext.send(keyLabel.text);
19     }
20 }
```

Embed the keyboard into an application

```
1  int main(int argc, char *argv[])
2  {
3      setenv("QT_IM_MODULE", "qtwskeyboard", true);
4
5      QGuiApplication app(argc, argv);
6
7      QQmlApplicationEngine engine;
8      engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
9
10     return app.exec();
11 }
```

Embed the keyboard into an application

```
1  import QtQuick 2.0
2  import QtQuick.Window 2.0
3
4  import com.kdab.KeyboardExample 1.0
5
6  Window {
7      width: 500
8      height: 500
9
10     visible: true
11
12     TextInput {
13         id: textinput
14         text: "Enter text"
15     }
16
17     Keyboard {
18         visible: Qt.inputMethod.visible
19     }
```

Custom keyboard example

Enter textHello WorldHello World

Hello World

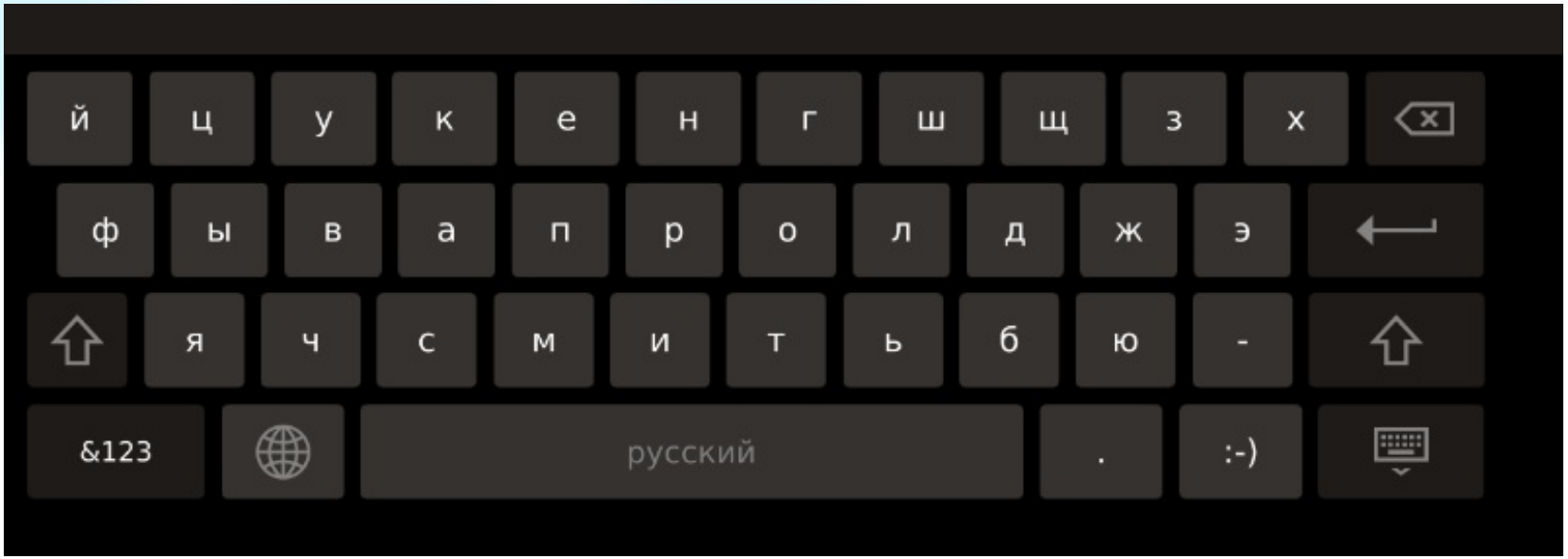
Available virtual keyboards for Qt

Use existing virtual keyboards for advanced usages like

- Multi language support
- Support for text correction
- Support for Chinese/Japanese/Korean

Qt Virtual Keyboard

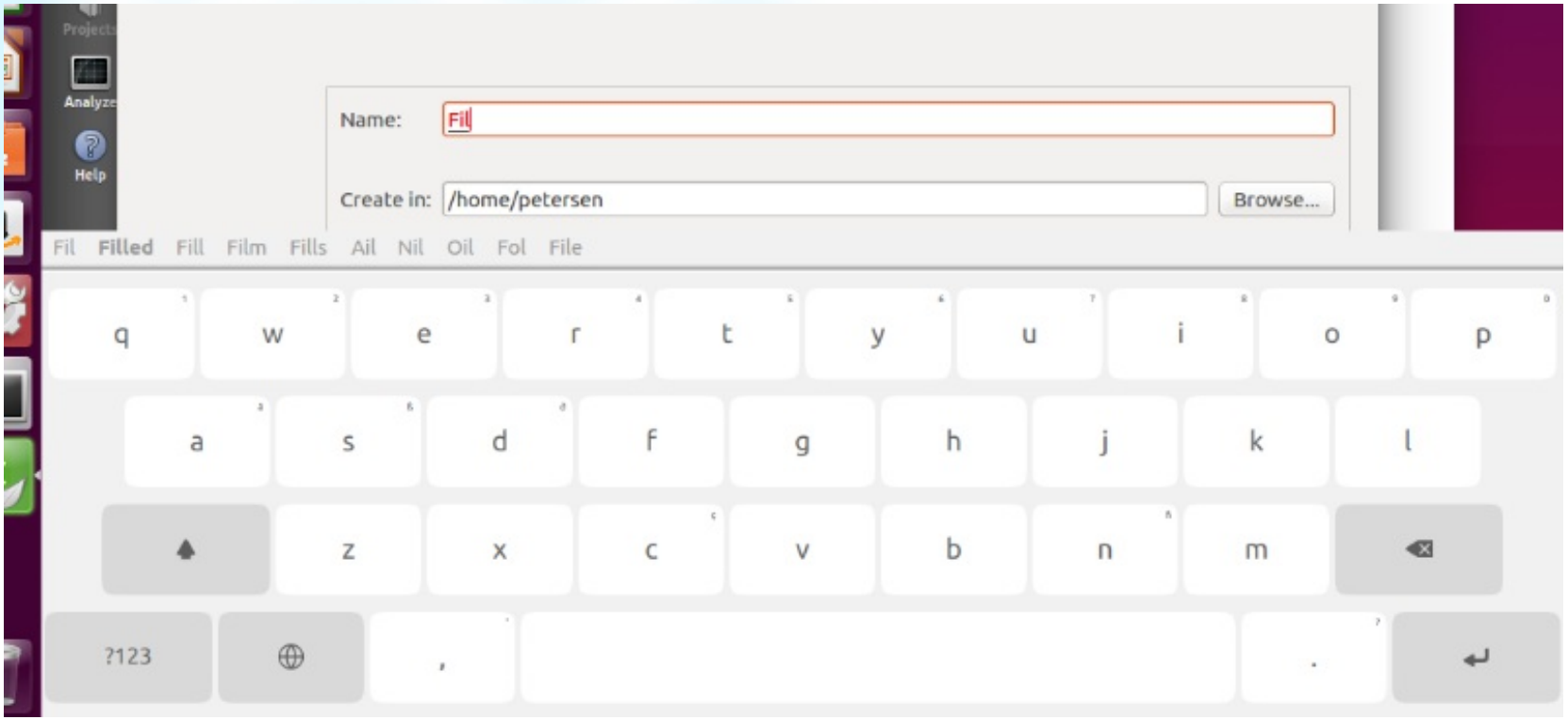
- Part of the Commercial Qt offering
- For xcb platform it displays automatically in a separate window
- For other QPA platforms it allows embedding in application
- Uses QML
- Supports multiple languages like: English, French, German, Russian, Arabic, ...
- Supports Chinese, Japanese and Korean
- Supports text correction
- QT_IM_MODULE=qtvirtualkeyboard


й ц у к е н г ш щ з х
ф ы в а п р о л д ж э
я ч с м и т ь б ю -
&123
русский
.
:-)

Ubuntu Keyboard

- Open Source: LGPL-3
- Displays keyboard in a separate window
- Uses QML with Ubuntu components
- Supports multiple languages like: English, French, German, Russian, Arabic, ...
- Supports Chinese
- Supports text correction and prediction
- QT_IM_MODULE=maliitphablet

Ubuntu Keyboard



- Input Method API in Qt
- Virtual Keyboards in Qt
- **Use Qt input method API in Qt applications**

Improve the user experience

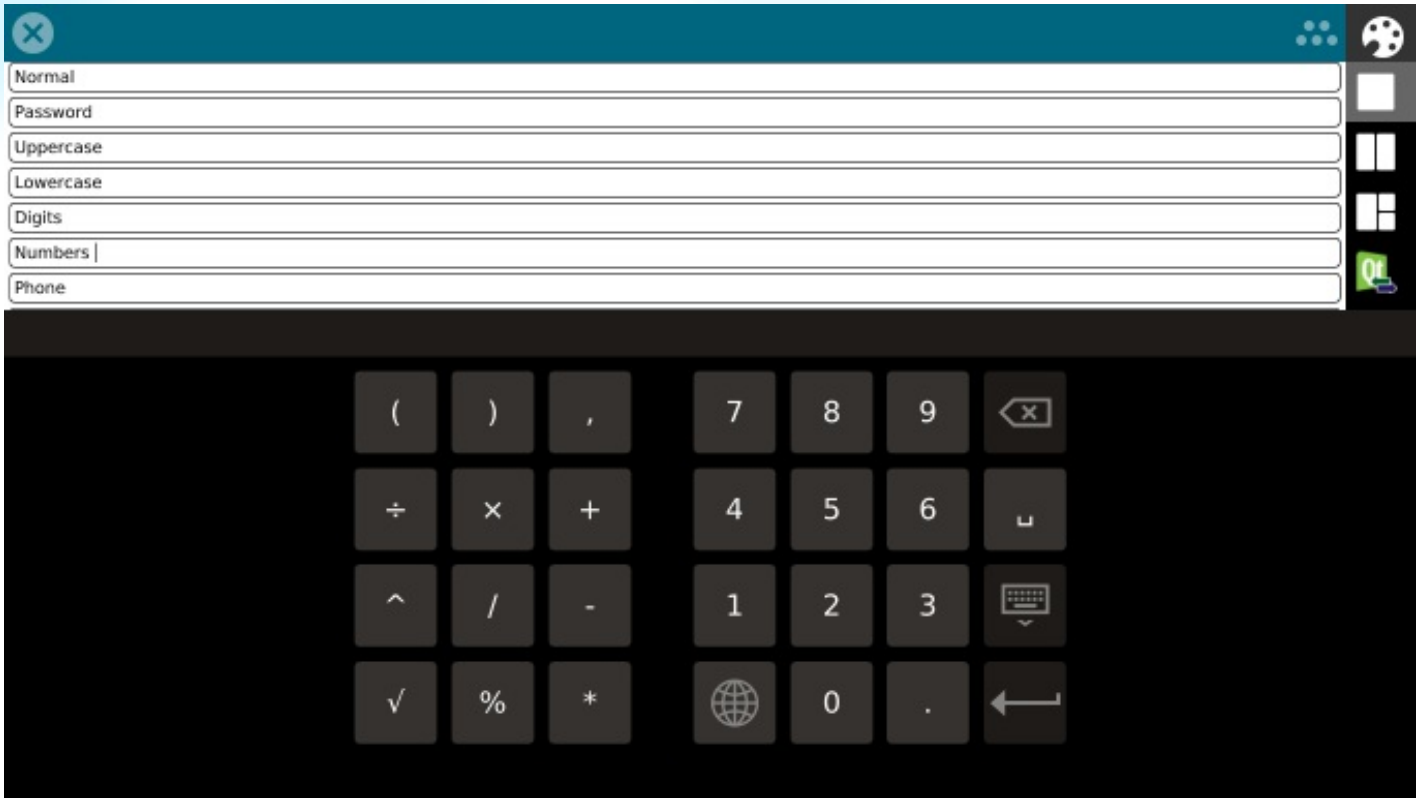
- Qt input fields have builtin support for the input method API but there are still ways for application developers to improve the user experience with virtual keyboards:
  - Define purpose of text input fields
  - Alter apperance of Return key
  - Change UI depending on keyboard

Define purpose of text input fields

- Keyboard can change layout depending on purpose of text field
- **Qt::ImHints** flags queried through **Qt::InputMethodQuery**
- Possible values for Qt::InputMethodHints:
  - **Qt::ImhNone** - No hints
  - **Qt::ImhHiddenText** - The input method should not show the characters while typing
  - **Qt::ImhDigitsOnly** - Only digits are allowed
  - **Qt::ImhFormattedNumbersOnly** - Only number input is allowed
  - **Qt::ImhDialableCharactersOnly** - Only characters suitable for phone dialing are allowed
  - **Qt::ImhEmailCharactersOnly** - Only characters suitable for email addresses are allowed
- Multiple hints can be combined. For example for password fields:
  - Qt.ImhNoAutoUppercase | Qt.ImhNoPredictiveText | Qt.ImhSensitiveData | Qt.ImhHiddenText

Define purpose of text input fields - example

```
1  TextInput {
2      id: input
3
4      inputMethodHints: Qt.ImhFormattedNumbersOnly
5  }
```



Alter apperance of Return key

- Can be used to display alternative key instead of Return
- Added in Qt 5.6
- **Qt::ImEnterKeyType** flags queried through **Qt::InputMethodQuery**
- Possible values for Qt::EnterKeyType:
  - **Qt::EnterKeyDone** - Show a "Done" button
  - **Qt::EnterKeySend** - Show a "Send" button
  - **Qt::EnterKeySearch** - Show a "Search" button
  - **Qt::EnterKeyReturn** - Show a Return button that inserts a new line
  - **Qt::EnterKeyNext** - Show a "Next" button which should be used to navigate to next input field
- Not all of these values are supported on all platforms. For unsupported values the default key will be used instead.

Alter apperance of Return key - Example

```
1  TextInput {
2      id: input
3
4      EnterKey.type: Qt.EnterKeySearch // Show a "Search" button
5  }
```

Change UI depending on keyboard

- When a virtual keyboard is shown it might overlap some parts of the application
- In particular not so nice to overlap the focused input field
- **QInputMethod::visible** property can be used to figure out if a virtual keyboard is displayed
- **QInputMethod::keyboardRectangle** property holds the virtual keyboard's geometry in window coordinates

Thank you!

www.kdab.com

tobias.koenig@kdab.com