LiveCode Game Academy - Managing different devices, testing and deploying

Contents

Android

Before you can deploy to Android devices you need to ensure you have Android deployment as part of your license. You also need to set up your system appropriately.

This section contains all the salient details you need to setup your system, but there are step-by-step lessons taking you through this in more detail for each platform here:

http://lessons.runrev.com/s/lessons/m/4069/l/27385-How-do-I-Become-an-Android-Developer-on-a-PC-

http://lessons.runrev.com/s/lessons/m/4069/l/27389-How-do-I-Become-an-Android-Developer-on-a-Mac-

Prerequisites for Android

Software Requirements

If you are intending to use the Android deployment pack on Windows, you will need:

- Windows XP/Vista or Windows 7
- LiveCode 4.6.1 or later
- The Java SDK: http://www.oracle.com/technetwork/java/javase/downloads/index.html.
- The Android SDK: http://developer.android.com/sdk/index.html.

If you are intending to use the Android deployment pack on Mac, you will need:

- Mac OS 10.5.x or later
- LiveCode 4.6.1 or later
- The Android SDK: http://developer.android.com/sdk/index.html

After you have installed the pre-requisites make sure you run the Android SDK Manager and have installed the 'SDK Platform Android 2.2, API 8, revision 2' package.

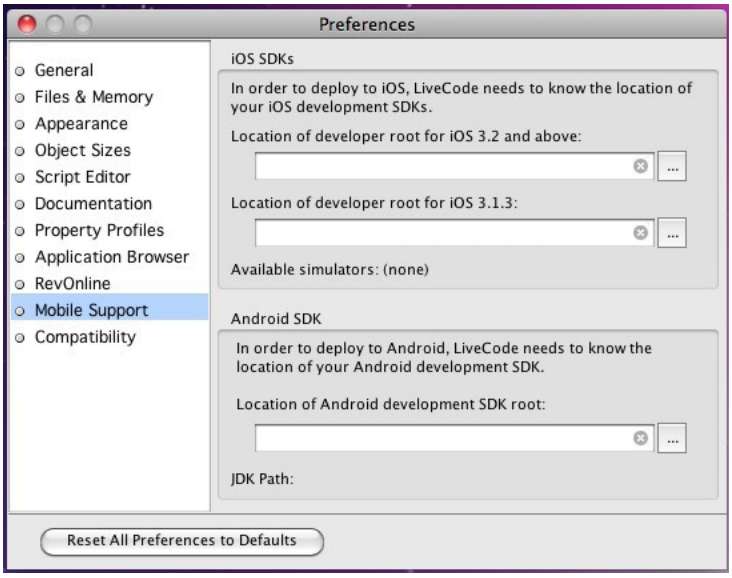
Configuring LiveCode for Android Support

After you have set up your system with the Java Development Kit and Android Development Kit, it is necessary to inform LiveCode where to find them.

Launch the LiveCode IDE and select Edit -> Preferences to launch the Preferences menu. Then select Mobile Support and you are presented with the dialog below.

This dialog allows you to configure the path to the Android SDK root, which you should already have installed. Select ... under Android SDK and choose the directory containing the Android SDK root.

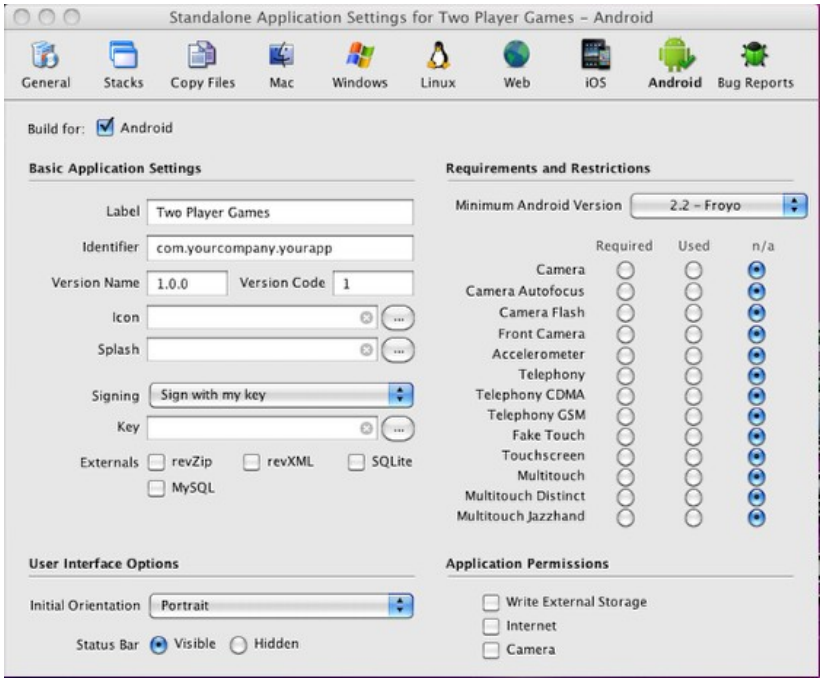
Validation checks are made once you specify the location of the Android SDK root. This ensures that you have selected a valid location and have the required Android components installed.



After doing so, the JDK path should be automatically located. If a path fails to appear for the JDK it means you have not correctly installed or configured your JDK.

Configuring an Android standalone

To configure a stack for Android, you use the Android deployment pane in the *Standalone Application Settings*. Select File -> Standalone Application Settings... from the LiveCode IDE. Then choose Android from the list of deployment options. This raises the Standalone Application Settings dialog for Android specific builds.



You can make a stack build for Android by selecting the Build for Android tick box and configure any other options you wish to include.

You can select the name of your application using the General option at the top of the pane and add files to your build by using the Copy Files option at the top of the pane.

Note: Making a stack build for Android disables building standalone mainstacks for any other non-mobile platforms. If you wish to share code and resources among platforms, you can either factor your application into multiple stacks, using a different mainstack for mobile and desktop targets or you can build your mobile and non-mobile standalones separately.

Note: Inclusions, Copy Referenced Files, Bug Reports and Stacks features are not available when building for Android. Use the CopyFiles feature if you wish to include multiple stack files in your application.

Configuring an emulated device

In order to run an Android project, you need either an emulated device running, or a real device configured for debugging connected.

Creating an emulated device is easily done using the Android SDK Manager that you will have previously installed:

- Make sure the SDK Manager is running.
- Choose 'Virtual Devices' from the left-hand list
- Click 'New...'
- Choose a name for your device
- Set the *Target* to at least Android 2.2 – API Level 8
- Fill in an *SD Card* size.
- Enable the *Snapshot* option (this isn't essentially but significantly speeds up subsequent launches of the emulator!)
- Then click *Create AVD*

After you have performed these steps, your newly created device should appear in the list of existing Virtual Devices, from which you can click *Start...* to launch it.

Any running Virtual Devices will appear in the list of Test Targets under the Development menu.

Configuring a real device

Instead of using the emulator, you can also launch LiveCode Android projects on real Android devices after they have been appropriately configured for debugging.

If you are running on Windows then before you can connect to a real device, you need to ensure the appropriate device driver is installed on your development machine. For details of how to do this see here:

http://developer.android.com/sdk/win-usb.html

If you are running on Mac, there is no need to install any drivers as it 'just works'. Once you have any necessary drivers installed, you must then put your device into debugging mode.

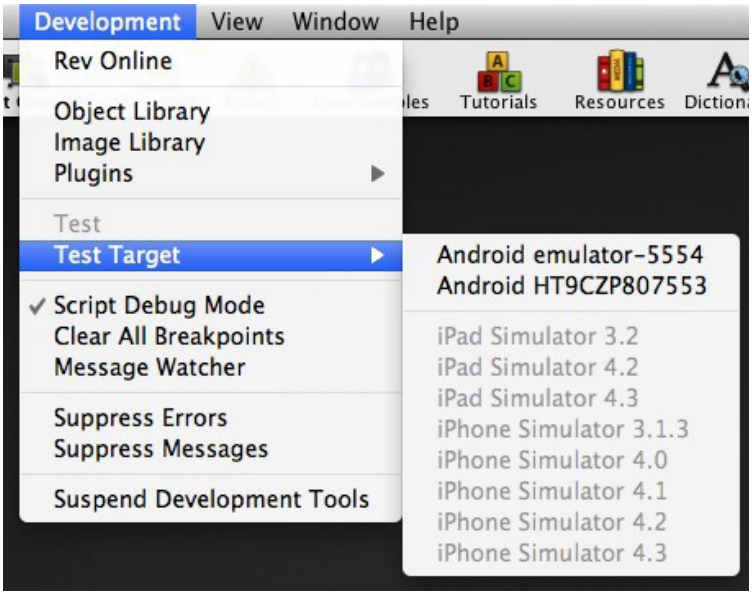
To do this, go to the home screen, press **MENU**, select **Applications > Development** and enable **USB debugging.**

Finally, simply connect your device via USB to your machine and it should (after a few moments) be available in the Test Target list.

Testing an Android application

Once you have your stack configured for Android, you can test it on either a real Android device or in the Android emulator.

First select the emulator or device you want to test on by selecting it from the list of Test Targets

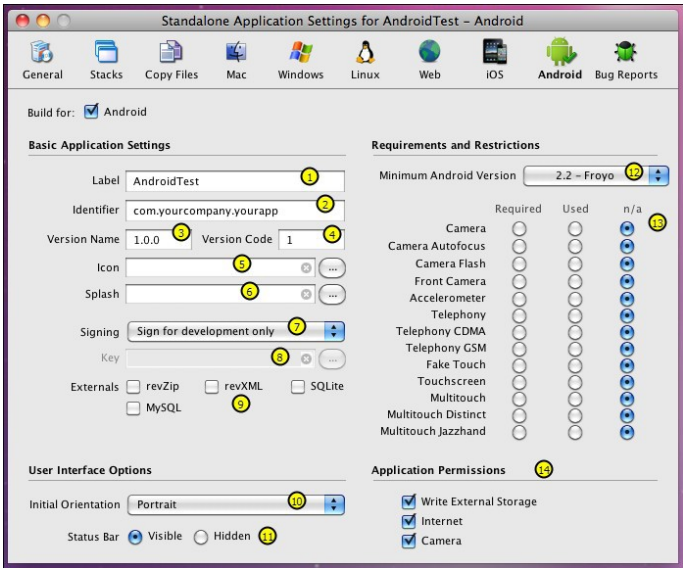


Then deploy the app to the simulator or device by sele by using the *Test* button on the menubar or by selecting Test from the Development menu.

The app will install and launch on the target device.

Configuring an Android Application

All Android applications have a manifest that is built into the package which controls many aspects of the applications requirements and functionality. To set the manifest up, you simply use the options presented in the Standalone Builder's Android pane, these will be used to construct a suitable manifest automatically:



Here the numbered items are as follows:

1. The string to display as the label of the application on the Launcher screen.
2. The unique identifier to use for the application, using the standard reverse domain name convention.
3. A human readable version string for the application.
4. An integer indicating the version of the application – this is used by the OS to compare versions of packages as opposed to the human readable string.
5. The PNG image file to use as the icon on the Launcher.
6. The image file to use in the personal and educational splash screens (this is not used when building with a commercial license).
7. Whether APKs are to be signed with the development key, a custom key or no key (this is only used when using 'Save as Standalone application').
8. The key-store file to use to sign the application when "Sign with my key" is selected at step

1. 7 (this is only used when using 'Save as Standalone application').

9. The extensions to include in the application.

    i. Choose 'revZip' if you are using any of the revZip commands and functions.

ii. Choose 'revXML' if you are using any of the revXML commands and functions.

iii. Choose 'SQLite' if you are using revDB along with the dbSQLite database driver.

iv. Choose 'MySQL' if you are using revDB along with the dbMySQLdatabase driver.

10. The initial orientation to start the application up in.

11. The initial visibility of the status bar.

12. The minimum Android version required by the application.

13. The features that should be added to the manifest. A required feature will only be visible to users who have devices that support the feature. A used feature will indicate to the user that this application uses the feature. A used feature will still be visible to devices which do not support the feature.

14. The permissions to be added to the manifest.

    i. Write external storage is required if your app will read or write files on external storage

2. (e.g. an SD card)

    ii. Internet is required if your app is accessing the internet.

    iii. Camera is required if your app is using any camera features.

Adding a launcher icon

All applications currently installed on an Android device are displayed on the launch screen. To customize the icon used for your application, you should provide an icon image (in PNG format) of size 72x72 and reference it using the appropriate option in the Android standalone settings pane.

Adding a splash image (personal and educational)

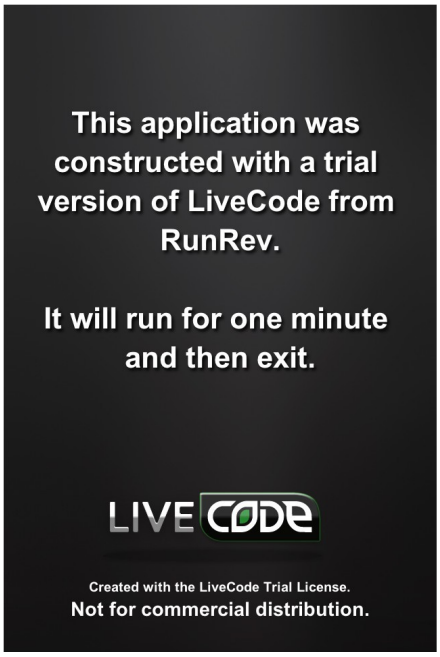
If you are using a personal or educational license, then you are restricted in what can be displayed as the launch image. In this case you should provide a (square) PNG image that will be placed inside a LiveCode branded banner (see below).

We recommend providing an image of 600x600 for the splash – this will give good results when resampled at the various resolutions and sizes required by the different Android devices.

Note: With these license types, the generated launch image will remain on screen for 5 seconds before being dismissed.

Adding a default launch image (trial)

If you are evaluating the Android deployment feature using a trial license, then you cannot configure a splash or launch image. Instead, all such applications will be built with the following launch image:



This image will remain on screen for 5 seconds before the application launches, and the application will quit after one minute.

iOS

Before you can deploy to iOS devices you need to ensure you have iOS deployment as part of your license. You also need to set up your system appropriately.

This section contains all the salient details you need to setup your system, but there are step-by-step lessons taking you through this in more detail here:

http://lessons.runrev.com/s/lessons/m/4069/l/23275-How-do-I-Become-an-iOS-Developer-

Prerequisites for iOS

Before you can use iOS deployment, you need to install the appropriate iOS SDKs available from Apple.

In order to get the iPhone SDK, you need to be 'registered iPhone developer'. You can register for this and download the SDK by visiting:

http://developer.apple.com/ios

LiveCode supports the following iOS SDKs:

Download	Platform	Simulators
Xcode 4.2	Lion	5.0, 4.3
Xcode 4.2 and iOS 5.0	Snow Leopard	5.0, 4.3
Xcode 3.2.6 and iOS 4.3	Snow Leopard	5.0, 4.3, 4.2, 4.1, 4.0

Make sure you have at least one SDK installed, otherwise you will not be able to use the iOS deployment feature.

As of 5.0.0-dp-1, support for iOS 3.x (and subsequently Leopard development) has been dropped. This is in order to fully support iOS 5 testing and development.

As of 5.0.0-dp-2, support for the iOS 5.0 simulator has been added. Though simulator builds take advantage of iOS 5.0 features, device builds do not. Support for iOS 5.0 features in device builds will be added when iOS 5.0 is officially released.

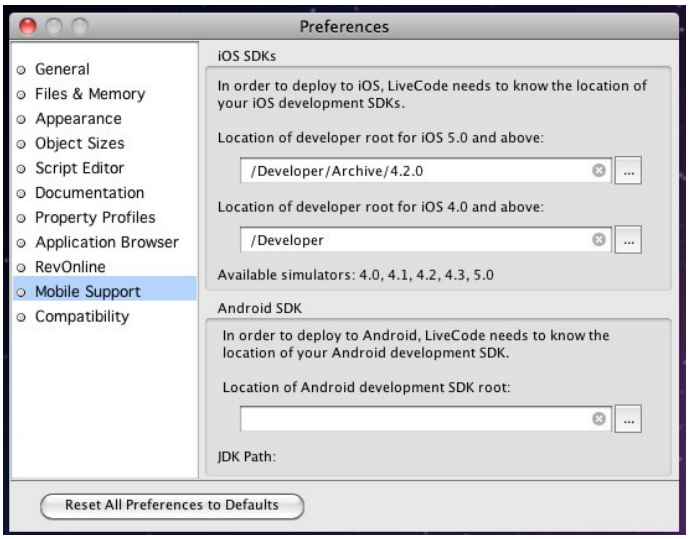
As of 5.0.0-rc-2, device builds use the iOS 5.0 SDK. In order to perform device builds, users must have the iOS 5.0 SDK installed (currently only available with Xcode 4.2).

Note: As a registered iOS developer you will be able to develop and run applications in the iPhone Simulator only. To build applications that can be run on an actual device you will need to enroll in the iOS Developer Programme.

Configuring LiveCode for iOS Support

After you have installed an iOS SDK, it is necessary to tell LiveCode where to find it (or them, if you have installed more than one).

To configure the paths to your installed SDKs, use the Mobile Support panel in Preferences.



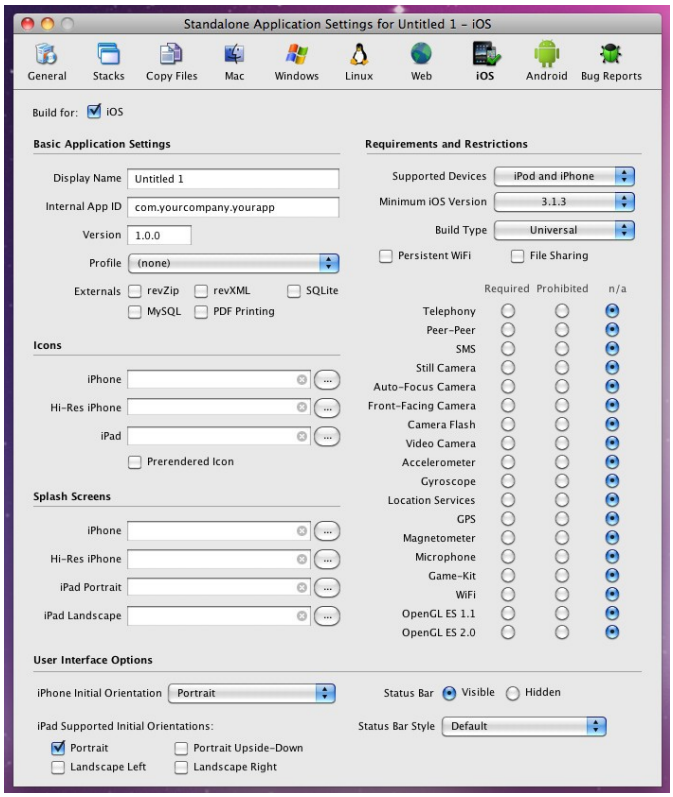
Use this pane to choose the correct SDK paths by using the '…' buttons next to the appropriate one.

You should choose the folder you selected when installing the SDK. (This defaults to '/Developer' in the iOS SDK installers).

When you have successively chosen your SDK(s), the list of simulators that you will have available will be updated.

Configuring an iOS Standalone

To configure a stack for iOS, you use the iOS deployment pane in the Standalone Application Settings dialog, available from the File menu:



This pane allows you to set the iOS-specific options for your application. You can also add files you wish to be included in the bundle using the ***Copy Files*** pane, and set the (bundle) name of your application on the *General* pane.

To make a stack build for iOS, simply check the ***Build for iOS*** button and configure any options that you wish.

Note: Making a stack build for iOS disables building for any other platform, however this is only true of the standalone's mainstack. If you wish to share code and resources among platforms, simply factor your application into multiple stacks, using a different mainstack for iOS and desktop targets.

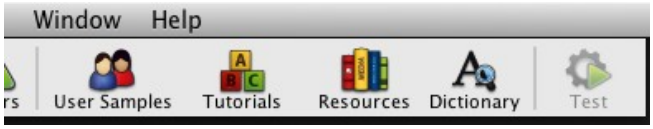
Note: The Inclusions, Copy Referenced Files, Bug Reports and Stacks features are ***not*** available when building for iOS. If you wish to include multiple stackfiles in your application, use the Copy Files feature instead.

Testing in the iOS Simulator

Once you have your stack configured for iOS, you can run it in the iOS Simulator.

First select the simulated device you want ti use for iOS testing from the list of Test Targets in the Development menu.

Then deploy the app to the simulator using the *Test* button on the menubar or by selecting Test from the Development menu.



The app will install and launch on the chosen simulator.

Testing on an iOS Device

Before you can begin testing your application on a real device, you will need to have several things in place:

1. Enrolment in the iPhone Developer Programme: this is required so that you can generate the necessary certificates and profiles.

2. A iPhone Developer Certificate: this is installed on your development machine and is used to digitally sign the application you wish to put onto an iPhoneOS device. Follow the instructions on the Certificates tab of the iPhone Developer Program Portal.

3. Registration of at least one iPhoneOS device in the program portal. You can add devices using the Devices tab of the iPhone Developer Program Portal.

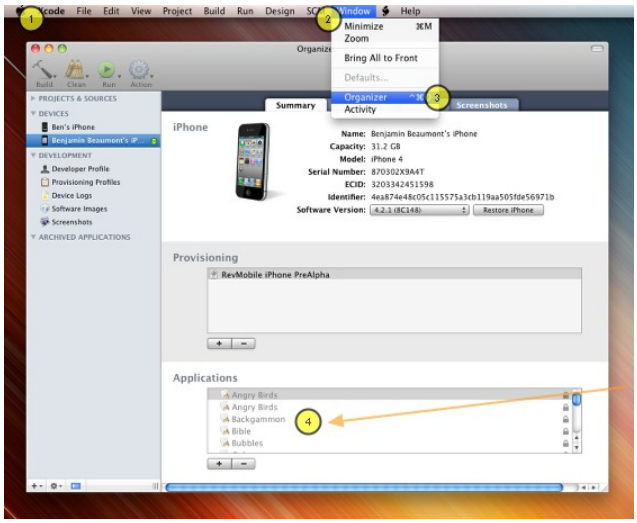
4. An App ID for your application. You can create App IDs using the App IDs tab of the iPhone Developer Program Portal. (Note that at this stage it isn't necessary for you to have a separate App ID for every app – you can use a single id for all your apps for testing/development purposes.)

5. A provisioning profile tying together your test device's id, you app id and your certificate. These can be created using the Provisioning tab of iPhone Developer Program Portal.

Once you have all these things ready, you should find that the 'Profile' drop-down menu in the iOS pane of the Standalone Settings dialog is populated with any provisioning profiles you have installed.

With a suitable profile chosen, you can simply use the Save as Standalone Application... item in the File menu to build an iOS app bundle in the same was as you would build a standalone for any other platform.

The next thing to do is to install the bundle on your test device. To do this, start up Xcode, and choose Window > Organizer. This will bring an interface allowing you to manage the applications, devices and profiles you are using for development.



Next, make sure you have your test device connected to your machine and choose it from the left hand list. If you haven't used the device for development before, you will be prompted to do so, and you'll then be presented with a list of installed applications.

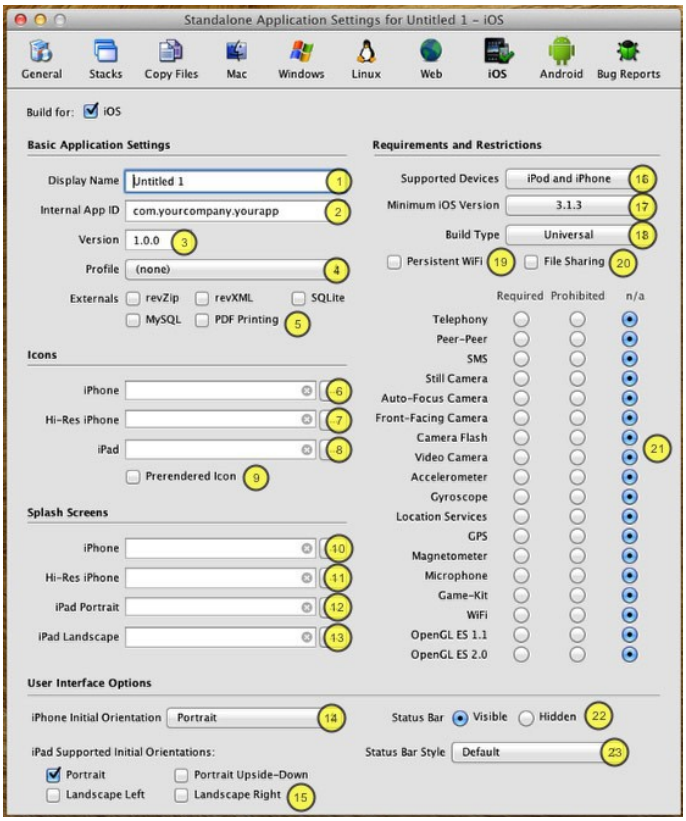
To get your newly prepared application on the device, simply drag the application bundle from the desktop into the Applications list – opting to install the appropriate provisioning profile if it has not been previously installed on the device.

Finally, navigate to the application on your device, and start it up!

Configuring an iOS Application

Setting plist options

All iOS applications have a plist that is built into the application bundle which controls many aspects of the applications requirements and functionality. To set the plist up, you simply use the options presented in the Standalone Builder's iOS pane, these will be used to construct a suitable plist automatically:



Here the numbered items are as follows:

1. The string to display as the label of the application on the SpringBoard (CFBundleDisplayName).
2. The bundle identifier to use for the application, in conjunction with the App Id present in a provisioning profile, this uniquely identifies an application (CFBundleId).
3. The version of the application (CFBundleVersion).
4. The provisioning profile to use when building the application to run on a device.
5. The extensions to include in the application:
    a. Choose 'revZip' if you are using any of the revZip commands and functions.
    b. Choose 'revXML' if you are using any of the revXML commands and functions.
    c. Choose 'SQLite' if you are using revDB along with the dbSQLite database driver.
    d. Choose 'MySQL' if you are using revDB along with the dbMySQL database driver.
6. The icon to display on the iPhone and iPod SpringBoard (CFBundleIconFile and CFBundleIconFiles). This icon should be 57x57 pixels.

7. The icon to display on the Hi-Res iPhone and Hi-Res iPod SpringBoard CFBundleIconFile and CFBundleIconFiles). This icon should be 114x114 pixels.

8. The icon to display on the iPad SpringBoard (CFBundleIconFile and FBundleIconFiles). This icon should be 72x72 pixels.

9. Determines whether the SpringBoard icon already has a tint and gloss applied. (UIPrerenderedIcon).

10. The iPhone and iPod image to use as the launch image (commercial), or the image to incorporate as the splash image (personal and educational), see ***Adding a default launch image*** or ***Adding a splash image*** for more details. This image should be 320x480 pixels and rotated to the initial iPhone orientation setting.

11. The Hi-Res iPhone and Hi-Res iPod image to use as the launch image (commercial), or the image to incorporate as the splash image (personal and educational), see Adding a default launch image or Adding a splash image for more details. This image should be 640x960 pixels and rotated to the initial iPhone orientation setting.

12. The portrait iPad image to use as the launch image (commercial), or the image to incorporate as the splash image (personal and educational), see Adding a default launch image or Adding a splash image for more details. This image should be 768x1024 pixels and rotated to the initial iPhone orientation setting.

13. The landscape iPad image to use as the launch image (commercial), or the image to incorporate as the splash image (personal and educational), see Adding a default launch image or Adding a splash image for more details. This image should be 1024x768 pixels and rotated to the initial iPhone orientation setting.

14. The initial iPhone and iPod orientation to start the application up in.

15. The set of supported iPad initial orientations. This is used to determine which splash screen will be displayed.

16. The devices supported by the application, iOS uses this to determine if an application should launch on iPod/iPhones and whether it should run in iPod/iPhone emulation mode on iPads (UIDeviceFamily).

17. The minimum iOS version required by the application (MinimumOSVersion)

18. The instruction set to build for. Universal will build for both Arm v6 and Arm v7 devices (but will subsequently produce a larger app). Arm v6 builds will run on both Arm v6 and Arm v7 devices. Arm v7 builds will only run on Arm v7 devices.

19. Determines whether the application requires a persistent WiFi connection (UIRequiresPersistentWiFi)

20. Determines whether the 'Shared Files' feature of iTunes is enabled for this application (UIFileSharingEnabled).

21. These options determine what facilities the application requires or prohibits on the device in order to be launched (UIRequiredDeviceCapabilities).

22. The initial visibility state of the status bar (UIStatusBarHidden).

23. The initial status bar style (UIStatusBarStyle).

Adding a splash image (personal and educational)

If you are using a personal or educational license, then you are restricted in what can be displayed as the launch image. In this case you should provide a (square) PNG image that will be placed inside a LiveCode branded banner (see below).

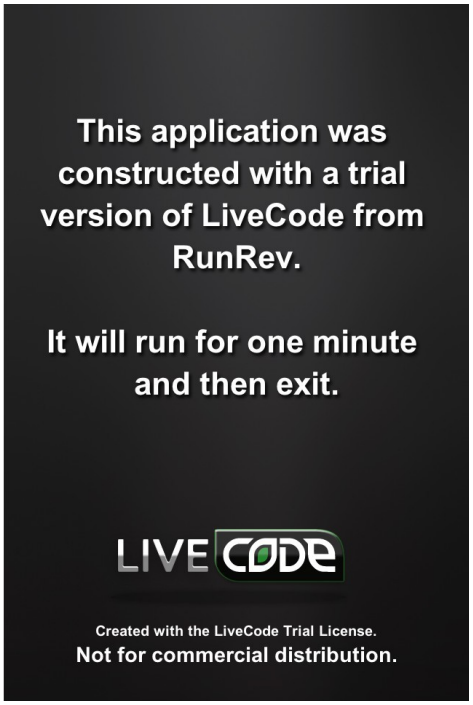
The plugin automatically generates a collection of launch images using this image depending on the target device settings you have specified in the plist.

We recommend providing an image of 600x600 for the splash – this will give good results when resampled at the various resolutions and sizes required by the different iOS devices.

Note: With these license types, the generated launch image will remain on screen for 5 seconds before being dismissed.

Adding a default launch image (trial)

If you are evaluating the iOS deployment feature using a trial license, then you cannot configure a splash or launch image. Instead, all such applications will be built with the following launch image:



This image will remain on screen for 5 seconds before the application launches, and the application will quit after one minute.

Deployment Features

Standalone builder messages

When building a mobile application for either a device (through ***Save as standalone***) or for simulation (by clicking ***Simulate)***, messages are sent to the application's main-stack to notify it before building starts, and after build has finished.

Before the application is built the following (optional) message is sent:

savingMobileStandalone *targetType, appBundle*

Where *targetType* is either "android release" or "android test", depending on the type of build; and *appBundle* is the path of the application bundle being built.

After the application is built (but before being launched in the simulator), the following (optional) message is sent:

mobileStandaloneSaved *targetType, appBundle*

Where the parameters are the same as before except if the build failed, in which case *appBundle* will be empty.

Note that if you make changes to the stack in savingMobileStandalone that you want to appear in the built application, you must save the stack before returning from the handler. The mobile standalone builder uses the stackfile as it is on disk after return from the message to build the app.

Handling differences between devices

When deploying to mobile devices, we face a number of unique challenges. These challenges fall under two general categories: handling differing device resolutions, and handling differences in how certain features need to be coded to accommodate the device.

Handling differences in device resolution

Mobile device function differently to the desktop in that an app must occupy the whole screen. This means that we can think of our app as running in a window which is automatically sized to fit the screen resolution. Therefore, if we want our app to run on more than just one specific type of mobile device, we generally cannot use absolute coordinates such as 100,100 or 740,360 to place objects. If we have an image we want to place in the center of the screen for example, a device sized 500 by 500 pixels will have the center of the screen at 250,250 – while a device sized 1000 by 1000 pixels will have the center of the screen at 500,500.

Therefore, rather than specifying a coordinate value for the center of the screen, we can use ratios to express the location on the screen that we want to place our object. For example, the center of the screen would be 0.5,0.5 – halfway across the screen and halfway down the screen. We can then write a function that will take a ratio value such as 0.5,0.5 or 0.1,0.85 and convert it into a set of coordinates while the app is

actually running on the device, as it will then be able to look at the size of the screen and output the correct coordinates for the ratios we provide.

As well as considering the placement of objects, we also need to consider the size of the objects we place on the screen. For example, if we have a background image sized 960x640 pixels (iPhone Retina size), it will not fill the whole screen if we run the app on an iPad (which is sized 1024x768). We have 3 options here – we can either have one background image that we stretch or tile to fit the whole screen, we can have multiple copies of the same image, sized to different resolutions, or we can have one set of 'master' images that we scale appropriately when the app is first run. The first solution can work in situations where we have a very simple background, such as a solid colour. However, for a professional look and feel where more complex images are involved, we ideally need to implement the 2nd or 3rd solution. We'll be focusing on the 3rd solution in this academy, but many of the principles involved are the same.

Handling differences in feature implementation.

Certain features sometimes need to be implemented differently from one device to the next. For example, we might want to do one thing on iOS and another thing on Android. In some (rare) cases the syntax you use to access a feature on a mobile device will differ between android and iOS. In order to compensate for this, we can use an if statement, or a switch statement, in which we check what the platform is and take appropriate action.

```
if the platform is "iPhone" then
   iPhoneStartTrackingLocation
else
   answer “Tracking unavailable!”
end if
```

```
switch the platform
   case "iPhone"
      answer "I'm an iPhone!"
      break
   case "Android"
      answer "I'm an Android!"
      break
   default
```

```
        answer ("I'm a desktop of type:" & the platform)
        break
end switch
```

As you can see, the "the platform" function is extremely useful. It returns "iphone" on any iOS device, "android" on any Android device, and a string describing the operating system type on any other device.