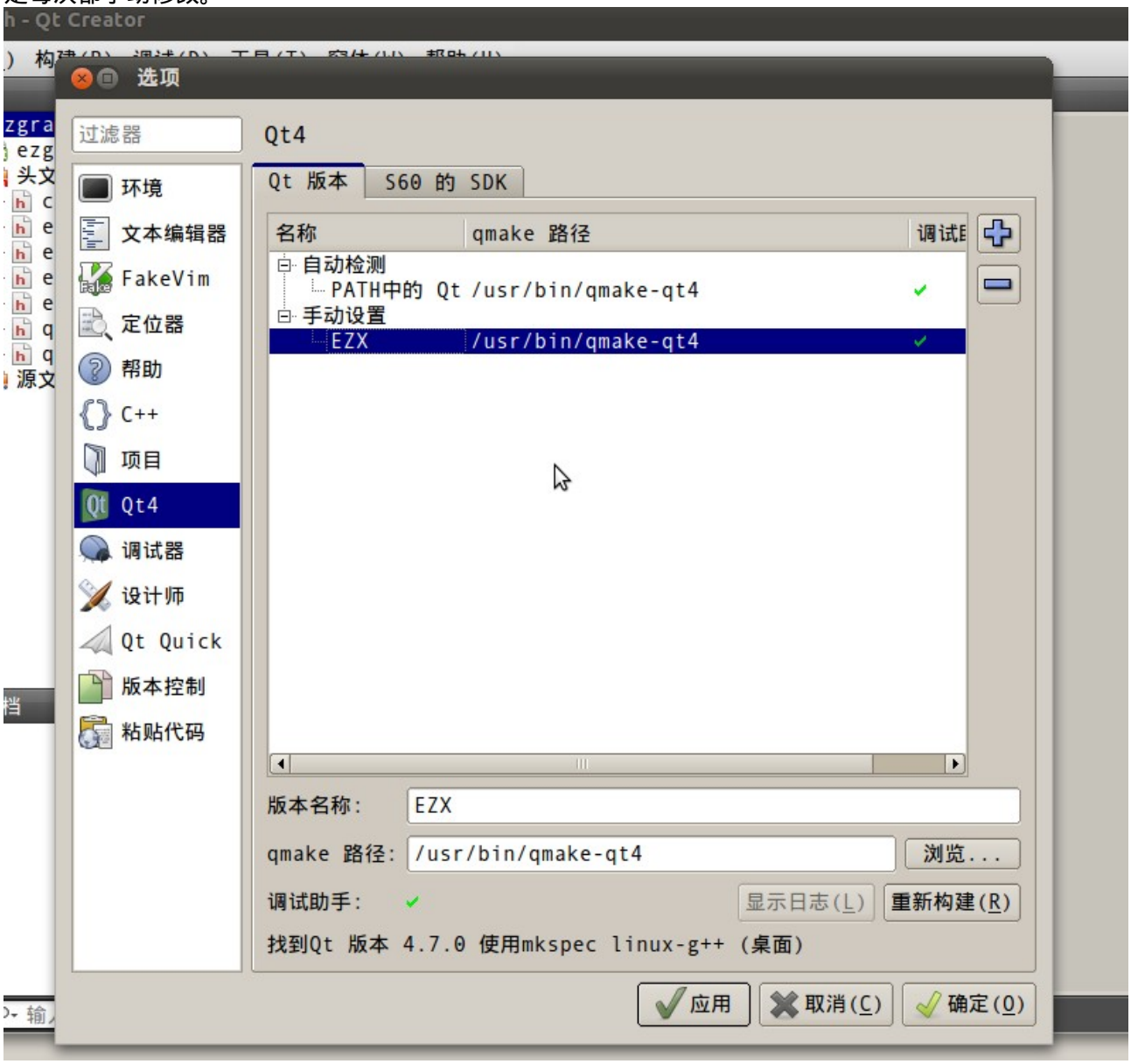
Qt Creator 中增加自定义编译环境

Wang Bin 2011-02-17
wbsecg1@gmail.com

为了使用 Qt Creator 这个图形化的集成开发环境我们甚至可以把交叉编译环境也加入到里面。以我的 motorola 的 EZX（使用 qt2.3.8）开发环境为例，来讲讲操作步骤。

在点击 工具-选项，弹出的对话框中选择 *Qt* 版本，在手动设置中增加一项，版本名称 为 EZX，*qmake* 路径 为 /usr/bin/qmake-qt4。点击确定。注意只能选择 qt4 的 qmake，否则版本无法识别，不能运行 qmake。这一点不知有没有其他方法。这里一个问题是选好 qmake 后默认的 mkspec 为 linux-g++（桌面），而我们希望是交叉编译的 mkspec，不知如何修改。下面的方法是每次都手动修改。



以上步骤完成后会在 左边那栏的 项目-*Qt* 版本里多出一项 *EZX* 。接下来，为了在编译是能正确调用 EZX 的开发环境，我们还要在/usr/share/qt4/mkspecs 中新建一个文件夹 linux-g++-montavista，存放 EZX 编译环境的设置。参考现有的文件比如 linux-g++中的 qmake.conf，我的 qmake.conf 为

```
#
```

```
#
# qmake configuration for linux-g++-montavista
#

MAKEFILE_GENERATOR      = UNIX
TARGET_PLATFORM         = unix #Qt4
TEMPLATE        = app
CONFIG                  += qt warn_off release
#incremental link_prl
QMAKE_INCREMENTAL_STYLE = sublib

#
# qmake configuration for common gcc
#

QMAKE_CC                = $(CCACHE) $(DISTCC) iwmmxt_le-gcc
QMAKE_CFLAGS                = -pipe
QMAKE_CFLAGS_DEPS       = -M
QMAKE_CFLAGS_WARN_ON        = -Wall -W
QMAKE_CFLAGS_WARN_OFF       = -w
QMAKE_CFLAGS_RELEASE = -O2 -mcpu=iwmmxt -mtune=iwmmxt
QMAKE_CFLAGS_DEBUG    = -g
QMAKE_CFLAGS_SHLIB      = -fPIC
QMAKE_CFLAGS_STATIC_LIB      += -fPIC #Qt4
QMAKE_CFLAGS_YACC       = -Wno-unused -Wno-parentheses
QMAKE_CFLAGS_THREAD   = -D_REENTRANT  #Qt4 +=

#Qt4
#QMAKE_CFLAGS_HIDESYMS   += -fvisibility=hidden
#QMAKE_CFLAGS_PRECOMPILE += -x c-header -c ${QMAKE_PCH_INPUT}
-o ${QMAKE_PCH_OUTPUT}
#QMAKE_CFLAGS_USE_PRECOMPILE += -include $
{QMAKE_PCH_OUTPUT_BASE}

QMAKE_CXX               = $(CCACHE) $(DISTCC) iwmmxt_le-g++
QMAKE_CXXFLAGS              = $$QMAKE_CFLAGS  -DQWS -fno-exceptions
-fno-rtti
QMAKE_CXXFLAGS_DEPS   = $$QMAKE_CFLAGS_DEPS
QMAKE_CXXFLAGS_WARN_ON    = $$QMAKE_CFLAGS_WARN_ON
QMAKE_CXXFLAGS_WARN_OFF   = $$QMAKE_CFLAGS_WARN_OFF
QMAKE_CXXFLAGS_RELEASE    = $$QMAKE_CFLAGS_RELEASE
QMAKE_CXXFLAGS_DEBUG= $$QMAKE_CFLAGS_DEBUG
QMAKE_CXXFLAGS_SHLIB  = $$QMAKE_CFLAGS_SHLIB
QMAKE_CXXFLAGS_YACC   = $$QMAKE_CFLAGS_YACC
QMAKE_CXXFLAGS_THREAD       = $$QMAKE_CFLAGS_THREAD  #Qt4 +=

QMAKE_INCDIR        = $(MONTAVISTA)/target/usr/include $
(MONTAVISTA)/target/usr/local/include
QMAKE_LIBDIR        = $(MONTAVISTA)/target/usr/lib $
(MONTAVISTA)/target/usr/lib $(MONTAVISTA)/target/usr/local/lib
QMAKE_INCDIR_X11   = /usr/X11R6/include
QMAKE_LIBDIR_X11   = /usr/X11R6/lib
```

```
QMAKE_INCDIR_QT          = $(QTDIR)/include $(EZXDIR)/include $
(QT_EXTDIR)/include
QMAKE_LIBDIR_QT          = $(QTDIR)/lib $(EZXDIR)/lib $(QT_EXTDIR)/lib
QMAKE_INCDIR_OPENGL    = /usr/X11R6/include
QMAKE_LIBDIR_OPENGL    = /usr/X11R6/lib

QMAKE_LINK             = iwmmxt_le-g++
QMAKE_LINK_SHLIB  = iwmmxt_le-g++
QMAKE_LINK_C         = iwmmxt_le-gcc #Qt4
QMAKE_LINK_C_SHLIB       = iwmmxt_le-gcc #Qt4
QMAKE_LFLAGS         =  #Qt4 +=
QMAKE_LFLAGS_RELEASE  =  #Qt4 +=
QMAKE_LFLAGS_DEBUG     =  #Qt4 +=
QMAKE_LFLAGS_APP  += #Qt4
QMAKE_LFLAGS_SHLIB       = -shared  #Qt4 +=
QMAKE_LFLAGS_PLUGIN    = $$QMAKE_LFLAGS_SHLIB  #Qt4 +=
QMAKE_LFLAGS_SONAME  = -Wl,-soname,  #Qt4 +=
QMAKE_LFLAGS_THREAD   =  #Qt4 +=
QMAKE_RPATH          = -Wl,-rpath,
QMAKE_LFLAGS_RPATH = -Wl,-rpath,
#Qt4: QMAKE_LFLAGS_RPATH

QMAKE_PCH_OUTPUT_EXT    = .gch

# -Bsymbolic-functions (ld) support
QMAKE_LFLAGS_BSYMBOLIC_FUNC = -Wl,-Bsymbolic-functions
QMAKE_LFLAGS_DYNAMIC_LIST = -Wl,--dynamic-list,

#
# qmake configuration for common linux
#

QMAKE_LIBS             =
QMAKE_LIBS_DYNLOAD      = -ldl
QMAKE_LIBS_X11             = -lXext -lX11 -lm
QMAKE_LIBS_X11SM  = -lSM -lICE
QMAKE_LIBS_NIS             = -lnsl
QMAKE_LIBS_QT              =  -lqte-mt -lezxappsdk -lipp-jp -lezxopenwindow
-lipp-miscGen -lezxappbase -lezxjpeg -lezxpm
QMAKE_LIBS_QT_THREAD  = -lpthread -lqte-mt -lezxappsdk -lipp-jp
-lezxopenwindow -lipp-miscGen -lezxappbase -lezxjpeg -lezxpm
QMAKE_LIBS_OPENGL        = -lGLU -lGL -lXmu
QMAKE_LIBS_OPENGL_QT  = -lGL -lXmu
QMAKE_LIBS_THREAD        = -lpthread -lqte-mt -lezxappsdk -lipp-jp
-lezxopenwindow -lipp-miscGen -lezxappbase -lezxjpeg -lezxpm

QMAKE_MOC              = $(QTDIR)/bin/moc
QMAKE_UIC              = $(QTDIR)/bin/uic

QMAKE_AR               = iwmmxt_le-ar cqs
QMAKE_RANLIB         =
```

```
QMAKE_TAR             = tar -cf
QMAKE_GZIP            = gzip -9f

QMAKE_COPY            = cp -f
QMAKE_COPY_FILE        = $(COPY)
QMAKE_COPY_DIR         = $(COPY) -r
QMAKE_MOVE            = mv -f
QMAKE_DEL_FILE              = rm -f
QMAKE_DEL_DIR               = rmdir
QMAKE_STRIP           = iwmmxt_le-strip
QMAKE_STRIPFLAGS_LIB    += --strip-unneeded
QMAKE_CHK_DIR_EXISTS    = test -d
QMAKE_MKDIR           = mkdir -p

#
# qmake configuration for common unix
#

QMAKE_LEX             = flex
QMAKE_LEXFLAGS              +=
QMAKE_YACC            = yacc
QMAKE_YACCFLAGS             += -d
QMAKE_YACCFLAGS_MANGLE  += -p $base -b $base
QMAKE_YACC_HEADER       = $base.tab.h
QMAKE_YACC_SOURCE       = $base.tab.c
QMAKE_PREFIX_SHLIB     = lib
QMAKE_PREFIX_STATICLIB  = lib
QMAKE_EXTENSION_STATICLIB = a
```

在编译前，我们先选好 项目-*Qt* 版本 中的 EZX，然后点击 构建步骤 qmake 那行右边的详情，额外的参数写入 -spec linux-g++-montavista。

最后，在下方的 构建环境变量编辑 QTDIR, PATH 等变量为你交叉编译所需的值，比如交叉编译器的

路径。

这些都做好后，就可以开始用 qt creator 进行交叉编译了。
这种方法有点美中不足的地方，比如每个工程都要在 项目里设置 qmake 的参数和环境变量，qmake 参数中 QMLJSDEBUGGER_PATH 去不掉，生成的 Makefile 会有很多不许要的东西比如 DIST=各种 prf 文件。