C++ Unchained: Extending the QML API of ArcGIS Runtime for Qt

Mark Cederholm
UniSource Energy Services
Flagstaff, Arizona

Part 1:
I don’t say “Qute”!

[but I might say “Q-awesome”]

What is Qt?

- Allows cross-platform development using a single C++ code base
- Compiles to native machine code for target platform
- Meta-object system
- Signals and slots
- Open source (commercial options available)

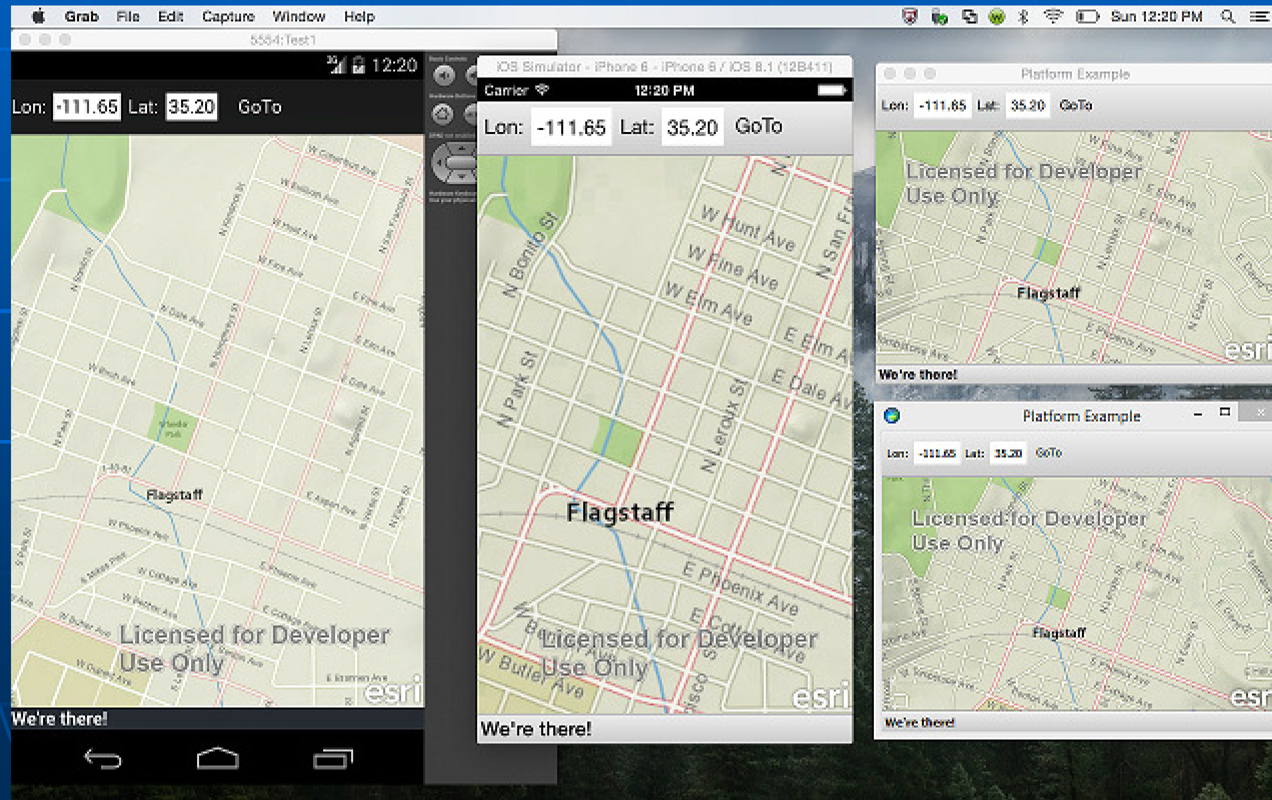
What is Qt Quick?

- QML: JSON-like interface markup language
- Allows embedded JavaScript code
- JavaScript engine runs at near C++ speed
- Can interact with C++ code-behind
- Allows separation of interface and application logic

ArcGIS Runtime for Qt 10.2.5

- QML API supports Android, iOS, Linux, OS X, and Windows
- Published as a QML plugin
- Map applications can be written entirely in QML/JavaScript
- C++ extends its capabilities

DEMO: A Simple Map Application



12:02:49 PM

Part 2: Exploring a QObject using QMetaObject

Getting the Meta-Object

- QMetaObject stores all the meta-information for a QObject subclass

```
QObject *pqObject = qobject_cast<QObject *>(someObject);
const QMetaObject *pqClass = pqObject->metaObject();
```

Enumerations

```
int iEnumCount = pqClass->enumeratorCount();
for (int i = 0; i < iEnumCount; i++)
{
        QMetaEnum qmEnum = pqClass->enumerator(i);
        QString qsEnumName = qmEnum.name();
        int iKeyCount = qmEnum.keyCount();
        for (int j = 0; j < iKeyCount; j++)
        {
                const char *name = qmEnum.key(j);
                int value = qmEnum.value(j);
        }
}
```

```
int iOffset = pqClass->enumeratorOffset();
```

Properties

```
int iPropCount = pqClass->propertyCount();
for (int i = 0; i < iPropCount; i++)
{
        QMetaProperty qProp = pqClass->property(i);
        const char *name = qProp.name();
        const char *typeName = qProp.typeName();
}
```

Methods

```
int iMethodCount = pqClass->methodCount();
for (int i = 0; i < iMethodCount; i++)
{
        QMetaMethod qMethod = pqClass->method(i);
        QByteArray qSignature = (qMethod.methodSignature());
        // Type can be Method, Signal, Slot, or Constructor
        QMetaMethod::MethodType mType = qMethod.methodType();
        const char *typeName = qMethod.typeName();
        int iParamCount = qMethod.parameterCount();
        for (int j = 0; j < iParamCount; j++)
        {
                QByteArray qParamName = qMethod.parameterNames()[j];
                QByteArray qParamType = qMethod.parameterTypes()[j];
        }
}
```

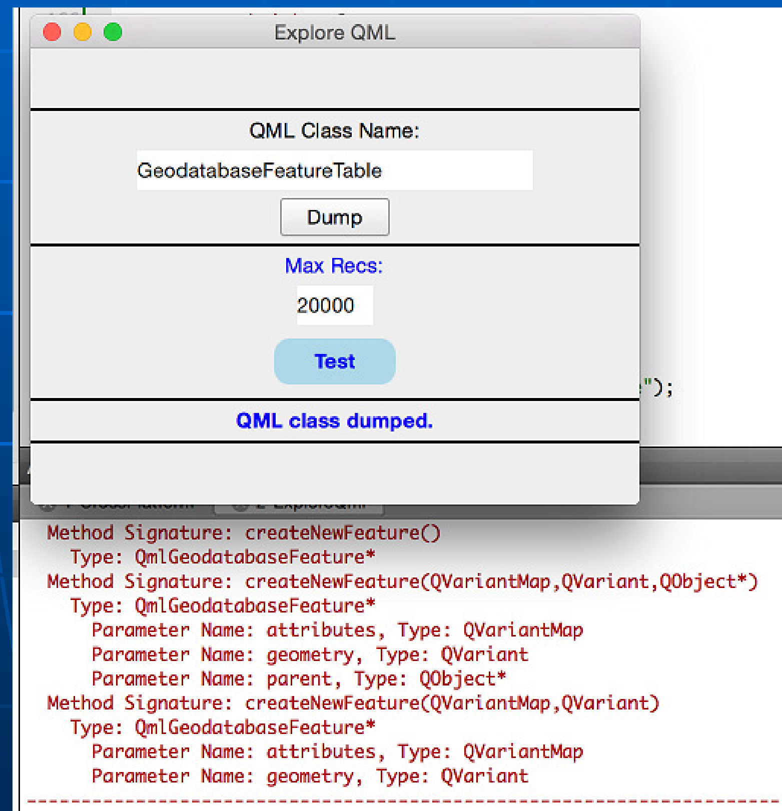
```
int iOffset = pqClass->methodOffset();
```

Types and QVariant

- QMetaType manages named types in the meta-object system
- Used as a helper to marshal types in QVariant and in queued signals and slots connections

```
int iReturnType = QMetaType::type(typeName);
QVariant qvRet(iReturnType, NULL);
```

DEMO: “Dumping” a QML Object



12:02:49 PM

Part 3: Interacting with a QML Object

Creating a QML Object in C++

```
QQmlEngine qEngine;
QQmlComponent component(&qEngine);
QString qsQML = "import ArcGIS.Runtime 10.25\nQuery { }";
component.setData(qsQML.toLocal8Bit(), QUrl());
QObject *pqQuery = component.create();
```

Creating a Runtime Object

```
// QObject *pqRuntime = handle to ArcGISRuntime singleton
bool bResult;
QObject *pqGdb;
bResult = QMetaObject::invokeMethod(
      pqRuntime,
      "createObject",
      Q_RETURN_ARG(QObject *, pqGdb),
      Q_ARG(QString, "Geodatabase"));
```

Getting and Setting Properties

- Property values are marshalled through QVariant
- setProperty: Common types are automatically cast

```
// QString qsParcelGDB = path to offline geodatabase
bool bResult pqGdb->setProperty("path", qsParcelGDB);
bool bValid = pqGdb->property("valid").toBool();
QObject * pqResult = _pqTab->
      property("queryFeaturesResult").value<QObject *>();
```

Invoking a Method: 1

```
bResult = QMetaObject::invokeMethod(pqRuntime,
        "createObject", Q_RETURN_ARG(QObject *, pqGdb),
        Q_ARG(QString, "Geodatabase"));
. . .
const char *typeName = "QmlGeodatabaseFeatureTable*";
int iReturnType = QMetaType::type(typeName);
QVariant qvRet(iReturnType, NULL);
void *dataRet = qvRet.data();
bResult = QMetaObject::invokeMethod(pqGdb,
        "geodatabaseFeatureTable",
        QGenericReturnArgument(typeName, dataRet),
        Q_ARG(QString, "Parcel"));
_pqTab = qvRet.value<QObject *>();
```

Invoking a Method: 2

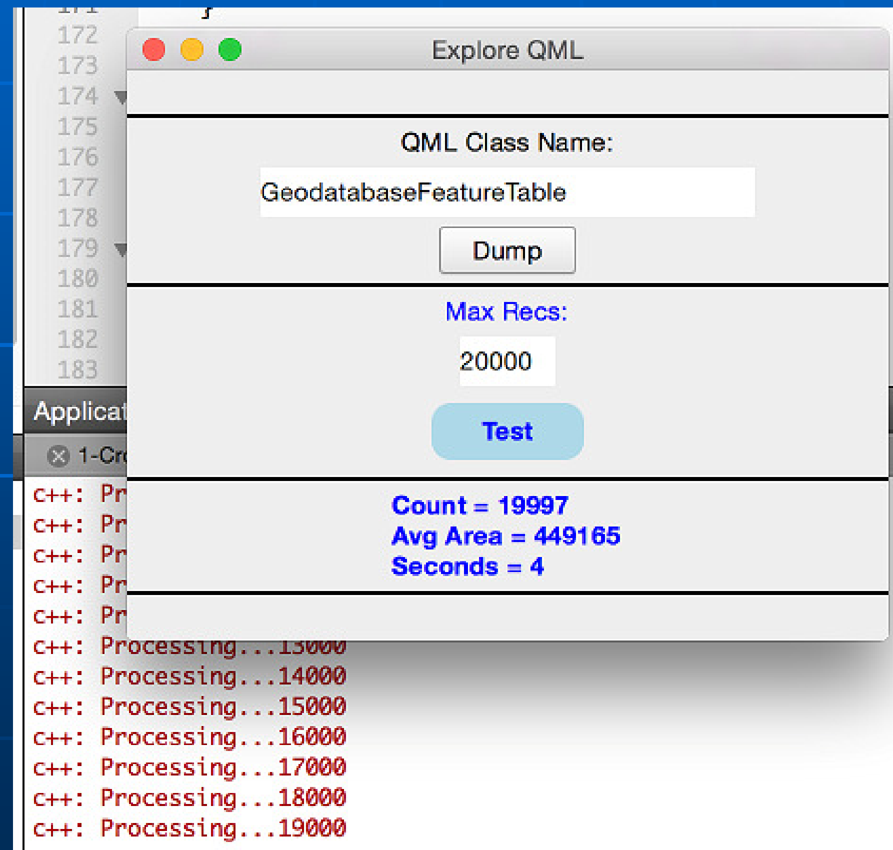
```
const QMetaObject *pqClass = pqGdb->metaObject();
QMetaMethod qMethod = pqClass->method(
      pqClass->indexOfMethod(
      "geodatabaseFeatureTable(QString)"));
const char *name = qMethod.typeName();
int iType = qMethod.returnType();
QVariant qvRet(iType, NULL);
void *dataRet = qvRet.data();
bResult = qMethod.invoke(pqGdb,
      QGenericReturnArgument(name, data),
      Q_ARG(QString, "Parcel"));
_pqTab = qvRet.value<QObject *>();
```

This is the most reliable approach

Connecting Signals and Slots

```
bResult = (bool)QObject::connect(
        _pqTab,
        SIGNAL(queryFeaturesStatusChanged()),
        this,
        SLOT(queryParcelStatusChanged()));
. . .
// QObject *pqQuery
QVariant qvArg = QVariant::fromValue(pqQuery);
bResult = QMetaObject::invokeMethod(_pqTab,
        "queryFeatures", Q_ARG(QVariant, qvArg));
```

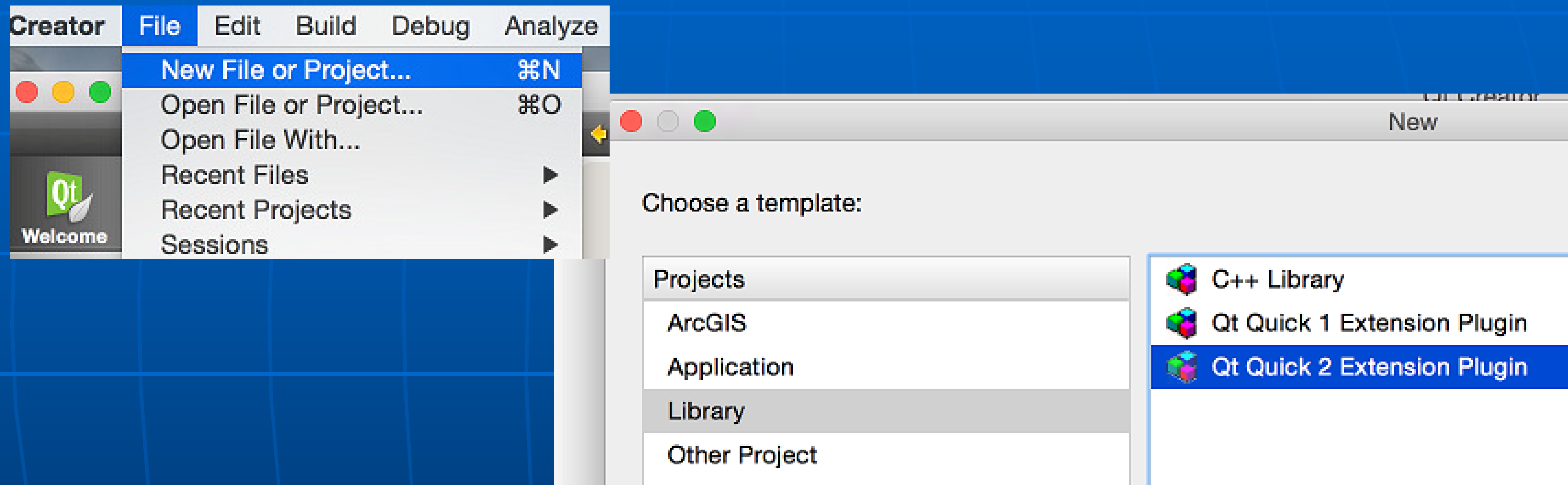
DEMO: Querying a Geodatabase



12:02:49 PM

Part 4:
Creating a QML Plugin

Creating a QML Plugin Project in Qt Creator



See "Runtime Qt Demo.txt" in sample code for details

12:02:49 PM

Declaring a QML Object

```
class QmlStreetQuery : public QObject
{
        Q_OBJECT
public:
        explicit QmlStreetQuery(QObject *parent = 0);
signals:
        void message(QString message, bool error);
        void streetData(QVariant data);
public slots:
        void findStreet(QObject *map, QString layerName,
               QString streetName);
private slots:
        void queryStreetStatusChanged();
private:
        QObject *_pqTab;
};
```

Declaring Properties

```
class StreetData : public QObject
{
      Q_OBJECT
      Q_PROPERTY(QString name READ name NOTIFY nameChanged)
      Q_PROPERTY(QObject* geometry READ geometry
            NOTIFY geometryChanged)
public:
. . .
      QString name() const;
      QObject* geometry() const;
signals:
      void nameChanged();
      void geometryChanged();
. . .
};
```

Registering a QML Object

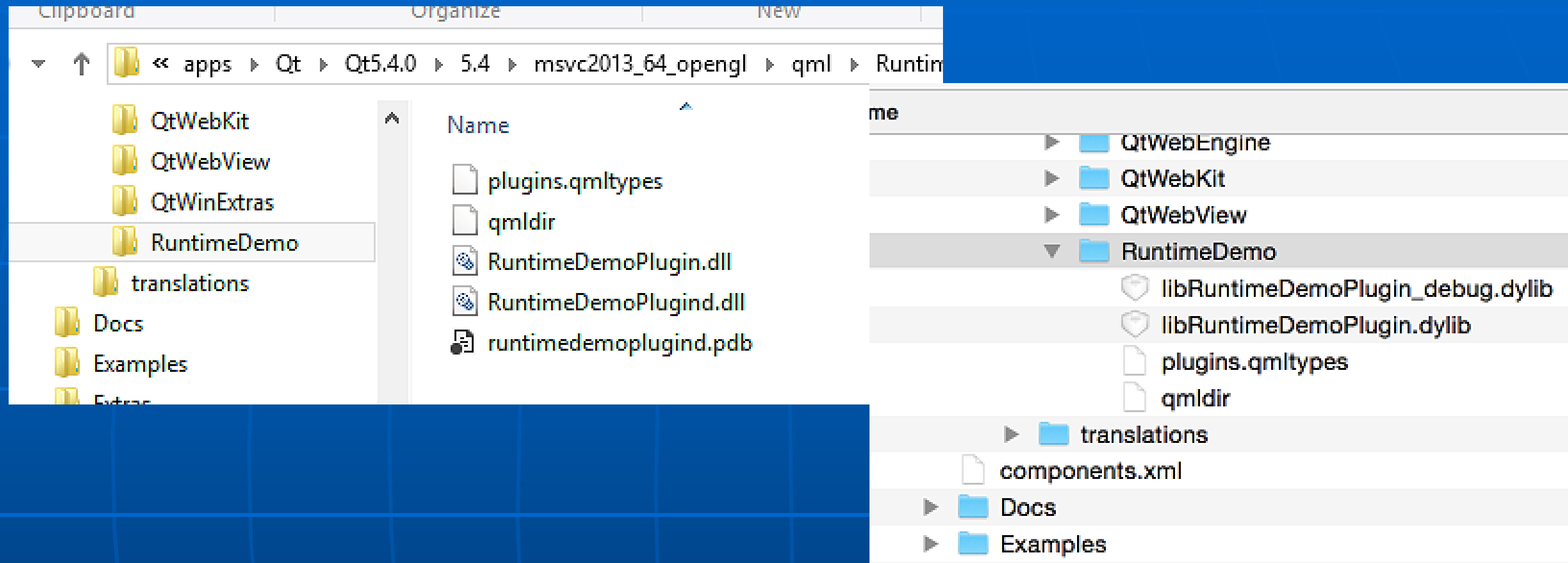
```
#include "runtimedemoplugin_plugin.h"
#include "qmlstreetquery.h"
#include <qqml.h>

void RuntimeDemoPlugin::registerTypes(const char *uri)
{
        // @uri com.devsummit.runtimedemo
        qmlRegisterType<QmlStreetQuery>(uri, 1, 0,
                "StreetQuery");
}
```

The qmldir File

```
module RuntimeDemo
plugin RuntimeDemoPlugin
classname RuntimeDemoPlugin
typeinfo plugins.qmltypes
```

```
qmlplugindump -noinstantiate -nonrelocatable
      RuntimeDemo 1.0 > plugins.qmltypes
```

The Plugin Folder



- Including the debug version allows debugging in Qt Creator

Using the QML Object

```
import RuntimeDemo 1.0

ApplicationWindow
{
. . .
	StreetQuery
	{
		id: streetQuery
		Component.onCompleted:
		{
			message.connect(alertMessage);
			streetData.connect(receiveData);
		}
	}
. . .
}
```

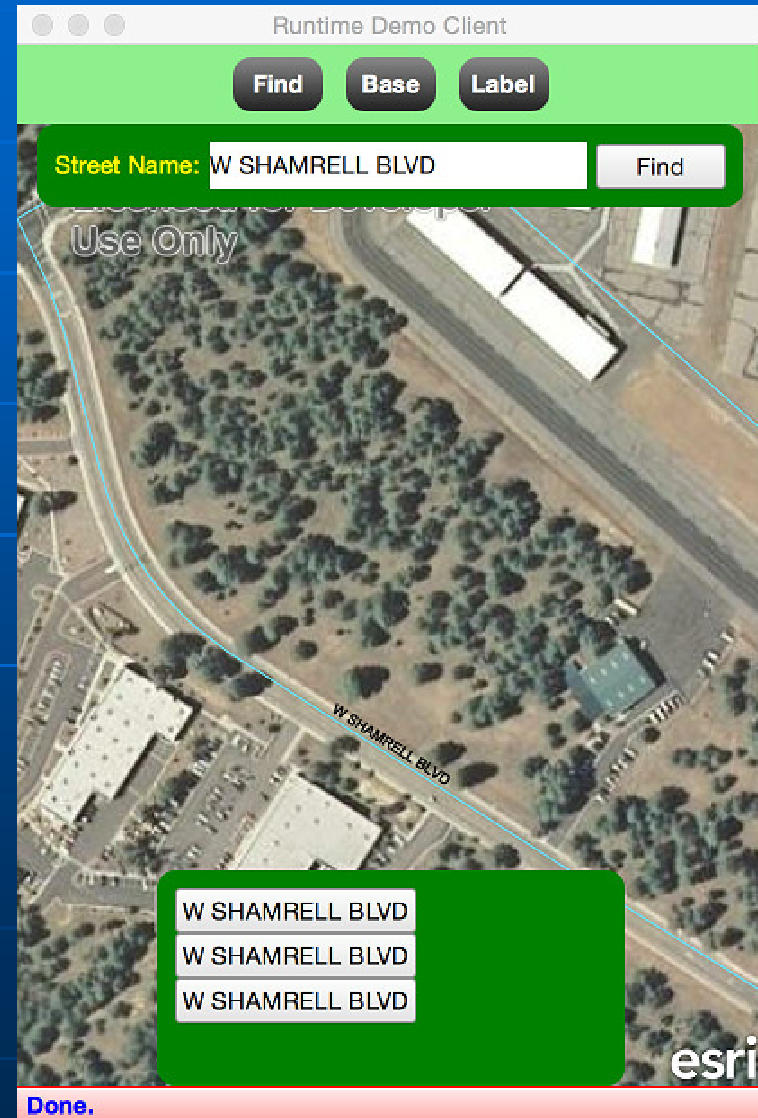
Using the QML Object (cont'd)

```
ApplicationWindow
{
. . .
      Button
      {
            text: "Find"
            onClicked: streetQuery.findStreet(myMap,
                  "Street Centerline",
                  txtStreetName.text);
      }
. . .
}
```

Using the QML Object (cont'd)

```
ApplicationWindow
{
. . .
        function alertMessage(message, error)
        {
                lblStatus.text = message;
                lblStatus.color = error ? "red" : "blue";
        }
        function receiveData(listData)
        {
                lvResults.model = listData;
                alertMessage("Done.", false);
                rectResults.visible = true;
        }
. . .
}
```

DEMO: QML Plugin



12:02:49 PM

Bonus: Loading External Layouts

(see sample code)

Questions?

- Mark Cederholm
  mcederholm@uesaz.com
- This presentation and sample code may be downloaded at:

http://www.pierssen.com/arcgis10/runtime.htm