密级状态：绝密（　　）　秘密（　　）　内部资料（　）　公开（ √ ）

RK3399 Linux Demo 应用说明

（技术部，第二系统产品部）

文件状态： [ ] 草稿 [√] 正式发布 [ ] 正在修改	文件标识：	RK3399 Linux Demo 应用说明
	当前版本：	V1.0
	作　　者：	陈锦森
	完成日期：	2017-01-16
	审　　核：	蓝斌元
	完成日期：	2017-01-16

福州瑞芯微电子有限公司

Fuzhou Rockchips Semiconductor Co . , Ltd

文档修改记录：

日期	修订版本	修订内容	修改人	核定人
2017-01-16	V1.0	初始版本。	陈锦森	蓝斌元

目录

1. 3399Linux 应用开发环境介绍

1.1 开发语言简介 Qt

3399LinuxDemo 应用使用 Qt 作为开发语言，Qt 是 1991 年奇趣科技开发的一个跨平台的 C++图形用户界面应用程序框架。它提供给应用程序开发者建立艺术级的图形用户界面所需的所有功能，同时良好的可扩展性也决定了 Qt 程序的可发展性。目前 Qt 支持的操作系统有:Windows、嵌入式 Linux、MS/Windows - 95、98、NT4.0、ME、2000、XP、Vista 等，这意味着使用 Qt 您只需一次性开发应用程序，无须重新编写源代码，便可跨不同桌面和嵌入式操作系统部署这些应用程序。

Qt 主要功能与特征如下：

① 直观的 C++ 类库：模块化 Qt C++ 类库提供一套丰富的应用程序生成块 (block)，包含了构建高级跨平台应用程序所需的全部功能。具有直观，易学、易用，生成好理解、易维护的代码等特点。

② 跨桌面和嵌入式操作系统的移植性：使用 Qt，您只需一次性开发应用程序，就可跨不同桌面和嵌入式操作系统进行部署，而无须重新编写源代码，可以说 Qt 无处不在（QtEverywhere） 。

③ 使用单一的源代码库定位多个操作系统；

④ 通过重新利用代码可将代码跨设备进行部署；

⑤ 无须考虑平台，可重新分配开发资源；

⑥ 代码不受担忧平台更改影响的长远考虑 ；

⑦ 使开发人员专注于构建软件的核心价值，而不是维护 API 。

⑧ 具有跨平台 IDE 的集成开发工具：Qt Creator 是专为满足 Qt 开发人员需求而量身定制的跨平台集成开发环境 (IDE)。Qt Creator 可在 Windows、Linux/X11 和 Mac OS X 桌面操作系统上运行，供开发人员针对多个桌面和移动设备平台创建应用程序。

⑨ 在嵌入式系统上的高运行时间性能，占用资源少。

1.2 开发工具简介 Qt Creator

Qt Creator 是一个用于 Qt 开发的轻量级跨平台集成开发环境。与 Qt 语言相辅相成，Qt Creator 能够跨平台运行，目前支持的系统包括 Linux(32 位及 64 位)，Mac OS X 和 Windows。Qt Creator 的设计使得开发人员能够利用 Qt 这个应用程序框架更加快速和轻易的开发任务。

另外，目前比较主流的 Qt 开发方式除了 Qt Creator IDE，还能利用 Visual Studio+Qt 进行便携的开发，在 Visual Studio 中下载安装 Qt 插件，能很容易的部署起 Qt 的开发环境。本套 Demo 程序开发环境为 Qt Creator。

Qt Creator 的主要特征包括：

① 复杂代码编辑器：Qt Creator 的高级代码编辑器支持编辑 C++ 和 QML (JavaScript)、上下文相关帮助、代码完成功能、本机代码转化及其他功能。

② 版本控制：Qt Creator 汇集了最流行的版本控制系统，包括 Git、Subversion、Perforce、CVS 和 Mercurial。

③ 集成用户界面设计器：Qt Creator 提供了两个集成的可视化编辑器：用于通过 Qt widget 生成用户界面的 Qt Designer，以及用于通过 QML 语言开发动态用户界面的 Qt Quick Designer*。

④ 项目和编译管理 ：无论是导入现有项目还是创建一个全新项目，Qt Creator 都能生成所有必要的文件。包括对 cross-qmake 和 Cmake 的支持。

⑤ 桌面和移动平台：Qt Creator 支持在桌面系统和移动设备中编译和运行 Qt 应用程序。通过编译设置您可以在目标平台之间快速切换。

⑥ Qt 模拟器：Qt 模拟器是诺基亚 Qt SDK 的一部分，可在与目标移动设备相似的环境中对移动设备的 Qt 应用程序进行测试。

2. Qt 应用开发及 Demo 程序说明

2.1 Qt 应用开发需知

2.1.1 关于 Qt 的核心构件—qmake 与 pro 配置

—qmake

Qmake 随着 Qt 的安装而被安装，在 Qt 中 qmake 扮演着一个极为重要的作用。

我们知道，手写 Makefile 是比较困难并且容易出错的，尤其是需要给不同的平台和编译器组合写几个 Makefile。使用 qmake，开发者创建一个简单的“项目”文件并且运行 qmake 生成适当的 Makefile，这个项目配置文件我们称之为 pro 文件。qmake 会注意所有的编译器和平台的依赖性，可以把开发者解放出来只关心他们的代码。Trolltech 公司使用 qmake 作为 Qt 库和 Qt 所提供的工具的主要连编工具。

qmake 也注意了 Qt 的特殊需求，可以自动的包含 moc 和 uic 的连编规则。

了解到上面的规则之后，在系统工程中编译应用(而不是 Qt Creator 中编译)时，我们可以先调用系统工程中的 qmake 工具生成 Makefile 文件，而后执行 make 命令编译应用即可。

—pro 配置

每个 Qt 应用程序在创建的时候都会随之生成一个 pro 配置文件，根据上面的解释，我们知道 qmake 会根据这个配置文件生成与具体平台相关的 Makefile 参与编译。在具体的 Qt 程序开发过程中，我们常常需要对 pro 文件进行一系列繁琐的配置，现归纳总结一些常用的配置变量如下：

1. 模板变量—TEMPLATE

模板变量告诉 qmake 为这个应用程序生成哪种 makefile，如 TEMPLATE = app

下面是可供使用的选择：

A> app -建立一个应用程序的 makefile。这是默认值，所以如果模板没有被指定，这个将被使用。

B> lib - 建立一个库的 makefile。

C> vcapp - 建立一个应用程序的 VisualStudio 项目文件。

D> vclib - 建立一个库的 VisualStudio 项目文件。

E> subdirs -这是一个特殊的模板，它可以创建一个能够进入特定目录并且为一个项目文件生成 makefile 并且为它调用 make 的 makefile。

2. 指定生成的应用程序信息—DESTDIR 与 TARGET

例如：

```
DESTDIR += ../bin                    // 指定生成的应用程序放置在../bin 目录下
```

TARGET = pksystem　　// 指定生成的应用程序包名字为 pksystem

3. 指定配置信息—CONFIG

CONFIG 用来告诉 qmake 关于应用程序的配置信息。

例如：CONFIG+= qt warn_on release

在这里使用“+=”，是因为我们添加我们的配置选项到任何一个已经存在中。这样做比使用“=”那样替换已经指定的所有选项是更安全的。

A> qt 部分告诉 qmake 这个应用程序是使用 Qt 来连编的。这也就是说 qmake 在连接和为编译添加所需的包含路径的时候会考虑到 Qt 库的。

B> warn_on 部分告诉 qmake 要把编译器设置为输出警告信息的。

C> release 部分告诉 qmake 应用程序必须被连编为一个发布的应用程序。在开发过程中，程序员也可以使用 debug 来替换 release

4. 指定执行 qmake 生成的编译文件的存储位置—UI_DIR、MOC_DIR、OBJECTS_DIR 与 RCC_DIR

例如：

UI_DIR += forms: 将.ui 文件 qmake 转换生成的 ui_*.h 文件存储在 forms 目录下

RCC_DIR += res: 将 Qt 的资源文件.qrc 转换生成的 qrc_*.h 存储在 res 目录下

MOC_DIR += moc : 含 Q_OBJECT 的头文件转换成标准.h 文件的存放在 moc 目录

OBJECTS_DIR += obj: 指定目标文件(obj)的存放在 obj 目录下

5. 指定程序编译时依赖的相关路径—DEPENDPATH

DEPENDPATH += . forms include qrc sources

6. Qt 文件包含路径

INCLUDEPATH += .　　// 除 Qt 应用目录外所包含路径

HEADERS += include/painter.h　　// 工程中包含的头文件

FORMS += forms/painter.ui　　// 工程中包含的.ui 设计文件

SOURCES += sources/main.cpp sources/painter.cpp　// 工程中包含的源文件

RESOURCES += qrc/painter.qrc　　// 工程中包含的资源文件

另外如：

LIBS += -L folderPath　//引入的 lib 文件的路径　-L：引入路径

Release:LIBS += -L folderPath // release 版引入的 lib 文件路径

Debug:LIBS += -L folderPath // Debug 版引入的 lib 文件路径

DEFINES += XX_XX_XXX //定义编译选项，在.h 文件中就可以使用 ：#ifdefine xx_xx_xxx

RC_FILE = xxx.icns

2.1.2 关于 Qt 的用户界面

日常开发中，Qt Creator 中编写应用程序有二种比较常用的方式：

1. 使用 Qt Designer 设计界面

采用 Qt Designer，使得快速创建用户界面成为可能。在 Qt Designer 环境中，所有的操作都采用可视化的操作，可拖放控件、关联信号与槽、设置特定控件的属性、调整特定控件的位置等，在以上的基础上再补充一些 css 效果，一个轻松的界面就构建出来了。

使用 Qt Designer 设计界面使得效果就在自己的掌控之中，可行与否一眼就能看的出来，且比以下手工书写界面代码的方式要快速一些。

Qt 中使用 Qt Designer 设计界面生成的界面文件为.ui 后缀。

Qt 控件引入.ui 文件一种简单的方式为：Qt 控件类继承 Ui::WpaGui(WpaGui 为.ui 界面文件名)，最后 setup(this)即可成功引入.ui 文件。

2. 手工书写界面代码

使用手工创建代码时，需要从 Qt 已有的 GUI 类库中选择一个类作为基类继承，并且添加必要的其它成员。通常，我们会选择从 QDialog、 QWidget、QMainWindow 等类中选择一个作为主窗体；然后创建其它的控件，并使用布局管理器布局这些控件；最后将该布局设置为主窗体的布局（setLayout 函数）。

使用代码书写界面的好处是可以比较容易运用一些布局对整体的界面布局进行宏观的调控，设置比例等，对应用机器分辨率不明确的应用程序有独到的好处。

2.1.3 关于 Qt 信号与槽的机制

信号槽机制与 Windows 下消息机制类似，消息机制是基于回调函数，Qt 中用信号与槽来代替函数指针，使程序更安全简洁。

信号和槽机制是 Qt 的核心机制，可以让编程人员将互不相关的对象绑定在一起，实现对象之间的通信。

—信号

当对象改变其状态时，信号就由该对象发射 (emit) 出去，而且对象只负责发送信号，它不知道另一端是谁在接收这个信号。这样就做到了真正的信息封装，能确保对象被当作一个真正的软件组件来使用。

—槽

用于接收信号，而且槽只是普通的对象成员函数。一个槽并不知道是否有任何信号与自己相连接。而且对象并不了解具体的通信机制。

信号与槽的连接

所有从 QObject 或其子类（例如 Qwidget）派生的类都能够包含信号和槽。因为信号与槽的连接是通过 QObject 的 connect() 成员函数来实现的。

connect(sender, SIGNAL(signal), receiver, SLOT(slot));

其中 sender 与 receiver 是指向对象的指针，SIGNAL() 与 SLOT() 是转换信号与槽的宏。

2.2 Qt Demo 程序说明

2.2.1 网络管理

—无线网络管理

在 Demo 应用程序中，无线网络管理的书写基础 Wpa_supplicant 书写。Wpa_supplicant 是 Linux 下无线连接的管理工具，我们可以用之来进行无线网络的配置。Linux 系统中已经自动集成了该软件。

在 Qt Demo 应用程序的./Setting/ wpa_supplicant 目录下为 Wpa_supplicant 的开源代码，目前已经集成在 Qt Demo 程序中，因此可以这么说，Qt Demo 应用只是提供一个友好的用户界面，交互的操作由 Qt5 完成，而底下实际的功能是由 wpa_supplicant 完成。

在 Demo 源码中 wpa 相关的操作都集成在 wapManager.cpp 类中，包括无线网络的开关、Wifi 数据的获取、无线网络的配置等等操作。要理解程序首先应该了解 wap_supplicant 的一些接口与使用。

首先要知道怎么样与后台的 wpa_supplicant 进行交互，wpa_supplicant 本身就提供了一套 C/C++的接口，供外面程序调用，接口的头文件为 wpa_ctrl.h，在 WpaManager 类的 cpp 文件中，可以清楚的看见 include 的头文件里，有#include “common/wpa_ctrl.h”。

在 wpa_ctrl.h 头文件中，包含一组宏定义的事件消息和 8 个函数接口。由于 wpa_supplicant 交互的方式是基于数据报文的，通过向外界发送事先定义好的事件消息，而外面这根据这些事件消息来确定下步要执行的动作。

譬如：#define WPA_EVENT_CONNECTED “CTRL-EVENT-CONNECTED ”

当程序从 wpa_supplicant 获得这个宏定义消息后，就可以确定已经连接上确定的 Wifi 网络了，而在获得消息后，外面程序则可以使用 8 个函数接口来操作 wpa_supplicant 的行为。

而这些函数接口为：

struct wpa_ctrl * wpa_ctrl_open(const char *ctrl_path);

void wpa_ctrl_close(struct wpa_ctrl *ctrl);

int wpa_ctrl_request(struct wpa_ctrl *ctrl, const char *cmd, size_t cmd_len,
char *reply, size_t *reply_len,
void (*msg_cb)(char *msg, size_t len));

int wpa_ctrl_attach(struct wpa_ctrl *ctrl);

int wpa_ctrl_detach(struct wpa_ctrl *ctrl);

int wpa_ctrl_recv(struct wpa_ctrl *ctrl, char *reply, size_t *reply_len);

int wpa_ctrl_pending(struct wpa_ctrl *ctrl);

int wpa_ctrl_get_fd(struct wpa_ctrl *ctrl);

wpa_ctrl_open 接口用来打开 wpa_supplicant 的控制接口，在 UNIX 系统里使用 UNIX domain sockets，而在 Windows 里则是使用 UDP sockets，当然接口的路径并不是固定的，可以根据配置文件内的路径设置来改变。

wpa_ctrl_close 接口自然是用于关闭控制接口。

wpa_ctrl_request 接口是用来发送控制命令至 wpa_supplicant，并且会接受命令成功执行与否的反馈消息。这是一个堵塞的动作，一般会至少等待 2 秒钟用来接受反馈的回复消息。如果有未经主动请求的消息接受，堵塞的时间则会更长。

wpa_ctrl_attach 接口是为控制接口注册一个事件监视，但注册成功后就可以开始接口事件消息。

wpa_ctrl_detach 接口则是取消控制接口的事件监视。

wpa_ctrl_recv 接口是在控制接口的事件监视注册成功后，用来接受事件消息，这是一个堵塞的操作，当没有可用的消息时，就会一直堵塞。

wpa_ctrl_pending 接口是用来检测是否有即将到来的事件消息。

wpa_ctrl_get_fd 接口则是来获得控制接口的文件描述符号。

2.2.2 音视频播放器

在 Qt5 中，废止了原本的多媒体框架 phonon，转而引入 QMediaPlayer，因此在 Demo 程序中用 QMediaPlayer 进行音视频播放器的开发。

QMediaPlayer 是一个多媒体流框架，其中封装了具体平台播放多媒体相关的操作，例如在 Linux 中使用 GStreamer 框架，而 Windows 中则不然，使用 QMeidaPlayer 能忽略平台的相关性，而 Qt 的平台无关性不谋而合。

在应用中使用 QMediaPlayer，除了需要添加必要的头文件之外，还需要在.pro(Qt 的工程配置文件)添加 QT += multimedia.

3. 应用编译步骤(参考)

在工程代码下，Demo 应用程序的路径在 app/carmachine 下，可以直接运行 rk_make.sh 脚本进行编译，编译生成可执行文件 Carmachine。

脚本文件编译的步骤与下面的步骤相似，自己编写的 Qt 应用可以按照下面的步骤进行编译。

① 项目源码拷贝到工程路径下：如

~/3399Linux/buildroot/output/build/qt5multimedia-5.6.1-1/examples/multimediawidgets 下准备参与编译

② 修改 Qt 项目的 pro 文件，添加 target.path 指定库文件的安装路径，如：

```
71
72 target.path = $$[QT_INSTALL_EXAMPLES]/multimediawidgets/Carmachine
73 INSTALLS += target
74
75
```

其中 Carmachine 为项目源码的名称，这句话将项目直接安装在项目的源码路径下。

注意：pro 文件与 Qt 下的编译工具 qmake 生成与各平台相关的 makefile 文件。

③ 在项目源码目录下执行 qmake 并打包 make

qmake 工具可以使用 buildroot/output/usr/bin 目录下的 qmake 工具

具体步骤如:(项目源码目录下)

/home/cjs/3399Linux/buildroot/output/host/usr/bin/qmake&&make -j8

如果未出现编译错误,根据 pro 文件的配置，在源码路径下可找到编译生成的可执行文件。

应用测试: 可将可执行文件烧写到文件系统中，可烧写在 buildroot/output/target/bin 下.

串口中执行安装命令：

如 cd bin&&./Carmachine -platform linuxfb -plugin EvdevTouch:/dev/input/event0

说明: 以前 Qt4 的程序在嵌入式 Linux 平台运行时，需要在命令行输入-qws 选项以启动 Qt 窗口系统服务，如" ./HelloWorld -qws"；而使用 Qt5 后，不再需要-qws，而需要通过-platform 来指定要使用的 QPA 插件，如" ./HelloWorld -platform linuxfb"，如果不指定，则用默认的 QPA 插件，默认的 QPA 插件通过上面的 QT_QPA_PLATFORM 变量指定。

说明：-plugin EvdevTouch:/dev/input/event0 指定触摸事件