Author: dailey.dai@openthinks.com

C++ and QML Interaction

QML is designed to be easily extensible through C++ code. The classes in the Qt QML module enable QML objects to be loaded and manipulated from C++, and the nature of QML engine's integration with Qt's meta object system enables C++ functionality to be invoked directly from QML. This allows the development of hybrid applications which are implemented with a mixture of QML, JavaScript and C++ code.

QML use C++ class & object

Define exported C++ type in QML

```
//backend.h
#ifndef BACKEND_H
#define BACKEND_H
#include <QObject>
#include <QString>
class BackEnd : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString userName READ userName WRITE setUserName NOTIFY userNameChanged)
public:
    explicit BackEnd(QObject *parent = nullptr);
    QString userName();
    Q_INVOKABLE void setUserName(const QString &userName);
signals:
    void userNameChanged(const QString &userName);

public slots:
    void logChange(const QString &userName);

private:
    QString m_userName;
};
#endif // BACKEND_H
```

- defined type need extend from `QObject` or its child
- use `Q_OBJECT` macro
- use `Q_PROPERTY` macro define property which could be accessed in QML
- use `Q_INVOKABLE` macro define method which could be invoked in QML
- `signal` & `slot` could be directly used in QML

```
//backend.cpp
#include "backend.h"
#include <QDebug>
BackEnd::BackEnd(QObject *parent) : QObject(parent){}
```

```
QString BackEnd::userName()
{
    return m_userName;
}
void BackEnd::setUserName(const QString &userName)
{
    if (userName == m_userName)
        return;
    m_userName = userName;
    emit userNameChanged(m_userName);
}
void BackEnd::logChange(const QString &userName){
    qDebug()<<"Changed Name:"<<userName;
}
```

Register C++ type and used in QML

Register C++ type into QML

- `qmlRegisterType` registers the C++ type in the QML system with the given name
- `qmlRegisterInterface` registers an existing Qt interface type
- `qmlRegisterUncreatableType` registers a named C++ type that is not instantiable but should be identifiable as a type to the QML type system
- `qmlRegisterSingletonType` registers a singleton type that can be imported from QML

```
//main.cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include "backend.h"

int main(int argc, char *argv[])
{
#if defined(Q_OS_WIN)
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif
    QGuiApplication app(argc, argv);
    //qmlRegisterType(const char *uri, int versionMajor, int versionMinor, const char
*qmlName)
    qmlRegisterType<BackEnd>("io.qt.examples.backend",1,0,"BackEnd");
    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;
    return app.exec();
}
```

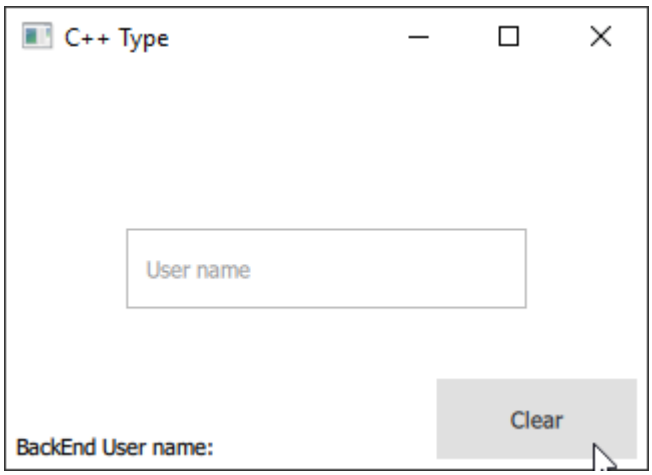
Import registered type

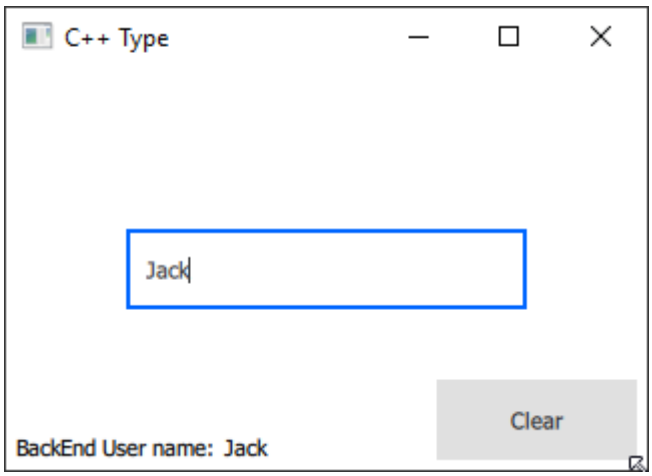
```
//main.qml
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.0
import io.qt.examples.backend 1.0
Window {
    visible: true
    width: 320
    height: 480
    title: qsTr("Hello World")
}
```

Instantiate registered type

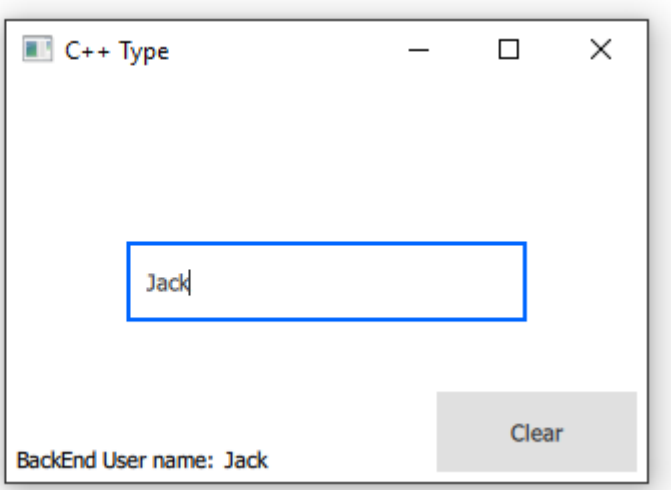
```
//main.qml
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.0
import io.qt.examples.backend 1.0
Window {
    visible: true
    width: 320
    height: 200
    title: qsTr("C++ Type")
    BackEnd{
        id:backend;
        Component.onCompleted: {//call backend signal and slot
            backend.userNameChanged.connect(backend.logChange);
        }
    }
    TextField{// call backend property
        text: backend.userName
        placeholderText: qsTr("User name")
        anchors.centerIn: parent
        onTextChanged: backend.userName = text;
    }
    Text{
        id:labelPrefixx;
        text:qsTr("BackEnd User name:");
        anchors.left: parent.left;anchors.leftMargin: 5;
        anchors.bottom: parent.bottom;anchors.bottomMargin: 5;
    }
    Text{
        id:labelBackend;
        anchors.left: labelPrefixx.right;anchors.leftMargin: 5;
        anchors.bottom: labelPrefixx.bottom;
    }
    Button{
        text:qsTr("Clear");
        anchors.right: parent.right;anchors.rightMargin: 5;
        anchors.bottom: labelPrefixx.bottom;
        onClicked: {// call backend invokeable method
```

```
            backend.setUserName(null);
        }
    }
    Connections{// connect backend signal
        target: backend;
        onUserNameChanged:{
            labelBackend.text=userName;
        }
    }
}
```

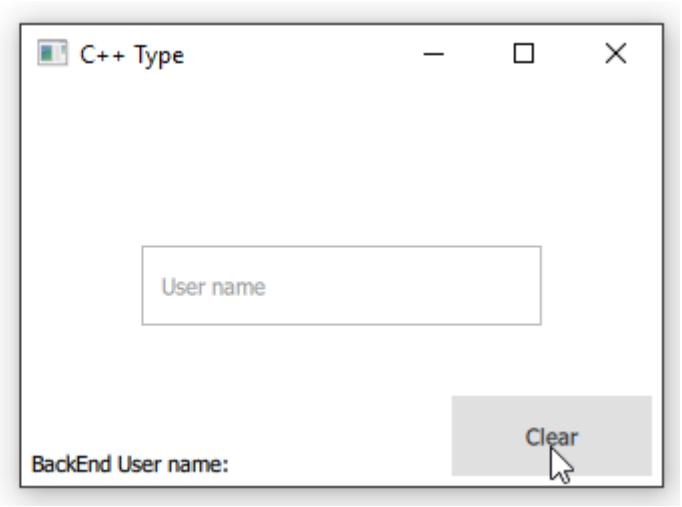





```
Changed Name: "J"
Changed Name: "Ja"
Changed Name: "Jac"
Changed Name: "Jack"
```



```
Changed Name: "J"
Changed Name: "Ja"
Changed Name: "Jac"
Changed Name: "Jack"
Changed Name: ""
```



Register C++ instance as property in QML

Register property in QML

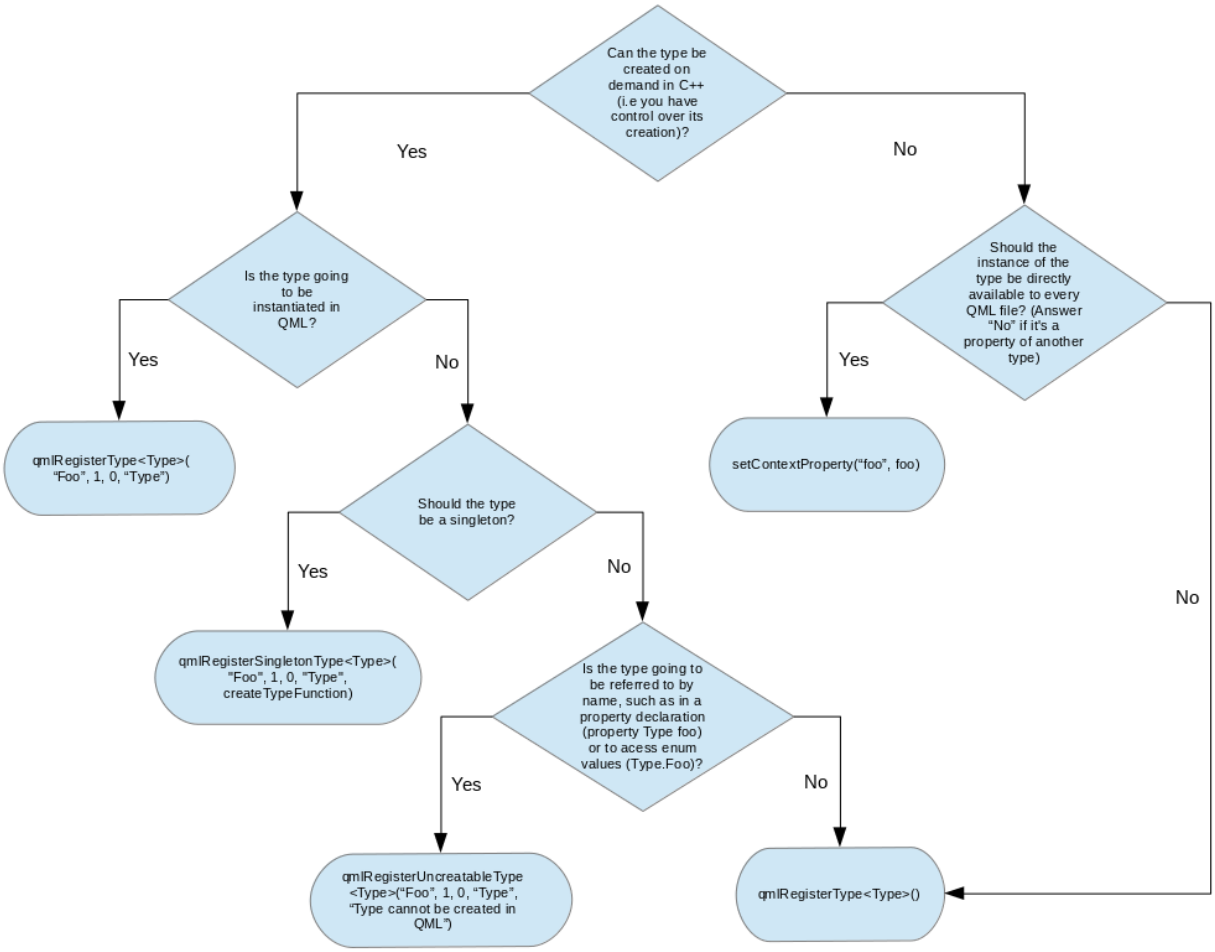
```cpp
//main.cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QtQml>
#include "backend.h"

int main(int argc, char *argv[])
{
#if defined(Q_OS_WIN)
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif
    QGuiApplication app(argc, argv);
    //qmlRegisterType<BackEnd>("io.qt.examples.backend",1,0,"BackEnd");
    QQmlApplicationEngine engine;
    engine.rootContext()->setContextProperty("backend",new BackEnd);
    engine.load(QUrl(QStringLiteral("qrc:/main2.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;
    return app.exec();
}
```

Use property name in QML

```
//main.qml
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.0
Window {
    visible: true
    width: 320
    height: 200
    title: qsTr("C++ Type")
    Component.onCompleted: {
```

```
        backend.userNameChanged.connect(backend.logChange);
    }
    // same as previous code
    // ...
    //
}
```

Choosing the Correct Integration Method Between C++ and QML



C++ use QML object

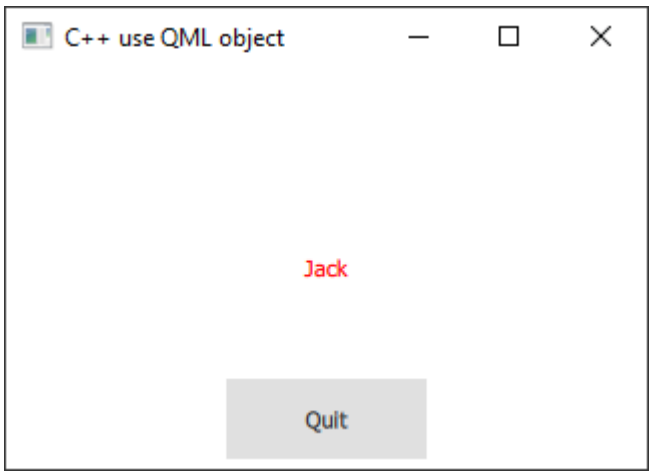
- `QObject::findChild()` find child item by its object name(not item id)
- `QMetaObject::invokeMethod()` invoke item method

```
//main.qml
import QtQuick 2.9
import QtQuick.Window 2.2
import QtQuick.Controls 2.0
Window {
    objectName:"rootObj";
    visible: true
    width: 320
    height: 200
    title: qsTr("C++ use QML object")
```

```
    Text{
        objectName: "userName";
        text:qsTr("Marry");
        anchors.centerIn: parent;
    }
    Button {
        objectName: "exitButton";
        text: qsTr("Quit")
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        anchors.bottomMargin: 5
    }
}
```

```
//main.cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QtQml>
#include <QMetaObject>
#include <QVariant>
#include <QDebug>
#include <QColor>

int main(int argc, char *argv[])
{
#if defined(Q_OS_WIN)
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    engine.load(QUrl(QStringLiteral("qrc:/main3.qml")));
    if (engine.rootObjects().isEmpty())
        return -1;
    QObject *root = NULL;
    //find root object in QML
    QList<QObject*> rootObjects = engine.rootObjects();
    for(int i=0,j=rootObjects.size();i<j;i++){
        if(rootObjects.at(i)->objectName() == "rootObj"){
            root = rootObjects.at(i);
            break;
        }
    }
    // find button and bind click signal to app slot quit
    QObject* exitButton = root->findChild<QObject*>("exitButton");
    if(exitButton){
        QObject::connect(exitButton,SIGNAL(clicked()),&app,SLOT(quit()));
    }
    // find text
    QObject* textLabel = root->findChild<QObject*>("userName");
    if(textLabel){
        // failed to invoke, no method or slot for QQuickText
```

```
        bool bRet = QMetaObject::invokeMethod(textLabel,"setText",Q_ARG(QString,"Jack"));
        qDebug()<<"call setText return - "<<bRet;

        textLabel->setProperty("text","Jack");
        textLabel->setProperty("color",QColor::fromRgb(255,0,0));
    }
    return app.exec();
}
```




```
output:
QMetaObject::invokeMethod: No such method QQuickText::setText(QString)
call setText return -  false
```