

ofxApp

Features (1)

- Startup / Loading Screen with progress (ofxStateMachine)
- CMS Asset download/cache/checksum management (ofxSimpleHttp, ofxAssets, ofxTagSystem)
- CMS JSON content loading & failover recovery (ofxMTJsonParser)
- CPU & GPU profiler (ofxTimeMeasurements)
- Dynamic texture load / unload (TexturedObject)

Features (2)

- Logging to file & screen (ofxSuperLog)
- Parameter Tweaking (ofxRemoteUI)
- Error Reporting to CMS (ofxSensu)
- Static Textures preloading + access by fileName + live asset edits (ofxAppStaticTextures + ofxAutoTexture)
- Analytics (ofxGoogleAnalytics)
- Fonts (ofxFONTStash)

Features (3)

- TUIO (ofxTuio)
- Screen Management (fullscreen, window, etc) (ofxScreenSetup)
- All behaviors highly configurable from a single config file (ofxJsonSettings)

Dependencies

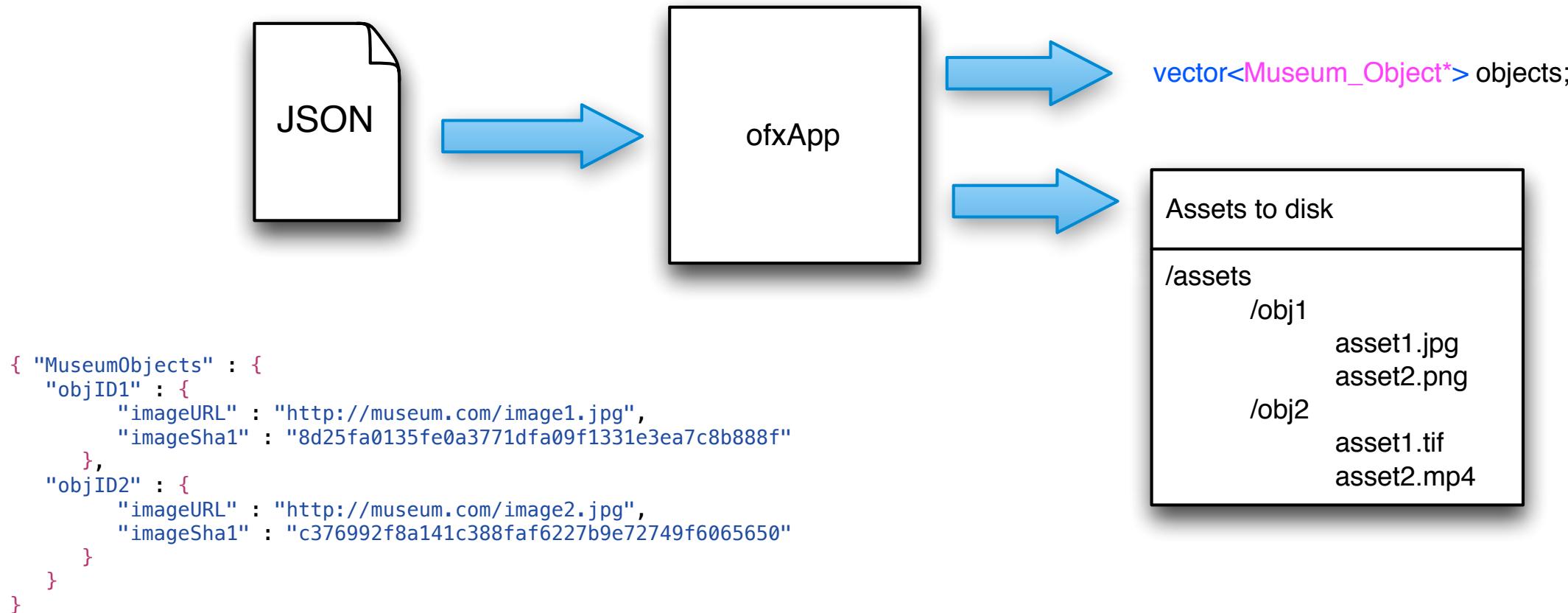
- ofxOSC
- ofxXmlSettings
- ofxFontStash
- ofxTuio
- ofxJson
- ofxSuperLog
- ofxAutoTexture
- ofxMtJsonParser
- ofxSimpleHttp
- ofxTagSystem
- ofxStateMachine
- ofxAssets
- ofxThreadSafeLog
- ofxProgressiveTextureLoad
- ofxTexturedObject
- ofxOpenCV
- ofxSensu
- ofxNetwork
- ofxGoogleAnalytics
- ofxJsonSettings
- ofxTimeMeasurements
- ofxRemoteUI
- ofxHistoryPlot
- ofxScreenSetup
- ofxMullion

Time in State: 0.1 seconds

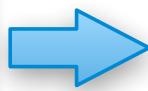
WILL LOAD CONTENT - doing stuff ...

ofxRemoteUIServer: CONNECTED (192.168.43.120)!

Content Abstraction



JSON
N Objects



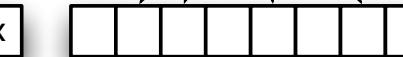
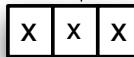
download
to disk



All Objects
in JSON

User Lambda /
Custom Parsing

Rejected*
by User
Lambda

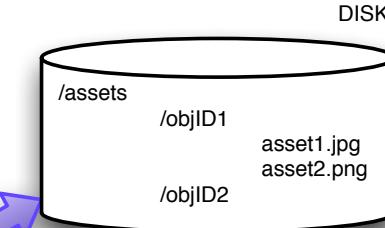


Object Asset
Integrity Check

Missing
Assets
List

http://museum.com/image1.jpg
http://museum.com/image2.jpg
http://museum.com/image3.jpg
http://museum.com/image4.jpg

Valid
Objects



```
{ "MuseumObjects" : {  
    "objID1" : {  
        "imageURL" : "http://image1.jpg",  
        "imageSha1" : "8d25fa0135fe0"  
    },  
    "objID2" : {  
        "imageURL" : "http://image2.jpg",  
        "imageSha1" : "c376992f8a141c38"  
    }  
}
```

File Download

Object Filtering

vector<Museum_Object*> objects;
Objects With
Valid Local Assets

Rejected by
Asset Policies

Content Abstraction

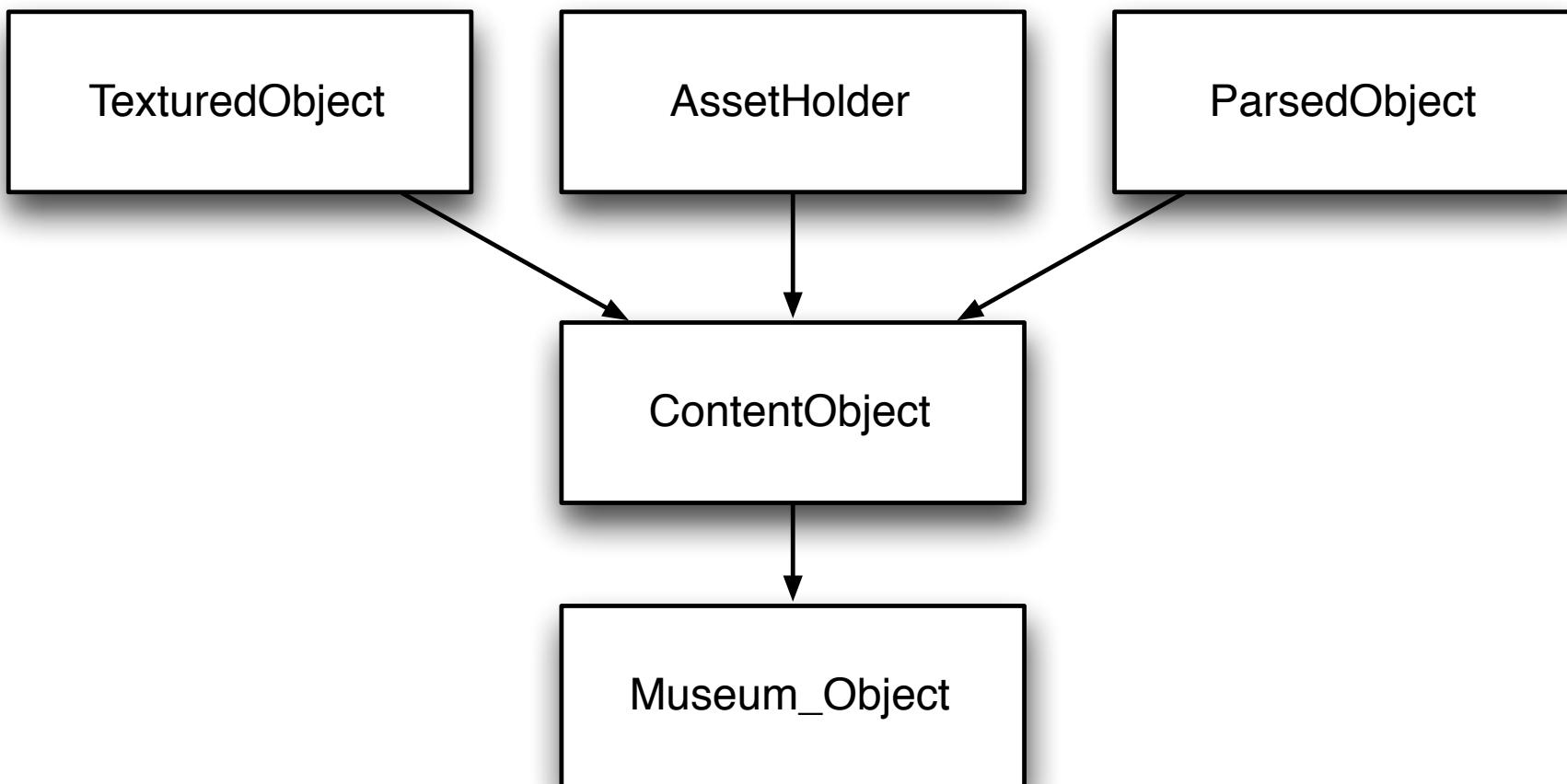
ofxApp needs you to answer a few questions ...

Content Abstraction

- **Where are the Objects?** Locate objects inside the JSON
- **What fields do you need** from each object?
- Is the **object valid?**
- What is the **Object UUID?**
- Does the object have **textures** you want to dynamically load and unload?

```
{ "MuseumObjects" : {  
    "objID1" : {  
        "imageURL" : "http://image1.jpg",  
        "imageSha1" : "8d25fa0135fe0"  
    },  
    "objID2" : {  
        "imageURL" : "http://image2.jpg",  
        "imageSha1" : "c376992f8a141c38"  
    }  
}
```

Content Object

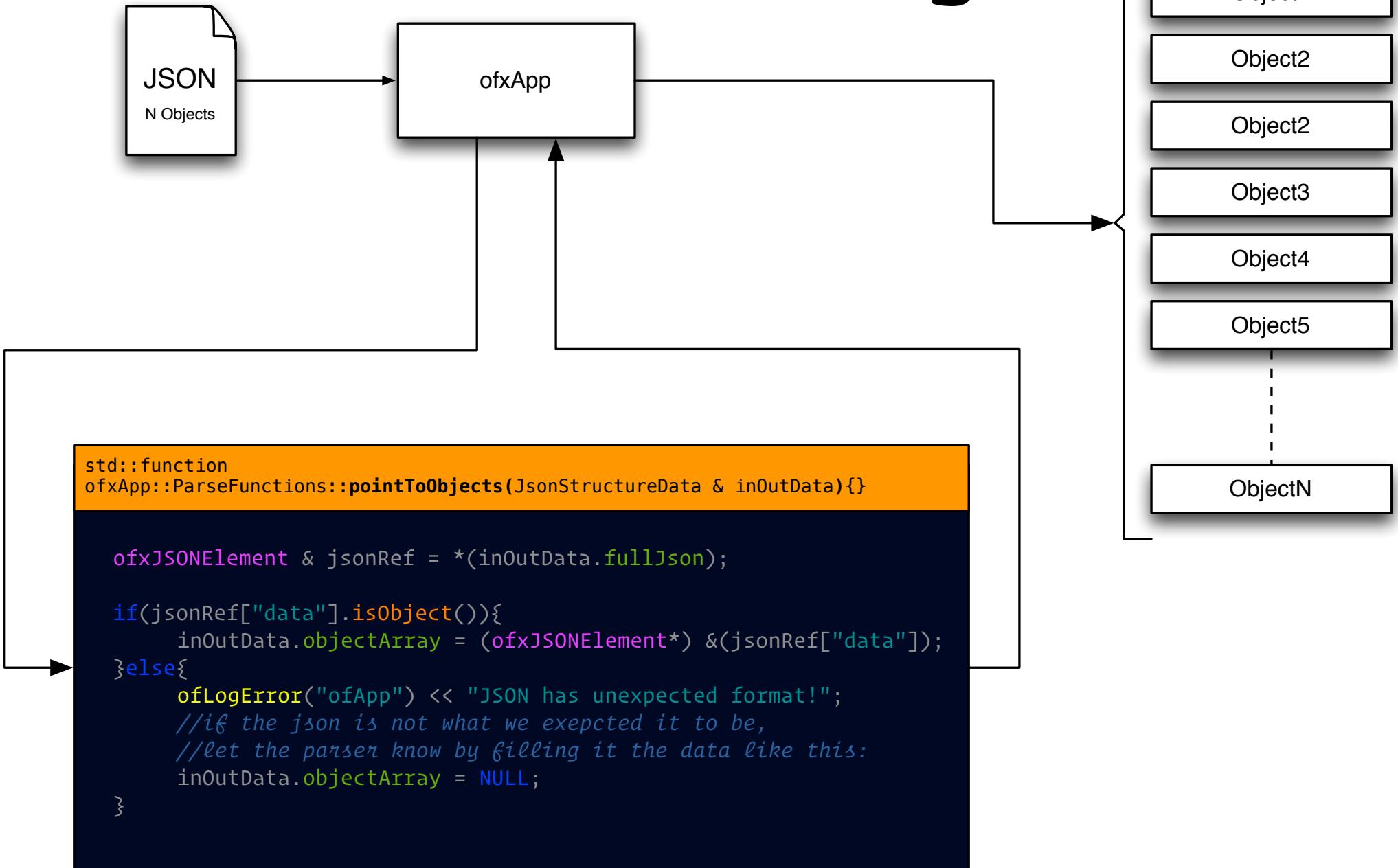


Custom Parsing

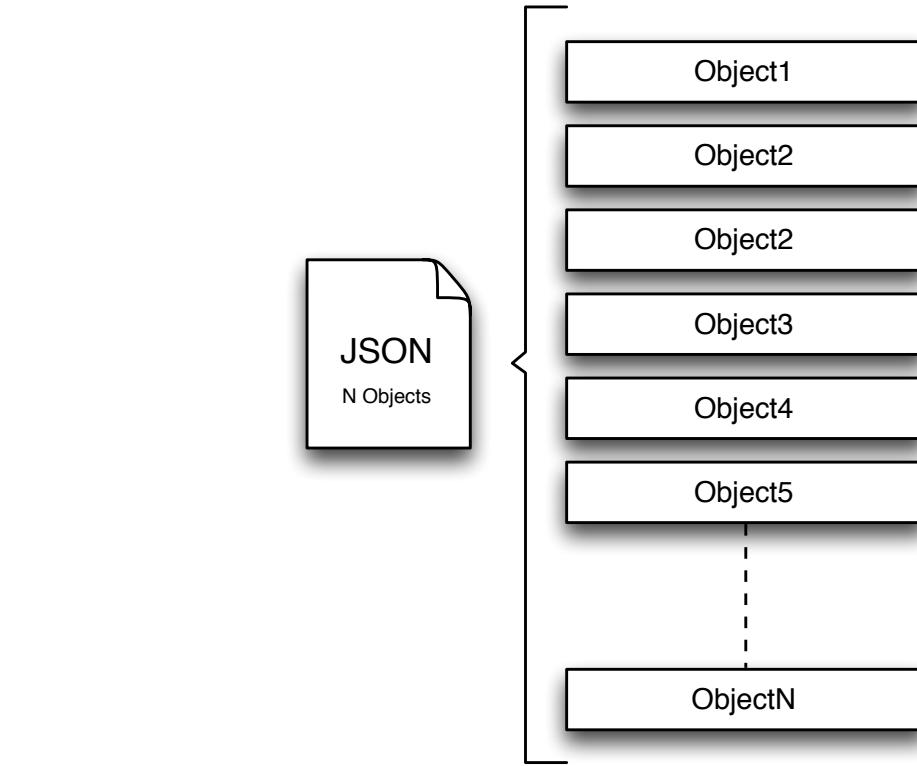
- (3) user lambdas need to be defined per each content source
- (4) if you need dynamic texture loading

```
std::function<void (ofxMtJsonParserThread::JsonStructureData &) > pointToObjects;
std::function<void (ofxMtJsonParserThread::SingleObjectParseData &) > parseOneObject;
std::function<void (ofxApp::CatalogAssetsData &) > defineObjectAssets;
std::function<void (ContentObject*) > setupTexturedObject;
```

Custom Parsing



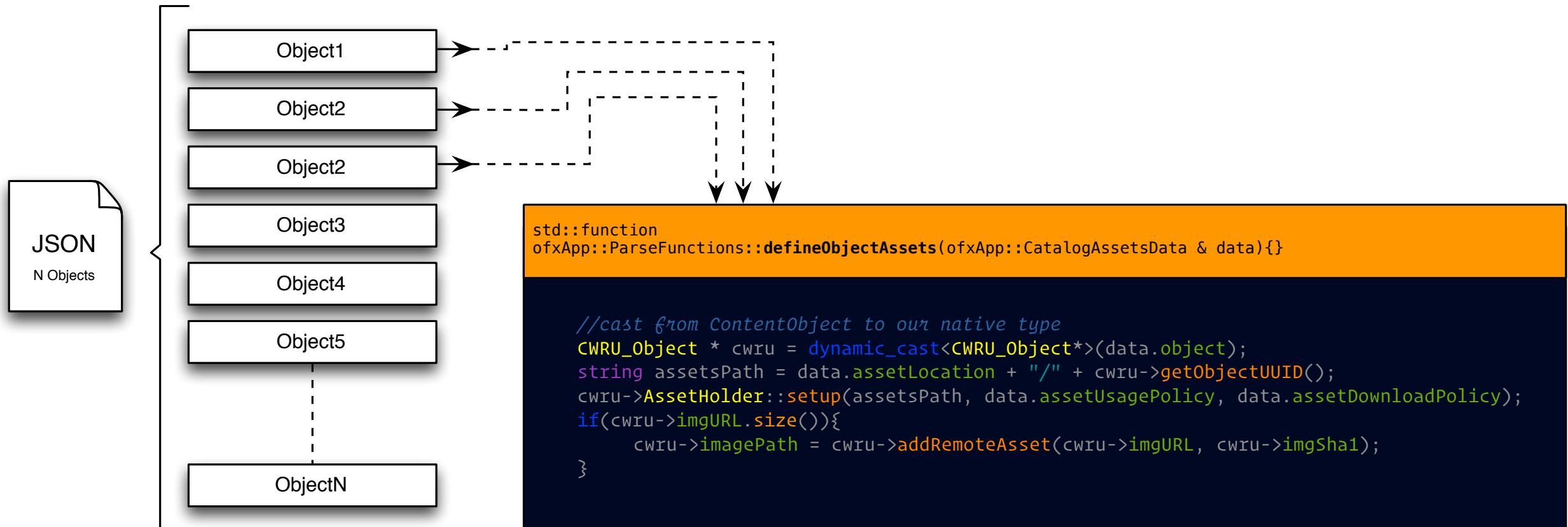
Custom Parsing



```
{  
  "53603249c3fc6b0981d482b4": {  
    "description": "bla bla",  
    "image": {  
      "checksum": "32f20d70f34491c7e7912f1ce385be3b8f6ed89d",  
      "uri": "http://129.22.220.12/asset/images/GTbyj.caldwell4.jpg"  
    },  
    "title": "Alumni Profile: Gregg Gillis",  
  },  
  "53603249c3fc6b0981d482b5": {  
    "description": "bla bla",  
    "image": {  
      "checksum": "f0459c1c6c92d88f9218d0cd52764ecf65f3e77f",  
      "uri": "http://129.22.220.12/asset/images/russo-brothers.jpg"  
    },  
    "title": "Alumni Profile: The Russo Brothers",  
  },  
}
```

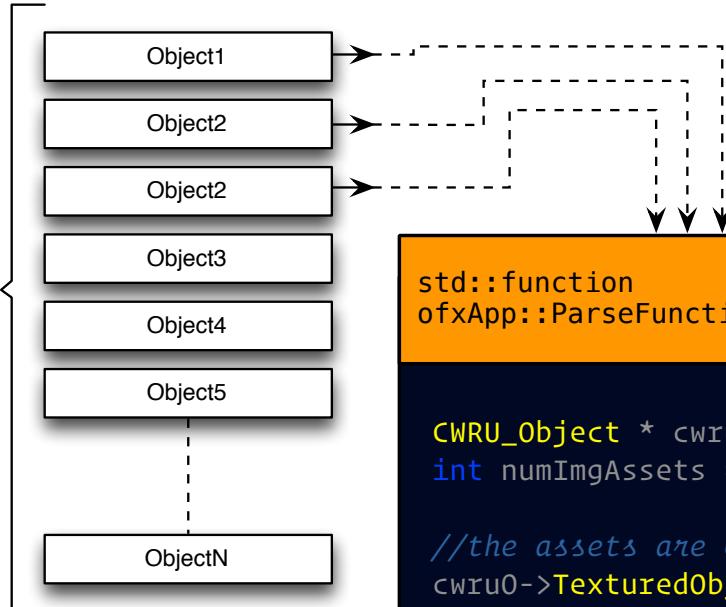
```
std::function  
ofxApp::ParseFunctions::parseOneObject(SingleObjectParseData & inOutData){}  
  
//pointers mess up the json syntax somehow  
const ofxJSONElement & jsonRef = *(inOutData.jsonObj);  
string title, description, imgURL, imgSha1;  
  
try{ //do some parsing - catching exceptions  
    title = jsonRef["title"].asString();  
    description = jsonRef["description"].asString();  
    imgURL = jsonRef["image"]["uri"].asString();  
    imgSha1 = jsonRef["image"]["checksum"].asString();  
}catch(Exception exc){  
    inOutData.printMutex->lock();  
    ofLogError("ofApp") << exc.what() << " " << exc.message() <<  
    " " << exc.displayText() << " WHILE PARSING OBJ " << inOutData.objectID;  
    inOutData.printMutex->unlock();  
}  
  
CWRU_Object * o = new CWRU_Object();  
o->title = title;  
o->description = description;  
o->imgURL = imgURL;  
o->imgSha1 = imgSha1;  
inOutData.objectID;  
//in this case we dont need to set the objectID back to the parser  
//bc this json happens to be a dictionary, not a list... so its  
//smart enough to get it from there.  
  
//this is how we "return" the object to the parser;  
inOutData.object = dynamic_cast<ParsedObject*> (o);
```

Custom Parsing



Custom Parsing

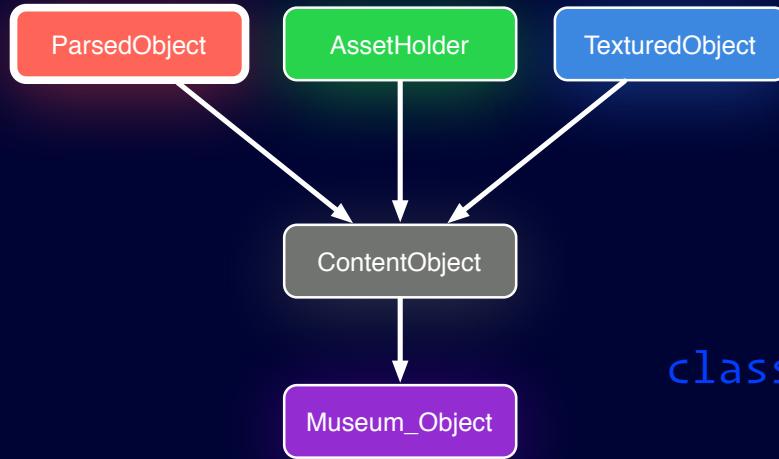
JSON
N Objects



```
std::function  
ofxApp::ParseFunctions::setupTexturedObject(ContentObject* texturedObject){}
```

```
CWRU_Object * cwru0 = dynamic_cast<CWRU_Object*>(texturedObject); //cast from ContentObject to our native type  
int numImgAssets = cwru0->AssetHolder::getNumAssets(); //this will always be 1 for this example, 1 img per object  
  
//the assets are owned by my extended object "AssetHolder"  
cwru0->TexturedObject::setup(numImgAssets, TEXTURE_ORIGINAL); //we only use one tex size, so lets choose ORIGINAL  
cwru0->TexturedObject::setResizeQuality(CV_INTER_AREA); //define resize quality (in case we use mipmaps)  
  
//this example show how to tackle certain problems; would not usually be necessary  
//but we need to check the pixel size of each image for TextureLoader to be able to work;  
//so we do that here.  
for(int i = 0; i < numImgAssets; i++){  
    ofxAssets::Descriptor & d = cwru0->AssetHolder::getAssetDescAtIndex(i);  
  
    switch (d.type) {  
        case ofxAssets::VIDEO: break;  
        case ofxAssets::IMAGE:{  
            auto info = ofxApp::utils::getImageDimensions(d.relativePath);  
            if(info.valid){  
                cwru0->imgSize = ofVec2f(info.width, info.height);  
            }else{  
                ofLogError("TexturedObject") << "cant load image at " << d.relativePath;  
            }break;  
        }  
        default: break;  
    }  
}
```

ParsedObject

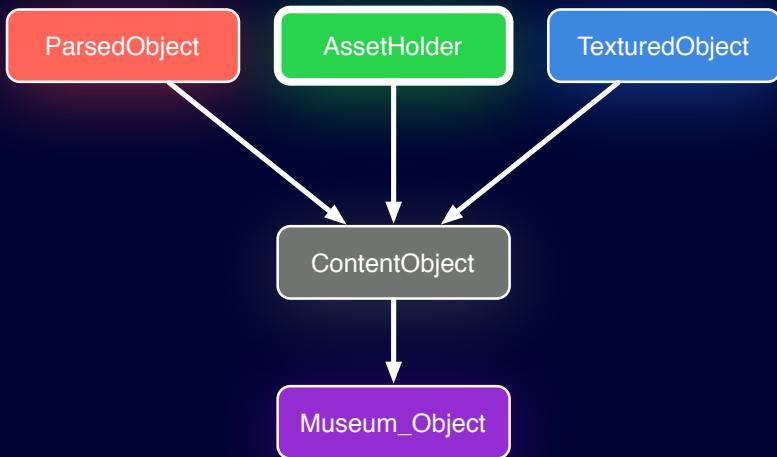


```
class ParsedObject{

public:
    string getObjectUUID(){return uuid;}
    void setObjectUUID(const string & uid){ this->uuid = uid; }

private:
    string uuid; //must be unique!
};
```

AssetHolder



```
class AssetHolder{
public:

    AssetHolder();
    void setup(const string& directoryForAssets,
               const ofxAssets::UsagePolicy & assetOkPolicy,
               const ofxAssets::DownloadPolicy & downloadPolicy);

    string addRemoteAsset(const string& url,
                         const string& sha1,
                         const vector<string>& tags = vector<string>(),
                         ofxAssets::Specs spec = ofxAssets::Specs(),
                         ofxAssets::Type type = ofxAssets::TYPE_UNKNOWN
                         );

    string addLocalAsset(const string& path, //path relative to data!
                         const vector<string>& tags = vector<string>(),
                         ofxAssets::Specs spec = ofxAssets::Specs(),
                         ofxAssets::Type type = ofxAssets::TYPE_UNKNOWN
                         );

    // Access To Assets...
    bool remoteAssetExistsInDB(const string& url);
    bool localAssetExistsInDB(const string& absPath);

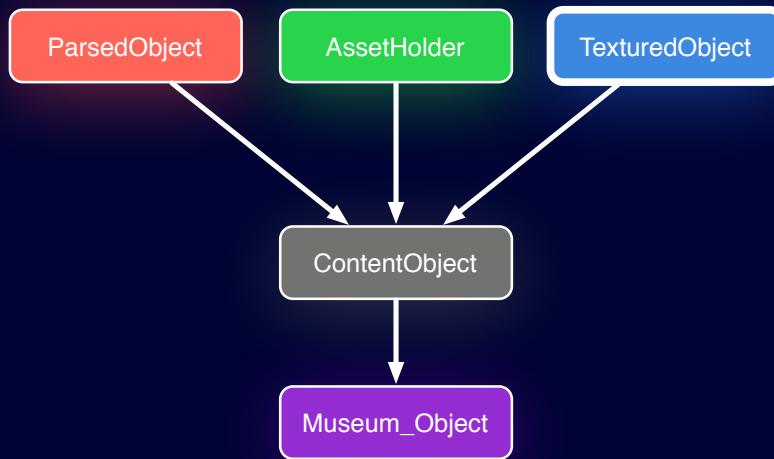
    // ... by key (relative path)
    ofxAssets::Descriptor& getAssetDescForPath(const string& path); //should be a relative path
    ofxAssets::Descriptor& getAssetDescForURL(const string& url);
    vector<ofxAssets::Descriptor> getAssetDescriptorsForType(ofxAssets::Type);

    // ... by Index
    int getNumAssets();
    ofxAssets::Descriptor& getAssetDescAtIndex(int i); //in add order

    // Read/Write custom User info //
    ofxAssets::UserInfo & getUserInfoForPath(const string& path);

    // Tags //
    void addTagsforAsset(const string & relPath, vector<string> tags);
    vector<ofxAssets::Descriptor> getAssetDescsWithTag(const string & tag);
```

TexturedObject



```
class TexturedObject{
public:
    TexturedObject();

    void setup(int numTextures, TexturedObjectSize size);
    bool isSetup(){return hasBeenSetup;}

    void update(float timeNow); //feed me ofGetElapsedTimef()!

// SUBCLASSES MUST IMPLEMENT these methods /////////////////////////////////
virtual ofVec2f getTextureDimensions(TexturedObjectSize s = TEXTURE_ORIGINAL, int index = 0) = 0;
virtual string getLocalTexturePath(TexturedObjectSize s = TEXTURE_ORIGINAL, int index = 0) = 0;

// CONFIG /////////////////////////////////
void setResizeQuality(int resizeQ){ resizeQuality = resizeQ; } // CV_INTER_NN, CV_INTER_LINEAR, CV_INTER_CUBIC

// TEXTURE COMMANDS /////////////////////////////////

ofTexture* requestTexture(TexturedObjectSize s = TEXTURE_ORIGINAL,
                           int index = 0,
                           bool highPriority = false,
                           bool withMipmaps = true
);

void releaseTexture(TexturedObjectSize s = TEXTURE_ORIGINAL,
                    int index = 0,
                    float delaySeconds = TexturedObjectConfig::one().getDefaultTextureUnloadDelay());

// STATUS COMMANDS /////////////////////////////////

bool isLoading(TexturedObjectSize s = TEXTURE_ORIGINAL, int index = 0);
bool isUnloading(TexturedObjectSize s = TEXTURE_ORIGINAL, int index = 0);

bool isReadyToDraw(TexturedObjectSize s = TEXTURE_ORIGINAL, int index = 0);
bool isFullyLoaded(TexturedObjectSize s = TEXTURE_ORIGINAL, int index = 0);
bool isWaitingForCancelToFinish(TexturedObjectSize s = TEXTURE_ORIGINAL, int index = 0);
```

- How are Content Sources defined?

"bin/data/configs/ofxAppSettings.json"

```
"ofxApp":{  
    "Content":{  
        "JsonSources":{  
            "CH" : {  
                "url" : "http://uri.cat/LP/ch_small.json",  
                "jsonDownloadDir": "CH_JsonDownloads",  
                "assetsLocation": "CH_assets",  
                "shouldSkipObjectPolicyTests" : false  
            },  
            "CWRU" : {  
                "url" : "http://uri.cat/LP/cwru.json",  
                "jsonDownloadDir": "CWRU_JsonDownloads",  
                "assetsLocation": "CWRU_assets",  
                "shouldSkipObjectPolicyTests" : false  
            }  
        }  
    }  
}
```

"src/ofxApp.cpp"

```
void ofApp::setup(){  
  
    //create my custom lambdas for parsing objects  
    ofxAppParsers myParsers = ofxAppParsers();  
  
    map<string, ofxApp::ParseFunctions> objectParsers;  
    objectParsers["CWRU"] = myParsers.cwru;  
    objectParsers["CH"] = myParsers.ch;  
  
    //start the ofxApp setup process  
    ofxApp::get().setup(objectParsers, this);  
}
```

- How are Content Sources defined?

“ofxAppParsers.h”

```
class ofxAppParsers{  
  
public:  
  
    ofxAppParsers();  
    ofApp::ParseFunctions cwrU;  
    ofApp::ParseFunctions ch;  
};
```

“ofxAppParsers.cpp”

```
ofxAppParsers::ofxAppParsers(){  
  
    // CWRU ////////////////////////////////  
  
    cwrU.pointToObjects = [](&ofxMtJsonParserThread::JsonStructureData & inOutData){};  
    cwrU.parseOneObject = [](&ofxMtJsonParserThread::SingleObjectParseData & inOutData){};  
    cwrU.defineObjectAssets = [](&ofxApp::CatalogAssetsData & data){};  
    cwrU.setupTexturedObject = [](ContentObject * texturedObject){};  
  
    // CH ////////////////////////////////  
  
    ch.pointToObjects = [](&ofxMtJsonParserThread::JsonStructureData & inOutData){};  
    ch.parseOneObject = [](&ofxMtJsonParserThread::SingleObjectParseData & inOutData){};  
    ch.defineObjectAssets = [](&ofxApp::CatalogAssetsData & data){};  
    ch.setupTexturedObject = [](ContentObject * texturedObject){};  
}
```

ofxAppDelegate

- How does your app communicate with ofxApp?

```
class ofAppDelegate{
public:

    //this will be your entry point to start loading stuff
    virtual void ofAppPhaseWillBegin(ofxApp::Phase) = 0;
    //after you are asked to start loading content, ofxApp will query every frame to check if you are done
    virtual bool ofAppIsPhaseComplete(ofxApp::Phase){return true;} //override this method if you are loading custom content
    virtual void ofAppDrawPhaseProgress(ofxApp::Phase, const ofRectangle & r){}; //override the loading screen drawing
    virtual string ofAppGetStatusString(ofxApp::Phase){return "";} //override the progress bar status text
    virtual float ofAppGetProgressForPhase(ofxApp::Phase){return -1;} //return [0..1] to report progressbar; -1 for indefinite

    //this is how your app gets all the parsed objects - up to you how you store them
    virtual void ofAppContentIsReady(const string & contentID, vector<ContentObject*> ) = 0;

    //tuio callbacks
    virtual void tuioAdded(ofxTuioCursor & tuioCursor){};
    virtual void tuioRemoved(ofxTuioCursor & tuioCursor){};
    virtual void tuioUpdated(ofxTuioCursor & tuioCursor){};

    //screen setup changed callback
    virtual void screenSetupChanged(ofxScreenSetup::ScreenSetupArg &arg){};

};
```

ofxAppDelegate

- You are given opportunities to do custom “setup” while ofxApp starts:

```
enum class Phase{  
    WILL_LOAD_CONTENT, //Before any content is loaded  
    DID_DELIVER_CONTENT, //After content has been delivered  
    WILL_BEGIN_RUNNING //Just before the app starts running  
};  
  
void    ofxAppPhaseWillBegin(ofxApp::Phase);  
bool    ofxAppIsPhaseComplete(ofxApp::Phase);  
  
void    ofxAppDrawPhaseProgress(ofxApp::Phase, const ofRectangle & r);  
string  ofxAppGetStatusString(ofxApp::Phase);  
float   ofxAppGetProgressForPhase(ofxApp::Phase);
```

ofxAppDelegate

- Your ofApp must subclass ofxAppDelegate

```
class ofApp : public ofBaseApp, public ofxAppDelegate{
public:

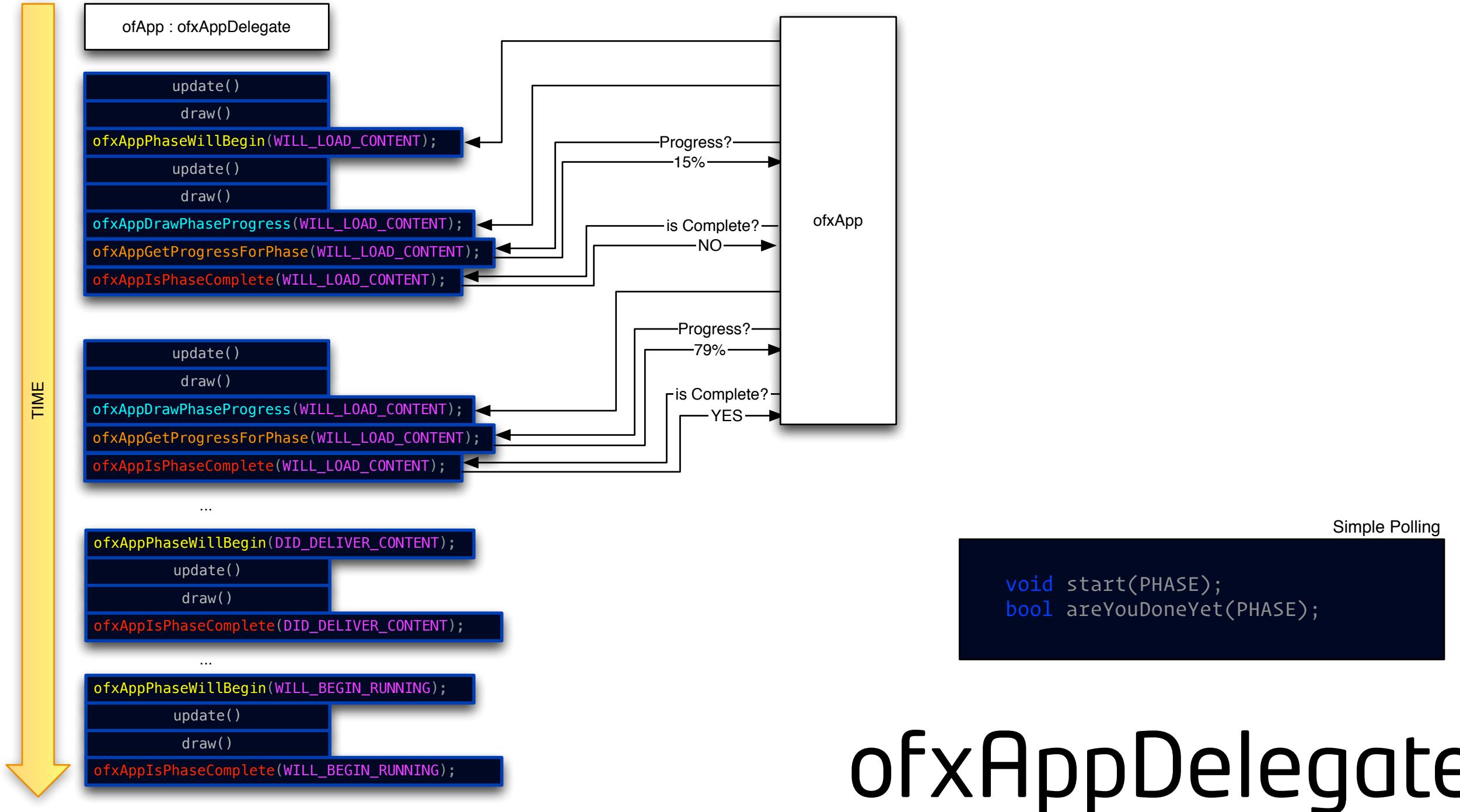
    void setup();
    void update();
    void draw();

    // ofxAppDelegate Callbacks /////////////////////////////////
    void      ofxAppPhaseWillBegin(ofxApp::Phase);
    bool     ofxAppIsPhaseComplete(ofxApp::Phase);

    void      ofxAppDrawPhaseProgress(ofxApp::Phase, const ofRectangle & r);
    string   ofxAppGetStatusString(ofxApp::Phase);
    float    ofxAppGetProgressForPhase(ofxApp::Phase);

    void      ofxAppContentIsReady(const string & contentID, vector<ContentObject*>);

};
```



ofxApp

- How does it take over?

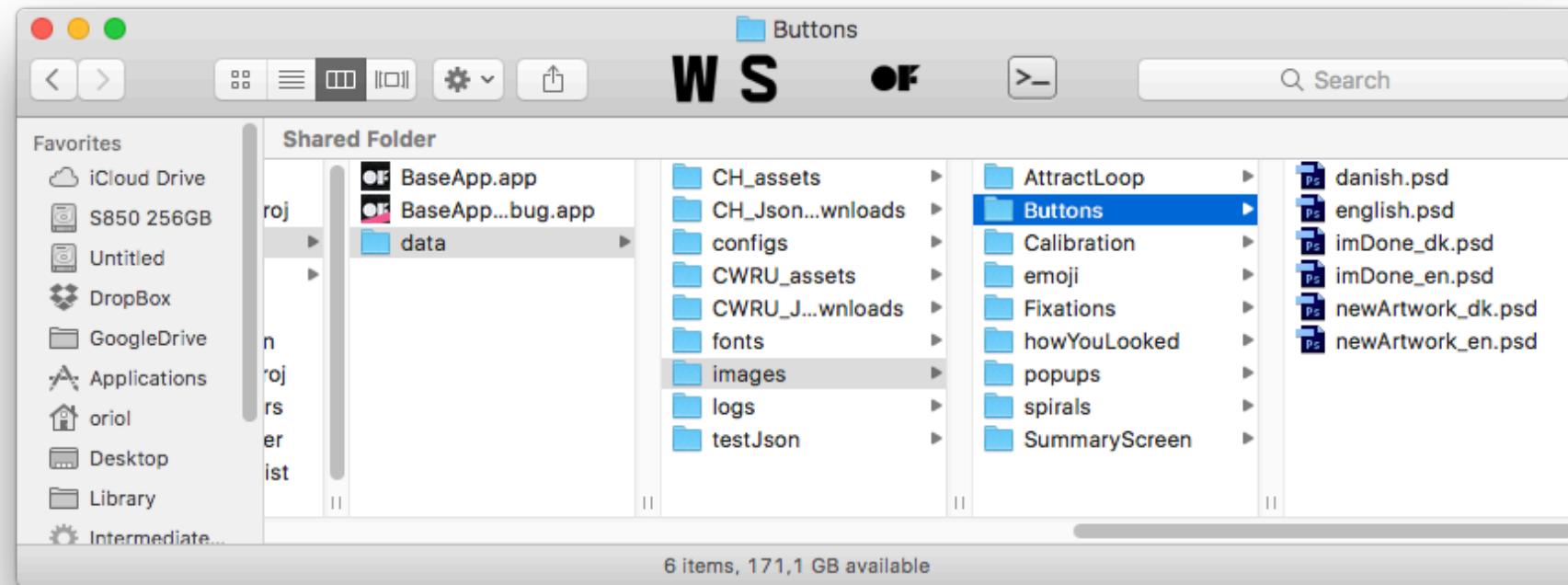
```
void ofApp::setup(){
    ofxApp::get().setup(this); //start the ofxApp setup process
}

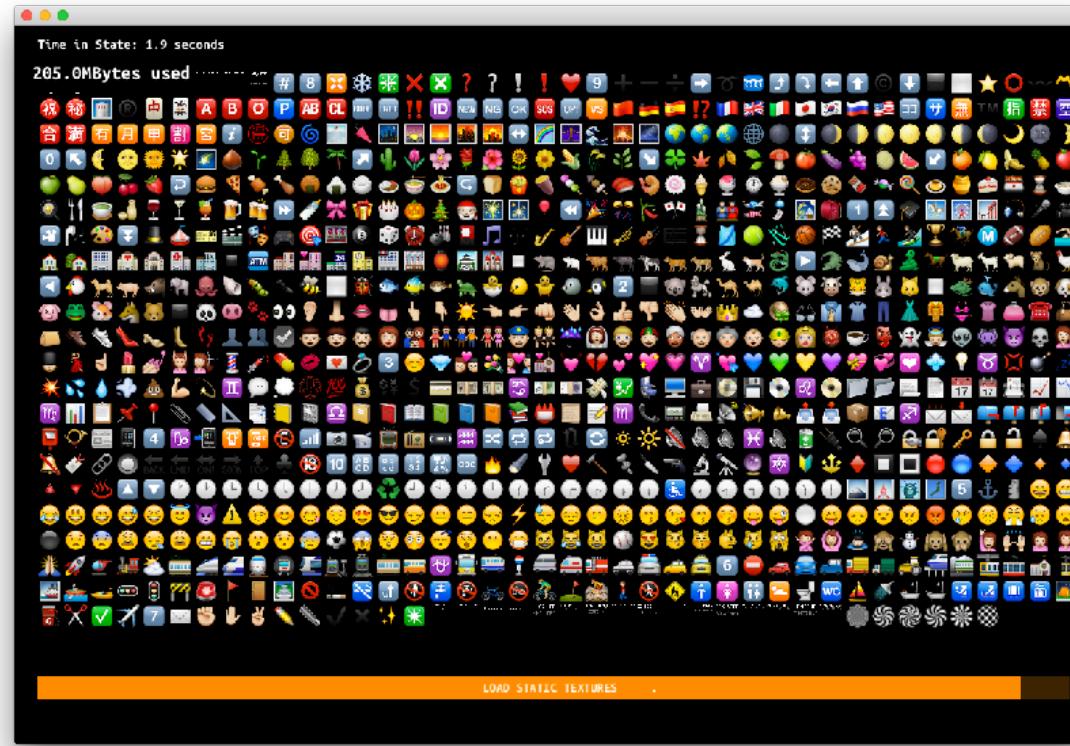
void ofApp::update(){
    if(ofxApp::get().getState() == ofxApp::State::RUNNING){
        //Your App updates here
    }
}

void ofApp::draw(){
    if(ofxApp::get().getState() == ofxApp::State::RUNNING){
        //Your App draws here
    }
}
```

ofxApp::StaticTextures

- Static Images (icons & such)





```
ofxAppStaticTextures: ###### Loaded "images/popups/timedOut.png" #####
ofxAppStaticTextures:   Name:"popups/timedOut" [372x164] Mipmap:0 Format:GL_RGBA8 Mem: 512.00 KB
ofxAppStaticTextures: ###### Loaded "images/popups/takeOffGlasses.png" #####
ofxAppStaticTextures:   Name:"popups/takeOffGlasses" [1080x166] Mipmap:0 Format:GL_RGBA8 Mem: 2.00 MB
ofxAppStaticTextures: ###### Loaded "images/popups/soSorry.png" #####
ofxAppStaticTextures:   Name:"popups/soSorry" [352x162] Mipmap:0 Format:GL_RGBA8 Mem: 512.00 KB
ofxAppStaticTextures: ###### Loaded "images/popups/sitUpStraight.png" #####
ofxAppStaticTextures:   Name:"popups/sitUpStraight" [576x162] Mipmap:0 Format:GL_RGBA8 Mem: 1.00 MB
ofxAppStaticTextures: ###### Loaded "images/popups/pressStart.png" #####
ofxAppStaticTextures:   Name:"popups/pressStart" [1450x180] Mipmap:0 Format:GL_RGBA8 Mem: 2.00 MB
```

ofxApp::StaticTextures

- Mipmaps? `GL_TEXTURE_2D` or `GL_TEXTURE_RECTANGLE_ARB` ?

There are two filename modifiers you can play with:

- `"_t2d"`: the texture should be loaded as `GL_TEXTURE_2D`
- `"_mip"`: the texture should have with `mipmaps`.

For example, files named like this, will receive this treatment:

- `"img.png"` : will load as `GL_TEXTURE_RECTANGLE_ARB` and no mipmaps
- `"img_mip.png"` : will load as `GL_TEXTURE_2D` with `mipmaps`
- `"img_t2d_mip.png"` : will load as `GL_TEXTURE_2D` with `mipmaps`
- `"img_t2d.png"` : will load as `GL_TEXTURE_2D` but no mipmaps will be created

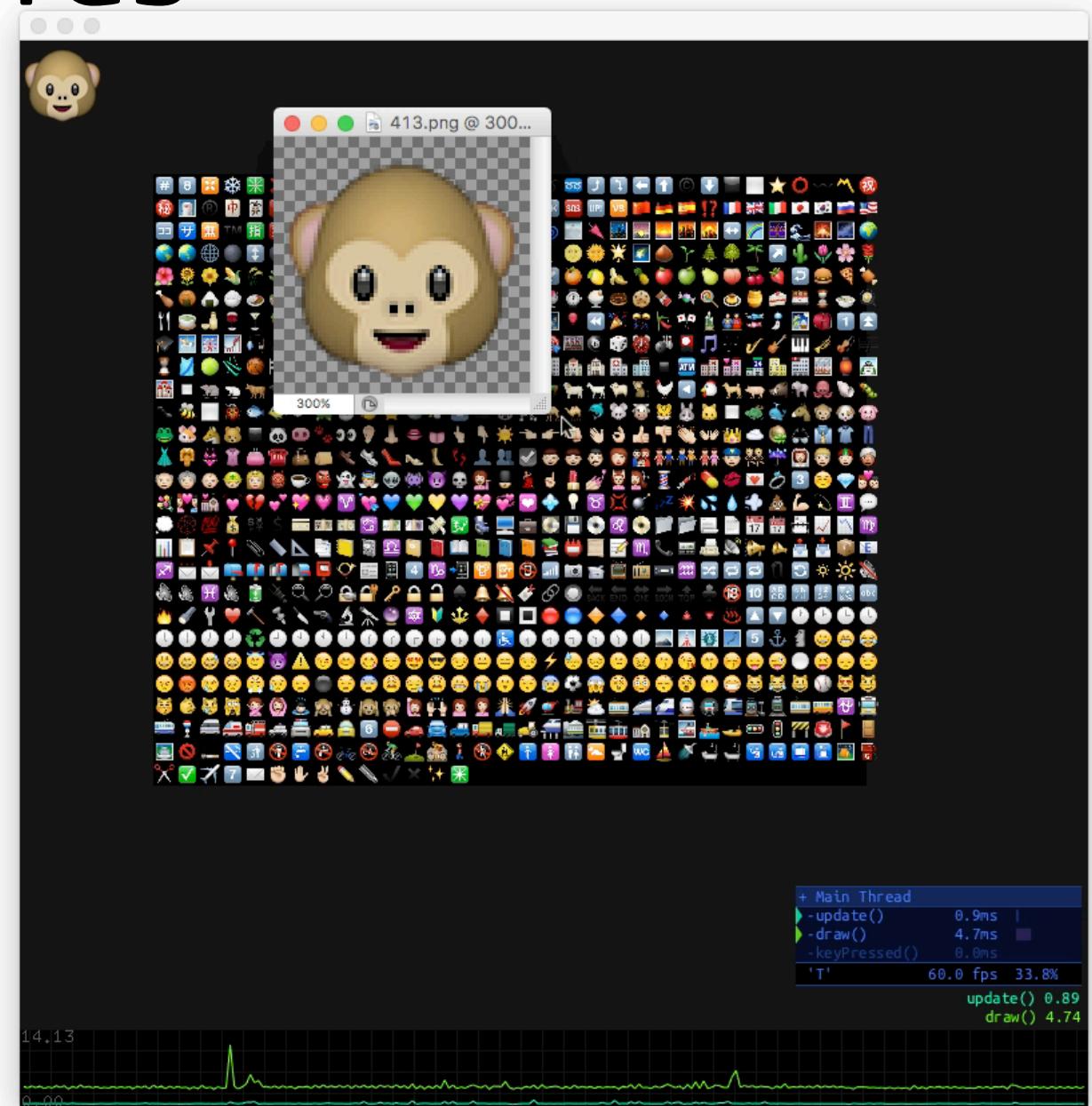
ofxApp::StaticTextures

- How To get them?

```
G_TEX("emoji/413")->draw(0,0);
```

ofxApp::StaticTextures

- Fast Edits: Textures are monitored for file changes; edit as the program runs and you will see live changes
- Live edits can be disabled entirely with a simple pre-processor macro



ofxApp :: Logging

- Readable; indented
- Colorized by module && by Log Level
- Timestamps optional
- File and/or Console and/or Screen
- Optionally synchronised (mutexed)

ofxApp :: Logging

```
[notice ]          ofxApp:  
[notice ]          ofxApp: [REDACTED]  
[notice ]          ofxApp: [REDACTED] GL Info  
[notice ]          ofxApp: [REDACTED]  
[notice ]          ofxApp:  
[notice ]          ofxApp: GL_RENDERER: NVIDIA GeForce GTX 760 OpenGL Engine  
[notice ]          ofxApp: GL_MAX_SAMPLES: 8  
[notice ]          ofxApp: GL_MAX_VIEWPORT_DIMS: 16384 x 16384  
[notice ]          ofxApp: GL_MAX_TEXTURE_SIZE: 16384  
[notice ]          ofxApp: GL_MAX_TEXTURE_UNITS: 8  
[notice ]          ofxApp: GL_MAX_TEXTURE_IMAGE_UNITS: 16  
[notice ]          ofxApp: GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS: 16  
[notice ]          ofxApp: GL_MAX_VERTEX_ATTRIBS: 16  
[notice ]          ofxApp: GL_MAX_VERTEX_UNIFORM_COMPONENTS: 4096  
[notice ]          ofxApp: GL_MAX_VARYING_FLOATS: 124  
[notice ]          ofxApp: GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS: 16  
[notice ]          ofxApp: GL_MAX_TEXTURE_IMAGE_UNITS: 16  
[notice ]          ofxApp: GL_MAX_TEXTURE_COORDS: 8  
[notice ]          ofxApp: GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT: 16  
[notice ]          ofxApp:  
[notice ]          ofxApp: setupRemoteUI()  
[notice ]          ofxRemoteUI: directoryPrefix set to 'configs'  
[notice ]          ofxFONTStash: Loaded font '/Volumes/LP/LP/REPOS/METLIFE_REPO/Sketches/ofxAppExample/bin/data/fonts/UbuntuMono-B.ttf' in texture (512 x 512)  
[notice ]          ofxApp: RemoteUI will save settings on quit: 1  
[notice ]          ofxRemoteUI: Using interface: en0      IP: 192.168.43.120      SubnetMask: 255.255.255.0  
[notice ]          ofxRemoteUI: Broadcasting my presence every 1sec at this multicast @ 192.168.43.255:25748  
[notice ]          ofxRemoteUI: Listening for commands at 192.168.43.120:37126  
[notice ]          ofxRemoteUI: Loading from XML! "configs/ofxRemoteUISettings.xml"  
[notice ]          ofxApp: setupErrorReporting()  
[notice ]          ofxSensu: Setting up with host: 127.0.0.1 port: 3030  
[notice ]          ofxApp: setupGoogleAnalytics()  
[notice ]          ofxGoogleAnalytics: Generating new UUID (9d2a7c61-9495-4315-a867-621fb7807f2f)  
[notice ]          ofxGoogleAnalytics: Generating new UUID (828c2a4c-ffca-47a4-a857-6cb4eed11585)  
[warning]          ofxGoogleAnalytics: Creating a new UUID for this app: 828c2a4c-ffca-47a4-a857-6cb4eed11585
```


ofxApp :: Logging

```
ofLogNotice("ofApp") << "Start User Process " << ofxApp::toString(s);
```

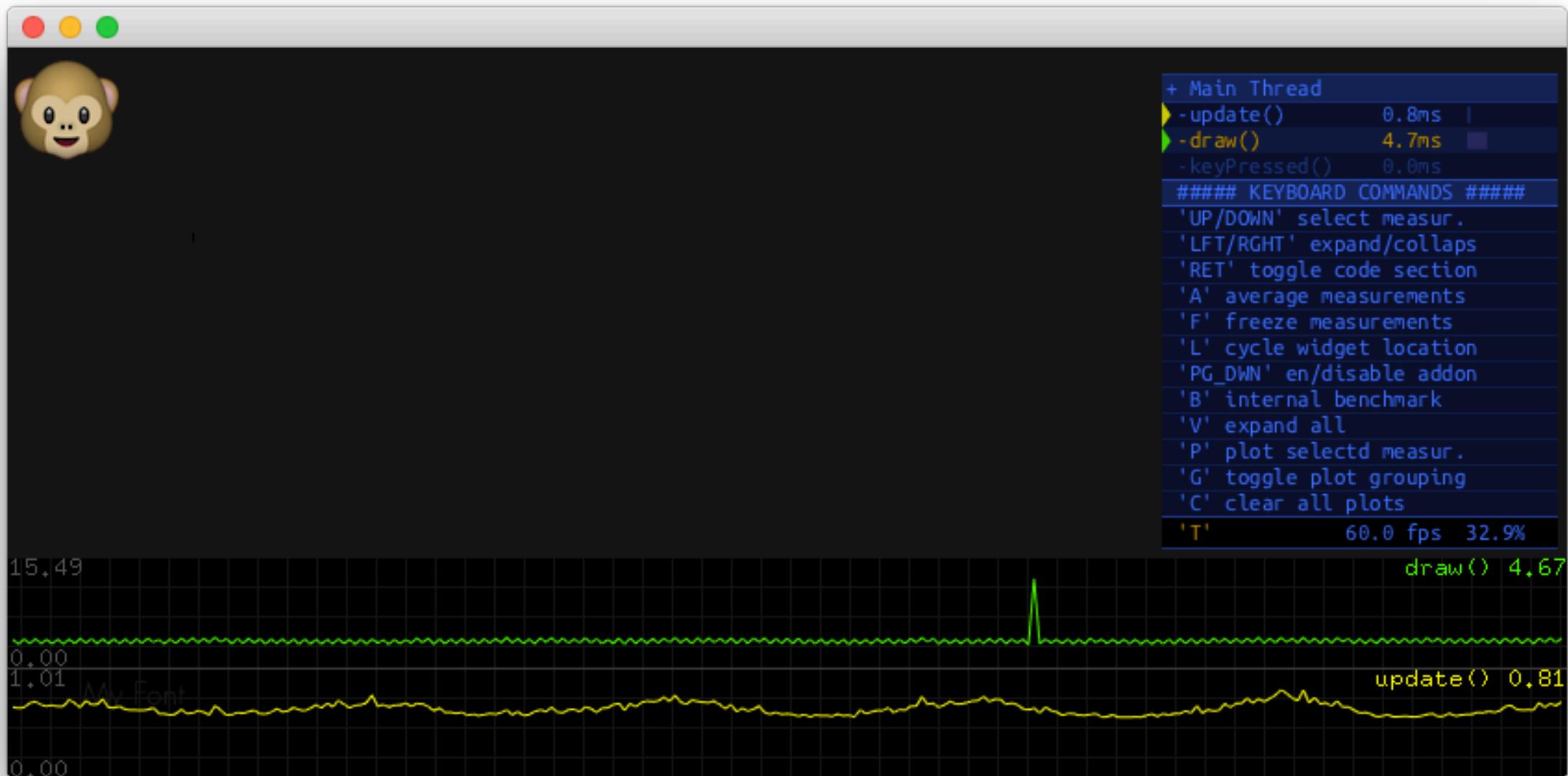
ofxApp :: Logging

logs/assetStatus.log

```
'https://images.collection.cooperhewitt.org/118668_e56c073e2456f489_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/31634_2366a6ebad71018a_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/35381_83365cef358c5525_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/68429_00ebc9b114f2369a_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/106060_8a004dcc7f53cff5_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/34058_f0da4d59f8c522f1_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/30564_09fcfac3c3ca54dc_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/30150_88a47f5c19481c17_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/24736_471fa14b1f552c1f_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/39102_50770b3c89bb14da_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/28146_9344719e5773f72b_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/68421_92631df45846be92_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/41672_3f9e4245715f79a4_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/34057_7058f5a0438109eb_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/11277_3d6aadade6e41043_z.jpg' EXISTS and SHA1 OK 😊
'https://images.collection.cooperhewitt.org/123489_891f9172791349c1_z.jpg' EXISTS and SHA1 OK 😊
```

ofxApp::Profiler

ofxTimeMeasurements



ofxApp::Profiler

- Measures time it takes for CPU to execute

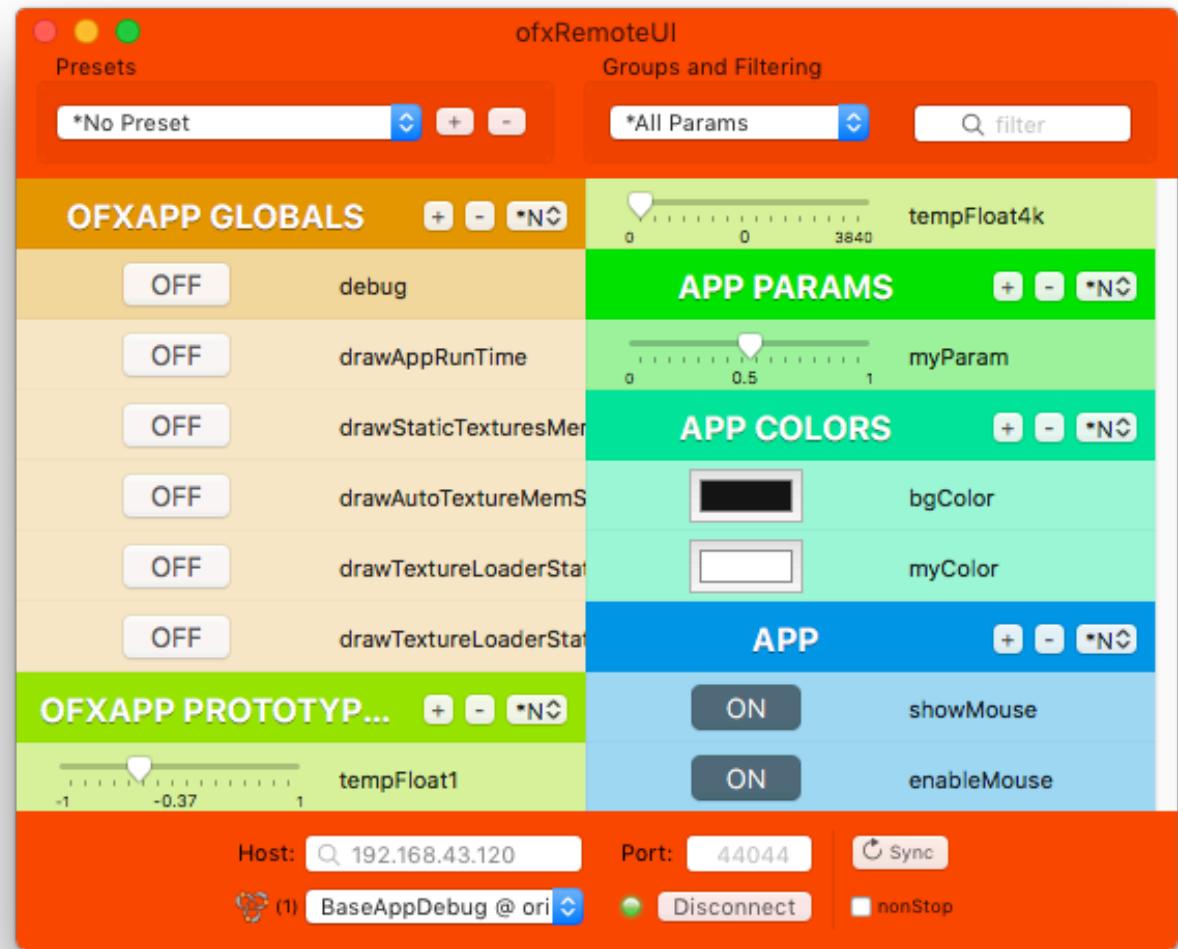
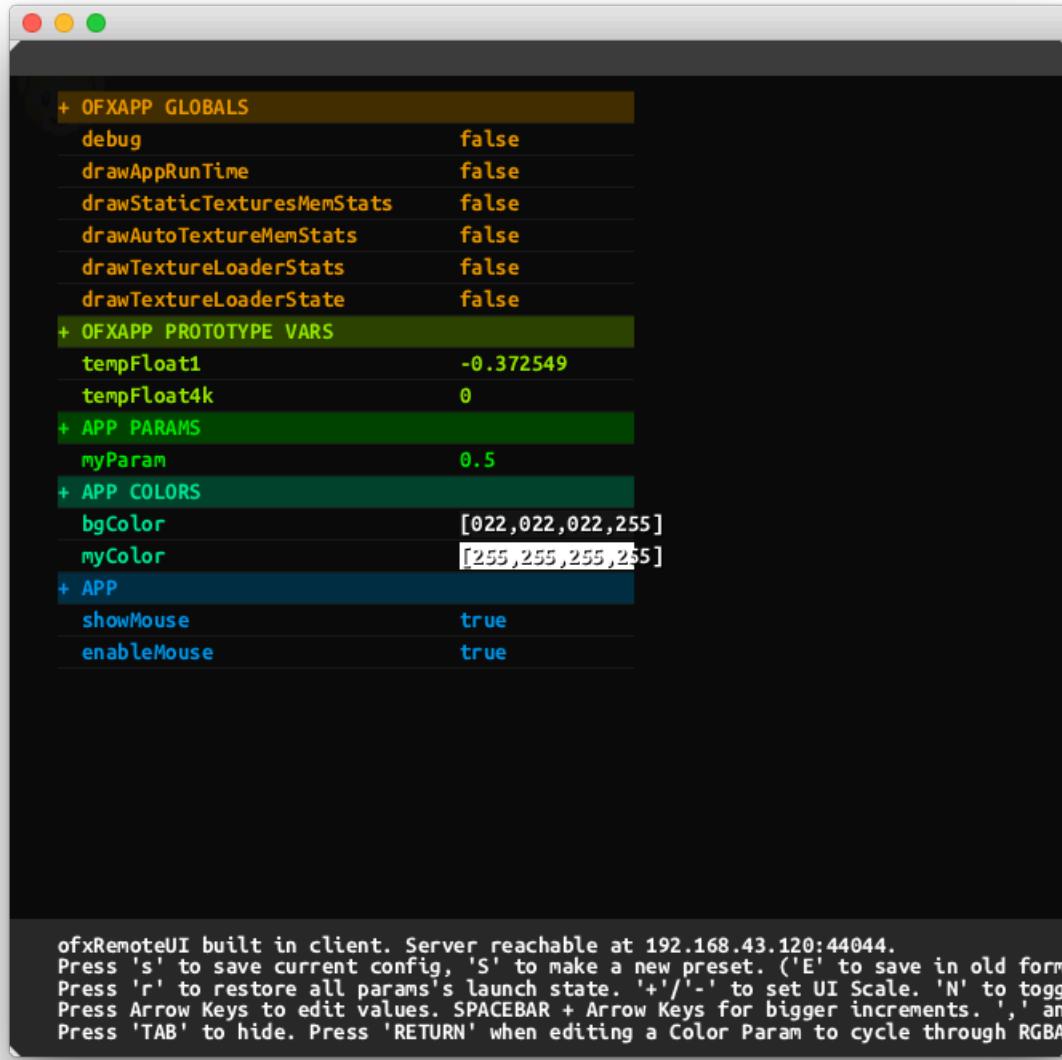
```
TS_START("myCustomDraw");
drawCustomThingies();
TS_STOP("myCustomDraw");
```

ofxApp::Profiler

- Also measures OpenGL execution times

```
TSGL_START("banana");
drawCustomThingies();
TSGL_STOP("banana");
```

ofxApp :: Parameters



ofxApp :: Parameters

```
bool debug = false;
bool drawAppRunTime = false;
bool drawTextureLoaderStats = false;
bool drawTextureLoaderState = false;
bool drawStaticTexturesMemStats = false;
bool drawAutoTextureMemStats = false;

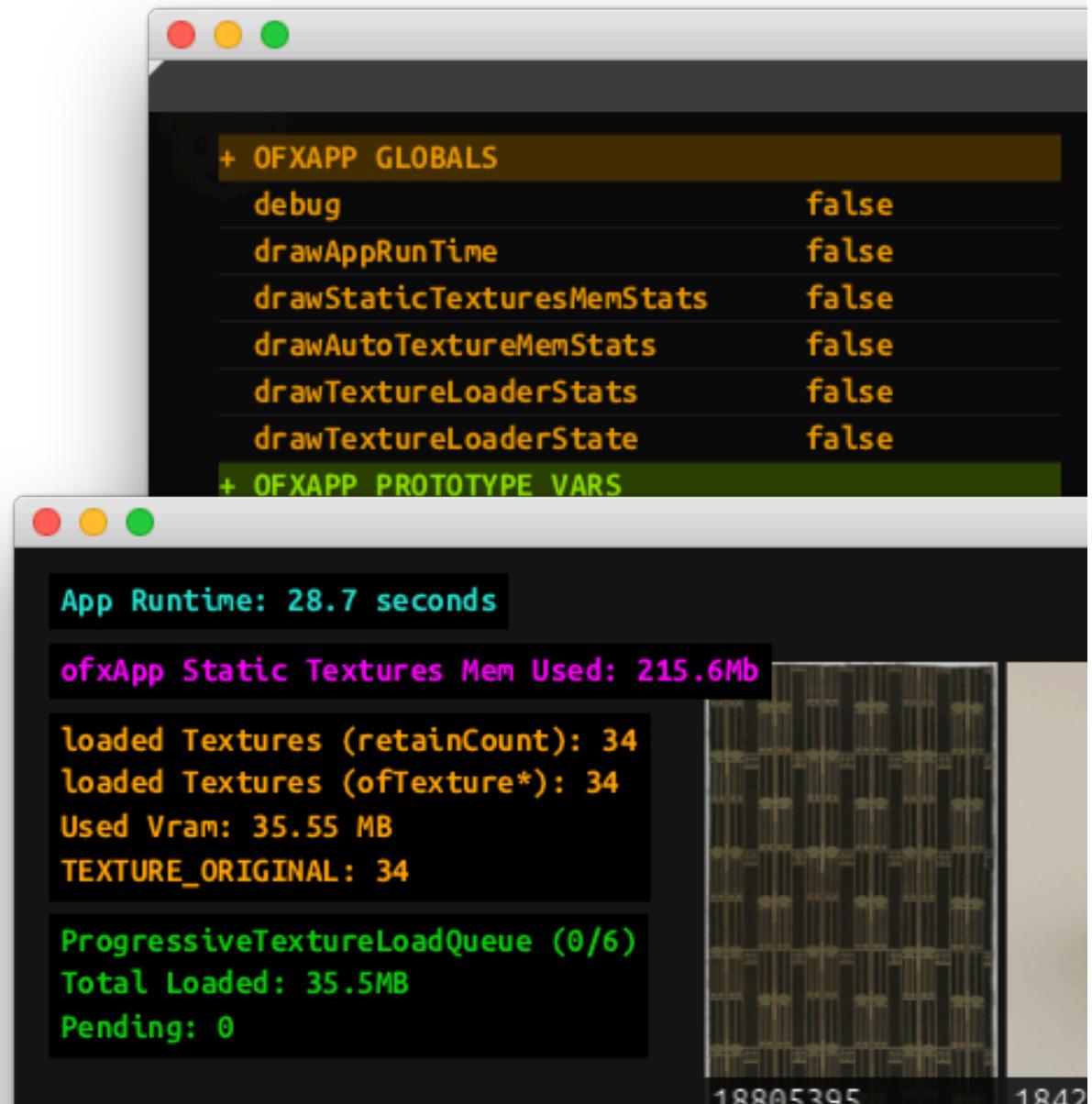
float tempFloat1 = 0;
float tempFloat4k = 0;

RUI_NEW_GROUP("OFXAPP GLOBALS");
RUI_SHARE_PARAM(debug);
RUI_NEW_COLOR();
RUI_SHARE_PARAM(drawAppRunTime);
RUI_SHARE_PARAM(drawStaticTexturesMemStats);
RUI_SHARE_PARAM(drawAutoTextureMemStats);
RUI_SHARE_PARAM(drawTextureLoaderStats);
RUI_SHARE_PARAM(drawTextureLoaderState);

RUI_NEW_GROUP("OFXAPP PROTOTYPE VARS");
RUI_SHARE_PARAM(tempFloat1, -1, 1);
RUI_SHARE_PARAM(tempFloat4k, 0, 3840);
```

ofxApp :: Status Info

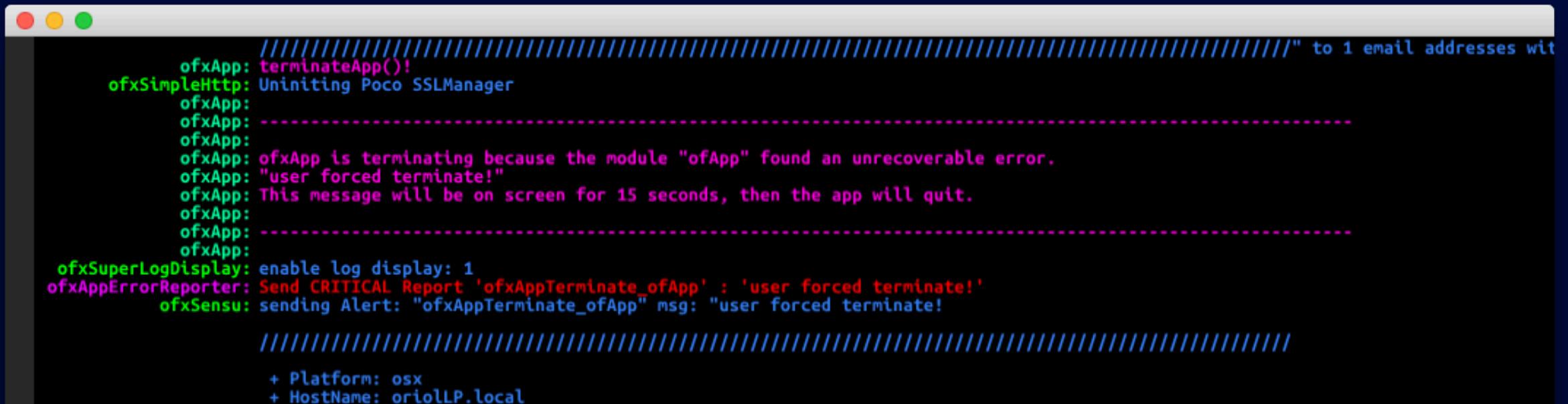
- VRAM Used in Static Textures
- VRAM Used in dynamic textures
- OpenGL error reports
- App Runtime



ofxApp :: Error Reporting

ofxSensu >> email alerts + error / recovery

```
OFXAPP_REPORT("myAlertID", "something went wrong", 2);
OFXAPP_REPORT_FILE("myAlertID", "something went wrong", 2, "path/to/my/File");
ofxApp::utils::terminateApp("ofApp", "user forced terminate!");
```



A screenshot of a terminal window on a Mac OS X system. The window title bar shows the standard red, yellow, and green close buttons. The main pane displays the following text:

```
ofxApp: terminateApp()!
ofxSimpleHttp: Uniniting Poco SSLManager
ofxApp:
ofxApp:
ofxApp:
ofxApp: -----
ofxApp: ofxApp is terminating because the module "ofApp" found an unrecoverable error.
ofxApp: "user forced terminate!"
ofxApp: This message will be on screen for 15 seconds, then the app will quit.
ofxApp:
ofxApp: -----
ofxApp:
ofxSuperLogDisplay: enable log display: 1
ofxAppErrorReporter: Send CRITICAL Report 'ofxAppTerminate_ofApp' : 'user forced terminate!'
ofxSensu: sending Alert: "ofxAppTerminate_ofApp" msg: "user forced terminate!"

+ Platform: osx
+ HostName: oriollP.local
```

ofxApp :: config file

```
{  
    "App": {  
        "frameRate" : 60,  
        "numAASamples" : 8,  
        "showMouse" : true,  
        "enableMouse" : true,  
        "maxThreads" : 8,  
        "window": {  
            "windowMode" : 8, //see ofxScreenSetup::ScreenMode; windowed=8, fullOneMonitor=1, fullAllMonitors=0  
            "customWidth": 800, //540  
            "customHeight": 800,  
            "customWindowPosition" : false,  
            "customPositionX" : 1950,  
            "customPositionY" : -450  
        },  
        "startupScreenViewport": { "x":0.0, "y":0.0, "w":1.0, "h":1.0 }, //define what part of the window is used to draw loading screen  
        "renderSize": { //get it with app.renderSize  
            "width" : 1080,  
            "height" : 1920  
        },  
        "mullions": {  
            "numX" : 4,  
            "numY" : 4,  
            "visibleAtStartup" : false  
        },  
        "onErrorHandlerMessage": "call 911", //TODO!  
        "TimeSample0fxApp" : false  
    },  
  
    "StateMachine": {  
        "onErrorHandlerRetryCount" : 0,  
        "onErrorHandlerWaitTimeSec" : 10  
    },
```

```
"Fonts": {
    "ofxApp": { //required by ofxApp, dont remove the "monospaced" & "monospacedBold" sections!
        "monospaced": {
            "fontFile" : "fonts/UbuntuMono-R.ttf",
            "atlassize" : 512,
            "lineHeight" : 1.4,
            "mipmaps" : false,
            "ipmapPadding" : 0,
            "uiScale" : 1.0
        },
        "monospacedBold": {
            "fontFile" : "fonts/UbuntuMono-B.ttf",
            "atlassize" : 512,
            "lineHeight" : 1.4,
            "mipmaps" : false,
            "ipmapPadding" : 0,
            "uiScale" : 1.0
        }
    },
    "user": { //place your custom fonts here
        "NoManSky": {
            "fontFile" : "fonts/geonms-webfont.ttf",
            "atlassize" : 512,
            "lineHeight" : 1.3,
            "mipmaps" : false,
            "ipmapPadding" : 0,
            "uiScale" : 1.0
        }
    }
},
```

```
"StaticAssets":{  
    "textures" : "images" //this is where the static assets live; under "data"  
},  
  
"RemoteUI":{  
    "enabled" : true,  
    "columnWidth" : 400,  
    "uiScale" : 1.0,  
    "useFontStash" : true,  
    "fontSize" : 15,  
    "fontFile" : "fonts/UbuntuMono-B.ttf",  
    "saveSettingsOnExit" : true,  
    "showUiDuringEdits" : false,  
    "automaticBackupsOnSave" : false,  
    "drawOnScreenNotifications" : true,  
    "notificationsScreenTime" : 5.0,  
    "logNotificationsScreenTime" : 10.0,  
    "paramWatches" : {  
        // "debug" : true  
        // "myColor" : false  
    }  
},  
  
"TimeMeasurements":{  
    "enabled" : true,  
    "uiScale" : 1.0,  
    "threadTimeDecay" : 0.985,  
    "frameRate" : 60,  
    "fontSize" : 13,  
    "fontFile" : "fonts/UbuntuMono-R.ttf",  
    "useFontStash" : true,  
    "msPrecision" : 1,  
    "plotResolution" : 0.33,  
    "plotH" : 60,  
    "removeExpiredThreads" : true,  
    "removeExpiredTimings" : false,  
    "widgetLocation" : 3,  
    "percentageAsGraph" : true  
},
```

```
"TextureLoader": {  
    "maxNumberSimultaneousLoads" : 3,  
    "textureLodBias" : -0.3,  
    "maxTimeSpentLoadingPerFrameMs" : 4.0, //ms  
    "scanlinesPerLoop" : 128,  
    "maxLoadRequestsPerFrame" : 64  
},  
  
"Logging":{  
    "logLevel" : 1, //0..5 (verbose, notice, warning, error, fatal, silent)  
    "toScreen": true,  
    "toFile" : true,  
    "toConsole" : true,  
    "deleteOldLogs" : true,  
    "logExpirationInDays" : 24,  
    "useFontStash" : true,  
    "fontSize" : 14,  
    "uiScale": 1.0,  
    "screenLogPanelWidth" : 0.95,  
    "maxScreenLines" : 4000,  
    "visibleAtStartup" : false,  
    "syncronizedLogging" : false,  
    "displayLogTimes" : true,  
    "ThreadSafeLog" : {  
        "alsoPrintToConsole" : true  
    }  
},  
  
"ErrorReporting":{  
    "enabled" : true,  
    //"host" : "127.0.0.1",  
    "host" : "192.168.103.10",  
    "port" : 3030,  
    "emails" : ["oriol@localprojects.com"]  
},
```

```
"GoogleAnalytics" : {
    "enabled" : true,
    "verbose" : true,
    "sendBenchmark" : true,
    "randomizedUUID" : true, //this might fake N different users, instead of same user each session
    "maxRequestsPerSession" : 250,
    "sendDataInterval" : 5, //seconds
    "googleID" : "UA-51706745-1",
    "appName" : "AppName",
    "appVersion" : "v1",
    "appID" : "myAppID",
    "appInstallerID" : "myAppInstallerID",
    "shouldReportFramerate" : true,
    "framerateReportInterval" : 60.0 //seconds
},
"UIO": {
    "enabled" : true,
    "port" : 3333
},
"Downloads": {
    "proxy": {
        "useProxy" : false,
        "proxyHost" : "",
        "proxyPort" : 8080,
        "proxyUser" : "",
        "proxyPassword" : ""
    },
    "credentials": {
        "username" : "",
        "password" : ""
    },
    "customHeaders": { //not implemented
        // "myHeaderName" : "myHeaderContent",
    },
    "verbose" : false,
    "maxConcurrentDownloads" : 5,
    "userAgent" : "", //not implemented
    "timeOutSec" : 20,
    "speedLimitKb" : 0, //per download - 0 means no limit
    "idleTimeAfterEachDownloadSec" : 0.0
},
```

```
"Content":{  
  "JsonSources":{  
    "CH": {  
      "url" : "file://testJson/ch_small.json",  
      //"url" : "http://uri.cat/LP/ch_small.json",  
      "jsonDownloadDir": "CH_JsonDownloads",  
      "assetsLocation": "CH_assets", //all assets will be in that folder, with a sub-folder (named after the object ID) per each object  
      "shouldSkipObjectPolicyTests" : false //set this to true to avoid the object cleanup according to the policies defined below  
    },  
    "CWRU": {  
      "url" : "file://testJson/cwru.json",  
      //"url" : "http://uri.cat/LP/cwru.json",  
      //"url" : "file://testJson/cwru_broken.json",  
      "jsonDownloadDir": "CWRU_JsonDownloads",  
      "assetsLocation": "CWRU_assets", //all assets will be in that folder, with a sub-folder (named after the object ID) per each object  
      "shouldSkipObjectPolicyTests" : false //set this to true to avoid the object cleanup according to the policies defined below  
    }  
  },  
  "AssetDownloadPolicy":{ //asset should be downloaded if...  
    "fileMissing" : true,  
    "fileTooSmall" : true,  
    "fileExistsAndNoSha1Provided" : true,  
    "fileExistsAndProvidedSha1Missmatch" : true,  
    "fileExistsAndProvidedSha1Match" : false //no need to re-download if file is on disk and sha1's match  
  },  
  "AssetUsagePolicy":{ //asset should be Used if...  
    "fileMissing" : false,  
    "fileTooSmall" : false,  
    "fileExistsAndNoSha1Provided" : false,  
    "fileExistsAndProvidedSha1Missmatch" : false,  
    "fileExistsAndProvidedSha1Match" : true  
  },  
  "ObjectUsagePolicy":{ //a JSON object should only be used if...  
    "allAssetsAreOK" : true, //if true, an object can only be used if ALL assets are OK  
    "minNumberImgAssets" : 1, //ie. if there are < N image assets, object will be rejected  
    "minNumberVideoAssets" : 0,  
    "minNumberAudioAssets" : 0  
  }  
}
```

Screenshot of a GitHub repository page for "local-projects / ofxApp".

The repository has 142 commits, 1 branch, 0 releases, and 1 contributor.

Recent commits:

- armadillu example shows texture load/unload with ofxInterface scrollview, consi... 376d246 7 hours ago
- example example shows texture load/unload with ofxInterface scrollview, consi... 7 hours ago
- exampleNoContent example shows texture load/unload with ofxInterface scrollview, consi... 7 hours ago
- src example shows texture load/unload with ofxInterface scrollview, consi... 7 hours ago
- addon_config.mk add missing addons 2 months ago
- readme.md make global app a singleton pattern, more readme, more phase name cha... 6 days ago

ofxApp

What is ofxApp? Its the basic skeleton for an interactive installation made in OpenFrameworks. It tries to simplify your life by offering some basic functionality without much required on your side.

Features

A quick summary of the features offered by ofxApp:

- Startup / Loading Screen with progress (ofxStateMachine)
- CMS Asset download/cache/checksum management (ofxSimpleHttp, ofxAssets, ofxTagSystem)
- CMS JSON content loading & failover recovery (ofxMTJsonParser)
- Time based profiler (ofxTimeMeasurements)

Display a menu Dynamic texture load / unlcad (TexturedObject)