Programmers rejoice:
QML makes business people understand
About me

My company

What I do at work

Where I live
What is it all about?

Agenda
- Motivation
- A real life example
- The hurdles of DSL creation
- What Qt has to offer
- PresentationSystem demo
- Extending Qml‘s type system
- When to use Qml for DSL creation and when not
A common problem in software development

You mean
If(obj != NULL) {
 name="Qt"
}?

#?=&f\ (%)
What if…

- Stakeholders and developers could share a common language?
- Changes to the software could be made in that language, too?
A real life example
The heating system example. Before.

- Implementation
- Build & Deploy
- Rollout to test site
- Gather data
- Adjust requirements

Up to 4 weeks, only in winter
The heating system example. After.

- Adjust software via DSL
- Cont. Integration
- Rollout to test site
- Gather data
- < 1 week
- Adjust requirements
Why...

...isn‘t there a DSL in every software project?
DSLs are hard to develop

Formal definition
- Grammar
- Schema
- Metamodel
- Abstract Syntax

Tooling
- Parser
- Validator
- Compiler
- Editor / IDE
DSL creation with XText

XText is a powerful, DSL-based language development framework

Steps to create a DSL using Xtext:

- Textual definition of syntax and semantic model for your DSL. Language: **Grammar**
- Configure generator that creates java packages and editor for your DSL. Language: **MWE2**
- Set up **Maven** builds for CI of your DSL
- Update Eclipse
- Use your DSL
- Not targeting Java? → Use **Xtend** for code generation
Programming languages compared

QML, WPF, Silverlight
Object oriented, declarative, special purpose, Model driven

C++, Java, C#
Object oriented, imperative, general purpose

C, COBOL
Procedure oriented, imperative

Assembler Code
Processor instruction set, no abstraction at all

Comes with an extensible type system!
Custom types can be made available in Qml

Types used are backed by C++ classes

CodeCompletion for custom types immediately available in Creator

Easy to learn

Effort needed to create a (simple) DSL is low

No other programming languages needed!
Extending the Qml type system

- Any custom types can be registered with QML‘s type system (but must inherit from QObject)
- Different forms of registration are available to define the runtime behaviour of your types:
  - Creatable types
  - Uncreatable types
  - Interfaces
  - Singletons
The QmlPresentationSystem

- is a DSL for creating slide decks
- makes use of QML’s extensible type system
- is easy to use (compared to programming a slide deck with C++ or any other general purpose language)
How to use your own custom types in Qml

1. Derive from QObject

Public class YourType : public QObject {...}

2. Define Properties

Q_PROPERT(Y(DataElementIds::EnDataElementIds identifier READ getListIdentifier MEMBER m_id)

3. Register type in Qml

qmlRegisterType<YourType>("com.your.namespace", 1, 0, „QmlTypeName");
How to use your own custom types in Qml

5. Run qmake

4. Import custom namespace in Qml file

```qml
import com.your.namespace 1.0
```

6. Ready to use custom type in Qml

```qml
YourType{
    displayName: „YourInstanceName"
    value: false
    identifier: YourId
}
```
Using singleton types

QObject and QJSValue types can be registered as singleton types.

Registering types that are defined in C++:

```c
int qmlRegisterSingletonType(const char * uri, int versionMajor, int versionMinor,
const char * typeName, QJSValue(*) ( QQmlEngine *, QJSEngine * ) callback)
```

```c
int qmlRegisterSingletonType(const char * uri, int versionMajor, int versionMinor,
const char * typeName, QObject(*) ( QQmlEngine *, QJSEngine * ) callback)
```
Using singleton types

Registering types that are defined in Qml:

```c
int qmlRegisterSingletonType(const QUrl & url, const char * uri, int versionMajor, int versionMinor, const char * qmlName)
```
Using uncreatable types

```c
int qmlRegisterUncreatableType(const char * uri, int versionMajor, int versionMinor,
const char * qmlName, const QString & message)
```

**Note:**
To use enums in Qml, they must be wrapped in a class.
Drawbacks you have to deal with

- Mingling with QtQuick types and QObject properties
- No integrated code generator for creation of non-Qt-code
- Editor is not always as smart as it could be
- Fixed syntax
See how it works
The sample application

HardwareLayer

GPIOs

DataSwitch

GpioMappings

DataLayer

DataElements
When is a Qml based DSL the right choice?
What is a domain?

Stakeholders’ domains

- Heating system
- Drive control
- Accounting
- Routing
- …

Your domains

- System composition
- StateMachines
- UIs
- Data layer
- …

Simply said: Everything is a domain
Think about a QML based DSL, when

- other tools would require too much effort
- simple DSL features are needed
- no code generation is required
- the DSL will mainly be used to define static aspects
Think about using other tools

- when you want your own syntax / semantics
- code generation for different languages is required
- when you want to have a clean DSL (without artifacts from QObject)
Q&A / Discussion