```
/*
 * Stewart Bracken Copyright 2014
 *
 */
#include "ofApp.h"


//--------------------------------------------------------------
void ofApp::setup() {

    ofSetVerticalSync(true);
    ofSetLogLevel("ofxLua", OF_LOG_NOTICE);
    ofSetEscapeQuitsApp(false);

    hasError= false;

    //file browser gui
    gui = new ofxUICanvas();
    ofAddListener(gui->newGUIEvent,this,&ofApp::guiEvent);

    //console gui
    guiConsole = new ofxUICanvas();
    ofAddListener(gui->newGUIEvent,this,&ofApp::guiConsoleEvent);

    ofSetLoggerChannel(ofPtr<ofGUILoggerChannel>(new ofGUILoggerChannel(this)));


    // scripts to run
    scripts.push_back("scripts/dragScript.lua");

    reset_directory_gui();

    currentScript = 0;

    // init the lua state
    lua.init(true);

    // listen to error events
    lua.addListener(this);

    ofGetFrameNum();

    // bind the OF api to the lua state
    lua.bind<ofxLuaBindings>();
```

```
    // run a script
    lua.doScript(scripts[currentScript]);

    // call the script's setup() function
    lua.scriptSetup();

    if(ofIsGLProgrammableRenderer()){
        ofLog()<<"YEA I'm Programmable!"<<endl;
    }
}

//--------------------------------------------------------------
void ofApp::update() {
    // call the script's update() function
    lua.scriptUpdate();
}

//--------------------------------------------------------------
void ofApp::draw() {
    // call the script's draw() function
    lua.scriptDraw();

    if(hasError){
        ofDrawBitmapStringHighlight(error, 9, 9);
    }
}

//--------------------------------------------------------------
void ofApp::exit() {
    // call the script's exit() function
    lua.scriptExit();

    // clear the lua state
    lua.clear();

    delete gui;
    delete guiConsole;
}

//--------------------------------------------------------------
void ofApp::keyPressed(int key) {

    if ( key == OF_KEY_ESC ){
        if ( gui->isVisible() ){
            gui->toggleVisible();
```

```
            gui->clearWidgets();
            guiConsole->toggleVisible();
        }else{
            reset_directory_gui();
            guiConsole->toggleVisible();
        }
    }

    lua.scriptKeyPressed(key);
}

//--------------------------------------------------------------
void ofApp::mouseMoved(int x, int y) {
    lua.scriptMouseMoved(x, y);
}

//--------------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button) {
    lua.scriptMouseDragged(x, y, button);
}

//--------------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button) {
    lua.scriptMousePressed(x, y, button);
}

//--------------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button) {
    lua.scriptMouseReleased(x, y, button);
}

//--------------------------------------------------------------
// ofxLua error callback
void ofApp::errorReceived(string& msg) {
    ofLogNotice() << "got a script error: " << msg;
    hasError = true;
    error = msg;

    addConsoleMessage(msg);
}

//--------------------------------------------------------------
void ofApp::reloadScript() {
    // exit, reinit the lua state, and reload the current script
    hasError = false;
```

```
    lua.scriptExit();
    lua.init(true);
    lua.bind<ofxLuaBindings>(); // rebind

    //Clear the gui console
    guiConsole->clearWidgets();

    //add the current script path to the lua path so require works correctly
    string fullpath =
ofFilePath::getAbsolutePath(ofToDataPath(scripts[currentScript]));
    string folder = ofFilePath::getEnclosingDirectory(fullpath);
    string new_path("package.path = '");
    new_path.append(folder);
    new_path.append("?.lua;' .. package.path;");
    lua.doString(new_path);

    ofResetElapsedTimeCounter();
    lua.doScript(scripts[currentScript]);
    lua.scriptSetup();
}

void ofApp::add_to_gui(string path){
    ofDirectory dir(path);
    if (!dir.isDirectory())
        return;

    //list all lua files, add gui for these
    dir.allowExt("lua");
    dir.listDir();
    for(int i = 0; i < dir.size(); ++i){
        string lua_file = dir.getPath(i);
        directory_map.insert(pair<string,string>(lua_file, path));
        gui->addButton(lua_file, false);
    }

    //list all directories and recursively appl this func
    dir = ofDirectory(path);
    dir.listDir();
    for(int i = 0; i < dir.size(); ++i){
        add_to_gui(dir.getPath(i));
    }
}

void ofApp::build_directory_gui(){
    string path = "./scripts/";
```

```
    gui->addLabel("./scripts/");
    add_to_gui(path);
}

void ofApp::reset_directory_gui(){
    build_directory_gui();
    gui->setVisible(true);
    gui->autoSizeToFitWidgets();
}

void ofApp::guiEvent(ofxUIEventArgs &e){
    string name = e.widget->getName();
    int kind = e.widget->getKind();
    if ( kind == OFX_UI_WIDGET_BUTTON && e.getButton()->getValue() == 0){
        scripts.clear();
        scripts.push_back(name);
        reloadScript();
    }
}

void ofApp::guiConsoleEvent(ofxUIEventArgs &e){

}

void ofApp::addConsoleMessage(const string& message){
    //guiConsole
    guiConsole->addLabel(message);
}

//--------------------------------------------------
void ofGUILoggerChannel::log(ofLogLevel level, const string & module, const string
& message){
    // print to cerr for OF_LOG_ERROR and OF_LOG_FATAL_ERROR, everything else to
       cout
    ostream& out = level < OF_LOG_ERROR ? cout : cerr;
    out << "[" << ofGetLogLevelName(level, true)  << "] ";
    // only print the module name if it's not ""
    if(module != ""){
        //out << module << ": ";
        return ofLog(level)<<module<<": "<<message;
    }else{
        app->addConsoleMessage(message);
    }
}
```

```
void ofGUILoggerChannel::log(ofLogLevel level, const string & module, const char*
format, ...){
    //TODO: this isn't supported yet by the gui console
    va_list args;
    va_start(args, format);
    log(level, module, format, args);
    va_end(args);
}

void ofGUILoggerChannel::log(ofLogLevel level, const string & module, const char*
format, va_list args){
    //thanks stefan!
    //http://www.ozzu.com/cpp-tutorials/tutorial-writing-custom-printf-wrapper-funct
ion-t89166.html
    FILE* out = level < OF_LOG_ERROR ? stdout : stderr;
    fprintf(out, "[%s] ", ofGetLogLevelName(level, true).c_str());
    if(module != ""){
        fprintf(out, "%s: ", module.c_str());
    }
    vfprintf(out, format, args);
    fprintf(out, "\n");
    //TODO: this isn't supported yet
}
```