Cordova

Install

Install nodejs and npm

```
sudo apt-get install nodejs npm
```

Install cordova

```
sudo npm install -g cordova
```

cordova need node and ubuntu install nodejs

```
sudo ln -s /usr/bin/nodejs /usr/local/bin/node
```

install cordova-browser from git to have linux support

```
sudo npm install -g  git://git.apache.org/cordova-browser.git
```

Update

```
sudo npm update -g
```

Install Android SDK

Create /etc/etc/profile.d/android.sh or add in .bash*profile <code> export ANDROID*HOME=/lab/android-sdk-linux/ export PATH=$PATH:$ANDROID*HOME/tools:$ANDROID*HOME/platform-tools </code>

Project

Create project

```
cordova create /lab/peper-memory org.csgalileo.peper.memory PeperMemory
```

Add android support

```
cd /lab/peper-memory
cordova platform list
cordova platforms add android
```

```
cordova platforms add browser

# temporary patch
# cp /usr/local/lib/node_modules/cordova-
browser/bin/templates/project/cordova/run ./platforms/browser/cordova/run
```

Add to platforms/browser/cordova/run

```
   case 'linux':
      spawn('chromium-browser', ['--test-type', '--disable-web-security', '-
-user-data-dir=/tmp/temp_chrome_user_data_dir_for_cordova_browser',
project]);
      break;
```

Crosswalk

Optionally it is possible to use crosswalk instead of android webview

Add android platform to project

```
cordova platforms add android
```

Download [https://crosswalk-project.org/documentation/downloads.html|crosswalk]

```
CROSSWALK=/lab/crosswalk/crosswalk-cordova-10.39.235.15-arm
rm -fR platforms/android/CordovaLib/*
cp -a ${CROSSWALK}/framework/* platforms/android/CordovaLib/
cp -a ${CROSSWALK}/VERSION platforms/android/
```

Add to platforms/android/AndroidManifest.xml

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Build both the CordovaLib and xwalk*core*library

```
cd platforms/android/CordovaLib/
android update project --subprojects --path . --target "android-19"
ant clean debug

cd .. # platforms/android/
rm -Rf ant-gen
rm -Rf ant-build
cd ../..
cordova build android
```

Manage project

View project detail

```
cordova info
```

Add plugins

```
cordova plugin add org.apache.cordova.device
cordova plugin add org.apache.cordova.console

# cordova plugin add https://github.com/donaldp24/CanvasCameraPlugin.git &&
cordova prepare
```

Search plugins

```
cordova plugin search <text>
```

Update corbova inside project

```
cordova platform update android
```

Test project

```
cordova emulate browser
```

Release

```
cordova build --release android
```

[http://developer.android.com/tools/publishing/app-signing.html sign and align]

Video autoplay

CordovaApp.java

```
super.init();
super.appView.getSettings().setMediaPlaybackRequiresUserGesture(false);
```

Posizionare un set di src dopo il device ready

```
$('#video1')[0].src =
"http://192.168.2.125:8000/lib/exe/fetch.php?media=contrade:pergari:pergari_
int.webm";
```

GPIO

jni contain NDK project

```
├── Android.mk
├── gpio.c
├── gpio.h
└── jni_gpio.c
```

Rebuild this project running 'ndk-build into jni folde'r

Android.mk

```
LOCAL_PATH := $(call my-dir)


include $(CLEAR_VARS)
LOCAL_MODULE := gpio_test
LOCAL_SRC_FILES := gpio.c
LOCAL_LDLIBS += -llog
LOCAL_CFLAGS += -DEXEC
include $(BUILD_EXECUTABLE)

include $(CLEAR_VARS)
LOCAL_MODULE := libgpio
LOCAL_SRC_FILES := gpio.c jni_gpio.c
LOCAL_LDLIBS += -llog
include $(BUILD_SHARED_LIBRARY)
```

gpio.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <poll.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>

#include "gpio.h"

#define LOG_TAG "GPIO"

#ifndef EXEC
#include <android/log.h>
#define  LOGD(...)
__android_log_print(ANDROID_LOG_DEBUG,LOG_TAG,__VA_ARGS__)
#else
#define LOGD(...) printf(">==< %s >==<
```

```
",LOG_TAG),printf(__VA_ARGS__),printf("\n")
#endif


void on_new_value(int val);
int read_gpio(char *path, void (*callback)(int));

int main(int argc, char **argv){
    LOGD("Hello!\n");
#ifdef HOST
    return 0;
#endif
    if(argc == 2)
        return read_gpio(argv[1], on_new_value);
    else
        LOGD("missing argument path");

    return -1;
}

int read_gpio(char *path, void (*callback)(int)){
    int fd = open(path, O_RDONLY);
    char buf[11];
    int res = 0;

    if(fd == -1){
        perror("error opening file");
        return -1;
    }

    struct pollfd gpio_poll_fd = {
        .fd = fd,
        .events = POLLPRI,
        .revents = 0
    };

    for(;;){
        res = poll(&gpio_poll_fd,1,-1);
        if(res == -1){
            perror("error polling");
            break;
        }

        if((gpio_poll_fd.revents & POLLPRI)  == POLLPRI){
            LOGD("POLLPRI");
            int off = lseek(fd, 0, SEEK_SET);
            if(off == -1) break;
            memset(&buf[0], 0, 11);
            size_t num = read(fd, &buf[0], 10*sizeof(char));
            callback(atoi(buf));
        }
```

```
        if((gpio_poll_fd.revents & POLLERR) == POLLERR){
            //seems always to be true ..
            //LOGD("POLLERR");
        }
    }
    return 0;
}

void on_new_value(int val){
    LOGD("interrupt received, val: %d", val);
}
```

gpio.h

```
#ifndef _GPIO_H_
#define _GPIO_H_
int read_gpio(char *path, void (*callback)(int));
#endif
```

jni*gpio.c <code> #include <jni.h> #include <stdlib.h> /* Header for class nativehelper*NativeGpio / *#include “gpio.h” #define LOGTAG “GPIO” #ifndef HOST #include <android/log.h> #define LOGD(...)* android*log*print(ANDROID*LOG*DEBUG,LOG*TAG,VA*ARGS*) #else #define LOGD(...) printf(“>==< %s >==< ”,LOGTAG),printf(VAARGS),printf(“\n”) #endif #ifndef _Included*nativehelper*NativeGpio #define _Included*nativehelper*NativeGpio #endif #ifdef* cplusplus extern “C” { #endif static void on*new*value(int val); static jmethodID cb*method*id; static jclass cb*class; static jobject cb*object; static JNIEnv *cbsaveenv; / * Class: csgalileo*NativeGpio * Method: readGpio * Signature: (Ljava/lang/String;Lcsgalileo/NativeGpio/GpioInterruptCallback;)V / JNIEXPORT void JNICALL JavacsgalileoNativeGpioreadGpio (JNIEnv env, jclass cls, jstring path, jobject callback){ cbclass = (env)→GetObjectClass(env, callback); if(cbclass == NULL){ LOGD(“callback interface not found”); return; } cbmethodid = (env)→GetMethodID(env, cbclass, “onNewValue”, “(I)V”); if(cbmethodid == NULL){ LOGD(“could not find callback method”); return; } cbobject = callback; cbsaveenv = env; const char *fname = (env)→GetStringUTFChars(env, path, NULL); LOGD(“path is %s”, fname); readgpio((char *)fname, onnewvalue); (env)→ReleaseStringUTFChars(env, path, fname); } static void onnewvalue(int val){ LOGD(“onnewvalue interrupt received, val: %d”, val); (*cbsaveenv)→CallVoidMethod(cbsaveenv, cbobject, cbmethodid, (jint)val); } #ifdef __cplusplus } #endif </code>*