Snippets details and examples

Snippet contents

- Place the cursor after snippet insertion: use double-dollar:

```
My snippet $$ cursor placement
```

- Replace multiple occurrences: dollar before and after

```
class $MyClass${
public:
    $MyClass$(){}
    ~$MyClass$(){}
};
```

- Upper/lower and first character case:

```
$upper:u$
$LOWER:l$
$cASE:c$
```

- Insert creator variable:

```
%{ActiveProject:Name}
%{UUID}
```

Snippet examples

Type: C++

Shortcut: ggcert

Code: QSslCertificate inspection and display

```
void certificateAnalysis(QSslSocket *socket)
{
    QSslCertificate cert = socket->peerCertificate();
    QList<QSslCertificate> certChain = socket->peerCertificateChain();
    certificateAnalysis(cert, certChain);
}

void certificateAnalysis(QSslCertificate certificate, QList<QSslCertificate>
certificateChain){
    QSslCertificate cert = certificate;
    if( cert.isNull() ){
        qInfo().noquote() << "The client has not presented a certificate";
        return;
```

```
    }
    QList<QSslCertificate> chain = certificateChain;
    qInfo().noquote() << "";
    qInfo().noquote() << "#################### Certificate analysis
####################";
    qInfo().noquote() << "Certificate chain length:" << chain.count();
    qInfo().noquote() << "Self-signed:" << cert.isSelfSigned();
    qInfo().noquote() << "Valid from:" << cert.effectiveDate().toString();
    qInfo().noquote() << "Valid until:" << cert.expiryDate().toString();
    qInfo().noquote() << "Serial number:" << cert.serialNumber();
    qInfo().noquote() << "Version:" << cert.version();
    qInfo() << "SHA256 digest:" << cert.digest(QCryptographicHash::Sha256);
    qInfo().noquote() << "";
    qInfo().noquote() << "Issuer information:";
    qInfo().noquote() << "\tIssuer display name:" << cert.issuerDisplayName();
    qInfo().noquote() << "\tIssuer Organization (O):" << cert.issuerInfo(
QSslCertificate::Organization ).join(" ");
    qInfo().noquote() << "\tIssuer Common Name (CN) :" << cert.issuerInfo(
QSslCertificate::CommonName ).join(" ");
    qInfo().noquote() << "\tIssuer Locality (L) :" << cert.issuerInfo(
QSslCertificate::LocalityName ).join(" ");
    qInfo().noquote() << "\tIssuer Organizational Unit (OU):" << cert.issuerInfo(
QSslCertificate::OrganizationalUnitName ).join(" ");
    qInfo().noquote() << "\tIssuer Country Name (C):" << cert.issuerInfo(
QSslCertificate::CountryName ).join(" ");
    qInfo().noquote() << "\tIssuer State/province (ST):" << cert.issuerInfo(
QSslCertificate::StateOrProvinceName ).join(" ");
    qInfo().noquote() << "\tIssuer Distinguished Name Qualifier:" <<
cert.issuerInfo( QSslCertificate::DistinguishedNameQualifier ).join(" ");
    qInfo().noquote() << "\tIssuer Serial Number:" << cert.issuerInfo(
QSslCertificate::SerialNumber ).join(" ");
    qInfo().noquote() << "\tIssuer E-Mail Address:" << cert.issuerInfo(
QSslCertificate::EmailAddress ).join(" ");
    qInfo().noquote() << "";
    qInfo().noquote() << "Subject information:";
    qInfo().noquote() << "\tSubject display name:" << cert.subjectDisplayName();
    qInfo().noquote() << "\tSubject Organization (O):" << cert.subjectInfo(
QSslCertificate::Organization ).join(" ");
    qInfo().noquote() << "\tSubject Common Name (CN) :" << cert.subjectInfo(
QSslCertificate::CommonName ).join(" ");
    qInfo().noquote() << "\tSubject Locality (L) :" << cert.subjectInfo(
QSslCertificate::LocalityName ).join(" ");
    qInfo().noquote() << "\tSubject Organizational Unit (OU):" <<
cert.subjectInfo( QSslCertificate::OrganizationalUnitName ).join(" ");
    qInfo().noquote() << "\tSubject Country Name (C):" << cert.subjectInfo(
QSslCertificate::CountryName ).join(" ");
    qInfo().noquote() << "\tSubject State/province (ST):" << cert.subjectInfo(
QSslCertificate::StateOrProvinceName ).join(" ");
    qInfo().noquote() << "\tSubject Distinguished Name Qualifier:" <<
cert.subjectInfo( QSslCertificate::DistinguishedNameQualifier ).join(" ");
    qInfo().noquote() << "\tSubject Serial Number:" << cert.subjectInfo(
QSslCertificate::SerialNumber ).join(" ");
    qInfo().noquote() << "\tSubject E-Mail Address:" << cert.subjectInfo(
QSslCertificate::EmailAddress ).join(" ");
    qInfo().noquote() << "";
    qInfo().noquote() << "Server public key:";
```

```
    qInfo().noquote() << "";
    qInfo().noquote() << cert.publicKey().toPem();

    if( cert.extensions().count() > 0 ){
        qInfo().noquote() << "Certificate extensions:";
        for( auto extension : cert.extensions() ){
            qInfo().noquote() << "\tExtension name:" << extension.name();
            qInfo().noquote() << "\t\tCritical:" << extension.isCritical();
            qInfo().noquote() << "\t\tSupported:" << extension.isSupported();
            qInfo().noquote() << "\t\tOID:" << extension.oid();

            QString variantType = extension.value().typeName();
            variantType = variantType.toLower();
            if( variantType == "qbytearray" ){
                qInfo().noquote() << "\t\tValue:" <<
extension.value().toByteArray();
            }
            else if( variantType == "qstring" ){
                qInfo().noquote() << "\t\tValue:" <<
extension.value().toString();
            }
            else if( variantType == "bool" ){
                qInfo().noquote() << "\t\tValue:" << extension.value().toBool();
            }
            else if(  variantType == "qvariantmap" ){
                for( auto [name,value] :
extension.value().toMap().asKeyValueRange() ){
                    QString nameValue;
                    QString map_name = name;
                    QVariant map_variant = value;
                    QString varType = map_variant.typeName();
                    varType = varType.toLower();
                    if( varType == "qbytearray" ){
                        nameValue += name % "=" % map_variant.toByteArray();
                    }
                    else if( varType == "bool" ){
                        nameValue += name % "=" %
QVariant(map_variant.toBool()).toString();
                    }
                    else if( varType == "qstring" ){
                        nameValue += name % "=" % map_variant.toString();
                    }
                    else{
                        nameValue += name % "=" % map_variant.toString();
                    }
                    qInfo().noquote() << "\t\tValue:" << nameValue;
                }
            }
            else{
                qInfo().noquote() << "\t\tValue:" << extension.value();
            }
        }
    }

    if( chain.count() > 0 ){
        qInfo().noquote() << "";
```

```
        qInfo().noquote() << "Certificate chain:";
        for( auto cert : chain ){
            qInfo().noquote() << "Certificate name:" <<
cert.subjectDisplayName();
            qInfo().noquote() << "Public key:";
            qInfo().noquote() << "";
            qInfo().noquote() << cert.publicKey().toPem();
        }
    }
    qInfo().noquote() << "#################### End of certificate analysis
####################";
    qInfo().noquote() << "";
}
```

Type: C++

Shortcut: ggcli

Code: Console application with QCommandLineParser

```
#include <QCoreApplication>
#include <QCommandLineOption>
#include <QCommandLineParser>

void initArgumentParser(QCoreApplication &app, QCommandLineParser &parser);
void initArgumentOptions(QCoreApplication &app, QCommandLineParser &parser);
void processArgumentOptions(QCoreApplication &app, QCommandLineParser &parser);

QList<QCommandLineOption> commandLineOptions;

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QCommandLineParser p;
    initArgumentParser(a,p);
    // return(a.exec());
    return(0);
}

void initArgumentParser(QCoreApplication &app, QCommandLineParser &parser){
    // TODO CODE: Application name
    app.setApplicationName("name");
    // TODO CODE: Application version
    app.setApplicationVersion("0.0.1");
    // TODO CODE: Organization name
    app.setOrganizationName("Golding's Gym");
    // TODO CODE: Organization domain (web presence)
    app.setOrganizationDomain("https://github.com/damiengolding");
    // Convenience options
    parser.addHelpOption();
    parser.addVersionOption();
    parser.setApplicationDescription("myapp description");
    // Init options from here, defined separately in initArgumentOptions
```

```
    initArgumentOptions(app,parser);
    // Process the supplied arguments into the parser
    parser.process(app);
    // Process the arguments supplied to the application so we can use them
    processArgumentOptions(app, parser);
}

void initArgumentOptions(QCoreApplication &app, QCommandLineParser &parser){
    // TODO CODE: Add options
    /*
        Various ways to create options:
            Simplest:
                parser.addOption({{"o","output"},"Write to file","file"});
            Create a list:
                QList<QCommandLineOption> commandLineOptions;
                QCommandLineOption op1("short name","description","long name");
                parser.addOption(op1);
                commandLineOptions.append(op1);
        Positional arguments
            parser.addPositionalArgument("list-types", "List supported types");
    */

}

void processArgumentOptions(QCoreApplication &app, QCommandLineParser &parser){
    // TODO CODE: Process supplied options
    /*
       Individually:
        if(parser.isSet(<QString name|QCommandLineOption>)){
            Either get the value of an option:
                QString s = parser.value(<QString name|QCommandLineOption>);
            Or operate on a switch, e.g. parser.isSet("verbose"):
        }
       Grouped as QString names:
            for(QString n : parser.optionNames()){
                As above, but no isSet(...) test need
            }
       Grouped as QCommandLineOptions:
            for(QCommandLineOption clo : commandLineOptions ){
                As above
            }
       Manage incorrect/unrecognised/missing options:
            for( QString opt : parser.unknownOptionNames() ){
                qWarning() << ""; // recoverable error
                qCritical() << ""; // non-recoverable error, usually a system
error such as read/write/execute privileges
                qFatal(""); // non-recoverable error, will exit and dump core
            }
        Poistional arguments:
            for(QString pos : parser.positionalArguments()){
                qDebug() << "[debug] Positional argument: " << pos;
            }
    */

}
```

Type: C++

Shortcut: ggcqt

Code: Custom Qt object - capable of being a QVariant

```
#include <QMetaType>
#include <QDebug>

class $name$
{
public:
    $name$() = default;
    ~$name$() = default;
    $name$(const $name$ &) = default;
    $name$ &operator=(const $name$ &) = default;
/*
      --- Custom constructor here ---
*/
    // $name$( const &etc arg1);

public:
    friend QDebug operator<<(QDebug debug, const $name$ &c)
    {
        QDebugStateSaver saver(debug);
        debug << "";
        return debug;
    }

    friend QDataStream & operator<<(QDataStream &arch, const $name$ &c )
    {
        arch << c.m_variable1;
        arch << c.m_variable2;
        return arch;
    }

    friend QDataStream & operator>>(QDataStream &arch, $name$ &c )
    {
        arch >> c.m_variable1;
        arch >> c.m_variable2;
        return arch;
    }

protected:

};

Q_DECLARE_METATYPE($name$);
#error "Qt custom type declared. Ensure that you are calling
qRegisterMetaType<$name$>(); before use, then delete this error."
```

Type: C++

Shortcut: ggdlldec

Code: Declaration of a function which can be used with rundll32.exe (*.h)

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>

__declspec(dllexport) void CALLBACK $EntryPoint$(
    HWND hwnd,
    HINSTANCE hinst,
    LPSTR lpszCmdLine,
    int nCmdShow);
```

Type: C++

Shortcut: ggdlldef

Code: Definition of a function which can be used with rundll32.exe (.cpp)

```
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}

void CALLBACK $EntryPoint$(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int
nCmdShow)
{
    int msgboxID = MessageBox(
        NULL,
        L"Hello World from Run32dll",
        L"Hello World",
        MB_ICONWARNING | MB_CANCELTRYCONTINUE | MB_DEFBUTTON2
    );

    switch (msgboxID)
```

```
    {
    case IDCANCEL:
        // TODO: add code
        break;
    case IDTRYAGAIN:
        // TODO: add code
        break;
    case IDCONTINUE:
        // TODO: add code
        break;
    }
}
```

Type: C++

Shortcut: ggdbgp

Code: Debug print support

```
public:
friend QDebug operator<<(QDebug debug, const $$&c)
{
    QDebugStateSaver saver(debug);
    debug.nospace() << "";
    return debug;
}
```

Type: C++

Shortcut: ggdnl

Code: Converts a QDomNodeList to a QList of QDomElement

```
QList<QDomElement> domElementList(const QDomNodeList &list){
    QList<QDomElement> ret;
    for( int i = 0; i<list.count();++i ){
        QDomNode node = list.at(i);
        QDomElement elem = node.toElement();
        if( elem.isNull() ) continue;
        ret.append(elem);
    }
    return(ret);
}
```

Type: C++

Shortcut: ggeconv

Code: Converts an enum declared with Q_ENUM to a string, and vice versa

```
#include <QString>
#include <QMetaEnum>
#include <QDebug>

template<typename E>
E EnumFromString(const QString &textValue){
    bool ok;
    auto enumResult = static_cast<E>(QMetaEnum::fromType<E>
().keyToValue(textValue,&ok));
    if(!ok){
        qDebug()<<"Could not convert " << textValue << " to enum.";
        return{};
    }
    return(enumResult);
}

template<typename E>
QString StringFromEnum(E value){
    const int intRepresentation = static_cast<int>(value);
    return( QString::fromUtf8(QMetaEnum::fromType<E>
().valueToKey(intRepresentation)) );
}
```

Type: C++

Shortcut: ggetg

Code: Events, transitions and guards for Qt State Machine Framework

```
#include <QAbstractTransition>
#include <QEvent>
#pragma once

/*!
 * \brief The EnumStruct class
   */

struct EnumStruct{
    // Implement the enum here
    // enum Events{
        // EnterIdling,
        // EnterOpening,
        // EnterSaving,
```

```
        // EnterCreating,
        // EnterSleeping,
        // EnterClosing
    // };
};

/*!
 * \brief The EnumEvent class
   */

struct EnumEvent : public QEvent
{
    EnumEvent(EnumStruct::Events eventEnum)
    : QEvent(QEvent::Type(QEvent::User+1)),
      value(eventEnum) {}

    EnumStruct::Events value;

};

/*!
 * \brief The EnumTransition class
   */

class EnumTransition : public QAbstractTransition
{
    Q_OBJECT

public:
    EnumTransition(EnumStruct::Events eventEnum)
        : m_value(eventEnum) {}

protected:
    bool eventTest(QEvent *e) override
    {
        if (e->type() != QEvent::Type(QEvent::User+1)) // EnumEvent
            return false;
        EnumEvent *ee = static_cast<EnumEvent*>(e);
        return (m_value == ee->value);
    }

    void onTransition(QEvent *) override {}

private:
    EnumStruct::Events m_value;
};
```

Type: C++

Shortcut: ggfsm

Code: Native implementation of a simple Finite State Machine

```
#include <QObject>

class IFiniteState;

/*!
 * brief The IFiniteStateMachine class
 */

class IFiniteStateMachine : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QList<IFiniteState*> actions READ getActions WRITE setActions
NOTIFY actionsChanged FINAL)

public:
    enum FiniteStateMachineExitCode{
        ExitSuccess,
        ExitFailure,
        ExitIndeterminate,
        ExitTerminated,
        ExitYes,
        ExitNo
    };
    Q_ENUM(FiniteStateMachineExitCode)

public:
    QList<IFiniteState*> getActions() const;
    void setActions(const QList<IFiniteState*> &newActions);

public slots:
    void init(){
        initActions();
    }

    virtual void start() = 0;
    virtual void stop() = 0;

protected: // Properties
    QList<IFiniteState*> actions;

protected: // Internal functions
    virtual void initActions() = 0;

signals:
    void running();
    void exiting(FiniteStateMachineExitCode);
    void actionsChanged();
```

```
private:

};

/*!
 * brief The IFiniteState class
 */

class IFiniteState : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QList<IFiniteState *> actions READ getActions WRITE setActions
NOTIFY actionsChanged FINAL)
    Q_PROPERTY(bool asynchronous READ getAsynchronous WRITE setAsynchronous
NOTIFY isAsynchronousChanged FINAL)

public:
    QList<IFiniteState *> getActions() const;
    void setActions(const QList<IFiniteState *> &newActions);

    bool getAsynchronous() const;
    void setAsynchronous(bool newAsynchronous);

protected: // Properties
    QList<IFiniteState*> actions;
    bool asynchronous = true;

public slots:
    virtual void enter() = 0;

signals:
    void exitSuccess();
    void exitFailure();
    void exitIndeterminate();
    void exitYes();
    void exitNo();
    void exitMaybe();
    void exitFatal();
    void actionsChanged();
    void isAsynchronousChanged();
private:
};

/*
    Inline functions
*/

inline QList<IFiniteState*> IFiniteStateMachine::getActions() const
{
    return actions;
}

inline void IFiniteStateMachine::setActions(const QList<IFiniteState*>
&newActions)
{
    if (actions == newActions)
```

```
        return;
    actions = newActions;
    emit actionsChanged();
}

inline QList<IFiniteState *> IFiniteState::getActions() const
{
    return actions;
}

inline void IFiniteState::setActions(const QList<IFiniteState *> &newActions)
{
    if (actions == newActions)
        return;
    actions = newActions;
    emit actionsChanged();
}

inline bool IFiniteState::getAsynchronous() const
{
    return asynchronous;
}

inline void IFiniteState::setAsynchronous(bool newAsynchronous)
{
    if (asynchronous == newAsynchronous)
        return;
    asynchronous = newAsynchronous;
    emit isAsynchronousChanged();
}
```

Type: C++

Shortcut: ggnic

Code: Obtains a NIC name from an IP Address

```
#include <QList>
#include <QNetworkInterface>
#include <QString>

QString IpAddressToNicName( const QString& host ){
    QList<QHostAddress> addresses = QNetworkInterface::allAddresses();
    for( auto address : addresses ){
        QString ipAddress = address.toString();
        if( ipAddress != host ) continue;
        QString nicName;
        QList<QNetworkInterface> interfaces = QNetworkInterface::allInterfaces();
        for( auto interface : interfaces ){
            QList<QNetworkAddressEntry> networkEntries =
interface.addressEntries();
            for( auto networkEntry : networkEntries ){
```

```
                QString addr = networkEntry.ip().toString();
                if( addr == ipAddress ){
                    return( interface.name() );
                }
            }
        }
    }
    return( QString() );
}
```

Type: C++

Shortcut: ggregion

Code: Simulates a #region #endregion block in C#

```
#pragma $region$ {
    $$
#pragma $region$ }
```

Type: C++

Shortcut: ggsing

Code: Singleton implementation

```
// Private constructor
    $SingletonTemplate$() {}

    public:
        // Remove ability to use the copy constructor(s)
        $SingletonTemplate$($SingletonTemplate$ const&) = delete;
        $SingletonTemplate$ &operator=($SingletonTemplate$ const&) = delete;
        // Provide a single, static method for retrieving the singleton instance
        static $SingletonTemplate$ &instance() {
            static $SingletonTemplate$ _instance;
            return _instance;
        }
```

Type: C++

Shortcut: ggstatm

Code: Magic Static implementation

```
template < typename T >
class $name$ final
{
  public:
```

```
    static T& GetInstance()
    {
        static T instance;
        return instance;
    }

private:
    $name$() = default;
    ~$name$() = default;

    $name$(const $name$&) = delete;
    $name$& operator=(const $name$&) = delete;
    $name$($name$&&) = delete;
    $name$& operator=($name$&&) = delete;

};
```

Type: C++

Shortcut: ggtest

Code: Private SLOTS for a class which is testable under Qt Test Framework

```
#ifdef QT_DEBUG
private slots: // for QTest module
    void initTestCase();
    void init();
    void cleanup();
    void cleanupTestCase();
    void $SubTestName$_data();
    void $SubTestName$();
    friend QDebug operator<<(QDebug debug, const $$&c){
        QDebugStateSaver saver(debug);
        debug.nospace() << "";
        return debug;
    }
#endif
```

Type: C++

Shortcut: ggminfo (and associated types)

Code: qInfo() and associated calls

```
qInfo() << "$$";
```

Type: Text

Shortcut: ggompb

Code: Project (*.pro) file entries for Open Multi-Procesing (OMP) and Boost

```
# OpenMP
win32{
    QMAKE_CXXFLAGS += -openmp
    LIBS += -lvcomp
    message("OpenMP for win32")
}
else{
    QMAKE_CXXFLAGS += -fopenmp
    QMAKE_LFLAGS += -fopenmp
    LIBS += -lopenmp
    message("OpenMP for unix")
}

# Boost
# INCLUDEPATH += "C:\Boost_1_81_0"
# Add one path at a time, then add individual library, e.g.
# LIBS += -L"C:\Boost_1_81_0\lib64-msvc-14.3" -lboost_program_options-vc143-mt-gd-x64-1_81.lib
```

Type: C++

Shortcut: ggqprng

Code: Qt Pseudo Random Number Generator

```
#include <QRandomGenerator>

int r = QRandomGenerator::global()->bounded(0,2);
$$
```

Type: C++

Shortcut: ggprng

Code: Native Pseudo Random Number Generator

```
#include <cstdlib>

std::srand((unsigned) time(NULL));
int r = std::rand() % 100 + 1;
$$
```

Type: C++

Shortcut: ggumdh

Code: initTestCase and cleanupTestCase functions using umdh

```
void $name$::initTestCase()
{
    quint64 pid = QCoreApplication::applicationPid();
    QString umdh32 = "C:\\Program Files (x86)\\Windows
Kits\\10\\Debuggers\\x86\\umdh.exe";

    QStringList argList;
    QString pidArg = "-p:" % QString::number(pid);
    QString fileArg = "-f:base.txt";
    argList << pidArg << fileArg;

    QProcess process;
    process.setProgram(umdh32);
    process.setArguments(argList);
    process.start();
    process.waitForFinished();

}

void $name$::cleanupTestCase()
{
    quint64 pid = QCoreApplication::applicationPid();
    QString umdh32 = "C:\\Program Files (x86)\\Windows
Kits\\10\\Debuggers\\x86\\umdh.exe";

    QStringList argList;
    QString pidArg = "-p:" % QString::number(pid);
    QString fileArg = "-f:leak.txt";
    argList << pidArg << fileArg;

    QProcess process1;
    process1.setProgram(umdh32);
    process1.setArguments(argList);
    process1.start();
    process1.waitForFinished();

    QProcess process2;
    QString compareDumps = "C:\\Program Files (x86)\\Windows
Kits\\10\\Debuggers\\x86\\umdh.exe";
    argList.clear();
    argList << "base.txt" << "leak.txt";
    process2.setProgram(compareDumps);
    process2.setArguments(argList);
    process2.start();
    process2.waitForFinished();

    QByteArray deltaString = process2.readAllStandardOutput();
```

```
    QFile f( "delta.txt" );
    if( !f.open(QIODevice::WriteOnly) ){
        qInfo() << "Couldn't open delta.txt for writing";
        return;
    }
    QTextStream ts( &f );
    ts << deltaString;

    f.close();

}
```