Qt Logging Framework

All you ever wanted to know about QDebug and friends

Kai Köhne

Senior Software Engineer, located in Berlin

Working for ~~Trolltech~~ ~~Nokia~~ Digia The Qt Company

Interests: #qtcreator #mingw #ifw #qtinstaller #qtlogging

Agenda

- Generating logging messages
- Categorized logging
- Formatting log messages
- Logging backends

Say Hello!

```
qDebug("hello");
qWarning("hello!!");
qCritical("HELLO!?!");
qFatal("giving up :(");
```

Logs message, message type, file, line, function

Macros since Qt 5.0:

```
#define qDebug \
  QMessageLogger(__FILE__, __LINE__, \
                 Q_FUNC_INFO).debug
```

qFatal aborts

printf() style logging

```
// print a QString
QString qstr = "Hello";
qDebug("QString: %s",
    qstr.toUtf8().constData());

// print a std::wstring
std::wstring wstr = L"Hello";
qDebug("wstring: %s",
    QString::fromStdWString(
        wstr.c_str())
        .toUtf8()
        .constData());
```

Compiler: system printf

- %s expects local 8 bit encoding
- %ls is wchar_t* (UTF-16 or UTF-16 or...)

Runtime: QString::vsprintf()

- %s expects UTF-8
- %ls needs to be UTF-16 (ushort*)

Stream logging

```
#include <Qdebug>
qDebug() << "Hello World";
qWarning() << QDate::currentDate();
qCritical() << QRect(0,10,50,40);
Hello World
QDate("2014-09-26")
QRect(0,10 50x40)
```

```
QString name = "World";
qDebug() << "Hello" << name;
Hello "World"
qDebug().nospace().noquote()
        << "Hello" << name;
HelloWorld
```

Convenient logging for a lot of Qt types

Tweak the formatting:

- nospace(), space()
- noquote(), quote() (Qt 5.4)

Stream logging

```
QStringList list;
// ...
{
  QDebug dbg = qDebug();
  dbg.nospace().noquote();
  dbg << "Contents:";
  for (int i = 0; i < list.size(); ++i)
    dbg << i << '/' << list.at(i) << " ";
}

Contents: 0/A 1/B 2/C
```

Assembling log entry

```
qDebug().nospace()
    << "RGB: " << hex << uppercasedigits
    << 0xff << 0x33 << 0x33;

RGB: FF3333
```

QTextStream manipulators

Stream logging

```
#include <QDebug>

struct RgbColor {
    uchar red, green, blue;
};

QDebug operator<<(QDebug dbg,
                  const RgbColor &color)
{
    QDebugStateSaver saver(dbg);
    dbg.resetFormat();
    dbg.nospace() << hex;
    dbg << "RgbColor("
        << color.red << color.green
        << color.blue << ")";
    return dbg;
}
```

QDebugStateSaver (Qt 5.1)

- Saves stream formatting state, and restores it on destruction
- Adds potential space on destruction

QDebug::resetFormat() (Qt 5.4)

- Immediately resets stream formatting state

QML/JS logging

```
console.log("testing", name);
console.debug("hello?", name);
console.info("Hello", name);
console.warn("Hello!", name);
console.error("no one there",
              name);

console.assert(answer==="hello"
          , "invalid answer");

console.trace();
console.count(var);
```

log()	QtDebugMsg
debug()	
info()	
trace()	
count()	
warn()	QtWarningMsg
error()	QtCriticalMsg
assert()	

Conditional Logging

QT_NO_DEBUG_OUTPUT ,
QT_NO_WARNING_OUTPUT

- Compile time
- No selection possible

Filter in message handler

- Somewhat complicated
- Expensive

Custom logic

- Environment variables
- Custom macros
- …

Categorized logging (Qt 5.2)

Split up logging messages in hierarchical categories.

Category is identified by it's name

```
category.subcategory.subsubcategory[...]
```

Logging of messages can be enabled or disabled based on the category and message type, at runtime.

Categorized logging (Qt 5.2)

```
#include <QLoggingCategory>

QLoggingCategory lcEditor("qtc.editor");

qCDebug(lcEditor) << "hello";
qCWarning(lcEditor) << "hello!!";
qCCritical(lcEditor) << "HELLO!?!";
qCDebug(lcEditor, "%s", "World"); //Qt5.3



Q_DECLARE_LOGGING_CATEGORY(lcEditor)
Q_LOGGING_CATEGORY(lcEditor,"qtc.editor")
```

QLoggingCategory

- Runtime representation of category
- Configured by registry

Q_DECLARE_LOGGING_CATEGORY, Q_LOGGING_CATEGORY

- Define category in global scope

Categorized logging (Qt 5.2)

```
#define qCDebug(category, ...) \
  for (bool qt_category_enabled = category().isDebugEnabled();
       qt_category_enabled; qt_category_enabled = false) \
    QMessageLogger(__FILE__, __LINE__, Q_FUNC_INFO,
                   category().categoryName()).debug(__VA_ARGS__)

#define Q_LOGGING_CATEGORY(name, ...) \
  const QLoggingCategory &name() \
  { \
    static const QLoggingCategory category(__VA_ARGS__); \
    return category; \
  }
```

Categorized logging (Qt 5.2)

Category configuration

- Default: Logging of all message types is enabled
- `QLoggingCategory(const char *, QtMsgType severityLevel)` (Qt 5.4)
  - Disables message types < severityLevel
- Category filter
- Textual logging rules

Categorized logging (Qt 5.2)

```
#include <QLoggingCategory>

QLoggingCategory::CategoryFilter oldFilter
                                        = 0;

void filter(QLoggingCategory *cat) {
    printf("Category registered: '%s'\n",
           cat->categoryName());
    oldFilter(cat);
}

// ...

oldFilter =
QLoggingCategory::installFilter(filter);
```

Low level hook to configure categories

Categorized logging (Qt 5.2)

```
*=true
qtc.*=false
qtc.editor=true
qtc.editor.debug=false
```

Logging Rules

<category>[.<type>] = true|false

- '*' wildcard as first and/or last
- Evaluated top to bottom

Categorized logging (Qt 5.2)

Sources

- `[Rules]` section of QtProject/qtlogging.ini (Qt 5.3)
- QLoggingCategory::setFilterRules(const QString &rules)
- `[Rules]` section of file set in QT_LOGGING_CONF (Qt 5.3)
- `QT_LOGGING_RULES` environment variable (Qt 5.3)

Recap

Logging entry consists of

- Type (QtDbgMsg, QtWarningMsg, QtCriticalMsg, QtFatalMsg)
- Category name
- Message text
- File, line, function information

Message Formatting (Qt 5.0)

Enrich debug output by printing metadata

- qSetFormatPattern()
- QT_MESSAGE_PATTERN environment variable

Default pattern (Qt 5.2):

```
%{if-category}%{category}: %{endif}%{message}
```

Message Formatting (Qt 5.0)

Format placeholders:

```
%{type} %{category} %{file} %{function} %{line} %{message}
%{appname} %{pid} %{threadid}
%{time [format]} %{backtrace [depth=N] [separator="..."]} (Qt 5.4)
%{if-category} %{if-warning} %{if-critical} %{if-fatal} … %{endif} (Qt 5.1)
%{if-category} … {endif} (Qt 5.2)
```

Backends

stderr

Windows Debugger Log

QNX slogger2 (Qt 5.0)

journald (Qt 5.3)

Android message handler (Qt 5.1)

Tweak selection with

QT_LOGGING_TO_CONSOLE=0

QT_LOGGING_TO_CONSOLE=1

Custom message handler

```
void messageHandler(QtMsgType type,
    const QMessageLogContext &context,
    const QString &message)
{
     static QMutex mutex;
     QMutexLocker lock(&mutex);
     static std::ofstream
            logFile("logfile.txt");
     if (logFile) {
         logFile
            << qPrintable(
                 qFormatLogMessage(type,
                        context, message))
            << std::endl;
     }
}
```

Must be thread-safe!

Recursion

- Checked if compiler supports thread_local
- Option: QThreadLocalStorage

qFormatLogMessage()

- uses message pattern (Qt 5.4)

Custom message handler

```
static QtMessageHandler originalHandler
                                        = 0;

int main(int argc, char *argv[])
{
    originalHandler =
   qInstallMessageHandler(messageHandler);
   // ...
}
```

qInstallMessageHandler()

- Returns current/default handler
- Message handlers can be chained

Summary

Qt 5 additions

- Logging metadata
- Categorized logging
- New backends (Android, JournalD)
- Customizing message output

Future

- QML/JS API for logging categories
- QtInfoMsg message type
- Log to file