旷明应用开发指南

部门	
文档编号	
版 本 号	V0.0.1
作　　者	

更新记录

日期	更新人	版本	备注
2024/12/14	QUA	V0.0.1	初版

目录

1、 引言

1.1 编写目的

本文旨在让客户快速了解旷明的儿童相机解决方案，并快速上手基于旷明的儿童相机 SDK 做产品化开发。

1.2 预期读者和阅读建议

本文档可提供给客户、研发人员、技术支持工程师和测试工程师使用。

1.3 缩略术语

词语	解释
SDK	Software Development Kit
XOS	旷明统一操作系统
BSP	板级支持包
QuaMM	旷明多媒体
QMGUIEngine	QM 图形库

1.4 参考资料

2、 QM Gui

旷明 Gui 引擎是一套优化并兼容 lvgl 的图形开发引擎，已经实现底层的渲染加速，支持复杂图层的高性能显示；

QMGui 支持丰富的 UI 控件，lvgl 图形库的图形库可参考官方文档，有如下控件：

- Base object（基础对象） (lv_obj)
- Arc（圆弧 ） (lv_arc)
- Animation Image （动画图像） (lv_animimg)
- Bar （进度条）(lv_bar)
- Button（按钮） (lv_button)
- Button matrix（矩阵按钮） (lv_buttonmatrix)
- Calendar（日历） (lv_calendar)
- Chart（图表） (lv_chart)
- Canvas（画布）(lv_canvas)
- Checkbox（复选框） (lv_checkbox)
- Drop-down list（下拉列表）(lv_dropdown)
- Image （图象）(lv_image)
- Image button （图像按钮）(lv_imagebutton)
- Keyboard（键盘） (lv_keyboard)
- Label （标签）(lv_label)
- LED（指示灯） (lv_led)
- Line （线条）(lv_line)
- List（列表） (lv_list)
- Lottie (lv_lottie)
- Menu （菜单）(lv_menu)
- Message box（消息框） (lv_msgbox)
- Roller （滚轮）(lv_roller)
- Scale （标尺）(lv_scale)
- Slider （滑动条）(lv_slider)
- Span（富文本） (lv_span)
- Spinbox （微调框）(lv_spinbox)
- Spinner（环形加载器） (lv_spinner)
- Switch （开关）(lv_switch)
- Table（表格） (lv_table)
- Tabview（选项卡） (lv_tabview)
- Text area（文本框） (lv_textarea)
- Tile view（平铺视图） (lv_tileview)
- Window （窗口）(lv_win)

2.1 旷明自定义组件

2.2 旷明已有的基础应用介绍

应用		
设置		
应用管理	App_manager	
应用间消息机制	Am_message	

3、 应用介绍

3.1 多应用

3.1.1 多应用的功能介绍

本章节介绍的多应用管理，仅对多应用产品有用。如果是单界面可以忽略；

多应用管理，是为了解耦应用之间的关系，方便可复用，比如主应用、设置应用、向导应用、视频播放器应用，等等；应用之间可以相互唤醒，可以相互发送消息来处理事件。

3.1.2 多应用的使用方法

任何应用都需要在 core/apps/src/appsmain.c 添加注册，比如{app_name}_init，主要函数要实现参考：

```
void app_a_init(void) {
    qua_app_ops *ops = malloc(sizeof(qua_app_ops));
    memset(ops, 0x0, sizeof(qua_app_ops));
    ops->on_create = app_a;
    ops->on_destory = app_a_exit;
    ops->on_message = app_a_handle_msg;
    app_manager_register("A", ops);
}
```

3.1.3 多应用的接口

接口名字	接口功能介绍
const char* app_manager_topname(void);	查询顶层的应用名
void app_manager_start(const char *app_name, void *start_data);	开启应用
void app_manager_register(const char *name, qua_app_ops *ops);	注册应用

bool app_manager_name_exist(const char *name);	查询应用是否已注册
void app_manager_exit(const char *app_name);	退出应用

3.1.4 应用的生命周期函数

如下声明周期函数可以在注册配置：
创建：on_create
销毁：on_destroy
唤起：on_wakeup
进后台：on_sleep

3.2 通用设置

3.2.1 设置的功能介绍

旷明提供的设置是比较通用的设置应用，如果符合产品定义，建议直接使用，可以支持用 json 来配置 UI；支持的控件有：开关、选项（适用于语言选择）、日期选择、功能条（比如查看版本号）；

3.2.2 设置的使用方法

配置功能宏：CONFIG_XOS_USE_APP_SETTING
设置文件的存放地址：product/{product_name}/xxx_setting.json
设置的 json 可以有多份，在启动设置时，传入 json 即可。
设置的配置修改后，参数会自动保存到 json 文件；

设置修改后，会获得一系列的更新事件：

初始化设置对象	menu_param xxx_setting_param = {0};
	xxx_setting_param.json_path = k_path_xxx_setting;
	menu_info* menu = menu_init(&xxx_setting_param);

读取设置项的值	char setting_value[60] = { 0 }; menu_get_item_value(menu, "module_xxx_item", setting_value);
添加设置项的回调	在上面的 menu_param 初始化时添加回调函数： param.callback = preview_setting_callback;

3.2.3 设置的 JSON 配置

关键参数介绍：

itemId	表示该设置项的 id
displayName	多语言 ID
Type	控件类型： Optionmenu：选项框 Button：按钮

Options	可选值
optionIndex	当前选择的值
oriIndex	初始值

3.2.3.1 设置项参考

如下是个设置项的配置参考：配置两条设置项：
拍照的大小和连拍的张数

```
{
  "version" : "0.1",
  "menus": [
    {
      "itemId": "camera_photo_pixel",
      "displayName": "ID_CAMERA_PHOTO_SIZE",
      "type": "optionmenu",
      "options":  [
        "ID_CAMERA_2M",
        "ID_CAMERA_4M",
        "ID_CAMERA_6M",
        "ID_CAMERA_8M",
        "ID_CAMERA_16M",
        "ID_CAMERA_32M",
        "ID_CAMERA_64M"
      ],
      "optionIndex": 0,
      "origIndex": 0
    },
    {
      "itemId": "camera_photo_continuous",
      "displayName": "ID_CAMERA_PHOTO_CONTINUOUS",
      "type": "optionmenu",
      "options":  [
        "ID_CAMERA_1_PIECE",
        "ID_CAMERA_3_PIECES",
        "ID_CAMERA_5_PIECES"
      ],
      "optionIndex": 0,
      "origIndex": 0
    },
}
```

3.2.3.2 系统设置参考

如下是系统设置的参考配置

```
{
```

```
  "version" : "0.1",
  "menus": [
    {
      "itemId": "common_language",
      "displayName": "ID_CAMERA_LANGUAGE",
      "type": "optionmenu",
      "options":  [
        "中文",
        "英文"
      ],
      "optionIndex": 0,
      "origIndex": 0
    },
    {
      "itemId": "common_setting_date",
      "displayName": "ID_DATE_AND_TIME",
      "type": "button"
    },
    {
      "itemId": "common_about",
      "displayName": "ID_CAMERA_ABOUT",
      "type": "button"
    }
  ]
}
```

4、 应用开发简介

4.1 Ubuntu 开发环境搭建

1、安装 Ubuntu linux 系统，版本一般基于最新的 LTS 版本；

2、基础如下依赖，用于编译整个系统

sudo apt-get install -y git-core gnupg flex bison gperf build-essential zip curl zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z-dev ccache libgl1-mesa-dev libxml2-utils xsltproc unzip libncurses5

3、如果报 dtc 错误，安装 sudo apt-get install device-tree-compiler

pip install xlrd==1. 2. 0

备注：应用开发建议使用的 IDE

Visual Studio Code，可以安装插件用于断点调试

4.2 需要关心的目录

SDK 总体的目录结构：

目录名字	说明	
Build	编译脚本	
Product	产品的配置	
Core/apps	应用目录	
Core/res	应用资源	
Core/package/external	第三方应用目录	

4.3 实战

4.3.1 源码位置

在 core/apps/下面创建，比如应用的名字 helloworld，其源码的目录位置 core/apps/helloword

4.3.2 资源位置

core/res/helloword
-skin //存放图片
-ttf //存放字体
-video
-voice //存放语音资源

编译时，这些资源会自动打包到 out/res/下面，在制作镜像烧录后，会在根目录下/res

源码中引用图片的方式:

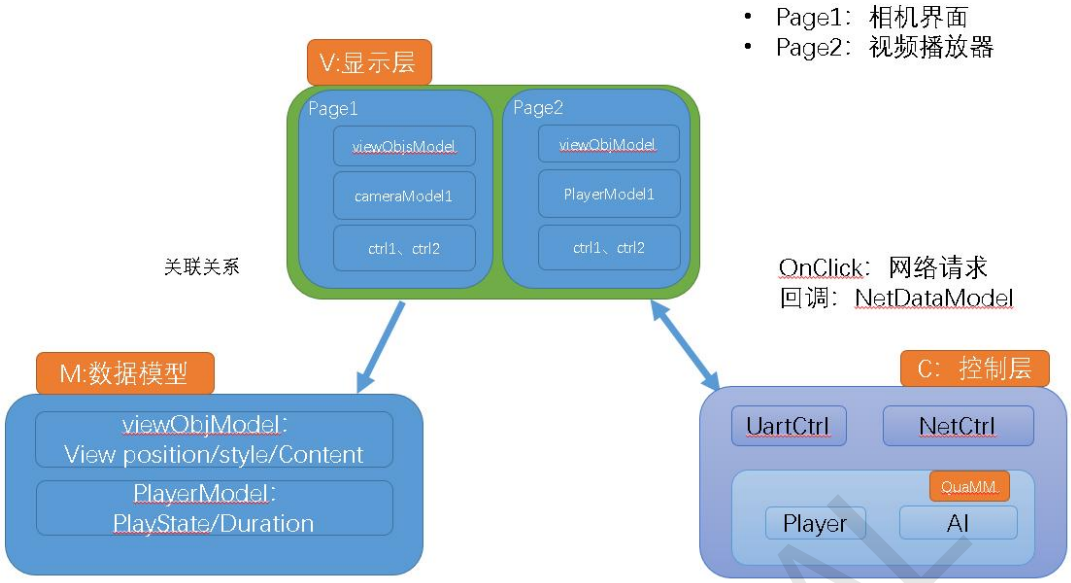
假如图片地址被自动拷贝到设备的对应地址是：/res/helloworld/res/skin/example.png
那么 lv_obj_t *image_obj = lv_image_create(superview);

lv_image_set_src(image_obj, “H:/res/helloworld/res/skin/example.png”)；

其他资源访问的方式类似；

4.3.3 MVC 应用架构

应用源码建议用 MVC 的结构，做下 view 与业务逻辑的分层方便维护；



5、 编译和烧写

5.1 编译应用：

清除：make distclean
配置 defconfig：make project_{product_name}_defconfig
(如果有修改 config 配置就要执行)
编译：make xos

5.2 编译运行于模拟器

./build.sh –S simulator –p helloworld
执行：./out/simulator/xos/bin/qxosui

6、 图形库配置指引

6.1 如何修改屏幕分辨率

1) 在 lv_conf_v9.h 修改宽高：
LV_USE_HOR_SIZE、LV_USE_VER_SIZE

6.2 颜色深度

支持 8 位、16 位、24 位、32 位，配置宏：lv_color_depth

6.3 功能选项

可详细参考 lv_conf.h 的介绍

7、 扩展功能

7.1 多语言的修改

7.1.1 多语言方案简介

旷明 XOS 应用开发使用的多语言字符串，统一放到 strings.xls 文件中进行维护，通过 python 脚本和 zlib 压缩算法生成指定语言的.c 文件和 string id 头文件，运行时根据编译配置的语言进行解压缩处理，实际应用中通过 STRID 来获取对应的实际字符串信息。

7.1.1.1 功能简介

旷明 XOS 应用 string 解决方案能够提供的功能主要包括：

- 多语言字符串支持
  - 支持多语言字符串压缩，比如中、英文
  - 支持通过 excel 表格维护多语言字符串
  - 支持 python 脚本对字符串进行压缩处理

7.1.1.2 系统框架

基于旷明 XOS SDK 的字符串系统框架图如下：

7.1.1.3 相关文档和工具

源文件：

\core\res\inc\los_language_ids.h：定义语言 ID 相关信息。
\core\res\inc\los_string_ids.h：与字符串 ID 相关的定义。
\core\res\src\QuaStrPackage.c：包含对外接口的文件。
\core\res\strings\src\raw_strings_**.c（chinese_cn, english_us,…）：原始字符串的文件。
\core\res\strings\src\raw_language_to_strings_map.c：语言到字符串映射关系的文件。
\core\res\strings\src\zipped_bytes_**.c（chinese_cn, english_us,…）：压缩后的数据文件。
\core\res\strings\src\zipped_language_to_bytes_map.c：压缩后语言到数据映射关系文件。
\tools\ubuntu\strings\trans_strings_res.sh：用于转换字符串资源的脚本文件。
\tools\ubuntu\strings\extract_strings_from_xlsx.py：从 Excel 文件提取字符串的脚本。
\tools\ubuntu\strings\deploy_source_files.sh：部署源文件的脚本。
\core\res\strings\strings_xls\strings.xlsx：存储字符串信息的 Excel 表格文件。

工具：
/tools/ubuntu/strings/extract_strings_from_xlsx.py
/tools/ubuntu/strings/trans_strings_res.sh

7.1.2 使用介绍

7.1.2.1 使用简介

打开 shell 并进入到 XOS 根目录，执行./tools/ubuntu/strings/trans_strings_res.sh 脚本，会在/core/res/strings 目录生成由/core/res/strings/strings.xlsx 维护的字符串相关的压缩数据维护文件。

7.1.2.2 方案原理介绍

字符串资源压缩方案通过 STRINGID 访问字符串，涉及以下文件和操作：

zlib.compress：用于压缩字符串资源。

strings.xlsx：存储字符串信息的表格文件。

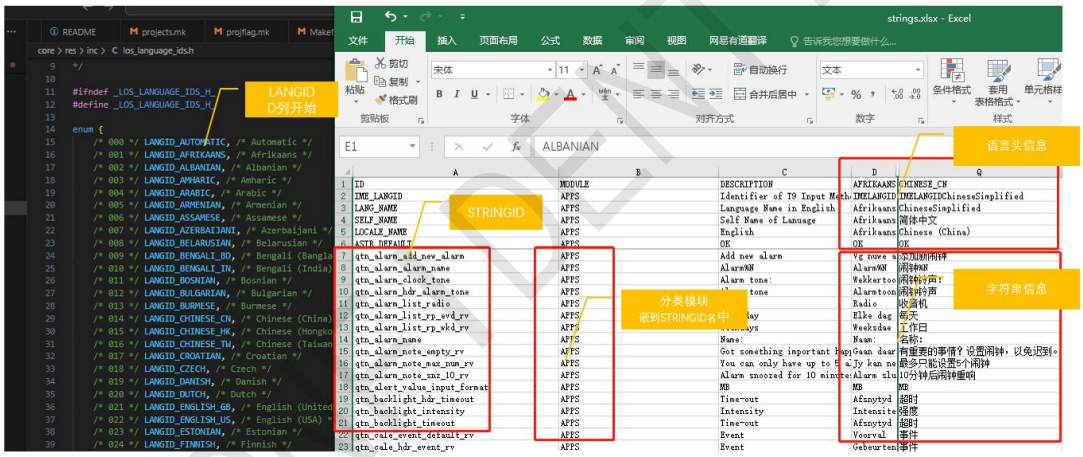
extract_strings.py：用于从表格文件中提取字符串信息。

generate zip*.c &.h：生成压缩后的字符串资源文件。

编译配置使用的语言：在编译过程中配置相关语言选项。

运行解压使用的语言：在运行时解压字符串资源并使用相应语言。

7.1.2.3 字符串表格



字符串通过 STRINGID 参与 XOS SDK 的使用，比如需要设置 label，可参考下面实例：

```
lv_obj_t * frame_content_label = lv_label_create(lv_scr_act());
lv_label_set_text(frame_content_label, GetTextRes(item->item_conts));
```

strings.xlsx 表格包含以下内容：

- 语言头信息： 定义了各种语言的 ID、名称等信息，例如在 \core\res\inc\xos_language_ids.h 文件中定义了枚举类型，包含如 STRID_CHINESE_CN（中文 - 中国）、STRID_ENGLISH_US（英语 - 美国）等语言 ID，以及对应的语言名称、自命名等信息。
- STRINGID：字符串资源的唯一标识符，其命名规则为 “STRINGID” + module name + string name，例如 STRID_APPS_QTN_ALARM_ADD_NEW_ALARM 表示应用程序（APPS）模块中添加新闹钟（QTN_ALARM_ADD_NEW_ALARM）的字符串 ID。
- 字符串信息：包含了不同语言下的字符串文本内容，如中文下 “添加新闹钟”，英文下 “Add new alarm” 等。
- 分类模块：嵌在 STRINGID 名中，用于区分不同的功能模块。

7.1.2.4 相关接口

对外接口在 QuaStrPackage.c 文件中，主要接口如下：

- QBool ResPkgManagerSetCurrentLang (int langId)：设置当前语言的函数，传入语言 ID 作为参数，返回布尔值表示设置是否成功。
- int ResPkgManagerGetCurrentLang (void)：获取当前语言的函数，返回当前设置的语言 ID。
- const char *ResPkgManagerGetLangSelfName (int langId)：根据语言 ID 获取语言的自命名，返回对应的字符串。
- const char * GetTextRes (ResID id)：根据资源 ID 获取对应的字符串资源，返回字符串指针。

7.1.2.5 使用示例

1. 字符串中文解压初始化设置
    - 调用 ResPkgManagerSetCurrentLang 函数设置当前语言为中文（假设中文语言 ID 为 LANGID_CHINESE_CN）。
    - 在需要显示字符串的地方，使用 GetTextRes 函数获取相应的中文字符串资源并显示。
2. 获取当前字符串语言信息
    - 使用 ResPkgManagerGetCurrentLang 函数获取当前设置的语言 ID。
    - 根据获取到的语言 ID，使用 ResPkgManagerGetLangSelfName 函数获取当前语言的自命名，例如“简体中文”。
3. 字符串英文解压初始化设置（类似中文设置步骤）
    - 调用 ResPkgManagerSetCurrentLang 函数设置当前语言为英文（假设英文语言 ID 为 LANGID_ENGLISH_US）。
    - 使用 GetTextRes 函数获取英文字符串资源并显示，如 “Add new alarm”。

7.2 资源替换（比如图片）

7.2.1 存放目录

core/res/{app_name}， 该目录下面可以存放各种资源：
图片：skin
字体：ttf
音频：voice
视频：video

7.2.2 如何引用

代码中需要定义一个路径, 让应用的图形控件去引用该路径：
比如: #define path_background k_path_prefix”bg.png”
代码使用：lv_img_set_src(img_obj, path_background);

7.2.3 资源如何被拷贝到设备

其原理是分为两个步骤：

1、编译时，各个应用的资源会统一拷贝到 out/xos/res/{app_name}下面，

2、每个产品目录（product）有个 pre_mkimg_copy_files.sh 文件，里面会把 out/xos/res 整体拷贝到/res/目录下，制作成镜像；

7.3 系统参数

XOS 采用 param.ini 文件的方式保存系统参数，并提供了读取参数和写入参数的 API。

param.ini 参考文件如下：

```
[sys.devinfo]
product_name                = icamera
vendor                      = quaming
sdk_verion                  = 1.00.01
hw_verion                   = 1.0
b_num                       = 12


[audio]
audio_sample_rate           = 8000


[video]
video_encoder               = h264
```

文件结构解释：

模块名：

[]表示参数所属模块

例如 sys.devinfo

系统参数：

模块名后每一行表示一个系统参数

例如：参数名 product_name，参数值为 icamera

7.3.1 参数模块初始化

参数模块通过 CONFIG_XOS_FWK_PARAM 宏控制，如果项目中需要使用 param 模块，需要打开这个宏。

头文件：

#include “param.h”

程序初始化时需要调用 param_init 接口，从 param.ini 文件中加载系统参数，然后才能调用参数的读取和写入接口。

API 接口	参数描述
int param_init(char *ini_path)	Ini_path:指定 param.ini 文件的路径，可以为空，则默认加载/data/config/param.ini

7.3.2 参数读取接口

调用示例：

#include “param.h”

读取 int 类型参数

int val = param_get_int("sys.devinfo:b_num", 0)

Entry:模块名：系统参数，中间用：连接

API 接口	参数描述
const char *param_get_string(const char *entry, const char *default_val)	Entry:表示参数名 格式为：“模块名:系统参数” default_val：默认值，如果没有查询到该系统参数，则返回默认值
int param_get_int(const char *entry, int default_val)	Entry:表示参数名 格式为：“模块名:系统参数” default_val：默认值，如果没有查询到该系统参数，则返回默认值。

7.3.3 参数写入接口

调用示例：

#include “param.h”

写入 int 类型参数

param_set_int("sys.devinfo:b_num", 2);

API 接口	参数描述
int param_set_int(const char *entry, int val)	Entry:表示参数名 格式为：“模块名:系统参数” Val:整型参数值
int param_set_string(const char *entry, const char *val)	Entry:表示参数名 格式为：“模块名:系统参数” val：字符串

7.3.4 恢复出厂设置

系统默认参数保存在/config/param.ini，首次启动系统将/config/param.ini 文件复制到/data/config/param.ini 目录下，后续系统参数的读写操作都针对/data/config/param.ini 文件。

恢复出厂设置会重新执行上述操作，用/config/param.ini 覆盖/data/config/param.ini。

7.4 调用多媒体

请参考：《QuaMM 多媒体接口使用说明》

7.5 第三方应用添加

请参考：《旷明新建应用指南》