qmake入門

日本Qtユーザー会

自己紹介

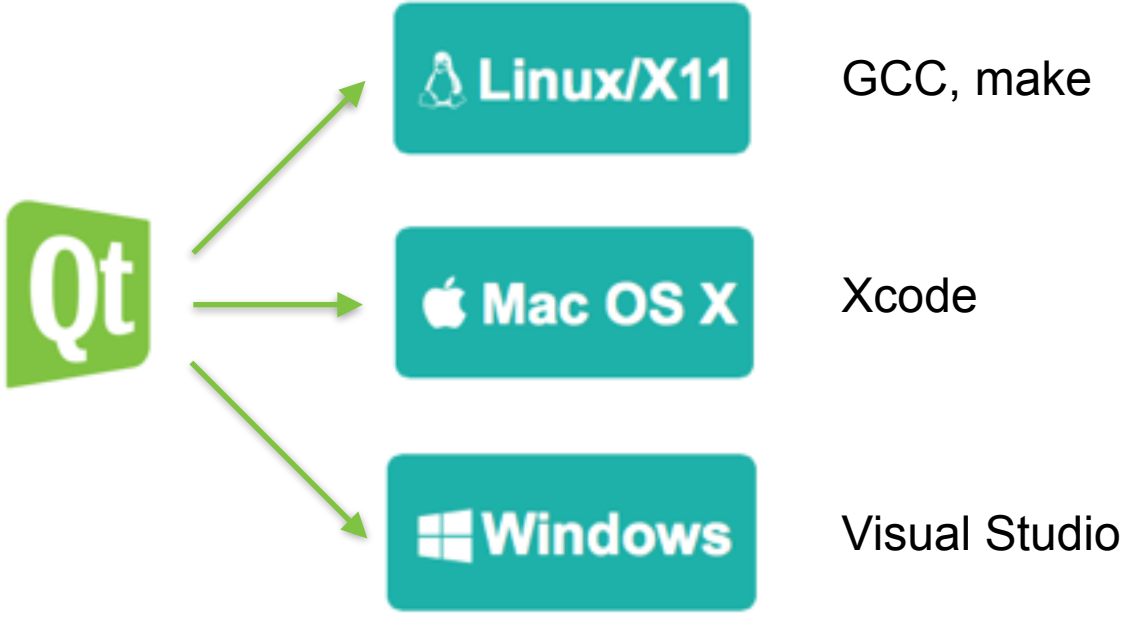
- Twitter ID : hermit4
- 日本Qtユーザー会　おやつ部部長
- フリーランスなので他の肩書きありません
- Qt3頃からQtの利用を開始
- 商用ライセンスユーザー

ごめんなさい

- あまり大きな声を出せる人ではないし、人前は苦手ですので、聴きにくかったら申し訳ない
- 既にQt CreatorにはqbsがQt同梱されているご時世なのですが、今回は、qbsの話はしません
- QtCreatorは最近使うようになったばっかりなのでコマンドラインの話が中心です
- 生成された後のプロジェクトファイルやMakefileの話はしません

qmakeの概要

qmakeとは



Qtはクロスプラットフォーム向けのフレームワークです
でも、ビルドのツールチェーンは各プラットフォームにお任せです

qmakeとは

.pro file

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = helloworld
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

→ Makefile

qmake -spec macx-xcode → .xcodeproj

qmake -tp vc → .vcxproj

helloworld.pro file

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = helloworld
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

- 変数
  - QT
  - TARGET
  - TEMPLATE
  - SOURCES
  - HEADERS
  - FORMS
- テスト関数
  - greaterThan

Qt5.5.0/Examples/Qt-5.5/qt3d/qt3d.pro

```
TEMPLATE = subdirs

SUBDIRS += \
    exampleresources \
    playground-qml \
     :
   skybox

# TODO Port the old examples to new APIs
#SUBDIRS += qt3d
#qtHaveModule(qml): SUBDIRS += quick3d

# Make all other subdirs depend on exampleresources
for(subdir, SUBDIRS) {
    !equals(subdir, exampleresources) {
        $${subdir}.depends += exampleresources
    }
}
```

- 変数
  - TEMPLATE
  - SUBDIRS
- テスト関数
  - for
  - equals
- コメント

qmakeの基礎

qmakeの基礎

- コマンド
  - qmake -project [-nopwd]
    プロジェクトファイルの生成
  - qmake [-makefile] [-spec spec][-tp prefix] [-t template] [-nocache|-cache file] [-nomoc] [-d] [-nodpend] files
    Makefileの生成
  - qmake [-query| -set varname value | -unset varname]
    プロパティ値の取得・設定・削除

qmakeの基礎 - プロパティ

```
QT_SYSROOT:
    :
    :
QT_HOST_PREFIX:/Users/hermit4/Qt5.5.0/5.5/clang_64
QT_HOST_DATA:/Users/hermit4/Qt5.5.0/5.5/clang_64
QT_HOST_BINS:/Users/hermit4/Qt5.5.0/5.5/clang_64/bin
QT_HOST_LIBS:/Users/hermit4/Qt5.5.0/5.5/clang_64/lib
QMAKE_SPEC:macx-clang
QMAKE_XSPEC:macx-clang
QMAKE_VERSION:3.0
QT_VERSION:5.5.0
```

-specを指定しない場合は、プロパティでプラットフォーム標準のプロパティに従ったSPECが利用されます

qmakeの基礎

- 入力ファイル
  - *.pro ファイル

    プロジェクトファイル
  - *.pri ファイル

    インクルードプロジェクトファイル、include()関数で読み込みます
  - *.prf ファイル

    機能ファイル
  - qmake.conf ファイル

    mkspecs下のファイル。-specで指定

qmakeの基礎 - Project File Elements

- 変数

  変数は、文字列のリストを保持するもので、簡単なプロジェクトは最低限の変数を指定するだけで作成できます。

  - TEMPLATE
  - CONFIG
  - QT
  - SOURCES
  - HEADERS
  - FORMS
  - RESOURCES
  - TRANSLATIONS

qmakeの基礎 - Project File Elements

- 変数

  他にもMakefileに必要な様々な変数が用意されています。

  - INCLUDEPATH
  - DEFINES
  - LIBS
  - DEPENDS
  - _PRO_FILE_
  - _PRO_FILE_PWD_

  変数によっては環境によってはMakefile等へ反映されない変数もあるので、注意が必要になります。

qmakeの基礎 - Project File Elements

- 変数
  - 値の代入

    TEMPLATE = app
  - 値の追加

    QT += core
  - ユニーク値の追加

    LIBS *= -lm
  - 値の削除

    QT -= gui
  - 値の置換

    DEFINES ~= s/QT_[DT].*/QT

    例えば、「DEFINES = QT_D_TEST QT_X_TEST」の場合、

    上記で$$DEFINES の値「 QT QT_X_TEST」に置き換わります

qmakeの基礎 - Project File Elements

- 変数
  - 変数の参照

    $$HOGE , $${HOGE}

    $${HOGE}_piyo
  - 環境変数の参照

    $$(HOME)
  - プロパティの参照

    $$[QMAKE_SPEC]
  - 注意事項
    - INCLUDEPATH += $(PWD)はMakefileでは“-I$(PWD)”に変換されます
    - message($$INCLUDEPATH) では”/home/hermit4”と展開されます

qmakeの基礎 - Project File Elements

- 空白・バックスラッシュ(¥)
  - 空白は、リストの区切り文字として使われます
  - 空白を含む文字列を一つの文字列として登録する場合は、ダブルクウォートかシングルクォートで囲む必要があります
  - 行末をバックスラッシュにすると、次の１行も合わせて連続した１行として処理されます
- コメント
  - # から行末まではコメントとして扱われます
  - “# xxxxx”となっていても、#から先はコメントになるので”閉じがなくてエラーになります

qmakeの基礎 - Project File Elements

- 組込関数

  qmakeではプロジェクトファイルで利用できる関数が幾つか定義されています。大別すると変換結果を返す変換関数と、真偽値を返すテスト関数に大別されます。

  - 変換関数
    - absolute_path
    - dirname
    - format_number
    - system
  - テスト関数
    - include
    - contains
    - equals
    - exists
    - system

qmakeの基礎 - Project File Elements

- スコープ構文

<condition>:<definition or command>

<condition> {
    <definition or command>
        :
}

<condition>を満たす場合に : の後の 1 行あるいは{}内のすべてが実行されます。入れ子にできるので

<condition>:<condition> {
    <definition or command>
}

という表記も可能です

qmakeの基礎 - Project File Elements

- スコープ構文 - condition

  conditionとして使えるのは以下の通り

  - テスト関数の結果

    contains(SOURCES, “debugprint.cpp”):DEFINES += DEBUGPRINT
  - プラットフォーム名(QMAKE_PLATFORMの値)

    win32:SOURCES += platform_w32.cpp

    !unix:SOURCES -= platform_unix.cpp
  - CONFIGに設定されたリスト

    release:DEFINES -= DEBUG_SETTING
  - QMAKESPEC 名

    linux-g++ : message(linux-g++ spec)
  - ‘:’ (AND), | (OR), ! (NOT)

qmakeの基礎 - TEMPLATE

どのようなMakefileを生成するのかを、TEMPLATE変数に値を設定することで選択できます。

- app
  アプリケーションの作成用Makefile
- lib
  ライブラリ作成用のMakefile
- subdirs
  ディレクトリ再帰用のMakefile
- aux [Makefile only]
  非コンパイル用のMakefile
- vcapp [Windows only]
- vclib [Windows only]
- vcsubdirs [Windows only]

qmakeの基礎 - CONFIG

qmakeの動作に関わる設定を行う変数で、リスト中の文字.prfファイルがあればそれを読み込み有効化します。

- qt (qt.prf)
- c++14 (c++14.prf)
- gcov (gcov.prf)

また、prfファイルはないものの、生成処理に影響を与える設定もあります。

- release
- debug
- staticlib
- dll
- order

qmakeの基礎 - アプリケーションの作成

- TEMPLATE = app
- CONFIG
  - windows
  - console
  - testcase

CONFIG += testcase を行うとMakefileにcheckターゲットが作成される。

make check時は以下の変数が利用できる

TESTRUNNER : テストの起動に呼び出されるツール

TESTARGS : テスト呼び出しに利用される引数

ex
make check TESTRUNNER=valgrind

qmakeの基礎 - ライブラリの作成

- TEMPLATE = lib
- CONFIG
  - dll
    共有ライブラリの生成
  - staticlib
    静的ライブラリの生成
  - plugin
    プラグインの作成

qmakeの応用

qmakeの応用 - サブプロジェクト

- TEMPLATE = subdirs
- SUBDIRS　対象となるディレクトリのリスト記述
- CONFIG
  - ordered
    SUBDIRS定義順に辿る

SUBDIR対象のディレクトリ値にはプロパティで依存プロジェクトを設定できる

```
SUBDIRS = app lib
app.depend = lib
```

qmakeの応用 - サブプロジェクト例

- ファイルツリー
  - myproj
    - myproj.pro
    - app/
      - app.pro
      - main.cpp
    - lib/
      - lib.pro
      - lib.pri
      - lib_global.h
      - hello.h
      - hello.cpp

myproj.pro

```
TEMPLATE = subdirs
SUBDIRS  = app lib
app.depends = lib
```

qmakeの応用 - サブプロジェクト例

lib_global.h

```
#pragma once

#include <QtCore/qglobal.h>

#if defined(LIB_COMPILE)
#  define LIBTESTSHARED_EXPORT Q_DECL_EXPORT
#else
#  define LIBTESTSHARED_EXPORT Q_DECL_IMPORT
#endif
```

hello.h

```
#pragma once

#include <QString>
#include "lib_global.h"

LIBTESTSHARED_EXPORT void hello(const QString& msg);
```

qmakeの応用 - サブプロジェクト例

hello.cpp

```
#include "hello.h"
#include <QTextStream>
QTextStream cout(stdout);

void hello(const QString& msg)
{
    cout << "Hello " << msg << endl;
}
```

main.cpp

```
#include "hello.h"

int main(int /* argc */, char* /* argv */ [])
{
    hello("world");
    return 0;
}
```

qmakeの応用 - サブプロジェクト例

lib.pro

```
TEMPLATE  = lib
QT        = core
CONFIG   += console
TARGET    = hello
SOURCES   = hello.cpp
HEADERS   = hello.h
DEFINES  += LIB_COMPILE
VERSION   = 0.0.1
```

lib.pri

```
LIBS += -L$$shadowed($${PWD}) -lhello
INCLUDEPATH += $${PWD}
DEPENDPATH  += $${PWD}
```

app.pro

```
TEMPLATE    = app
SOURCES     = main.cpp
linux:QMAKE_LFLAGS += -Wl,-rpath,\'\$\$ORIGIN/../lib\'
include(../lib/lib.pri)
```

qmakeの応用 - Install

- TARGETのinstallはINSTALLS変数とtarget.pathを使う

lib.pro

```
TEMPLATE  = lib
QT        = core
CONFIG   += console
TARGET    = hello
SOURCES   = hello.cpp
HEADERS   = hello.h
DEFINES  += LIB_COMPILE
VERSION   = 0.0.1
INSTALLS += target
target.path = /usr/local/lib
```

app.pro

```
TEMPLATE    = app
SOURCES     = main.cpp
linux:QMAKE_LFLAGS += -Wl,-rpath,\'\$\$ORIGIN/../lib\'
include(../lib/lib.pri)
INSTALLS += target
target.path = /usr/local/bin
```

qmakeの応用 - Install

- 追加のファイルインストールを記載することもできます

readme

```
app for QtJS
```

app.pro

```
TEMPLATE    = app
SOURCES     = main.cpp
linux:QMAKE_LFLAGS += -Wl,-rpath,\'\$\$ORIGIN/../lib\'
include(../lib/lib.pri)
INSTALLS += target docs
target.path = /usr/local/bin
docs.files = readme
docs.path = /usr/local/share/app
```

- Makefileまで「'」や「$$」などを引き継ぎたい場合は、¥でエスケープする必要があります

qmakeの応用 - Functionの追加

- 置換関数やテスト関数を独自に定義できます

```
defineReplace(name) {
    :
    :
    :
}
```

```
defineTest(name) {
    :
    :
    :
}
```

qmakeの応用 - Functionの追加

```
defineReplace(headersAndSources) {
    variable = $$1
    names = $$eval($$variable)
    headers =
    sources =

    for(name, names) {
        header = $${name}.h
        exists($$header) {
            headers += $$header
        }
        source = $${name}.cpp
        exists($$source) {
            sources += $$source
        }
    }
    return($$headers $$sources)
}
```

- headersAndSources()
  - 引数は 1 つのvariable名
  - variable中のリストを basenameとする.hと.cpp のリストを生成する

qmakeの応用 - Functionの追加

```
defineTest(isFiles) {
files = $$ARGS

    for(file, files) {
        !exists($$file) {
            return(false)
        }
    }
    return(true)
}
```

- isFiles()
  - リストを受け取る
  - リストがすべてファイルならtrue
  - ファイル以外があればfalse

qmakeの応用 - Functionの追加の検証

myfunction.pro

```
defineTest(isFiles) {
    :
}
defineReplace(headersAndSources) {
    :
}
TEMPLATE = aux
T1 = test1 test2 test3
message($$headersAndSources(T1))
isFiles(test1.cpp):message("test1.cpp is file")
!isFiles(test1):message("test1 is not file")
```

```
myfunction$ qmake
Project MESSAGE: test1.h test2.h test1.cpp test2.cpp
Project MESSAGE: test1.cpp is file
Project MESSAGE: test1 is not file
```

- ファイルツリー
  - myfunction
    - myfunction.pro
    - test1.cpp
    - test1.h
    - test2.cpp
    - test2.h
    - test2.bak

qmakeの応用 - Featuresの追加

- 複数のプロジェクトに独自の機能を追加できます

myfeature.prf

```
message(myfeature.prf loaded)
```

myfeature.pro

```
TEMPLATE = aux
CONFIG += myfeature
```

```
$ ~/Qt5.5.0/5.5/gcc_64/bin/qmake
Project MESSAGE: loaded myfeature
```

qmakeの応用 - カスタムターゲット

- カスタムのターゲットも可能です

customtarget.pro

```
TEMPLATE = aux

mytarget.target      = testfile
mytarget.commands    = touch $$mytarget.target
mytarget.depends     = mytarget2
mytarget2.commands   = @echo start custom target build
QMAKE_EXTRA_TARGETS += mytarget mytarget2
```

Makefile

```
testfile: mytarget2
        touch testfile

mytarget2:
        @echo start custom target build
```

qmakeの応用 - カスタムコンパイラ

- あるinputからあるoutputを作るためのルール定義

qttranslations/translations/translations.pro

```
updateqm.input = TRANSLATIONS
updateqm.output = $$MODULE_BASE_OUTDIR/translations/${QMAKE_FILE_BASE}.qm
updateqm.commands = $$LRELEASE ${QMAKE_FILE_IN} -qm ${QMAKE_FILE_OUT}
silent:updateqm.commands = @echo lrelease ${QMAKE_FILE_IN} && $
$updateqm.commands
updateqm.name = LRELEASE ${QMAKE_FILE_IN}
updateqm.CONFIG += no_link target_predeps
QMAKE_EXTRA_COMPILERS += updateqm
```

TRANSLATIONS内のリストから、.qmファイルを生成するためのルール

まとめ

- qmakeの概要
  - qmakeとは、簡単な例
- qmakeの基礎
  - コマンド、プロパティ、プロジェクトファイル構成、
- qmakeの応用
  - サブプロジェクト、Install、Functionの追加、Featuresの追加
  - カスタムターゲット、カスタムコンパイル

ご静聴ありがとうございました

http://qt-users.jp/