Forum Nokia

Qt Communication

Andreas Jakl

Senior Technical Consultant
Forum Nokia

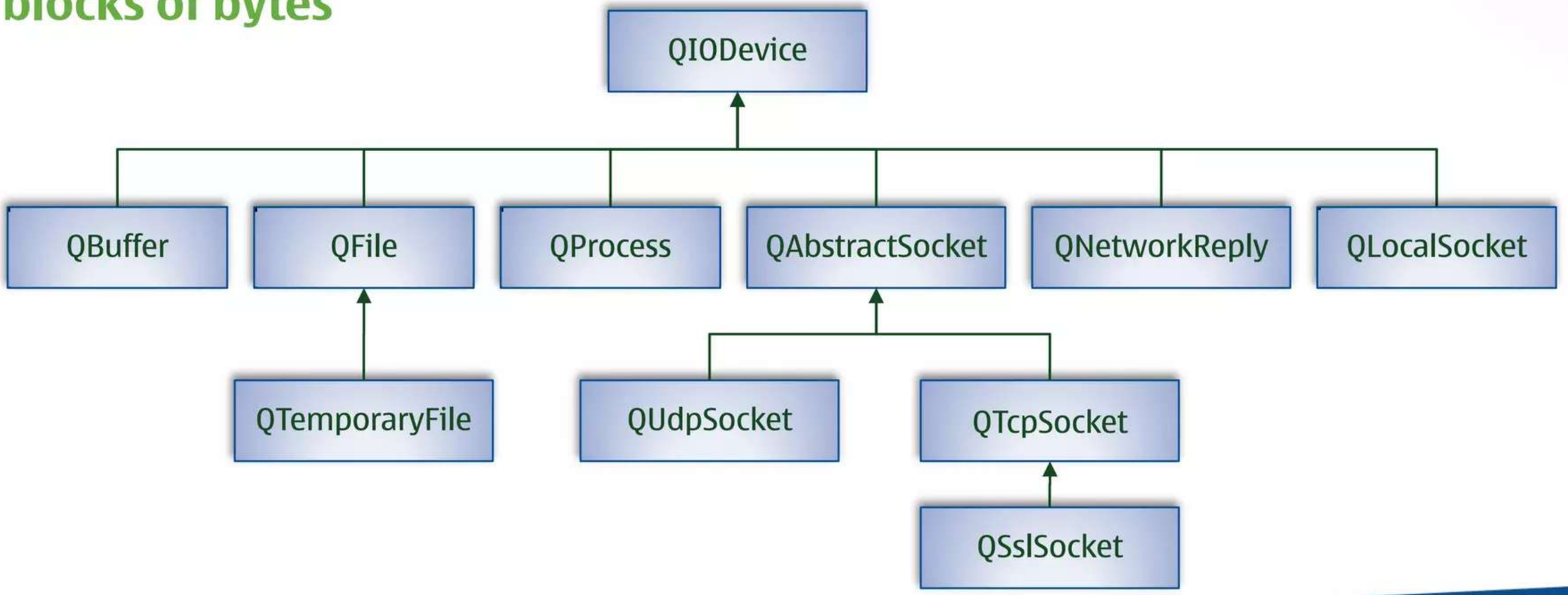
20 September, 2010
v3.0.0

Contents

- Devices and streams
- Files
- Sockets and HTTP
- XML
- Internationalization

Device Abstraction

- QIODevice is base class for all “devices” capable of reading and writing blocks of bytes



Devices

- **Sequential devices**
  - Data can be accessed only once
  - Start from first byte, progress serially to last byte
  - QProcess, Q*Socket
- **Random access devices**
  - Bytes can be read any number of times from any position
  - QFile, QTemponaryFile, QBuffer

Streams

- **Higher-level stream classes**
  - Take care of byte ordering, text encoding
  - **QDataStream**
    - Binary data
    - 100% independent of host system
    - Implements serialization of basic data types (C++ and Qt)
    - Overload << and >> operators to add support for custom data types
  - **QTextStream**
    - Read/write words, lines, numbers
    - Supports formatting
    - Operates on QIODevice, QByteArray or QString

Example: Console Application

- **No need for UI**
- **Necessary changes for the project file (.pro):**

```
TEMPLATE = app
QT = core
CONFIG += console
CONFIG -= app_bundle
SOURCES += main.cpp
```

  - Only `QtCore` module is required
  - Configuration `console` enables console output on Windows
  - Removing `app_bundle`: don't create bundle in Mac OS X

Opening Files

- **No event loop required**
  - → no `QCoreApplication`
- **Error Handling**
  - Through standard error stream (`std:cerr`) → defaults to console
  - `<iostream>` has overload for `std::string` → convert `QString` with `.toStdString()`

main.cpp

```
#include <QtCore>
#include <QDebug>
#include <iostream>
#include <stdio.h>

int main(int argc, char* argv[])
{
  QFile outFile("data.dat");
  if (!outFile.open(QIODevice::WriteOnly)) {
    std::cerr << "Cannot open file for writing: " <<
       outFile.errorString().toStdString() << std::endl;
    return 1;
  }
```

Writing Data

- **Data representation**
  - **Specify data format version** to maintain forward and backward compatibility
  - **Platform independent** (default: big endian encoding)

main.cpp

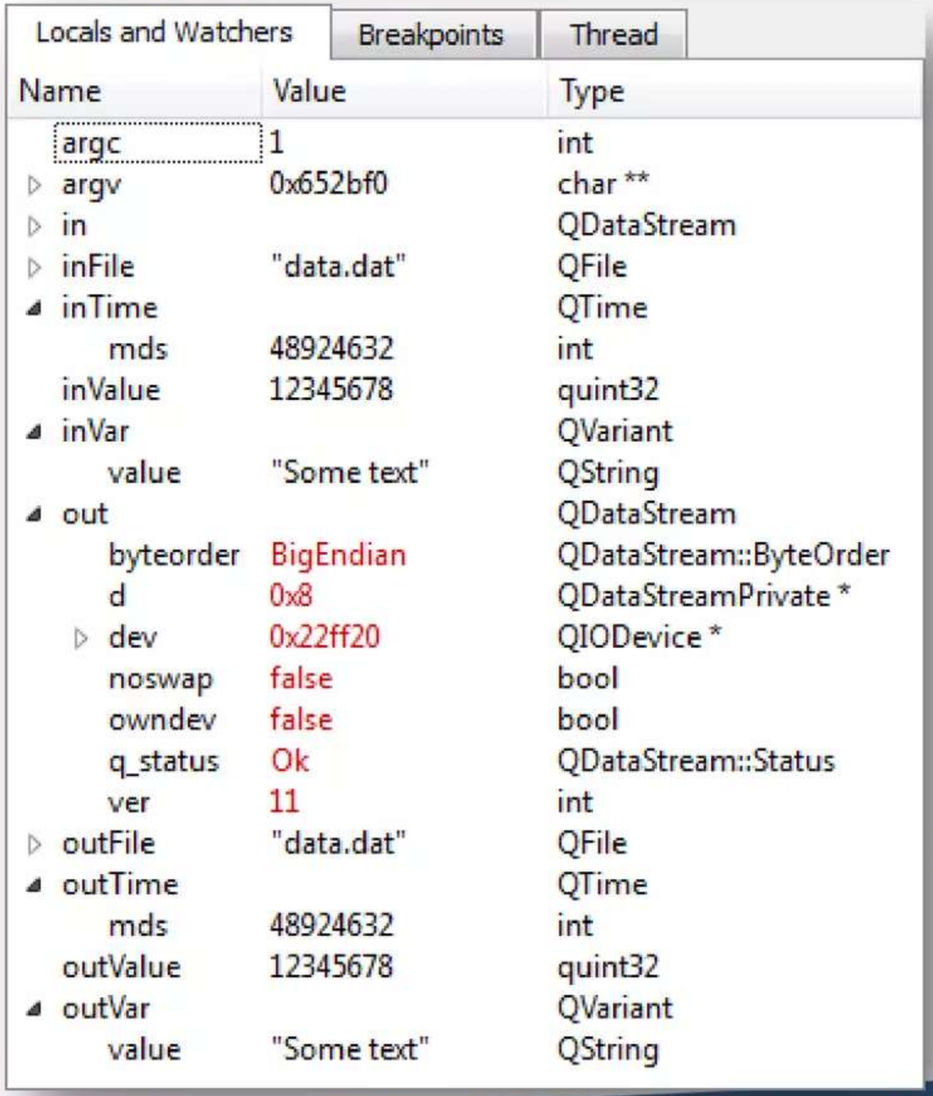
```
QDataStream out(&outFile);
 // Serialization format of data stream changes with
 // new version of Qt to accommodate new
 // functionality.
 out.setVersion(QDataStream::Qt_4_5);

 // Use specific Qt datatypes to make sure integers
 // are the same on all platforms!
 quint32 outValue = 12345678;
 QTime outTime = QTime::currentTime();
 QVariant outVar("Some text");
 out << outValue << outTime << outVar;

 outFile.close();
```

Recommended: extend the example with brief file header.
1. Magic String: check if file is really from your app.
2. Version number: correctly import from old versions of your settings file.

Data Representation



Input stream

Input variables

Output stream

Output variables

Reading Files

- **Reading similar to writing**
  - Use same order and data stream version
  - Streams don't save variable type
- **QVariant recognizes type**
  - **Here:** output using qDebug() instead of standard streams
  - **Advantage:** direct serialization of Qt data types

main.cpp

```
QFile inFile("data.dat");
 if (!inFile.open(QIODevice::ReadOnly)) {
   std::cerr << "Cannot open file for reading: " <<
     inFile.errorString().toStdString() << std::endl;
   return 1;
 }

 QDataStream in(&inFile);
 in.setVersion(QDataStream::Qt_4_5);

 quint32 inValue;
 QTime inTime;
 QVariant inVar;

 // Read values in same order as they were written
 in >> inValue >> inTime >> inVar;

 qDebug() << "Variant type:" << inVar.typeName() <<
             ", contents:" << inVar.toString();

 inFile.close();
 return 0;
}
```

```
Variant type: QString , contents: "Some text"
```

Communication

Image Credit: NASA

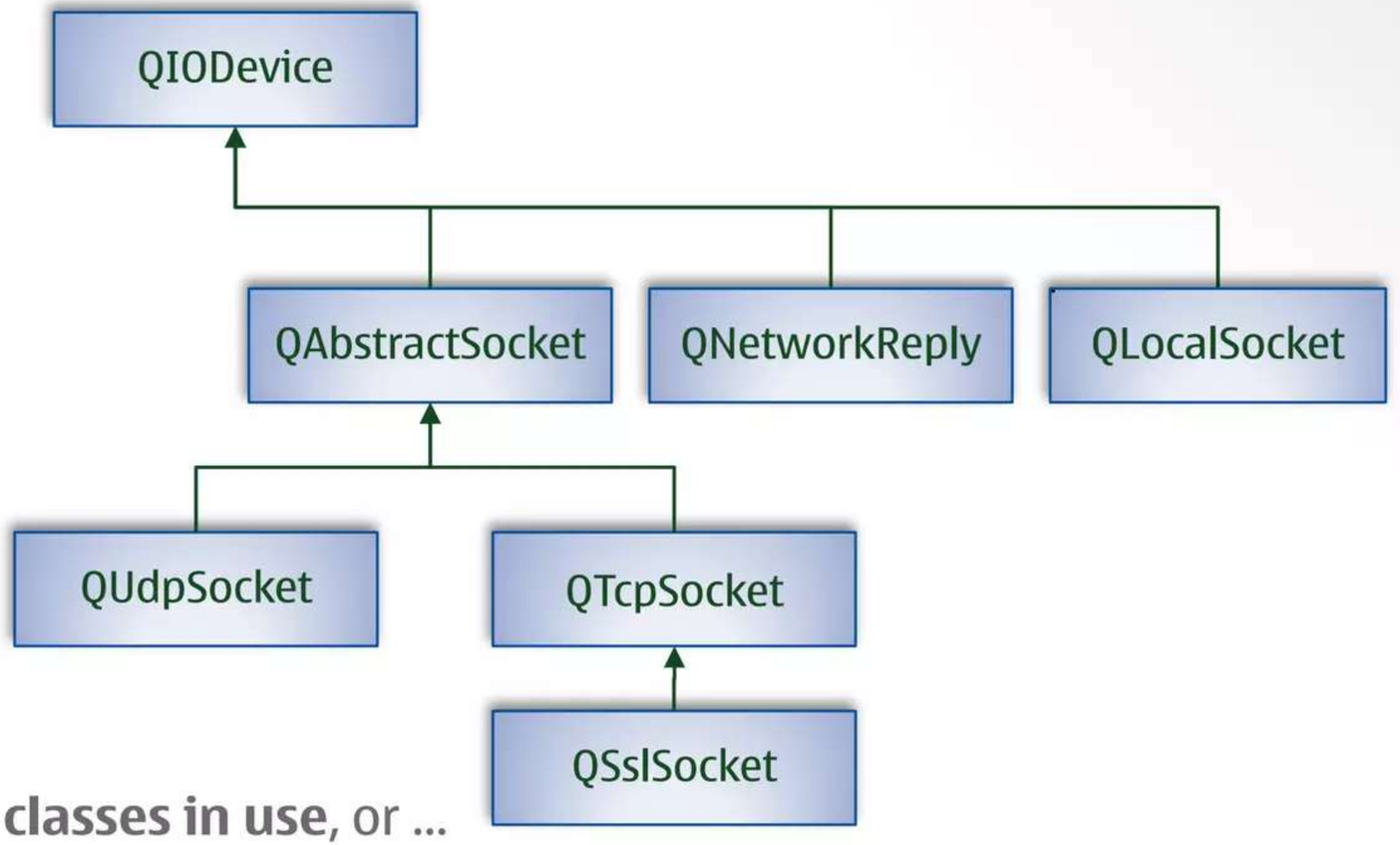
Qt Modules

- **Networking classes in *QtNetwork* extension module**
- **Using the module**
  - Insert to **project file (.pro):**

```
QT += network
```

  - **Source code:**
    - Include appropriate headers of **classes in use**, or ...
    - Use meta-include that contains **whole QtNetwork module**

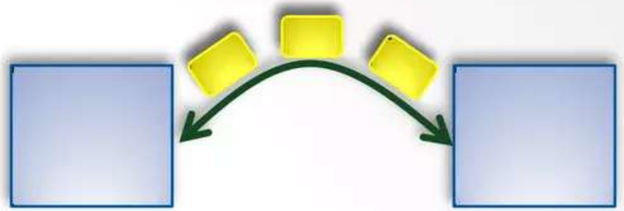
```
#include <QtNetwork>
```



Sockets

- **Socket**
  - Logical communication endpoint between 2+ software processes
- **Communication**
  - **Peer-to-Peer:** Two similar processes communicate
  - **Client-Server:** Different roles, e.g. web server & browser

Connection



- **Connection-oriented – "reliable"**
  - First establishes end-to-end-connection, then sends data
  - Connection is present → can check delivery order, arrival, errors, ...
- **Connectionless – "unreliable"**
  - Requires destination address each time data is sent (datagrams)
  - Each received packet is treated independently
  - No guarantees on order, duplication, delivery

TCP

- **Stream-oriented protocol ("reliable")**
  - Often preferable to HTTP in the mobile context: less overhead
- **Implementation provided by: `QTcpSocket`**
  - Either use **instance directly** or **derive** from it
  - Operations performed **asynchronously**
  - **Status changes** and errors: emitted via **signals**
  - Indirectly derived from **`QIODevice`**: allows `QDataStream` and `QTextStream`

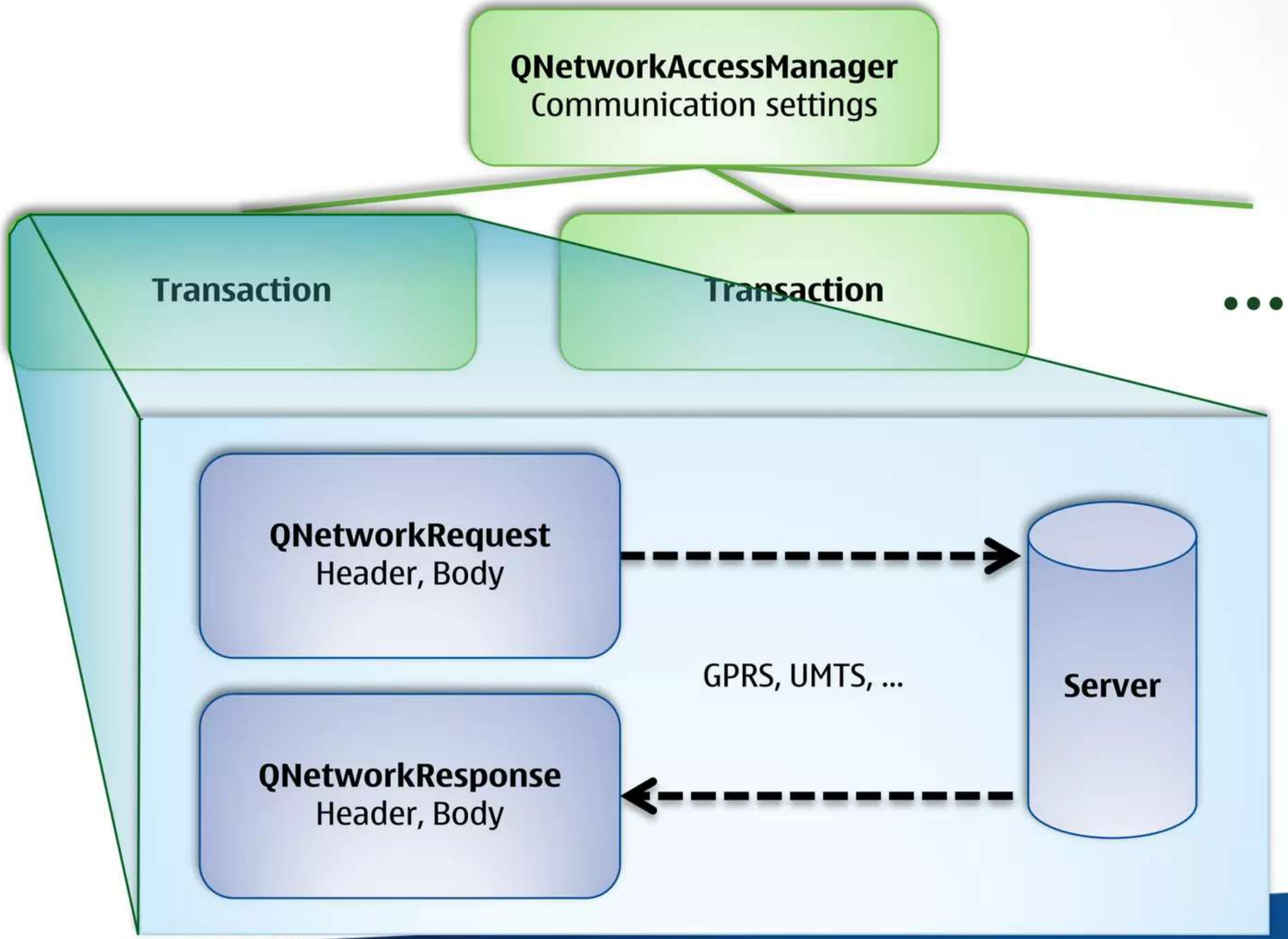
Servers and UDP

- **TCP Server: QTcpServer**
  - Single- or multi-threaded
  - See *Fortune Server* example from Qt SDK
- **UDP ("unreliable")**
  - **Connectionless**
  - No extra server class required
  - Instead: use `bind()` **of** `QUdpSocket`
  - **Data transfer:** smaller datagrams

High Level Network Operations

- **For communicating with services:**
  - Use high level classes for common protocols
  - **QNetworkAccessManager**
    - Coordinates multiple network requests
    - Creates and monitors the requests
    - Currently supports: HTTP, FTP and file access
  - **QNetworkRequest**
    - Represents individual request
    - Contains request header, URL, etc.
  - **QNetworkReply**
    - Created by manager
    - Contains response header, URL, reply contents, etc.

The dedicated `QHttp` and `QFtp` classes are deprecated.


QNetworkAccessManager
Communication settings
Transaction
Transaction
QNetworkRequest
Header, Body
GPRS, UMTS, ...
QNetworkResponse
Header, Body
Server

HTTP Protocol

- **Higher level protocol, based on TCP**
  - **Stateless protocol**
  - Request and response are self-contained
- **Well-known from web**
  - Client **browser sends request** for website to server
  - **Server responds** with web page contents
- **Mobile world**
  - HTTP-based **web services**: Client only requests specific data
  - **Response:** usually **XML**
  - Concept used by **AJAX requests** (Asynchronous Javascript and XML) → Web 2.0

Image Credit: Mozilla Foundation

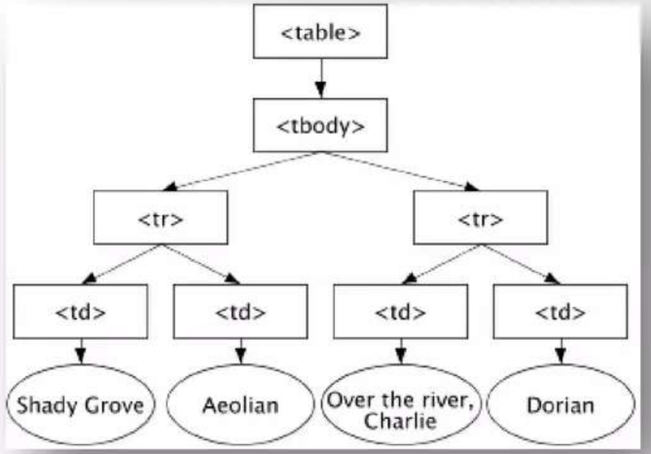
XML (Extensible Markup Language)

- **Properties**
  - General-purpose specification for creating custom markup languages
  - Textual data format
  - Markup (usually tags – e.g., <html>) and content
- **Qt**
  - Add ***xml* module** to project file (.pro)

Sample XML file
Truncated KML data from Google Maps

```
<kml>
  <Response>
    <name>FH Hagenberg, Austria</name>
    <Placemark id="p1">
      <address>
        Fachhochschule Hagenberg,
        4232 Hagenberg im Muehlkreis,
        Oesterreich
      </address>
      <ExtendedData>
        <LatLonBox north="48.3760743"
                   south="48.3612490"
                   east="14.5310893"
                   west="14.4990745"/>
      </ExtendedData>
      <Point>
       <coordinates>14.5150819,
                    48.3686622,0</coordinates>
      </Point>
    </Placemark>
  </Response>
</kml>
```

Parsing XML



- **DOM (Document Object Model)**
  - Standard of W3C
  - `QDomeDocument` builds hierarchical tree
  - Contains all nodes of XML file

Non-consecutive access to elements (e.g., web browser)
Easy to transfer DOM tree back into XML file

High memory requirements (especially for mobile devices)

Parsing XML

- **SAX (Simple API for XML)**
  - Event-driven API: `QXmlSimpleReader`
    - Triggers events; e.g., when encountering opening / closing tag
    - Override virtual event handler methods

More lightweight than DOM

More difficult to manipulate structure of data
(already handled data is discarded)
Parsing logic distributed over various functions, based on tag type, not on currently parsed contents

Parsing XML

- **Pull Parsing**
  - Iterator over XML elements (`QXmlStreamReader`)
  - Application controls process of parsing the file
  - Methods pull tokens from reader one after another

Lightweight
More straightforward to understand and maintain

More difficult to manipulate structure of data

Parsing XML

- **XQuery / XPath (W3C)**
  - Language to query XML data structure
  - Doesn't require manual procedural programming
  - Implementation: `QtXmlPatterns`

Easy access to specific information within XML

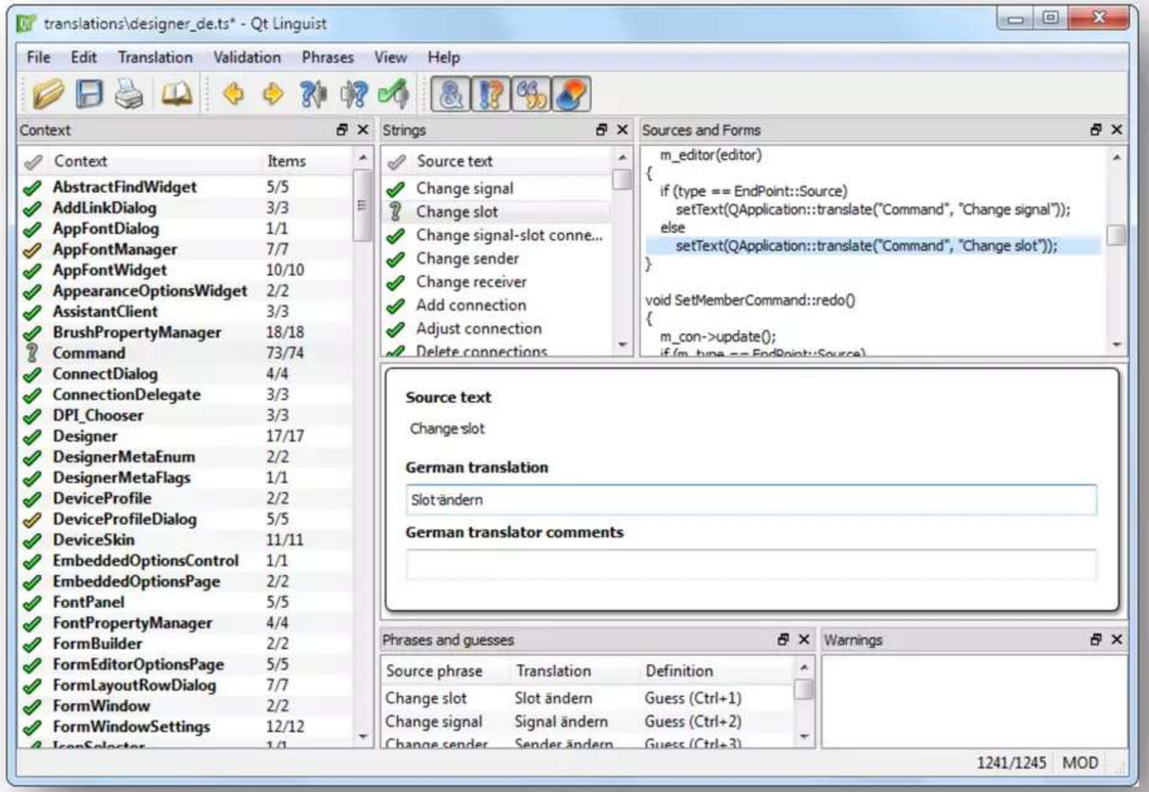
No support for updating XML documents
Lacks full text search

Internationalization

Internationalization

- **... is more than language**
  - Spelling
  - Ligatures: fi ⇨ fi
  - Formats (numbers, dates, currencies)
  - Non-spacing or diacritical marks (accents / umlauts in European languages)
  - Special line breaking behaviour
  - Character encoding
  - Presentation conventions (bidirectional writing)
  - Input techniques
- **Internationalization support built into Qt widgets & tools**

Qt Linguist



- **Tool to translate your application**
  - **Translation files** (.ts, xml-based) extracted from your source code
  - Qt Linguist only needs xml file → simple for external translators
- **Provides validation and preview (for Qt Designer-generated UIs)**

Preparing your Application

- **Mark user-visible strings for translation with tr()**
  - Inside functions in `QObject` subclasses that use the `Q_OBJECT` macro:

```
label = new QLabel("Hello World"), this);
```

```
label = new QLabel(tr("Hello World"), this);
```

  - **Other text-positions within your source code:**
    - Use `tr()` from other `QObject`-derived class
    - `QCoreApplication::translate()`
    - Completely outside functions: `QT_TR_NOOP()` / `QT_TRANSLATE_NOOP()` macros

Translation Context

- **Translation might be different according to context**
  - "Open" for file in German: "Öffnen"
  - "Open" for Internet connection in German: "Aufbauen"
- **Additional Information for the Translator**
  - **Class name** automatically provided by Qt Linguist
  - **Custom comments** through 2nd parameter of `tr()`:
    - Add explanation for context or usage area

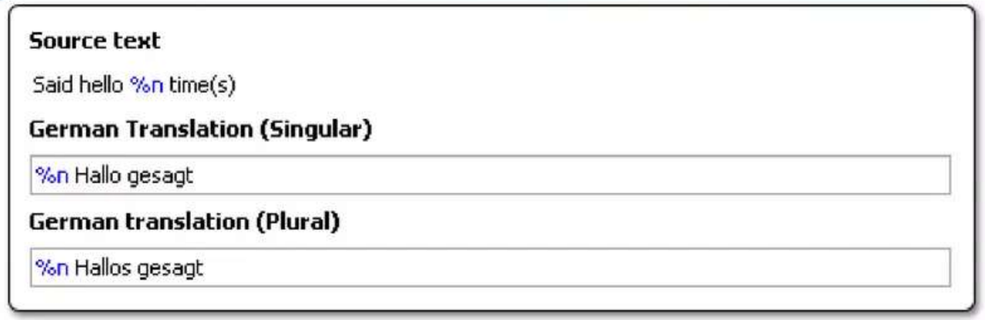
```
setWindowTitle(tr("Welcome", "Window title"));
```

  - Provide even more through **TRANSLATOR comments**

Plural, Keyboard Accelerators

- **Plural: provide extra translations depending on a value**

```
int nrHellos = 1;
label2 = new QLabel(tr("Said hello %n time(s)", "", nrHellos));
```



  - More information:

    http://qt.nokia.com/doc/qq/qq19-plurals.html

- **Also translate keyboard accelerators**

```
exitAct = new QAction(tr("E&xit"), this);
exitAct->setShortcut(tr("Ctrl+Q", "Quit"));
```

Add Languages

- **Edit the `.pro` file and add desired translation(s):**

```
TARGET = translator1
TEMPLATE = app
SOURCES += main.cpp \
    mywidget.cpp
HEADERS += mywidget.h
FORMS +=
TRANSLATIONS = translator1_de.ts \
               translator1_fr.ts
```

- **Run `lupdate <.pro-filename>`**
  - Finds translatable strings in source code, headers and Qt Designer files
  - Generates/updates XML file for each language
  - Translate these files with Qt Linguist

```
C:\Qt\Workspace\translator1>lupdate translator1.pro
Updating 'translator1_de.ts'...
    Found 3 source text(s) (3 new and 0 already existing)
Updating 'translator1_fr.ts'...
    Found 3 source text(s) (3 new and 0 already existing)
```

Finished Translations?

- **Run `lrelease <.pro-filename>`**
  - Produces compact binary `.qm` files out of `.ts` files
  - Only integrates translations marked as “finished”

```
C:\Qt\Workspace\translator1>lrelease translator1.pro
Updating 'C:/Qt/Workspace/translator1/translator1_de.qm'...
    Generated 3 translation(s) (3 finished and 0 unfinished)
Updating 'C:/Qt/Workspace/translator1/translator1_fr.qm'...
    Generated 0 translation(s) (0 finished and 0 unfinished)
    Ignored 3 untranslated source text(s)
```

Loading Translations

- **To use translations: QTranslator**
  - Usually initialized at **beginning of main()**

```
// Get locale of the system
QString locale = QLocale::system().name();
// Load the correct translation file (if available)
QTranslator translator;
translator.load(QString("translator1_") + locale, qApp->applicationDirPath());
// Adds the loaded file to list of active translation files
app.installTranslator(&translator);
```

Loading Translations II

Locale: de_AT (Austrian dialect of German language)

- **Locale:**
  - e.g., `de_AT`
- **QTranslator::load()**
  - **Second parameter:** directory of .qm file(s)
  - Automatically tries to **load more generic translations**

```
// Get locale of the system
QString locale = QLocale::system().name();
// Load the correct translation file (if available)
QTranslator translator;
translator.load(QString("translator1_") + locale, qApp->applicationDirPath());
// Adds the loaded file to list of active translation files
app.installTranslator(&translator);
```

Troubleshooting?

- **Not loading the translations?**
  - Make sure the translation files are found
  - Copy *.qm to directory of executable if in doubt!

Forum Nokia

Thank You.