**1.Getting Started with CocosBuilder 3**

Thank you for downloading CocosBuilder 3! CocosBuilder is a great, free (MIT license) tool for creating interfaces, animations, scenes, level maps, particle systems or just about anything you can imagine. You can use CocosBuilder together with Cocos2d for JavaScript, which will run on HTML5, iOS and Android, cocos2d-iphone or cocos2d-x.

**You can get back to this guide at anytime by selecting CocosBuilder Documentation from the Help menu.**



CocosPlayer and Cocos2d for JavaScript

The easiest way to use CocosBuilder is together with Cocos2d for JavaScript. Download the CocosBuilder source code which includes the CocosPlayer project. Run CocosPlayer on either your device or in Simulator. CocosBuilder will automatically pair with CocosPlayer if they are running on the same wifi network. If more than one computer on the same network are running CocosBuilder, use codes to pair the devices with CocosBuilder.

Good examples to get started with CocosBuilder and Cocos2d for JavaScript are the CocosDragonJS project and CrystalCraze. You will also find the API documentation for Cocos2d for JavaScript in the Help menu.

Get Cocos2d and CCBReader

To use CocosBuilder with cocos2d-iphone or cocos2d-x you will need to download the CCBReader. CCBReader is a set of classes that parses the files that CocosBuilder exports and builds your cocos2d node graphs for you. It will also handle any animations that you create. [Download the CCBReader and examples](http://cocosbuilder.com/?page_id=11) compatible with this version of CocosBuilder.

A good way to learn how to use CocosBuilder together with cocos2d is to checkout and read the examples, they are very well commented and easy to understand. Start by opening the CocosBuilderExample.ccbproj file located in /CocosBuilderExample/Resources with CocosBuilder. Then open the CocosBuilderExample.xcodeproj with Xcode.

Tutorials

Links to tutorials for how to use CocosBuilder will be posted on the [CocosBuilder website](http://cocosbuilder.com/). At the site you will also find the latest news and other useful information.

**2.Setting up a New Project**

This section will guide you through the process of setting up a new project with CocosBuilder. The process is slightly different depending on if you will be using Cocos2d for JavaScript, cocos2d-iphone or cocos2d-x.

Creating a Project for Cocos2d for JavaScript

To start a new Cocos2d for JavaScript project, simply select New Project… from the File menu. Choose a name for your project and save it. Your new project will be setup with an example MainScene.ccb file and MainScene.js script file. To try it out, either run it in CocosPlayer by selecting Run in CocosPlayer from the File menu or publishing it by selecting Publish. If you publish the files you can test it in your browser by double clicking the index.html file in the Published-HTML5 directory.

Creating a Project for cocos2d-iphone

It's best to setup your Xcode project before creating your CocosBuilder project. Make sure that the cocos2d templates are installed and create a new cocos2d-iphone project in Xcode (if you do not already have a project you want to use with CocosBuilder).

Once you have the Xcode project set up, create your CocosBuilder project. Select New Project… from the File menu. Save the CocosBuilder project in the Xcode project's Resources directory.

All images and sprite sheets you put in the Resources directory will now show up automatically in CocosBuilder's project view on the left side of the main window, but remember that you will need to add them to your Xcode project if you want to use them in the compiled version of your app.

Now update the settings of your project. Choose Project Settings…, in the settings window deselect the JavaScript based project checkbox. Save your changes and open the Publish Settings…. Set the iOS export path to your Resources directory, check the Flatten paths when publishing checkbox and check the Only publish ccb-files checkbox. Make sure that the iOS export target is selected, but the Android and HTML5 ones are unchecked.

Note: The settings described above assumes that you are managing all the resources yourself, and that you are using yellow folders (groups) in Xcode.

Publishing Your Project

Before you can import the files you create in CocosBuilder in your code you will need to publish them. To publish all ccb-files in a project, select Publish in the File menu. For each ccb-file a new ccbi-file is created. The ccbi files are very compact binary files with all unnecessary information stripped out. Include the ccbi-files in your Xcode project.

To load a ccbi-file from your code, make sure that you have included the CCBReader classes in your Xcode project. Then call the CCBReader's nodeGraphFromFile method.

CCNode\* myNodeGraph = [CCBReader nodeGraphFromFile:@"myFile.ccbi"];

Project Settings

By default CocosBuilder will export your ccb-files to the same directory as the project file, but you can change in which directories CocosBuilder should look for resources and to where ccb-files should be exported. It is also possible to export ccb-files together with your images to a zip-file.

Access the project settings by selecting Project Settings… from the File menu.

Publish to zip-file

You have the option to publish to a zip file. If this option is used a zip file called ccb.zip will be placed in your destination directory. To be able to load the zip-file in your code you need to:

1. Include the SSZipArchive classes, in addition to the CCBReader, in your Xcode project.
2. In your project's build settings, add CCBENABLEUNZIP to your Preprocessor Macros.
3. Replace cocos2d's default CCFileUtils class by calling [CCBFileUtils sharedFileUtils] before your cocos2d is loaded (you will need to import CCBReader.h to do this).
4. Unpack the zip file by calling [CCBReader unzipResources:@"ccb.zip"]

Flatten file paths

In the project settings, you can choose to flatten file paths (this is the default setting). If you do this, all file paths in the exported ccbi-files will be flattened (e.g. mySubDirectory/myImage.png will be transformed to myImage.png). If you use the zip option in combination, the files will be copied to a single directory within the zip file. If you are publishing without the zip option, the file structure will still be kept when copying resources to their destination. So when the resources are added in Xcode, they need to be added as yellow folders (which is usually the default option). If you choose not to flatten the paths you should add your resources as blue folders in Xcode.

**3.Working with ccb-files**

This section assumes that you already have set up your CocosBuilder project. It will give you some tips on how to work with ccb-files.

Adding Nodes

When creating a new ccb-file you define which type of the root node it uses. The root node cannot be deleted and you cannot change the type of the root node (unless you create a new file). There are some properties, e.g. the position, of the root node that you cannot change. These properties will be grayed out in the property inspector.



To add new nodes to a file, click the icons in the toolbar. Each icon corresponds to a cocos2d class. CocosBuilder has support for many of cocos2d's standard node types and a few extensions (such as CCControlButton and CCScale9Sprite). By default the node is added as a sibling to the selected node or, if no node is selected, as a child to the root node. By holding down the shift key when adding an object it will instead be added as a child to the selected node.

You can also add images by dragging them from the left hand project view onto the canvas area or the timeline. Images that you drag will automatically become CCSprite nodes. If you drop an image onto a CCMenu it will become a CCMenuItemImage.

Editing Properties

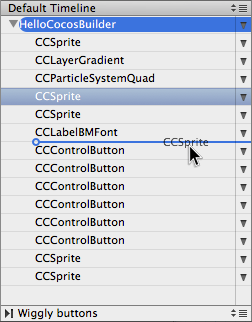
When a node in your ccb-file is selected its properties will show up in the right hand inspector. Change the properties by editing the values. To get a description of each property, please refer to the [cocos2d api documentation](http://www.cocos2d-iphone.org/api-ref/2.0.0/).



The properties that affect positioning of the node can be edited visually in the canvas area. To move a node simply select it and drag it around, to change scaling drag the handles and to rotate drag the handles while holding down the option key. To edit the anchor point of a node, grab it and drag it.

Changing zOrder

Unlike other properties the zOrder is not set in the properties inspector in CocosBuilder. Instead the zOrder is determined by the ordering of objects in the hierarchy view. All objects you add to a ccb-file will show up in the hierarchy view, to simply drag and drop them to the position that you require.



When loaded into your app the zOrder of your nodes will be numbered from 0 to (N-1), where N is the number of children the parent node has. If you need to insert objects programmatically at a specific zOrder, the recommended way is to use a CCNode as a placeholder which you either replace or add your children to directly in your code.

Guides and Sticky Notes

Guides and sticky notes have no effect when loaded into your app, but can make it easier to align objects or save comments to other developers or designers in the ccb-file.



To create a new guide, drag it out from the bottom or left side ruler. To move it, hold down the command key while dragging it. You can remove a guide by dragging it out of the visible area.

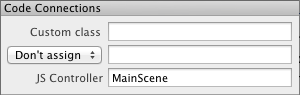
Sticky notes are created by choosing Add Sticky Note from the View menu. Edit or remove the sticky note by double clicking it.

**4a.Connecting with Cocos2d JS**

For any non-trivial use of CocosBuilder you will need to connect your ccb-files with your JavaScript code. This section explains how this works.

Using Custom Classes

The way you can link your code with CocosBuilder is to use controller objects. The controller object is created at the same time as your ccbi-file is loaded. To set the name of the controller objects class, select the root node in your ccb-file and enter the name in the JS Controller filed of Code Connections.



When the complete ccbi-file has loaded. cc.BuilderReader will call the onDidLoadFromCCB method. (It's optional to implement this method, if it doesn't exist it will not be called.)

This is an example of what a very simple controller class with only the onDidLoadFromCCB method implemented can look like:

//

// MainScene class

//

var MainScene = function(){};

MainScene.prototype.onDidLoadFromCCB = function()

{

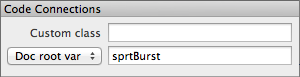
cc.log("File was loaded");

};

Linking Member Variables

References to objects in your ccbi-file can be linked to member variables when the file is loaded. These member variables can be either in the root node's controller object, in which case it must have been assigned a controller object. You can also assign them to a custom object, which is optionally passed as the owner object to the cc.BuilderReader. To load ccbi-file and assign members to it's owner, the node graph needs to be loaded by calling cc.BuilderReader.load("MyInterfaceFile", owner) or cc.BuilderReader.loadAsScene("MyInterfaceFile", owner);

To link a node in your ccb-file, simply select the object, under Code Connections change the Don't assign popup menu to either Doc root var or Owner var. Then enter the name of your member variable to the right of the popup menu.



When the ccbi-file is loaded in your game, the member variable will automatically be assigned to either your owner or controller object. To access it from inside the class use the this keyword.

MainScene.prototype.onDidLoadFromCCB = function()

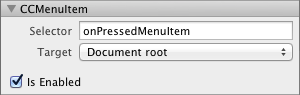
{

cc.log("CocosBuilder variable: " + this.sprtBurst);

};

Adding Callbacks to Menus

To add a callback when a CCMenuItemImage is tapped, simply select the CCMenuItemImage in CocosBuilder, then add the name of the method you want to call in the Selector field. Set the target either to the Document root or the Owner.



The callback will send the CCMenuItemImage as its only parameter to the method that you specify. You can also choose to leave the parameter out.

MainScene.prototype.onPressedMenuItem = function()

{

cc.log("Pressed menu item!");

};

Adding Callbacks to CCControl

Support for CCControl is coming soon to Cocos2d JS.

Options for Loading ccb-files

CocosBuilder documents, or ccb-files, needs to be published into a compact binary format, ccbi, before they can be loaded into your application. Once published they can be easily loaded with a single line of code. To load a node graph call the cc.BuilderReader.load method as follows.

var myNode = cc.BuilderReader.load("MyNodeGraph");

For your convenience, CCBReader can also wrap your node graph in a scene. To load your ccbi-file in a scene call cc.BuilderReader.loadAsScene.

var myScene = cc.BuilderReader.loadAsScene("MyScene");

Passing an Owner Variable

Sometimes you need to be able to access member variables from and get callbacks to another object than the root node of a ccb-file. To do this you will need to pass a owner to the CCBReader. To get the variable or callback assigned to the owner, make sure that you've selected owner when declaring the member variable name or callback in CocosBuilder. Then call the cc.BuilderReader.load or cc.BuilderReader.loadAsScene method but add an extra variable with the owner.

// Setup myCallbackObject somewhere in your code;

var myNode = cc.BuilderReader.load("MyNodeGraph", myCallbackObject);

Accessing Variables and Callbacks in a sub ccb-file

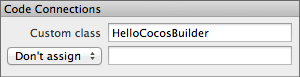
If you are using sub ccb-files specifying the root node as target will refer to the root node of the sub ccb-file. The owner target is the object that you pass to the CCBReader.

**4b.Connecting with cocos2d-iphone**

For any non-trivial use of CocosBuilder you will need to connect your ccb-files with your code. This section explains how this works.

Using Custom Classes

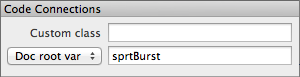
The way you can link your code with CocosBuilder is to use custom classes. To assign a custom class to an object in CocosBuilder, just select the object then enter the name of your custom class in the property inspector. Remember that your custom class needs to be a sub class of the selected object. When loading the ccbi-file CCBReader will initialize your custom class using the super class's init method, then set all of the objects properties. If you need to do any extra initialization of your object after the properties are set, CCBReader will call the didLoadFromCCB method.



Please note that CCBReader will not be able to use any custom init methods. You will need to have a working plain init method for all custom classes that you use with CocosBuilder. You can use custom classes for any node in you ccb-file, however the most common use is to override the root node.

Linking Member Variables

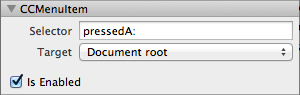
References to objects in your ccbi-file can be linked to member variables when the file is loaded. These member variables can be either in the root node of the document, in which case it must have been assigned a custom class. You can also assign them to a custom object, which is optionally passed as the owner object to the CCBReader. To load ccbi-file and assign members to it's owner, the node graph needs to be loaded by calling either the nodeGraphFromFile:owner: or the sceneWithNodeGraphFromFile:owner: methods.



To link an object, simply declare them in the header file, either in the object you plan to pass as the owner or in the custom class of your document root object. Select the object in CocosBuilder, under Code Connections change the Don't assign popup menu to either Doc root var or Owner var. Then enter the name of your member variable to the right of the popup menu.

Adding Callbacks to Menus

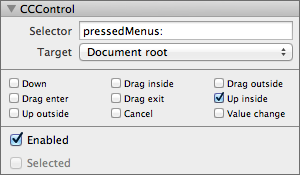
To add a callback when a CCMenuItemImage is tapped, simply select the CCMenuItemImage in CocosBuilder, then add the name of the method you want to call in the Selector field. Set the target either to the Document root or the Owner.



The callback will send the CCMenuItemImage as its only parameter to the method that you specify (it uses the id type, and is often named sender). You can also chose to leave the parameter out.

Adding Callbacks to CCControl

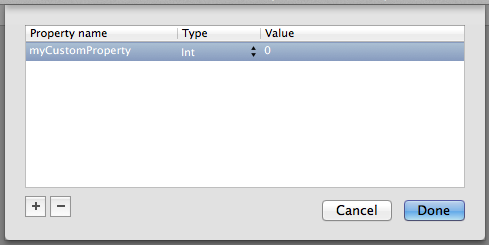
Adding a callback to a CCControl is similar to adding a callback to a CCMenuItemImage, it just have a few extra options.



Tick the event types that you want to receive callbacks for. For CCControlButton it is most common to use the Up inside callback only. Select your target, either Document root or Owner, and the name of your callback method. The callback method can optionally take two arguments, the sender (i.e. the CCControl) and the type of event. The event types are defined in CCControl.h.

Using Custom Properties

Sometimes it can be useful to be able to specify a number of extra properties to a custom class. If you have set the custom class of a node a Edit Custom Properties button will appear at the bottom of the inspector. Pressing the button brings up the custom properties window.



Here you can add the custom properties, make sure that the same properties are defined in your code. You can use int, float, BOOL or NSString properties. For NSString the property should be defined as retain or copy, and for the other types as assign.

If you use custom properties for your root node and include it as a sub ccb-file in another document you will be able to override the values of the properties. Just select the sub ccb-file and edit the values in the property inspector.

Options for Loading ccb-files

CocosBuilder documents, or ccb-files, needs to be published into a compact binary format, ccbi, before they can be loaded into your application. Once published they can be easily loaded with a single line of code. To load a node graph, add the CCBReader.h and CCBReader.m files to your Cocos2D [project](http://foo.com/), then call the nodeGraphFromFile: method as follows.

CCNode\* myNode = [CCBReader nodeGraphFromFile:@"MyNodeGraph.ccbi"];

You may need to cast the returned value depending on what sort of object is the root node in your ccbi-file and how you will use it in your code. For instance, if you load a CCParticleSystem, use the following code.

CCParticleSystem\* myParticles = (CCParticleSystem\*) [CCBReader nodeGraphFromFile:@"MyParticleSystem.ccbi"];

For your convenience, CCBReader can also wrap your node graph in a scene. To load your ccbi-file in a scene call sceneWithNodeGraphFromFile:

CCScene\* myScene = [CCBReader sceneWithNodeGraphFromFile:@"MyScene.ccbi"];

Passing an Owner Variable

Sometimes you need to be able to access member variables from and get callbacks to another object than the root node of a ccb-file. To do this you will need to pass a owner to the CCBReader. To get the variable or callback assigned to the owner, make sure that you've selected owner when declaring the member variable name or callback in CocosBuilder. The call the nodeGraphFromFile:owner: or sceneWithNodeGraphFromFile:owner: method of CCBReader when loading your file.

MyCallbackClass\* callbackClass = [[MyCallbackClass callbackClass alloc] init];

CCNode\* myNode = [CCBReader nodeGraphFromFile:@"MyNodeGraph.ccbi" owner:callbackClass];

Accessing Variables and Callbacks in a sub ccb-file

If you are using sub ccb-files specifying the root node as target will refer to the root node of the sub ccb-file. The owner target is the object that you pass to the CCBReader.

**4c.Connecting with cocos2d-x**

For any non-trivial use of CocosBuilder you will need to connect your ccb-files with your code. This section explains how this works.

Using Custom Classes

The way you can link your code with CocosBuilder is to use custom classes. To assign a custom class to an object in CocosBuilder, just select the object then enter the name of your custom class in the property inspector. Remember that your custom class needs to be a sub class of the selected object.

When loading the ccbi-file, you need to create two custom classes (could be one). The custom loader class derives from cocos2d::extension::CCLayerLoader.

The custom layer class derives from cocos2d::extension::CCBSelectorResolver ,cocos2d::extension::CCBMemberVariableAssigner , and cocos2d::extension::CCNodeLoaderListener classes.

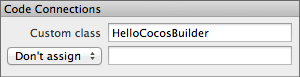
In your custom loader class you need to add code for initialization such as

public:

CCB\_STATIC\_NEW\_AUTORELEASE\_OBJECT\_METHOD(CustomLayerLoaderClass, loader);

CCB\_VIRTUAL\_NEW\_AUTORELEASE\_CREATECCNODE\_METHOD(CustomLayerClass));

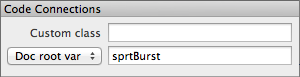
CCBReader will initialize your custom layer class using the loader class's loader method.



Please note that CCBReader will not be able to use any custom init methods. To use a custom init method, you can call it from the custom layer class constructor.

Linking Member Variables

References to objects in your ccbi-file can be linked to member variables when the file is loaded. These member variables can be either in the root node of the document, in which case it must have been assigned a custom class.



To link an object, simply declare them in the header file,

To initialize member variables, you can override the onAssignCCBMemberVariable function in the custom layer class and use something like

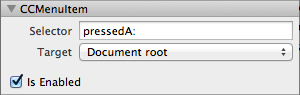
CCB\_MEMBERVARIABLEASSIGNER\_GLUE(this, "sprtBurst", CCSprite \*, this->mSprtBurst);

where sprtBurst is the property name set in CocosBuilder.

Select the object in CocosBuilder, under Code Connections change the Don't assign popup menu to either Doc root var or Owner var. Then enter the name of your member variable to the right of the popup menu.

Adding Callbacks to Menus

To add a callback when a CCMenuItemImage is tapped, simply select the CCMenuItemImage in CocosBuilder, then add the name of the method you want to call in the Selector field. Set the target either to the Document root or the Owner.



The callback will send the CCMenuItemImage as its only parameter to the method that you specify (it uses the id type, and is often named sender). You can also chose to leave the parameter out.

In your custon class, you override the onResolveCCBCCMenuItemSelector function and add code such as

CCB\_SELECTORRESOLVER\_CCMENUITEM\_GLUE(this, "pressedA:", MenuTestLayer::onMenuItemAClicked);

inside the function.

where MenuTestLayer is the name of your custom class.

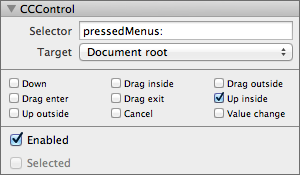
MenuTestLayer::onMenuItemAClicked can then be declared as

void MenuTestLayer::onMenuItemAClicked(cocos2d::CCObject \*pSender) {

}

Adding Callbacks to CCControl

Adding a callback to a CCControl is similar to adding a callback to a CCMenuItemImage, it just have a few extra options.



Tick the event types that you want to receive callbacks for. For CCControlButton it is most common to use the Up inside callback only. Select your target, either Document root or Owner, and the name of your callback method. The callback method can optionally take two arguments, the sender (i.e. the CCControl) and the type of event. The event types are defined in CCControl.h.

In the custom class, you override the onResolveCCBCCControlSelector function and add code such as

CCB\_SELECTORRESOLVER\_CCCONTROL\_GLUE(this, "pressedMenus:", MenuTestLayer::onPressedMenus);

inside the function. Where MenuTestLayer is the name of your custom class.

MenuTestLayer::onPressedMenus can then be declared as

void HelloCocosBuilderLayer::onMenuTestClicked(CCObject \* pSender, cocos2d::extension::CCControlEvent pCCControlEvent) {

}

Options for Loading ccb-files

CocosBuilder documents, or ccb-files, needs to be published into a compact binary format, ccbi, before they can be loaded into your application. Once published they can be easily loaded with a single line of code. To load a node graph, add the CCBReader.h and CCBReader.m files to your Cocos2D [project](http://foo.com/), then call the nodeGraphFromFile: method as follows.

CCBReader \*ccbReader = new cocos2d::extension::CCBReader(ccNodeLoaderLibrary);

CCNode\* myNode = ccbReader->readNodeGraphFromFile("MyNodeGraph.ccbi");

The initialization for ccNodeLoaderLibrary can be done in two ways.

1. If you're using a custom class: CCNodeLoaderLibrary \* ccNodeLoaderLibrary = CCNodeLoaderLibrary::newDefaultCCNodeLoaderLibrary();
2. ccNodeLoaderLibrary->registerCCNodeLoader("HelloCocosBuilderLayer", HelloCocosBuilderLayerLoader::loader());
3. In this case, HelloCocosBuilderLayer is the name of the custom class specified in CocosBuilder
4. If not using a custom class, you can initialize a default NodeLoader:  CCNodeLoaderLibrary \* ccNodeLoaderLibrary = CCNodeLoaderLibrary::newDefaultCCNodeLoaderLibrary();

You may need to cast the returned value depending on what sort of object is the root node in your ccbi-file and how you will use it in your code. For instance, if you load a CCParticleSystem, use the following code.

CCParticleSystem\* myParticles = (CCParticleSystem\*) ccbReader->readNodeGraphFromFile("MyParticleSystem.ccbi");

For your convenience, CCBReader can also wrap your node graph in a scene. To load your ccbi-file in a scene call sceneWithNodeGraphFromFile:

CCScene\* myScene = ccbReader->sceneWithNodeGraphFromFile("MyScene.ccbi");

Passing an Owner Variable

Sometimes you need to be able to access member variables from and get callbacks to another object than the root node of a ccb-file. To do this you will need to pass a owner to the CCBReader. To get the variable or callback assigned to the owner, make sure that you've selected owner when declaring the member variable name or callback in CocosBuilder. The call the nodeGraphFromFile(file, owner) or sceneWithNodeGraphFromFile(file, owner) method of CCBReader when loading your file.

HelloCocosBuilderLayer \*pOwner = new HelloCocosBuilderLayer();

CCNode\* myNode = ccbReader->readNodeGraphFromFile("MyNodeGraph.ccbi", pOwner);

Accessing Variables and Callbacks in a sub ccb-file

If you are using sub ccb-files specifying the root node as target will refer to the root node of the sub ccb-file. The owner target is the object that you pass to the CCBReader.

Example

Please take a look at HelloCocosBuilderLayer.h, HelloCocosBuilderLayer.cpp, and HelloCocosBuilderLayerLoader.h from the ExtensionsTest in the TestCPP project of cocos2d-x.

Setting scale and design size

For CocosBuilder based projects in cocos2d-x, the AppDelegate needs to setup the game to read the correct resources from the correct directories from the project files. It is based on the device screen size. You also need to set the scaling factor and the design resolution size of the GL view.

For portrait mode, you can add this code to AppDelegate.cpp in AppDelegate::applicationDidFinishLaunching

CCSize designSize = CCSizeMake(320, 480);

CCSize resourceSize = CCSizeMake(320, 480);

CCSize screenSize = CCEGLView::sharedOpenGLView()->getFrameSize();

std::vector<std::string> searchPaths;

std::vector<std::string> resDirOrders;

TargetPlatform platform = CCApplication::sharedApplication()->getTargetPlatform();

if (platform == kTargetIphone || platform == kTargetIpad)

{

searchPaths.push\_back("Published-iOS"); // Resources/Published-iOS

CCFileUtils::sharedFileUtils()->setSearchPaths(searchPaths);

if (screenSize.height > 768)

{

resourceSize = CCSizeMake(1536, 2048);

resDirOrders.push\_back("resources-ipadhd");

}

else if (screenSize.height > 640)

{

resourceSize = CCSizeMake(768, 1536);

resDirOrders.push\_back("resources-ipad");

}else if (screenSize.height > 480)

{

resourceSize = CCSizeMake(640, 960);

resDirOrders.push\_back("resources-iphonehd");

}

else

{

resDirOrders.push\_back("resources-iphone");

}

CCFileUtils::sharedFileUtils()->setSearchResolutionsOrder(resDirOrders);

}

else if (platform == kTargetAndroid || platform == kTargetWindows)

{

if (screenSize.height > 960)

{

resourceSize = CCSizeMake(640, 960);

resDirOrders.push\_back("resources-large");

}

else if (screenSize.height > 480)

{

resourceSize = CCSizeMake(480, 720);

resDirOrders.push\_back("resources-medium");

}

else

{

resourceSize = CCSizeMake(320, 568);

resDirOrders.push\_back("resources-small");

}

CCFileUtils::sharedFileUtils()->setSearchResolutionsOrder(resDirOrders);

}

pDirector->setContentScaleFactor(resourceSize.width/designSize.width);

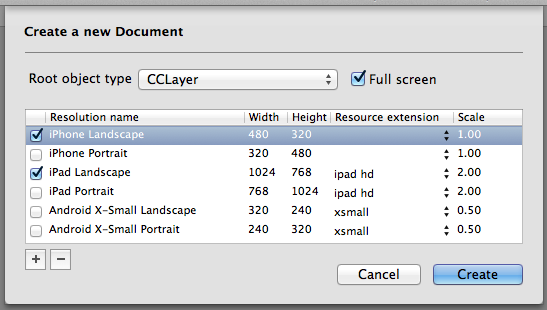
CCEGLView::sharedOpenGLView()->setDesignResolutionSize(designSize.width, designSize.height, kResolutionShowAll);

For landscape mode, you can change the order of resolutions. So (320, 480) becomes (480,320) , (640, 960) becoms (960, 640) etc. in all places.

**5.Working with Multiple Resolutions**

A common scenario when creating apps or games is to target multiple devices with different screen sizes. CocosBuilder provides advanced tools for supporting multiple resolutions and relative positioning. You can use the same file to have support for iPhone and iPad and a multitude of Android devices (if you are using cocos2d-x).

When creating a new file you select the resolutions that you want to support natively. Each resolution setting has a number of properties, such as width, height, resource extension and global scale.



CocosBuilder's resolution settings only reflects what is displayed in CocosBuilder, and are not exported to your ccbi-files. However, the default settings in CocosBuilder corresponds to the default settings in cocos2d. If you make changes to the default settings, please be aware that you may also need to make corresponding changes in your code when loading the file.

If you need to edit the resolutions after a document has been created you can do so by selecting Edit Resolutions… in the View menu.

Selecting the Right Resources

An important aspect of getting the multiple resolution support to work satisfactory is to select the correct resources for the correct display. The Resource extension determines which resources are selected for a particular resolution. For instance, for the ipad hd setting the resources with the -ipad extension are selected first hand, if no such resource is found resources with the -hd extension is used. If none of the listed extensions are found for a particular resource, the fallback is the resource without an extension. You place all your images, with the different resolution extensions, in a directory within your projects resource path.

The project view will only list resources without a resolution extension. Resources with an extension are hidden.

Supporting Retina Display

CocosBuilder works entirely with points and not pixels. This means that you will only be seeing non-retina display layouts inside CocosBuilder. When you load the ccbi-files in cocos2d, cocos2d will select the -hd or -ipadhd (for retina iPad) resources. This is the reason why you won't have the option to select -ipadhd as an extension when editing your resolution settings.

Content Sizes

By default cocos2d only supports absolute sizes for setting the contentSize of a node. Included with CCBReader is an extension which allows setting the contentSize relatively to the nodes parent. (If you want to use this feature programmatically, include the CCNode+CCBRelativePositioning.h file.)

There are six different options for setting the size of a node in CocosBuilder:



1. Absolute: This sets the size to an absolute value (in points).
2. Percentage of container size: This sets the content size to a percentage of its parents content size. If it is the root node of the ccb-file it will (by default) be a percentage of the devices screen size.
3. Relative container size: This option will calculate the nodes content size by subtracting the width/height values that you enter with the width/height value of the parent node. It can also be seen as an inset of the parents content size.
4. Horizontal percentage of container, fixed height: The width is set as a percentage, while the height is absolute (points).
5. Vertical percentage of container, fixed width: The height is set as a percentage, while the width is absolute (points).
6. Multiply by resolution scale: This option multiplies the value that you enter by the resolution scale factor which is specified for the current resolution.

Relative Positioning

All relative positioning in CocosBuilder is in relation to the position and content size of a nodes parent. In particular, if you are using the percentage option it will only work if the parent node has a size.



1. Absolute: This sets the position to an absolute value (in points), same as cocos2d would normally use.
2. Relative top-left: This sets the position relative to the top-left corner of the parent node.
3. Relative top-right: This sets the position relative to the top-right corner of the parent node.
4. Relative bottom-right: This sets the position relative to the bottom-right corner of the parent node.
5. Percentage of container: Uses a percentage of the parent's content size to set the position relative to the bottom-right corner of the parent node. E.g. using the value 50,50 will place the node in the middle of its parent.
6. Multiply by resolution scale: This option multiplies the value that you enter by the resolution scale factor which is specified for the current resolution.

Relative Scaling

You can use relative scaling for any object's scale or for some float based properties (e.g. the font size of CCLabelTTF).



1. Absolute: CocosBuilder will use the scale you provide regardless of which resolution is currently used.
2. Multiply by resolution scale: The value will be multiplied by the resolution scale factor for the current resolution.

Options When Loading ccbi-files

The resolution sizes are not saved in the ccbi-files, by default the screen size is used as the parent size when the files are loaded. If you have used a custom size you may need to pass this size to the loader. To do this you will need to use the nodeGraphFromFile:owner:parentSize: or sceneWithNodeGraphFromFile:owner:parentSize: methods.

CGSize mySize = CGSizeMake(100.0f, 100.0f);

CCNode\* myNode = [CCBReader nodeGraphFromFile:@"myNode.ccbi" owner:NULL parentSize:mySize];

Before loading your ccbi-files you can set the resolution scale you want to use. The default resolution scale is 1 for iPhone and 2 for iPad, but sometimes it can be useful to use other scale factors.

[CCBReader setResolutionScale: 2.5f];

Useful Tips!

* It is always better to design for multiple resolutions from the start in a project, rather then trying to convert an existing layout to fit different devices.
* If you are planning on using letter boxing, the multiply by resolution scale option can be very useful.
* Combine setting the anchor point of an object with the relative positioning options to pin nodes to corners or sides of the screen.
* You can achieve very complex behaviors for the multiple resolutions by nesting different positioning and size options.

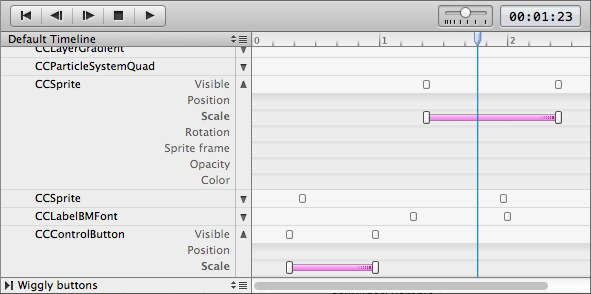
Don't be afraid to experiment with the different options, it can be complex at first sight, but once you get the hang of it you will have many options for laying out your scenes.

**6.Working with Animations**

You can use CocosBuilder for creating character animations, animating complete scenes or just about any animation you can imagine. The animation editor has full support for multiple resolutions, easing between keyframes, boned animations and multiple timelines to name a few of the features.

The Basics

In the bottom of the main window you can find the timeline. You use the timeline to create your animations.



By default your ccb-file has a single timeline that is 10 seconds long. CocosBuilder edits animations at a frame rate of 30 frames per second, but when you play back the animation in your app it will use whatever you have set cocos2d to use (the default in cocos2d is 60 fps). The current time is displayed in the top right corner, and has the format minute:second:frame. The blue vertical line also shows the current time. Click the time display to change the duration of the current timeline.

Adding Keyframes

Animations in CocosBuilder are keyframe based. You can add keyframes to different properties of a node and CocosBuilder will automatically interpolate between the keyframes, optionally with different types of easing.

To add a keyframe, first expand the view of the node by clicking the triangle to the right of the name of the node. This will reveal all the animatable properties of the node. What can be animated varies slightly depending on what type of node you have selected. Once the properties are visible you can click the property in the timeline with the option key held down. This will create a new keyframe at the time of the click. Alternatively, you can create a new keyframe at the time of the time marker by selecting a node then choosing Insert Keyframe in the Animation menu.

Keyframes are automatically added at the current time if you transform a node in the canvas area, given that the transformed property already has one or more keyframes in the timeline.

Editing Keyframes

You edit a specific keyframe of a node by moving the time marker to the time of the keyframe and selecting the node. You can focus on a keyframe by double clicking it (which will select the node and move the time marker).

You can select keyframes and move them together by dragging a selection box around them. You can also copy and paste keyframes between nodes. Make sure you only have one selected node when pasting the keyframes. The keyframes will be pasted starting at the time of the time marker.

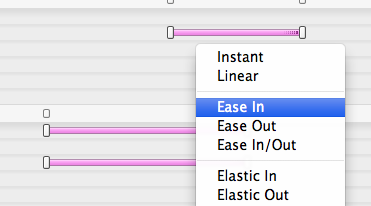
If you have selected a set of keyframes it is possible to reverse the order of them by selecting Reverse Selected Keyframes in the Animation menu. Use the Stretch Selected Keyframes… option to speed up or slow down an animation by a scaling factor.

Importing a Sequence of Images

If you have an animation created by sprite frames it can be tedious to move each individual frame to the timeline. CocosBuilder simplifies this process by automatically importing a sequence of images. Select the frames that you want to import in the left hand project view, then select a CCSprite in the timeline. Now choose Create Frames from Selected Resources in the Animation menu. The frames will automatically be created at the start of the marker. If you need to slow down the animation, select the newly created keyframes and use the Stretch Selected Keyframes… command.

Applying Easing

CocosBuilder offers a carefully selected subset of the easings provided by cocos2d. To apply an easing right click between two keyframes and select the type of easing that you want to apply.

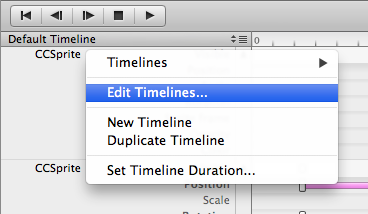


Some of the easings have additional options, after the easing has been applied you can right click again and select Easing Setting… from the popup menu.

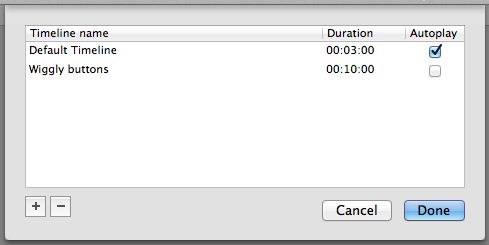
Using Multiple Timelines

A very powerful feature of CocosBuilder's animation editor is the ability to have multiple timelines in a single file. You can name the different sequences and play them back from your code by using their name. It's even possible to smoothly transition between the different timelines.

To select, add or edit your timelines use the timeline popup menu:



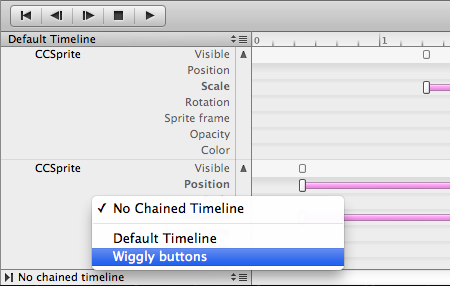
In the edit timelines dialog you can get an overview of your timelines, rename them, add new ones and (optionally) set one of the timelines to automatically start playback directly when the ccbi-file is loaded by your app.



Properties in timelines that do not have keyframes set share their values across timelines. E.g. if you move one node in one timeline it will be moved in all timelines as long as they do not have a keyframe set for the position property. It can sometimes be useful to add a single keyframe to a property just to override the shared value for a specific timeline.

Chaining Timelines

You can automatically play back a sequence of timelines by chaining them. You can also use this feature for automatically looping a timeline.



To have a timeline play in sequence, click the No chained timeline text and select the timeline you want to play right after the current one.

Playing Back Animations in Code

To programmatically control the animations you create with CocosBuilder you will need to retrieve the CCBAnimationManager. The animation manager will be assigned to the nodes userObject when the ccbi-file is loaded.

CCNode\* myNodeGraph = [CCBReader nodeGraphFromFile:@"myFile.ccbi"];

CCBAnimationManager\* animationManager = myNodeGraph.userObject;

The animation manager will be returned as an autoreleased object. To play back a specific timeline call the runAnimationsForSequenceNamed: method. If a timeline is currently playing it will be immediately stopped when calling this method.

[animationManager runAnimationsForSequenceNamed:@"My Timeline"];

Optionally, you can use a tween duration to smoothly transition to the new timeline. Where possible linear interpolations will be used for the transition.

[animationManager runAnimationsForSequenceNamed:@"My Timeline" tweenDuration:0.5f];

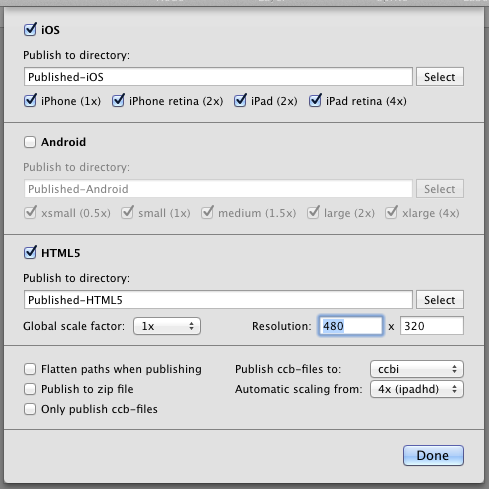
It is also possible to receive a callback whenever a timeline has finished playing. You will receive these callbacks even if another timeline is chained in sequence. Use the CCBAnimationManagerDelegate to receive the callbacks.

**7.Automatic Scaling and Sprite Sheets**

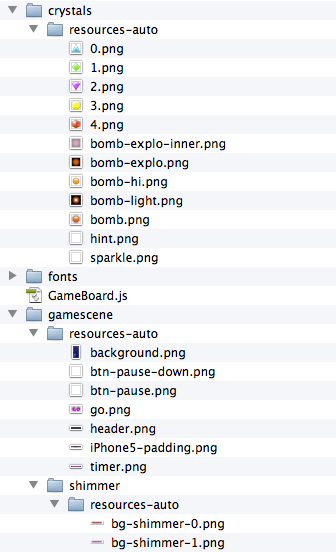
CocosBuilder can manage your assets and automatically scale them for different devices and resolutions. It can even create and scale the sprite sheets for you.

Automatic Scaling

To have CocosBuilder automatically scale your assets, first set your design resolution in your Publish Settings. CocosBuilders reference scale is the original iPhone resolution (equals 1x). By default images will automacially be scaled down from a design resolution of 4x, or the iPad retina display. You can change this, but keep in mind that using a lower resolution can force CocosBuilder to scale up images, which will make them look bad on high resolution devices. In the publish settings you can also choose which resolutions you want to export your images and sprite sheets to.



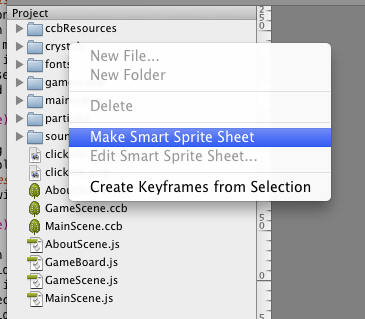
For the automatic scaling to work you will need to tell CocosBuilder that the images are in design resolution. You do this by putting the images in a sub-directory named resources-auto. Even though the resources are in the sub-directory, CocosBuilder will show them as being in it's original folder.



If you are not happy with the automatic scaling for some of the resources, it's always possible to override it. Simply create a directory next to the resources-auto directory and name it after the resolution you want to override. The images that you put in this directory will override any images in the resources-auto directory. E.g. to override images for iPhone retina devices, name the directory resources-iphonehd.

Smart Sprite Sheets

CocosBuilder can quickly turn a directory of images into a sprite sheet. To do this, simply right click the directory in the Project view and select Make Smart Sprite Sheet. The directory should only contain images (and optionally sub-directories for different resolutions or automatic scaling as described above).



The folder will turn pink and have a happy smiley face to indicate that it is a smart sprite sheet. To edit the properties of the sprite sheet, right click it again and select Edit Smart Sprite Sheet…. You can choose from a selection of output formats, compression and dithering. Dithering will only be used for lossy compression pvr formats. HTML 5 still doesn't support pvr textures, therefore PNG images will be exported for HTML 5 regardless of settings.

If you need to access the smart sprite sheet's images from code, the name of the sprite sheet will be the name of the directory with the .plist extension added. The names of the sprite frames will be the name of the full path to the image, including the directories that it is in (but excluding any resource specific directories, such as resources-auto)\_.

Sound Conversions

Sounds needs to be saved in different formats for different platforms to play back optimally. CocosBuilder conveniently handles this for you. To use the automatic conversion feature, make sure that all your sound effects are saved in wav format. Longer sounds, such as background music should be saved in mp3 format. When the sounds are published they will be converted to suitable formats for the different platforms.

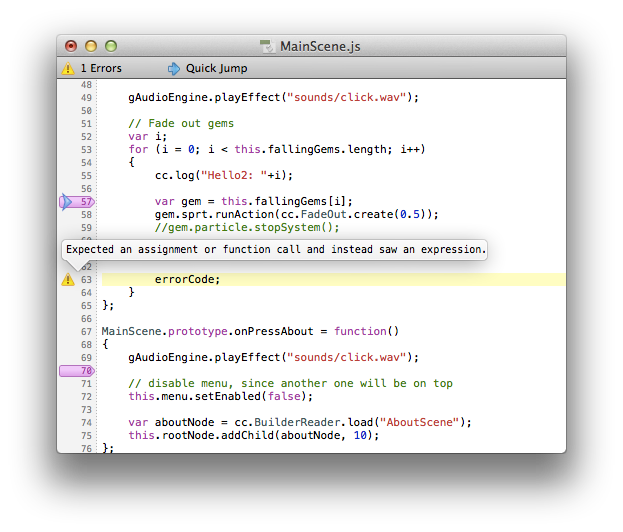
**8.Debugging JavaScript Code**

The latest version of CocosBuilder and CocosPlayer adds support for debugging JavaScript code. Make sure that debugging is enabled when building CocosPlayer (see the README.md file in the CocosPlayer folder for more details).

Setting Breakpoints

CocosBuilder will automatically connect to the debugger of CocosPlayer when you run your project. There are no extra steps required.

To set a breakpoint, simply click the line numbers where you want to break in CocosBuilder's text editor. When your game hits the breakpoint execution will halt and the line will be highlighted in the text editor. To continue execution hit the play button or the step button in the CocosPlayer Console.



Inspecting variables

When a breakpoint has been hit and execution halted, you can inspect the variables of the current scope. To do this, use the JavaScript command line in the CocosPlayer Console. Whatever you type on the command line will be evaluated in the current context and the returned value will be printed in the console. To inspect a variable, simply type the name of the variable and hit the enter key.

Known issues

The JavaScript debugger is in alpha state, and there may still be bugs and instabilities.

1. Occasionally, CocosPlayer will need to be restarted
2. Breakpoints set on invalid lines will be ignored (e.g. empty lines)
3. Breakpoints are not sent to CocosPlayer until after execution of the program has started, breakpoints set in startup code may be missed

If a value of a variable is false or 0, the debugger will output an invalid return from eval response

**9.Get the Source**

CocosBuilder is an open source project released under MIT license. You can get the complete source code from [GitHub](https://github.com/cocos2d/CocosBuilder). To build CocosBuilder you first need to download and install git. You can get it from [here](http://git-scm.com/download/mac), if you do not have it installed already.

Run the following commands to download the complete source code:

git clone https://github.com/cocos2d/CocosBuilder.git

cd CocosBuilder

git submodule update --init --recursive

When pulling updates you may also need to update the submodules. To get the latest version of CocosBuilder run the following commands (when inside the CocosBuilder directory):

git pull

git submodule update --init --recursive

CocosBuilder is developed on the develop branch, once the develop is reasonably stable it is merged into the master branch. When a new version it gets its own branch. For instance v2 holds the code for version 2.0.

Contributing

If you are contributing bug fixes or other features, please send a pull request to the develop branch. For bug fixes and smaller additions, it is fine to just send a pull request. If you are planning a larger addition please post in the cocos2d forums before starting so that the addition is in line with the overall roadmap and that nobody else is currently working on the same features.

**X1.CocosBuilder - Create Node Plug-in**

Node plug-ins can be used to add custom type of objects that can be used in and exported from CocosBuilder. The objects needs to be a direct or indirect sub-class of CCNode.

Setting up the project

The easiest way to setup your Xcode project for a new plug-in is to duplicate the example project in the PlugIn Nodes folder.

Now, open the project and slowly click twice on the project name in the file view to rename it. Name it after the class you want to create a plug-in for.

You will be asked to rename some project content items. Leave everything checked and click Rename. You now have a new project for your plug-in.

To compile and test the plug-in, first make sure that you have built a copy of CocosBuilder and that it is located in the build directory. The click the Run button in the plug-in project. This will compile the plug-in and copy it into CocosBuilder's PlugIns folder (inside the app bundle). For testing and debugging, double click the CocosBuilder program in the build directory. You can see the output of the program using Console.app, set the filter to CocosBuilder to avoid output from other applications.

Basic plug-in structure

To make a working node plug-in, you will need to add your class to the plug-in project and edit the CCBPProperties.plist file. When loading your custom class CocosBuilder/CCBReader will create it using alloc and default init method, then assign all the object's properties. Therefore, it is required that your class can be initialized using the init method only (without using a custom init-method).

The classes you add to a plug-in is linked in runtime against the Cocos2d library, only the header files are included in the project. If your node-object uses more than one class, those classes shouldn't be included in other plug-ins or there may be conflicts when loading the plug-ins.

For many types of nodes you can just drop their classes into the plug-in, setup their editable properties in the CCBPProperties.plist file and you are good to go. Sometimes it may be necessary to use a different class for displaying a node in the editor then when loading it into an app. Most often, it is easiest to use a sub-class of your node and override some of it's behavior. E.g. you can disable animations or user actions in the sub-class. If you use this approach you should name your sub-class with the CCBP prefix (e.g. CCBPMyCocosNode if your class is CCMyCocosNode).

Adding a plug-in built into CocosBuilder

Note that only plug-ins that are very general should be included in CocosBuilder by default. These are, for instance, core cocos2d classes and the GUI components bundled with CocosBuilder.

In order to create a built in plug-in, you need to perform the following steps:

1. Create a new Bundle target for the plug-in, name it after the plug-ins main class.
2. In the plug-in target's build settings, set "Wrapper Extension" to "ccbPlugNode" and set "Other Linker Flags" to "-undefined dynamic\_lookup".
3. Select the CocosBuilder target and go to Build Phases. Add your new plug-in target to the Target Dependencies by dragging and dropping from the Targets on the left side.
4. Also in the Build Phases, add your plug-in it to the Copy PlugIns phase.
5. When adding classes and resource to the plug-in, be carful to only add them to the plug-in target.

CCBPProperties.plist format

Most of the plug-in magic happens in the CCBPProperties.plist file. It defines all properties of your class that can be edited in CocosBuilder and other information about the plug-in. The file is structured as follows.

Required keys

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| className | String | The name of the main class when loaded into an app, e.g. CCMyCocosNode |
| editorClassName | String | The name of the class used by the editor (often the same as className), e.g. CCBPMyCocosNode |
| inheritsFrom | String | The name of the class's super class, this is used to display the inspector panels of the super class, e.g. CCSprite |
| canHaveChildren | Boolean | Yes, if it should be possible to add children to this node in CocosBuilder |
| properties | Array | List of PlugInProperty as described below |

Optional keys

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| propertiesOverridden | Array | The format is the same as for the properties-key, but it replaces a property of a super class |
| spriteFrameDrop | Dictionary | Specifies what class of object will be created when a sprite frame is dropped on the node. E.g. CCMenuItemImage for a CCMenu, see SpriteFrameDrop described below for details |
| requireChildClass | Array | Array of String:s that defines which classes are allowed as children for this node. E.g. CCMenu only allows CCMenuItemImage as children |
| requireParentClass | String | Specifies if this node can only be added as child to a specific type of node. E.g. CCMenuItemImage can only be added to CCMenu. |

PlugInProperty

A PlugInProperty defines how a property should be displayed in CocosBuilder and how it should be loaded into an app by CCBReader. It is a dictionary with the following keys. Which property type:s are supported and how they are serialized (for the default value) is defined in the Property Types document.

Required keys

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| type | String | The property type, e.g. Point or Float |
| displayName | String | What label should be associated with the type, e.g. Content Size |
| name | String | The name of the property to set on the node (not required for type Separator, SeparatorSub and StartStop) |

Optional keys

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| default | n/a | Default value, serialized as described in the Property Types document |
| readOnly | Boolean | Set to YES if this property should be read only, e.g. YES for contentSize in CCSprite |
| dontSetInEditor | Boolean | If set to YES this property will not be set or read from the node by the editor. Instead, the value will be saved in a separate variable. Requires the default value to be specified |
| platform | String | Set this key if the property should only be used on a specific platform. Valid values are Mac or iOS. |
| affectsProperties | Array | Array of String:s that defines which other properties should be updated when this value changes. E.g. when changing the texture of a sprite, it affects its contentSize |
| extra | String | Different uses for different type:s, see the Property Types document for details. |

SpriteFrameDrop

The SpriteFrameDrop structure is used to specify the behavior when a sprite frame is dropped onto a scene from the assets palette.

Required keys

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| className | String | The name of the class to create, e.g. CCSprite |
| property | String | The name of the property to assign the dropped sprite frame to, e.g. displayFrame for CCSprite |

**X2.CocosBuilder - Create export plug-in**

CocosBuilder has support for creating custom export plug-ins. Bundled with CocosBuilder comes the default ccbi-exporter, but it is easy to add new output formats.

Setup a new plug-in project

The easiest way to setup a new plug-in project is to duplicate and rename the example plug-in. Open the PlugIn Exporters folder and duplicate the Export Example folder. Rename it to what you want your new export plug-in to be called.

Now open the Xcode project in the new folder. Rename the project by clicking slowly twice on the projects name. The name you use here is what will be displayed in the save dialog when selecting Publish As… in CocosBuilder.

You will now be asked to rename various project content items. Keep everything selected and click the Rename button.

The plug-in uses a custom class to export your CocosBuilder document. This class's name needs to be unique for each plug-in. Therefore, you will need to rename the plug-in's main class. This is easiest done by opening the file, right clicking on the class's name and selecting Refactor -> Rename.

Enter a name that corresponds to your plug-in's name and use the CCBX prefix. You will also need to open the Info.plist file and change the Principal class property to the name of your new class.

To compile and test the plug-in, first make sure that you have built a copy of CocosBuilder and that it is located in the build directory. The click the Run button in the plug-in project. This will compile the plug-in and copy it into CocosBuilder's PlugIns folder (inside the app bundle). For testing and debugging, double click the CocosBuilder program in the build directory. You can see the output of the program using Console.app, set the filter to CocosBuilder to avoid output from other applications.

Write the plug-in

The export plug-in is very simple in it's structure. It's main class should contain two methods that CocosBuilder will call. The first method, extension, simply returns a string with the extension that the exported file should use. For instance, this is what the default ccbi exporter uses.

- (NSString\*) extension

{

return @"ccbi";

}

The second method, exportDocument: is where all the magic of the exporter happens. It will receive a NSDictionary that contains the complete document. The document structure can be found in the CCB File Format. The exportDocument: should create and return an autoreleased NSData object with the contents of the file to export. The example file uses NSPropertyListSerialization to create a plist representation of the document (same as ccb file save).

- (NSData\*) exportDocument:(NSDictionary\*)doc

{

return [NSPropertyListSerialization dataFromPropertyList:doc format:NSPropertyListBinaryFormat\_v1\_0 errorDescription:NULL];

}

**X3.CocosBuilder - CCB File Format**

This is a description of the CocosBuilder save file format. It is also used internally in CocosBuilder and is used by export plug-ins. Published files are (by default) saved in the CCBi file format.

Structure

CCB files are simple Plist-files with a ccb extension. To dissect a CCB file in Xcode, simple rename the extension to .plist and open it with Xcode.

Document (top level)

The root node in a CCB file is a dictionary. It contains the following properties.

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | | **Type** | **Comment** |
| fileType | String | | Must be CocosBuilder |
| fileVersion | Number | | Must be 4 |
| stageWidth | Number | | The width of the stage in points |
| stageHeight | Number | | The height of the stage in points |
| centeredOrigin | Boolean | | Yes if the stage origin is centered, no if it is bottom left |
| exportPlugIn | String | | Optional, the extension of the export plug-in last used |
| exportPath | String | | Optional, the full path to the file last published |
| nodeGraph | Dictionary | | Node (as defined below) |
| guides | Array | | Array of Guide (as defined below) |
| notes | Array | | Array of Note (as defined below) |
| resolutions | Array | | Array of ResolutionSetting (as defined below) |
| currentResolution | Number | | Index of the currently selected resolution setting |
| sequences | Array | | Array of Sequence (as defined below) |
| currentSequenceId | Number | | Id of the currently selected sequence |

Node

Represents a CCNode or a sub class of a CCNode.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| baseClass | String | Name of base class plug-in used, e.g. CCNode or CCSprite |
| customClass | String | Name of super class defined in editor, e.g. HelloCocosBuilder |
| memberVarAssignmentName | String | Name of member var this node is assigned to |
| memberVarAssignmentType | Number | 0 = don't assign, 1 = assign to root node, 2 = assign to owner |
| properties | Array | Array of Property (as defined below), this defines all properties that are assigned to the node |
| children | Array | Array of Node, list of all children's of this node |
| selected | Number | True if node is selected |
| seqExpanded | Number | True if node is expanded, so that animatable properties are visible |
| displayName | String | Name used for displaying the node |
| animatedProperties | Dictionary | Keys are String representations of sequence ids, values are AnimatedProperties (as defined below) |

Guide

Represents a guide in the canvas area.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| orientation | Number | The orientation of the guide, 0 = horizontal, 1 = vertical |
| position | Number | Position of the guide in pixels |

Note

Represents a sticky note in the canvas area.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| text | String | The text on the sticky note (can be left out) |
| width | Number | Width of sticky note in pixels |
| height | Number | Height of sticky note in pixels |
| xPos | Number | x position of the note in pixels |
| yPos | Number | y position of the note in pixels |

ResolutionSetting

Represents a resolution setting.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| name | String | The name of the resolution setting |
| width | Number | Width in points of the resolution |
| height | Number | Height in points of the resolution |
| ext | String | Extensions separated by space for use with the resolution |
| scale | Number | Resolution scale factor |
| centeredOrigin | Number | Has centered origin |

Sequence

Represents information about a timeline.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| scale | Number | The scale currently used for displaying the sequence |
| offset | Number | Offset by current horizontal scroll |
| length | Number | Length of sequence (in seconds) |
| resolution | Number | Resolution of timeline (should be 30) |
| name | String | Name of the sequence |
| sequenceId | Number | Id number of the sequence |
| chainedSequenceId | Number | Id number of the sequence to be played after this one, -1 if none is selected |
| autoPlay | Number | True if this sequence should be automatically played when the ccbi file is loaded |

Property

Represents a property of a Node. Too see how different properties are serialized, please see Property Types.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| type | String | Name of the type of property, property types are described below |
| name | String | Name of the property, e.g. contentSize or position |
| value | n/a | Different property types are serialized in different ways |
| platform | String | Optional, can be iOS or Mac. Set if a property is only supported for a specific platform |

AnimatedProperties

Represents the animated properties of a node, for a specific sequence. Keys are the name of the properties and values are SequenceNodeProperty (as defined below).

SequenceNodeProperty

Represents an animated property for a specific node and sequence.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| name | String | Name of property |
| type | Number | Type of the property (as described in Animated Properties) |
| keyframes | String | Array of Keyframe (as described below) |

Keyframe

Represents a keyframe.

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| value | Object | Serialization of a value (as describe in Animated Properties) |
| type | Number | Type of the property (as described in Animated Properties) |
| name | String | Name of property |
| time | Number | The time of the keyframe (in seconds) |
| easing | Dictionary | KeyframeEasing (as described below) |

KeyframeEasing

Represents the easing of a keyframe (from the associated keyframe to the following).

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Comment** |
| type | Number | 0 = Instant, 1 = Linear, 2 = CubicIn, 3 = CubicOut, 4 = CubicInOut, 5 = ElasticIn, 6 = ElasticOut, 7 = ElasticInOut, 8 = BounceIn, 9 = BounceOut, 10 = BounceInOut, 11 = BackIn, 12 = BackOut, 13 = BackInOut |
| opt | Number | Options for the easing. Only saved for cubic (rate value) and elastic (period value) easings |

**X4.CocosBuilder - CCBi File Format**

This is a description of the CocosBuilder export (publish) file format for Cocos2d-iphone. It is a binary file format designed to be as compact as possible and very quick to load. This document covers version 4 of the ccbi file format, relased with CocosBuilder 3. If you are implementing or porting a reader for the ccbi file format, you may want to also have a look at the CCBReader, which is the reference implementation.

Basic types

The file format is built upon a set of basic types, such as integers, floats and strings.

BYTE

A single unsigned integer saved in a byte

BOOLEAN

Saved as a 0 or 1 in a single byte

UINT

A unsigned integer, stored using Elias gamma encoding ( http://en.wikipedia.org/wiki/Eliasgammacoding ). Before being stored, 1 is added to the number to support 0 values. The last byte is padded with 0:s so that the next written value is aligned to even bytes.

SINT

A signed integer, stored using Elias gamma encoding. Before being stored, the number is transformed to a positive integer using bijection. 0 -> 1, -1 -> 2, 1 -> 3, -2 -> 4, 2 -> 5, etc.

FLOAT

The first byte of the float defines how it is stored. Possible values are:

1. 0 -> 0.0f
2. 1 -> 1.0f
3. 2 -> -1.0f
4. 3 -> 0.5f
5. 4 -> Saved as INT
6. 5 -> Saved as full 4 byte float number

STRING

Strings are saved as Java-style UTF8:s. The first two bytes define the length of the string, then the string is saved using UTF8 encoding (without a trailing \0);

Strings and String cache

STRING\_CACHE

To save space, all strings are written to a string cache at the beginning of the file.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| UINT | numStrings | Number of strings in the string cache |
| STRING | str[0] | First string in cache |
| STRING | str[1] | Second string in cache |
| … |  |  |
| STRING | str[numStrings-1] | Last string in cache |

CSTRING

A cached string is saved using a single UINT which refers to the strings index in the string cache.

Header

HEADER

The header is used to ensure the file has the right type and version.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| BYTE | magic0 | Must be 'i' |
| BYTE | magic1 | Must be 'b' |
| BYTE | magic2 | Must be 'c' |
| BYTE | magic3 | Must be 'c' |
| UINT | version | Must be 4 |
| BOOLEAN | jsControlled | True if the file is using a JS Controller object |

Animation Sequences

SEQUENCE

Information about one of the sequences in this document.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| FLOAT | duration | Length of the sequence in seconds |
| CSTRING | name | Name of the sequence |
| UINT | sequenceId | Id of the sequence |
| SINT | chainedSequenceId | Id of a chained sequence or -1 if none |

SEQUENCES

Provides information about the animation sequneces (timelines) used by this document. The actual keyframes are saved together with the nodes.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| UINT | numSequences | Number of sequences used in this document |
| SEQUENCE | seq[0] | First sequence |
| SEQUENCE | seq[1] | Second sequence |
| … |  |  |
| SEQUENCE | seq[numSequences-1] | Last sequence |

Node graph

PROPERTY

Represents a property of a node. For type IDs and how they are serialized, see the Property Types document.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| UINT | typeID | ID of the type |
| CSTRING | propertyName | The name of the property |
| BYTE | platform | The type of platform the property is supported for. Can be 0 = any platform, 1 = iOS only, 2 = Mac only |
| n/a | serializedValue | Value serialized as described in the Property Types document |

KEYFRAME

Represents a keyframe used in animations, part of NODESEQUENCEPROPERTY.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| FLOAT | time | The time in seconds of the keyframe |
| UINT | easingType | The type of easing used in between this keyframe and the following. (Easing types defined in table below) |
| FLOAT | easingOpt | Setting used with the easing (rate for cubic and period for elastic). This is only saved if easing type is 2,3,4,5,6, or 7 |
| n/a | serializedValue | Value serialized as describe in the Property Types document under Animated Properties |

Easing types used by KEYFRAME:

|  |  |
| --- | --- |
| **Value** | **Easing type** |
| 0 | Instant |
| 1 | Linear |
| 2 | Cubic in |
| 3 | Cubic out |
| 4 | Cubic in/out |
| 5 | Elastic in |
| 6 | Elastic out |
| 7 | Elastic in/out |
| 8 | Bounce in |
| 9 | Bounce out |
| 10 | Bounce in/out |
| 11 | Back in |
| 12 | Back out |
| 13 | Back in/out |

NODESEQUENCEPROPERY

A property used in an animation sequence for a specific node, contains a set of keyframes. All keyframes are saved ordered by there time.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Type** | **Name** | **Description** | | CSTRING | name | Name of the sequence | | UINT | type | The type of the property (defined in Animated Properties) | | UINT | numKeyframes | Number of keyframes used by this property | | KEYFRAME | keyframe[0] | First keyframe | | KEYFRAME | keyframe[1] | Second keyframe | | … |  |  | | KEYFRAME | keyframe[numKeyframes-1] | Last keyframe | |

NODE\_SEQUENCE

An animation sequence associated with a node.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| UINT | numProperties | Number of animated properties in this sequence |
| NODE\_SEQUENCE\_PROPERTY | property[0] | The first property used in the sequence |
| NODE\_SEQUENCE\_PROPERTY | property[1] | The second property used in the sequence |
| … |  |  |
| NODE\_SEQUENCE\_PROPERTY | property[numProperties-1] | The last property used in the sequence |

NODE

Represents a node or a node graph (if the node has children). Nodes that uses the CCBFile class (sub ccb-files) are handled slightly different when loaded. The node graph associated with the ccbFile property should replace the CCBFile node, but some the properties by the CCBFile node should override the ones in the ccbFile property. The properties that should be overriden are; all extra properties, and position, rotation, scale, tag and visible.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| CSTRING | class | Name of the nodes class |
| CSTRING | jsController | Name of the js controller, only written if jsControlled is set in the HEADER |
| UINT | memberVarAssignmentType | The target this node should be assigned as a variable for. 0 = No target, 1 = documents root node, 2 = owner (as passed to CCBReader) |
| CSTRING | memberVarAssignmentName | The name of the variable this node should be assigned to (only written if memberVarAssignmentType != 0) |
| UINT | numSequences | Number animation sequences saved for this node |
| NODE\_SEQUENCE | sequence[0] | The first sequence of this node |
| NODE\_SEQUENCE | sequence[1] | The second sequence of this node |
| … |  |  |
| NODE\_SEQUENCE | sequence[numSequences-1] | The last sequence of this node |
| UINT | numRegularProperties | Number of regular properties saved for this node |
| UINT | numExtraProperties | Number of regular properties saved for this node |
| PROPERTY | property[0] | The first property of this node |
| PROPERTY | property[1] | The second property of this node |
| … |  |  |
| PROPERTY | property[numRegularProperties+numExtraProperties-1] | The last property of this node |
| UINT | numChildren | Number of children of this node |
| NODE | child[0] | The first child of this node |
| NODE | child[1] | The second child of this node |
| … |  |  |
| NODE | child[numProperties-1] | The last child of this node |

Overall document structure

DOCUMENT

The top structure of a CCBi document.

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| HEADER | header | Document header |
| STRING\_CACHE | stringCache | All strings referenced by this file |
| SEQUENCES | sequences | Information about the animation sequences used in this file |
| NODE | rootNode | The root node of this document (which can also have children) |

**X5.CocosBuilder - Property Types**

This document contains a list of the supported property types of CocosBuilder, and how they are serialized in the CCB file format and in node plug-ins. For serialized values, first the dictionary representation is displayed (for CCB), follow by the binary representation (CCBi). The properties ID number, as used in CCBi is written in parentheses after the type name.

Main properties

The main properties are used both in the CCB file format, in the CCBi file format, for displaying the inspector panel and for creating node plug-ins.

Position (0)

Represents a CGPoint (point), and position type.

Array[ Number(x), Number(y), Number(type) ]

FLOAT x

FLOAT y

UINT type

Point (2), PointLock (3)

Represents a CGPoint (point).

Array[ Number(x), Number(y) ]

FLOAT x

FLOAT y

Size (1)

Represents a CGSize.

Array[ Number(width), Number(height) ]

FLOAT width

FLOAT height

ScaleLock (4)

Represents two float properties, the property names with X and Y appended and if the ratio is locked.

Array[ Number(x), Number(y), Boolean(locked) ]

FLOAT x

FLOAT y

Flip (15)

Represents two BOOL properties, the property names with X and Y appended.

Array[ Boolean(x), Boolean(y) ]

BOOLEAN x

BOOLEAN y

Float (7), Degrees (5)

Represents a float value.

Number(value)

FLOAT value

FloatVar (8)

Represents two properties, the property with the standard name and one with Var appended.

Array[ Number(value), Number(valueVar) ]

FLOAT value

FLOAT valueVar

Integer (6), IntegerLabeled (20)

Represents an int value. (For IntegerLabeled, see extra properties below.)

Number(value)

SINT value

Byte (12)

Represents an int value in the range 0-255.

Number(value)

BYTE value

Check (9)

Represents a BOOL value.

Boolean(value)

BOOLEAN value

Text (18)

Represents a NSString value.

String(text)

CSTRING text

SpriteFrame (10)

Represents a CCSpriteFrame loaded from a sprite sheet or created from a image file.

Array[ String(spriteSheetFile, spriteFile) ]

CSTRING spriteSheetFile

CSTRING spriteFile

spriteSheetFile: Relative name of spriteSheetFile, can be empty string (not NULL).

spriteFile: Name of sprite in sprite sheet, or name of file if if sheet is empty string.

Texture (11)

Represents a CCTexture loaded from an image file.

String(spriteFile)

CSTRING spriteFile

spriteFile: Relative name of image file

FntFile (17)

Represents a FNT file as used by CCLabelBMFont.

String(fntFile)

CSTRING fntFile

fntFile: Relative name of the FNT file

FontTTF (19)

Represents a name of a TTF font as used by CCLabelTTF

String(fontName)

CSTRING fontName

fontName: Name of the Font, e.g. Helvetica

Color3 (13)

Represents a ccColor3B.

Array [ Number ( r), Number (g), Number (b) ]

BYTE r

BYTE g

BYTE b

r, g, b: Color components as integers in the range 0-255

Color4FVar (14)

Represents two ccColor4F properties, the latter with "Var" appended.

Array [

Array[ Number ( r), Number (g), Number (b), Number (a) ],

Array[ Number ( rVar), Number (gVar), Number (bVar), Number (aVar) ]

]

FLOAT r

FLOAT g

FLOAT b

FLOAT a

FLOAT rVar

FLOAT gVar

FLOAT bVar

FLOAT aVar

r, g, b, a: Color components as float in the range 0-1

rVar, gVar, bVar, aVar: Color components as float in the range 0-1, assigned to property with "Var" appended to it's name

Blendmode (16)

Represents a ccBlendFunc property with src and dst components.

Array[ Number(src), Number(dst) ]

UINT src

UINT dst

src, dst: Blend func components as integers.

Block (21)

Represents a block property. The block will contain a callback to a target and selector and is created when loading the ccbi file in the app.

Array[ String(selector), Number(target) ]

CSTRING selector

UINT target

selector: Name of selector target: Target is 0 = None, 1 = Root node, 2 = Owner

Extra properties used by plug-ins

The extra properties are used only by plug-ins to display additional information in the inspector panel.

Separator

The separator is only in CocosBuilder to display a separator/header in the inspector panel. Only its displayName is used and name can be left out.

SeparatorSub

This property is identical to Separator, only it is not as prominent and hasn't got a background color.

IntegerLabeled

This property allows a plug-in to present a drop down menu for setting an integer. The integer is serialized identically to an ordinary Integer type, but the property has the extra field set. The extra field has the the format: < name 1 >|< int value 1 >[|< name 2 >|< int value 1 >[|…]]. For example (as used in CCParticleSystem):

Gravity mode|0|Radius mode|1

StartStop

The StartStop property is used to display a start and stop button in the inspector panel. Set the names of the buttons using the displayName, but with a | separating the two names. For instance:

Start Particles|Stop Particles

Clicking the buttons will cause different selectors of the selected node to be called. Which selectors to call is specified by the extra field, and the two selectors are separated by |. E.g:

resetSystem|stopSystem

Animated Properties

Animated properties are saved somewhat differently from normal properties and are used in keyframes and base values.

Visible (2)

Represents a BOOL value.

Boolean(value)

BOOLEAN value

Degrees (2)

Represents a float value.

Number(value)

FLOAT value

Position (3)

Represents a CGPoint (point), the position type is set by the base value.

Array[ Number(x), Number(y), Number(type) ]

FLOAT x

FLOAT y

ScaleLock (4)

Represents two float properties, the property names with X and Y appended, if it is locked is set by the base value.

Array[ Number(x), Number(y) ]

FLOAT x

FLOAT y

Byte (5)

Represents an int value in the range 0-255.

Number(value)

BYTE value

Color3 (6)

Represents a ccColor3B.

Array [ Number ( r), Number (g), Number (b) ]

BYTE r

BYTE g

BYTE b

r, g, b: Color components as integers in the range 0-255

SpriteFrame (7)

Represents a CCSpriteFrame loaded from a sprite sheet or created from a image file.

Array[ String(spriteFile, spriteSheetFile) ]

CSTRING spriteSheetFile

CSTRING spriteFile

spriteSheetFile: Relative name of spriteSheetFile, can be empty string (not NULL).

spriteFile: Name of sprite in sprite sheet, or name of file if if sheet is empty string.

**XX1.CocosBuilder 3 Roadmap**

Overall goal is to create a complete stable integrated environment for Cocos2d JS (and cocos2d-x/iphone).

JavaScript improvements

Integrated visual debugger

Set and remove breakpoints from the text editor, automatically connect to the debugger using the IP number received from rendevouz & CocosPlayer connection. Send debugger commands through JS console and display returned results in CocosBuilder console. Also improve display of cc.log, and possibly hide output from CCLog. Buttons for continue, step, show current line in text view.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/5)

Improved text editor

In the text editor, add support for:

1. Better syntax coloring
2. Syntax check using jslint or jshint
3. Better autocomplete, with all cocos2d default objects built in (and if possible - using output from jslint/jshint)
4. Drop down menu for quick jumps to different functions in the file
5. Better text editor support for JS Debugger

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/11)

Export sounds to multiple formats

Currently all sounds are converted to mp3 for HTML5 by the publisher. It will need to be converted to ogg also, as mp3 is not supported by all browsers.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/27)

Scaling options for Android

Add different preset options for how to scale the graphic context for Android. These should include (but there could be more options):

1. Using the native resolution of the device
2. Using the closest iOS resolution and letterbox
3. Using the closest iOS resolution's width and adjust the height to keep the x/y aspect ratio

This depends on [Cocos2d-x plist-file for initial configuration](https://github.com/cocos2d/cocos2d-js/issues/6) being done.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/28)

Include Chipmunk libs in HTML5 publish

CocosBuilder should include Chipmunk as part of publishing for HTML5. This should be optional and set with a checkbox in the project settings.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/39)

Run in browser

Add a menu option to run a project directly in the web browser.

1. This includes, publishing to HTML5 (already done)
2. Running a web-server at some local port
3. Opening a web browser (possibly configurable which one, or selecting from a sub - menu) with the correct local address to the published files

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/12)

Ease of use and polish

Preview of assets

When a file is selected in the project view, a preview of the file should be displayed. Preview includes:

* Image, sound etc
* Size of image or sound
* Different versions of an image (iphone, iphonehd etc)

Easier to create new ccb- and JS-files

Add button for creating new files at bottom of project view.

Add files by drag and drop

Make it possible to drag and drop files between folders, and to import images to Resources folder by dropping them on the project view. It should also be possilbe to delete files from within CocosBuilder.

Hide/lock layers

Add option to hide and/or lock a node in the node graph. A hidden node is not visible in CocosBuilder, but the visible property will not be affected on export. This feature is just to aid editing.

Save all

Option to save all open CCB files (without closing them).

Save files before publishing

Option to save all files before publishing. Currently changes to ccb-files are not exported until the file has been saved.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/13)

Publisher

Options for generating sprite sheets for Android

Add the option to have a different compression settings for sprite sheets when publishing for Android. For instance, it is possible that the user wants to publish to PVRTC on iOS, but these are not supported on Android so it should be possible to use another compression type for Android. In future, different compression types for web could also be an option.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/31)

Error handling

Check for invalid resources

Check for invalid resources, both when used inside CocosBuilder and during publishing process. Improve the error window that is displayed if publishing fails. Catch more error during the publishing process (overflow in sprite sheets, invalid ccb-files etc).

Warn if project is not setup correctly

Display a warning if project is not setup correctly. Specifically when setting the resource path inside the publishing path which will cause infinite recursion.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/32)

CCBReader

cocos2d-x

Simplify connections to cocos2d-x by auto-generating the glue code. Currently the CCBReader for cocos2d-x requires the user to write glue code for mapping name of functions to the actual functions, also when creating classes the class names needs to be mapped to the actual classes. (This is because c++ lacks introspection and reflection.)

Examples of current glue code:

CCB\_MEMBERVARIABLEASSIGNER\_GLUE(this, "sprtBurst", CCSprite \*, this->mSprtBurst);

CCB\_SELECTORRESOLVER\_CCMENUITEM\_GLUE(this, "pressedA:", MenuTestLayer::onMenuItemAClicked);

This glue code could be moved to a single class, which would handle all lookups for selectors, member variables and classes. CocosBuilder should generate this class, thus the user would not need to worry about the glue code. Methods that were not implemented would generate compiler warnings, which also would be helpful when trouble shooting.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/8)

Documentation

Update CocosBuilder documentation

Update CocosBuilder's documetation to include all updates related to CocosBuilder 3.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/29)

Tutorials

Create tutorials for using CocosBuilder. In particular:

* Tutorial for setting everything up and getting started
* Tutorial for animations
* Tutorial for using multiple resolutions in a single file and handling resources
* Tutorial for creating a simple game from start to finish and run it on iOS and Android

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/30)

Possible additions

These are ideas that possibly also could make it into CocosBuilder 3.

Simplified installation of CocosPlayer

Add a menu option to CocosBuilder to automatically install CocosPlayer in Simulator or on a device (user will need to have provisioning profiles and certificates setup for this to work). When hitting Run in CocosPlayer, CocosPlayer could be automatically launched in Simulator if no device is connected.

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/35)

Integration with cocos2d command line tool

Use Cocos2d command line tool for publishing files, creating new projects and possibly communicating with CocosPlayer. (This assumes that the command line tool is finished.)

[Github Issue](https://github.com/cocos2d/cocos2d-js/issues/36)