iOS Deployment Release Notes (R36)

Table of Contents

Overview

LiveCode now incorporates facilities for deploying to iOS. These facilities include the ability to build iOS applications that run in a variety of simulator versions as well as on iPhone, iPod Touch and iPad devices.

In addition to supporting many of the desktop engine's features, the iOS engine hooks into many iOS-specific features. Please see the *iOS Specific Features* section for more details.

For information on what parts of the Desktop feature set are currently implemented when deploying to iOS, please see the *What Works* section.

Note: *If you have not purchased the iOS deployment pack, you can still try out iOS deployment features, but any built apps will have a forced banner for 5 seconds on startup, and will quit after one minute.*

Note: *iOS deployment is only supported on Macs running the latest versions of Leopard or Snow Leopard and require installation of an appropriate iOS SDK.*

Getting Started

Choosing an SDK

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 3.2.6 and iOS 4.3	Snow Leopard	4.3, 4.2, 4.1, 4.0, 3.2
Xcode 3.2.5 and iOS 4.2	Snow Leopard	4.2, 4.1, 4.0, 3.2
Xcode 3.2.4 and iOS 4.1	Snow Leopard	4.1, 4.0, 3.2
Xcode 3.2.1 and iOS 3.1.3	Snow Leopard	3.1.3
Xcode 3.1.4 and iOS 3.1.3	Leopard	3.1.3

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

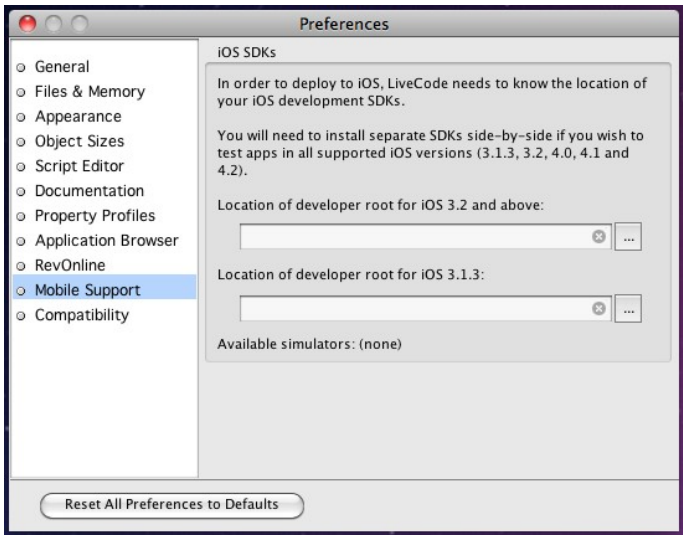
If you wish to test applications in all versions of the simulator (3.1.3, 3.2, 4.0, 4.1 and 4.2) then it is necessary to be running on Snow Leopard, and to have installed the iOS 4.2 SDK *and* the iOS 3.1.3 SDK in **separate** locations.

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

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*.



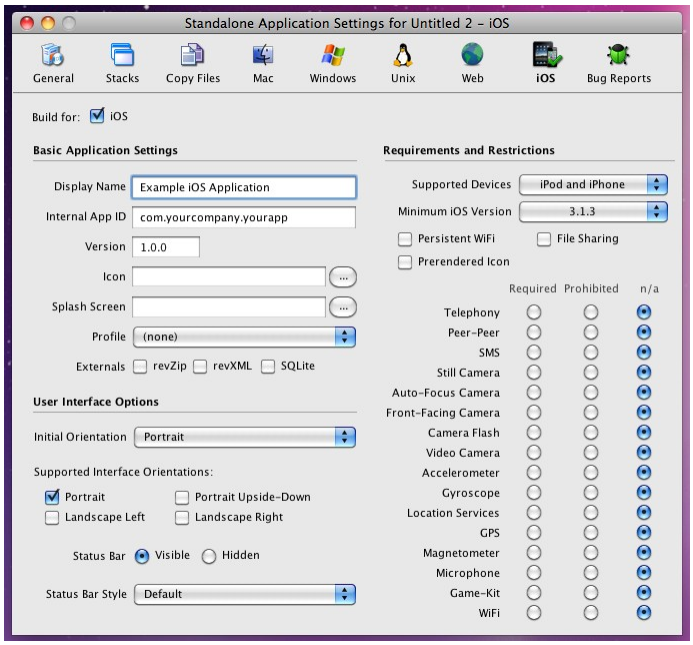
Use this pane to choose the correct SDK paths by using the '…' buttons next to the appropriate one. You should choose the folder you selected when installing the SDK. (This defaults to '/Developer' in the iOS SDK installers).

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

Note: *On startup if SDKs have not been previously configured, LiveCode will check to see if there is a recognised SDK at /Developer.*

Configuring an iOS standalone

To configure a stack for iOS, you use the new 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.

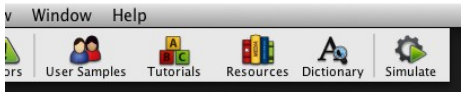
To make a stack build for iOS, simply check the *Build for iOS* button and configure any options that you wish.

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.*

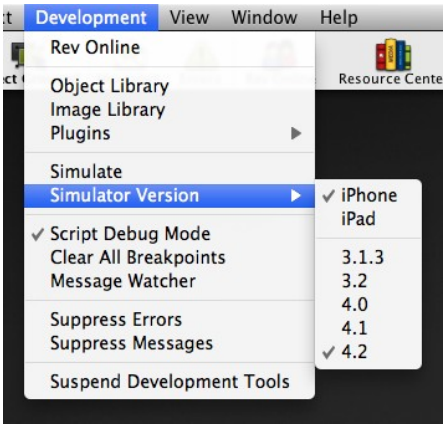
Running in the simulator

Once you have a stack configured for iOS, you can run it in the iOS Simulator by using the *Simulate* button on the menubar:



This button will be enabled for any stack that has been configured for iOS deployment, and clicking it will launch the stack in the simulator, terminating a running simulation if any.

You can also access the *Simulate* action from the Development menu. Additionally this is where you can configure which simulator version to use:



Here you can choose both version, and device type you wish to simulate. (Be aware that 3.2 is iPad only, and only 4.2 supports both iPad and iPhone/iPod Touch). Any setting you choose here will take effect the next time you use the *Simulate* button or menu-item.

Note: *If the* Simulate *button or menu-item remains disabled, even if you have configured a stack for iOS deployment, it probably means you haven't configured your SDKs correctly. In this case, check that there are available simulators in the* Mobile Support *pane of* Preferences.

A first project

Once you have installed an iOS SDK and configured LiveCode for it, it is easy to run a simple project:

1. Create a new main stack via **File > New Mainstack**.
2. Rename your new main stack to *Hello World*
3. Drag and drop a button onto the new main stack, and call it *Click Me*
4. Edit the *Click Me* button script and enter the following:

```
on mouseUp
 answer "Hello World!" with "ok"
end mouseUp
```

5. Save the *Hello World* stack.
6. Bring up the *Standalone Application Settings* dialog from the *File* menu, switch to the iOS pane and make sure 'Build for iOS' is checked.
7. Make sure your test stack is active and then click *Simulate* on the menubar.
8. Click the *Click Me* button in the simulator to see your script in action!

You can try the stack out in different versions of the simulator, simply by selecting the version you want from the *Development* menu.

Building for a real 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.

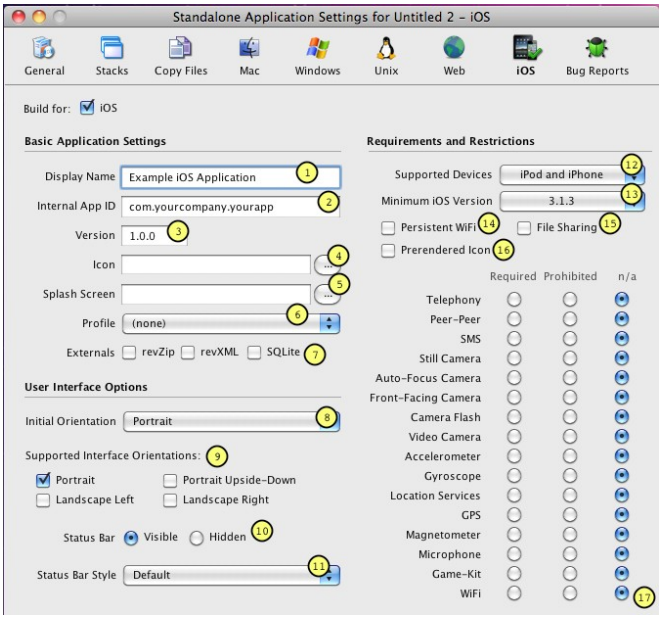
To get your newly prepared application on the device, simply drag the application bundle from the desktop into the *Applications* list – opting to install the appropriate provisioning profile if it has not been previously installed on the device.

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 icon to display on the SpringBoard, see *Adding a SpringBoard icon* for more details (CFBundleIconFile and CFBundleIconFiles).
5. The image to use as the launch image (commercial), or the image to incoporate as the splash image (personal and educational), see *Adding a default launch image* or *Adding a splash image* for more details.
6. The provisioning profile to use when building the application to run on a device.
7. 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.
8. The initial orientation to start the application up in (UIInterfaceOrientation).

9. The set of (initial) interface orientations your application supports, iOS uses this key to determine what launch image to display (UISupportedInterfaceOrientations).
10. The initial visibility state of the status bar (UIStatusBarHidden).
11. The initial status bar style (UIStatusBarStyle).
12. 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).
13. The minimum iOS version required by the application (MinimumOSVersion)
14. Determines whether the application requires a persistent WiFi connection (UIRequiresPersistentWiFi)
15. Determines whether the 'Shared Files' feature of iTunes is enabled for this application (UIFileSharingEnabled).
16. Determines whether the SpringBoard icon already has a tint and gloss applied. (UIPrerenderedIcon).
17. These options determine what facilities the application requires or prohibits on the device in order to be launched (UIRequiredDeviceCapabilities).

More details of the plist options can be found in the iOS Reference Document.

Adding a SpringBoard icon

All applications currently installed on an iOS device are displayed on the *SpringBoard* – the home screen user interface you get presented with when the device is switched on.

Depending on what devices your application runs you should provide between 1 and 3 icons:

- <name>.png – a 57x57 icon for use on old (non-Retina) iPods and iPhones
- <name>-114.png – a 114x114 icon for use on Retina display capable iPods and iPhones
- <name>-72.png – a 72x72 icon for use on iPads

Here, <name> is anything you choose, the plugin will copy the files into the app bundle with the correct final name to be picked up by the OS.

You should always provide a 57x57 icon, and the plugin will automatically look for appropriately named icons at the other sizes to include.

Adding a default launch image (commercial)

On startup of an iOS application the SpringBoard will initially display a static image – this image stays on screen until the application has completely finished initializing and is ready to update the screen.

If you are using a commercial license then you have complete control over the launch image. You should provide between 1 and 4 images as follows:

- <name>.png – a 320x480 image for use on old (non-Retina) iPods and iPhones
- <name>@2x.png – a 640x960 image for use on Retina display capable iPods and iPhones

- <name>-Portrait.png – a 768x1024 image for use on iPads when in portrait mode
- <name>-Landscape.png – a 1024x768 image for use on iPads when in landscape mode

Here, <name> is anything you choose, the plugin will copy the files into the app bundle with the correct final name to be picked up by the OS.

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).

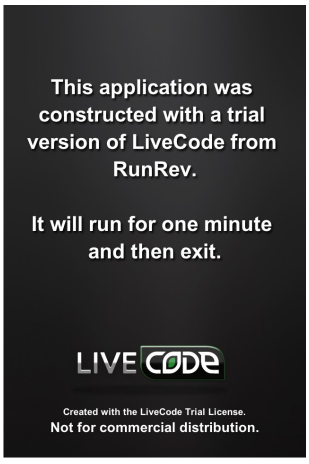
The plugin automatically generates a collection of launch images using this image depending on the target device settings you have specified in the plist.

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.

Adding custom fonts

In iOS 3.2 and later, the ability was introduced to allow applications to bundle custom fonts which then become available to the app (and only that app) while it is running.

To take advantage of this feature, all you need to do is reference the files of any fonts you wish to include in the *Copy Files* pane. These files can either be a direct file reference, or contained in one of the folder references. The Standalone Builder will treat any files that end with the extension *ttf* or *ttc* as font files to use in this way.

Any fonts included in this way will appear in ***the fontNames*** and can be used in the same way as any other font on the system.

Important: *Make sure you have an appropriate license for the fonts you choose to bundle with your app like you would any other media such as sounds, images and videos.*

Adding custom externals

More details about developing and adding custom externals to apps will be made available in due course in concert with an updated *Externals SDK*.

Copy files restrictions

It appears that (at least) the simulator does not like specific folder names being present at top-level inside the app-bundle. In particular, attempting to copy files in that result in a top-level folder called 'resources' (any case) will cause simulation to fail.

To help identify these cases, the Copy Files pane will warn you when you add files that could cause this issue. Additionally, when an app is built (either for simulation or for deployment) an appropriate error message will be displayed and the operation will cease.

Note: *At this stage we do not know if this problem is limited to 'resources' or whether there are others too. If you find you get an 'unknown error' when trying to simulate, try renaming some of your top-level 'copy files' folders and see if it goes away. If it does please let us know what folder names caused the problem so we can add them to our checks.*

General Engine Features

Engine version

The current release of the iOS engine was derived from the 4.0 version of the desktop engine. This means that features present in 4.5.x that you might expect to work in the iOS engine will not be present at this time. In particular, the engine version is fixed at 4.0.0 and the build number at 950.

We are working on reintegrating the iOS port of the engine with the main desktop engine, and versioning will once again become unified in a future release.

What doesn't work

The following features have no effect:

- clipboard related syntax and functionality (planned for a future release)
- drag-drop related syntax and functionality (no support on mobile devices)
- printing syntax and functionality (planned for a future release)
- setting the mouseLoc (no support on mobile devices)
- backdrop related syntax and functionality (no support on mobile devices)
- cursor related syntax and functionality (no support on mobile devices)
- socket syntax and functionality (planned for a future release)
- audioclips/videoclips/player functionality (use the 'play' and 'play video' syntax described later)
- strong encryption support (planned for a future release)
- dbMysql, dbPostgreSQL, dbODBC and custom externals (planned for a future release)
- paint tools (planned for a future release)

What does work

The following things do work as expected:

- rendering of controls with non-system themes (default is Motif theme)
- date and time handling
- gradients, graphic effects and blending
- any non-platform, non-system dependent syntax (maths functions, string processing functions, behaviors etc.)
- revZip, revXML and dbSqlite

Debugging

At present the options available for debugging applications running on target devices is limited.

Obviously, scripts will work in a similar fashion between Desktop and Mobile so this helps.

There is, however, a simple means of logging from an emulated target device. The LiveCode command form:

put *string*

Will write the string out to the standard error stream. These messages will be visible in *Console.app* when running in the simulator, and in the *Console* tab of the Xcode *Organizer* for a given target device while it is connected to the host computer.

Windowing and Stacks

The mobile engine uses a very simple model for window management: only one stack can be displayed at a time.

The stack that is displayed is the most recent one that has been targeted with the **go** command.

The currently active stack will be the target for all mouse and keyboard input, as well as be in receipt of a **resizeStack** message should the orientation or layout of the screen change.

The **modal** command can also still be used, and will cause the calling handler to block until the modal'ed stack is closed as with the normal engine. Note, however, that performing a further **go stack** from a modal'ed stack will cause the new stack to layer above the modal stack – this will likely cause many headaches, so it is probably best to avoid this case!

At this time menus and other related popups will not work correctly, as these are implemented in the engine (essentially) as a specialized form of **go stack** they will cause the current stack to be overlaid completely, with various undesirable side-effects.

Note: *The 'go in window' form of the 'go stack' command will not work correctly in the iOS and must not be used. Since there is only one stack/window displayed at once on this platform, a generic 'go stack' should be used instead.*

System Dialogs – answer and ask

The iOS engine supports a restricted version of the *answer* and *ask* commands – both using the system-provided UIAlertView class.

The answer command can be used in this form:

answer *message* [**with** *button* **and** …] [**titled** *title*]

This will use the iPhone standard alert popup with the given buttons and title. The last button specified will be marked as the default button.

The ask command can be used in this form:

ask [**question** | **password**] *prompt* **[with** *initialAnswer*] [**titled** *title*]

If neither question nor password is specified, question is assumed. The value entered by the user will be retured in *it*. If the user cancelled the dialog, **the result** will contain *cancel*.

Note: *You cannot nest calls to ask/answer on iOS. If you attempt to open an ask or answer dialog while one is showing, the command will return immediately as if the dialog had been cancelled.*

Non-file URL access

The iOS engine has support for fetching urls, posting to urls and downloading urls in the background. Note that the iOS engine does not support libUrl, and as such there are some differences between url handling compared to the desktop.

The iOS engine supports the following non-file URL access methods:

- GET for http, https and ftp URLs
- POST for http and https URLs
- PUT for ftp URLs

Note: *When using URLs for these protocols be aware that the iOS system functions used to provide them are much stricter with regards the format of URLs – they must be of the appropriate form as specified by the RFC standards. In particular, in FTP urls, be careful to ensure you urlEncode any username and password fields appropriately (libUrl will allow characters such as '@' in the username portion and still work – iOS will not be so forgiving).*

To fetch the google home page you can do:

put url ("http://www.google.com") **into** tGooglePage

To post data to a website, you can use:

post tData **to url** tMyUrl

To upload a file to an FTP server you can use:

put tData **into url** "ftp://ftp.myftpserver.com"

To download a url in the background, you can use:

load url tMyUrl **with message** "myUrlDownloadFinished"

Note that, the callback message received after a **load url** will be of the form:

myUrlDownloadFinished *url*, *status*, *data*

Here, *data* is the actual content of the url that was fetched (assuming an error didn't occur).

Progress updates on ongoing url requests are communicated via the *urlProgress* message. This message is periodically sent to the object whose script initiated the operation. It can have the form:

urlProgress *url,* "contacted"

urlProgress *url,* "requested"

urlProgress *url,* "loading", *bytesReceived* , [*bytesTotal*]

urlProgress *url,* "uploading", *bytesReceived*, [*bytesTotal*]

urlProgress *url,* "downloaded"

urlProgress *url,* "uploaded"

urlProgress *url,* "error", *errorMessage*

Note that pBytesTotal will be empty if the web server does not send the total data size.

You can also download a url direct to a file – this is particularly useful when downloading large files since the normal 'url' chunk downloads into memory. To do this use:

libUrlDownloadToFile *url*, *filename*

Unlike the libUrl command of the same name, this command will block until the download is complete, and will notify progress through the **urlProgress** message as described above.

When using GET and POST with http(s) URLs you can use the *httpHeaders* global property to configure the headers to send. This works the same as the desktop engine, any specified headers overriding those of the same key that would normally be sent, and any new keys being appended.

Note: *The order of the arguments passed to urlProgress changed in revision 18, to make them consistent with other callbacks, and also with the libUrl status callback.*

Out-of-bounds group scrolling

Two properties *unboundedHScroll* and *unboundedVScroll* now enable you to configure whether scroll values for a group can be set to values outside of the actual content bounds. This makes it much easier to support the standard iOS bouncing features in scrollers.

This change has been made to both the iOS engine, and the main desktop engine. See the main release notes for more details.

Externals

The revZip, revXML and dbSqlite (via revDB) externals can now be used on iOS.

To include these components, simply check the appropriate boxes on the iOS Standalone Settings Pane.

Note: *Like the iOS engine itself, the iOS externals are currently derived from those which shipped with version 4.0.*

Snapshots

The iOS engine supports both the object and screen snapshot variants of the import and export snapshot commands.

To fetch a snapshot of an object use:

import snapshot from [**rectangle** *rect* **of**] *object*

export snapshot from [**rectangle** *rect* **of**] *object*

To fetch a snapshot of the screen use:

import snapshot from rectangle *rect*

export snapshot from rectangle *rect*

In the screen snapshot case, co-ords are given relative to the top-left of the screen and include the status bar.

Note: *There does not seem to be a way to render the status bar without using private features of the iOS API. Therefore, if your snapshot rectangle includes part of the screen where the status bar is, it will be clipped out.*

iOS-specific engine features

This version of the LiveCode iOS engine includes a wide-range of features specific to iOS devices. These are described in the following sections.

Multi-touch events

Touches can be tracked in an application by responding to the following messages:

- **touchStart** *id*
- **touchMove** *id*, *x*, *y*
- **touchEnd** *id*
- **touchRelease** *id*

The *id* parameter is a number which uniquely identifies a sequence of touch messages corresponding to an individual, physical touch action. All such sequences start with a **touchStart** message, have one or more **touchMove** messages and finish with either a **touchEnd** or a **touchRelease** message.

A **touchRelease** message is sent instead of a **touchEnd** message if the touch is cancelled due to an incoming event such as a phone-call.

No two touch sequences will have the same *id*, and it is possible to have multiple (interleaving) such sequences occurring at once. This allows handling of more than one physical touch at once and, for example, allows you to track two fingers moving on the iPhone's screen.

The sequence of touch messages is tied to the control in which the touch started, in much the same way mouse messages are tied to the object a mouse down starts in. The test used to determine what object a touch starts in is identical to that used to determine whether the pointer is inside a control. In particular, invisible and disabled controls will not considered viable candidates.

Mouse events

The engine will interpret the first touch sequence in any particular time period as mouse events in the obvious way: the start of a touch corresponding to pressing the primary mouse button, and the end of a touch corresponding to releasing the primary mouse button.

This means that all the standard LiveCode controls will respond in a similar way as they do in the desktop version – in particular, you will receive the standard mouse events and *the mouseLoc* will be kept updated appropriately.

Note that touch messages will still be sent, allowing you to choose how to handle input on a per-control basis.

Motion events

An application can respond to any motion events generated by iPhoneOS by using the following messages:

- **motionStart** *motion*

- **motionEnd** *motion*
- **motionRelease** *motion*

Here *motion* is the type of motion detected by the device. As of iPhoneOS 3.0, the only motion that is generated is "shake".

When the motion starts, the current card of the defaultStack will receive **motionStart** and when the motion ends it will receive **motionEnd**. In the same vein as the touch events, **motionRelease** is sent instead of **motionEnd** if an event occurs that interrupts the motion (such as a phone call).

Accelerometer support

You can enable or disable the iPhone's internal accelerometer by using:

iphoneEnableAccelerometer [*interval*]

iphoneDisableAccelerometer

Enabling the accelerometer will cause **accelerationChanged** events to be delivered to the current card of the defaultStack at the specified interval. The interval should be specified in seconds, and is the approximate time between delivery of messages. Note that the interval is constrained by hardware-specific minimums and maximums (which are left unspecified by Apple).

The **accelerationChanged** message takes a single parameter pSample, which consists of four values:

x,y,z,t

Here x, y and z are the acceleration along those axes relative to gravity. The t value is a relative measurement of how much time has passed – you can use the difference between the time values in two **accelerationChanged** events to give an indication of how much time passed between the samples.

Photo Picking Support

You can hook into the iPhone's native photo picker by using

iPhonePickPhoto *source*, [*maxwidth*, [*maxheight*]]

Here *source* is one of:

- *library* – the photo is taken from the device's photo library
- *camera* – the photo is taken using the device's default camera
- *rear camera* – the photo is taken using the device's rear camera (if present)
- *front camera* – the photo is taken using the device's front camera (if present)
- *album* – the photo is taken from the device's recent camera roll

The *maxwidth* and *maxheight* parameters constrain the maximum size of an image. The chosen image will be scaled down proportionally to fit within the size specified. If either size specified is 0, then the parameter is ignored.

If the source type isn't available on the target device, the command will return with result *"source not available"*. If the user cancels the pick, the command will return with result *"cancel"*. Otherwise

a new image object will be created on the current card of the default stack containing the chosen image.

When running on an iPhone, the photo-picker is displayed using the standard iOS fullscreen overlay view.

When running on an iPad, the photo-picker is displayed using a standard iOS pop-over. In this case, the pop-over is positioned relative to **the rect of the target** at the time the iphonePickPhoto command was called.

Note: *The image object is cloned from the templateImage, so you can use this to configure settings before calling the picker.*

Keyboard Input

Surprisingly, the SDK does not provide direct control over the iPhoneOS software keyboard. However, an attempt has been made to provide some level of support for text input entry. If you have a text field which is focusable (**traversalOn** true), then whenever it has focus the iPhone keyboard will appear and allow basic text editing functionality.

While it is possible to use the non-Roman keyboards to enter text, for scripts which have combining and/or input method type requirements the input will be incorrect. For example, languages such as Russian can be entered correctly, but Korean will not work as expected.

The auto-capitalization, auto-correction, copy/paste, undo/redo and selection point magnification features that are present in standard iPhone text entry fields are not supported.

Configuring keyboard type

You can configure the type of keyboard that will be displayed by using the **iphoneSetKeyboardType** command:

iphoneSetKeyboardType *type*

Where *type* is one of:

- default – the normal keyboard
- alphabet – the alphabetic keyboard
- numeric – the numeric keyboard with punctuation
- url – the url entry keyboard
- number – the number pad keyboard
- phone – the phone number pad keyboard
- contact – the phone contact pad keyboard
- email – the email keyboard
- decimal – the decimal numeric pad keyboard (iOS 4.1+)

The keyboard type setting takes effect the next time the keyboard is shown – *it does not affect the currently displaying keyboard, if any*.

Similarly you can configure the type of return key displayed on the keyboard using the

iphoneSetKeyboardReturnKey command:

iphoneSetKeyboardReturnKey *returnKey*

Where *returnKey* is one of:

- default – the normal return key
- go – the 'Go' return key
- google – the 'Google' return key
- join – the 'Join' return key
- next – the 'Next' return key
- route – the 'Route' return key
- search – the 'Seach' return key
- send – the 'Send' return key
- yahoo – the 'Yahoo' return key
- done – the 'Done' return key
- emergency call – the 'emergency call' return key

Again, setting the return key only takes effect the next time the keyboard is shown.

If you wish to configure the keyboard options based on the field that is being focused, simply use the commands in an *openField* handler of the given field. The keyboard is only shown after this handler returns, so it is the ideal time to configure it.

Activation notifications

The following messages will be sent to the current card of the default stack when the keyboard is shown or hidden:

keyboardActivated

keyboardDeactivated

Handle these messages to move controls or change the display layout to take account of the restricted screen area that will be available.

Orientation handling

The iOS engine includes support for automatic handling of changes in orientation and in so doing gains use of the smooth iOS standard animation rotation animation (note this replaces the previous approach of using *iphoneRotateInterface* which no longer does anything).

Example: *You can find a simple stack using the orientation handling features in the IDE resources folder (open using the* ***Help > Example Stacks and Resources*** *menu item). The stack can be found at:* ***Mobile Examples/Orientation Example.livecode***

Auto-rotation support

You can configure which orientations your application supports, and also lock and unlock changes

in orientation.

The engine will automatically rotate the screen whenever the following are true.

- it detects an orientation change
- the orientation is in the currently configured 'allowed' set
- the orientation lock is off

Such a rotation may result in a *resizeStack* message being sent since rotating at 90 degrees switches width and height.

Querying orientation

You can fetch the current device orientation using the **iphoneDeviceOrientation()** function. This returns one of:

- *unknown* – the orientation could not be determined
- *portrait* – the device is being held upward with the home button at the bottom
- *portrait upside down* – the device is being held upward with the home button at the top
- *landscape left* – the device is being held upward with the home button on the left
- *landscape right* – the device is being held upward with the home button on the right
- *face up* – the device is lying flat with the screen upward
- *face down* – the device is lying flat with the screen downward

Similarly, you can fetch the current interface orientation using the **iphoneOrientation()** function. This returns one of *portrait, portrait upside down, landscape left* and *landscape right*. With the same meanings as for device orientation.

Controlling auto-rotation

To configure which orientations your application supports use:

iphoneSetAllowedOrientations *orientations*

Here *orientations* must be a comma-delimited list consisting of at least one of *portrait*, *portrait upside down*, *landscape left* and *landscape right*. The setting will take effect the next time an orientation change is effected – the interface's orientation will only be changed if the new orientation is among the configured list. You can query the currently allowed orientations with the **iphoneAllowedOrientations()** function.

To lock or unlock orientation changes for a time use:

iphoneLockOrientation and **iphoneUnlockOrientation**

The orientation lock is nestable, and when an unlock request causes the nesting to return to zero, the interface will rotate to match the devices current orientation (assuming it is in the set of allowed orientations). You can query the current orientation lock state with the **iphoneOrientationLocked()** function.

Orientation changed notification

An application will receive an ***orientationChanged*** message if the device detects a change in its position relative to the ground, and you can use the **iphoneDeviceOrientation()** function to find out the current orientation. This message is sent to the current card of the default stack.

The ***orientationChanged*** message is sent **before** any automatic interface rotation takes place thus changes to the orientation lock state and allowed set can be made at this point and still have an effect. If you wish to perform an action after the interface has been rotated, then either do so on receipt of ***resizeStack***, or by using a *send in 0 millisecs* message.

Initial orientation handling

On startup, the engine reads the settings of 'initial orientation' and 'supported orientations' from the plist (as configured by the iOS standalone settings pane). It uses the supported orientations it finds to initialize the orientations allowed by autorotation (i.e. iphoneSetAllowedOrientations), and the initial orientation it finds to ensure the interface starts the correct way round.

To ensure that your application works in only specific orientations from the outset, you need only configure the options in the standalone builder. In particular, *you need take no further action in script*.

Note: *Prior to R20, you had to place some code in the startup handler to lock an application to a given orientation. This should be removed when building with this release as it is no longer necessary and might cause unwanted behavior.*

Resolution handling

The new iPhone 4 has a display with double the resolution in both horizontal and vertical directions. By default, iOS handles this by mapping one logical 'point' to two physical 'pixels' with applications (rev included) interpreting everything in terms of logical points. This means that apps targetted for older devices will run identically on the newer iPhone 4 devices.

As **the screenRect** and associated properties all deal in logical points, they do not reflect the actual device resolution at which the app is being displayed. To fetch the device screen's resolution in pixels use the **iphoneDeviceResolution()** function. This will return a string in the form *width*, *height* – with the values being given in pixels.

To use the full resolution of such high-resolution devices, use the command:

iphoneUseDeviceResolution *(true | false)*

If passed true, rev will ensure that co-ordinates and sizes specified in rev are treated as being in pixels, rather than logical points. In particular, when changed, a resizeStack message will be sent notifying in the size change of the current main-stack, and functions and properties (such as **the screenRect**) will reflect co-ordinates in pixels.

Note: The notion of pixel and logical point remains valid on older devices, its just that it is always 1-1 thus using this command will have no effect there.

The scale of the devices screen (relative to a non-Retina display) can be queried using **iphoneDeviceScale()**. This function will return 2 if the display is a Retina display, or 1 otherwise.

Location handling

Basic support is present for CoreLocation – the framework that allows tracking of the device's position.

To start tracking the location of the device use:

iphoneStartTrackingLocation

To stop tracking the location of the device use:

iphoneStopTrackingLocation

You can detect changes in location by handling the ***locationChanged*** message. This message is sent to the current card of the default stack. If location tracking cannot be started (typically due to the user 'not allowing' access to CoreLocation) then a ***locationError*** message is sent instead.

The current location of the device can be fetched by using the **iphoneCurrentLocation**() function. If location tracking has not been enabled this function returns empty. Otherwise it returns an array with the following keys:

- *horizontal accuracy* – the maximum error in meters of the position indicated by *longitude* and *latitude*
- *latitude* – the latitude of the current location, measured in degrees relative to the equator. Positive values indicate positions in the Northern Hemisphere, negative values in the Southern.
- *longitude* – the longitude of the current location, measured in degrees relative to the zero meridian. Positive values extend east of the meridian, negative values extend west.
- *vertical accuracy* – the maximum error in meters of the *altitude* value.
- *altitude* – the distance in meters of the height of the device relative to sea-level. Positive values extend upward of sea-level, negative values downward.
- *timestamp* – the time at which the measurement was taken, in seconds since 1970.

If the latitude and longitude could not be measured, those keys together with the *horizontal accuracy* key will not be present. If the altitude could not be measured, that key together with the *vertical* accuracy will not be present.

Email composition

Basic support

A version of **revMail** has been implemented that hooks into the iPhone's MessageUI framework. Using this, you can compose a message and request that the user send it using their currently configured mail preferences.

The syntax of **revMail** is:

revMail *toAddress*, [*ccAddress*, [*subject*, [*messageBody*]]]

Where the address fields are comma separated lists of email address. If any of the parameters are not present, the empty string is used instead.

Upon return, **the result** will be set to one of:

- *not configured* – if the user has turned off or has not setup mail access on their device
- *cancel* – if the user chooses to cancel the send
- *saved* – if the user chose to save the message in drafts
- *sent* – if the user elected to send the email
- *failed* – if sending the email was attempted, but it failed

Note that once you've called the **revMail** command you have no more control over what the user does with the message – they are free to modify it and the addresses as they see fit.

Advanced support

More complete access to iOS's mail composition interface is gained by using one of the following commands:

iphoneComposeMail *subject*, [*recipients*, [*ccs*, [*bccs*, [*body*, [*attachments*]]]]]

iphoneComposeUnicodeMail *subject*, [*recipients*, [*ccs*, [*bccs*, [*body*, [*attachments*]]]]]

iphoneComposeHtmlMail *subject*, [*recipients*, [*ccs*, [*bccs*, [*body*, [*attachments*]]]]]

All commands work the same, except different variants expect varying encodings for the *subject* and *body* parameters:

- *subject* – the subject line of the email. If the Unicode form of the command is used, this should be UTF-16 encoded text.
- *recipients* – a comma -delimited list of email addresses to place in the email's 'To' field.
- *ccs* – a comma-delimited list of email addresses to place in the email's 'CC' field.
- *bccs* – a comma-delimited list of email addresses to place in the email's 'BCC' field.
- *body* – the body text of the email. If the Unicode variant is used this should be UTF-16 encoded text; if the HTML variant is used then this should be HTML.
- *attachments* – either **empty** to send no attachments, a single attachment array or a one-based numeric array of attachment arrays to include.

The attachments parameter consists of either a single array, or an array of arrays listing the attachments to include. A single attachment array should consist of the following keys:

- *data* – the binary data to attach to the email (not needed if *file* present)
- *file* – the filename of the file on disk to attach to the email (not needed if *data* present)
- *type* – the MIME-type of the data.
- *name* – the default name to use for the filename displayed in the email

If you specify a file for the attachment, the engine's does its best to ensure the least amount of memory is used by asking the OS to only load it from disk when needed. Therefore, this should be the preferred method when attaching large amounts of data.

For example, sending a single attachment might be done like this:

```
put "Hello World!" into tAttachment["data"]
```

```
put "text/plain" into tAttachment["type"]
put "Greetings.txt" into tAttachment["name"]
iphoneComposeMail tSubject, tTo, tCCs, tBCCs, tBody, tAttachment
```

If multiple attachments are needed, simply build an array of attachment arrays:

```
put "Hello World!" into tAttachments[1]["data"]
put "text/plain" into tAttachments[1]["type"]
put "Greetings.txt" into tAttachments[1]["name"]
put "Goodbye World!" into tAttachments[2]["data"]
put "text/plain" into tAttachments[2]["type"]
put "Farewell.txt" into tAttachments[2]["name"]
iphoneComposeMail tSubject, tTo, tCCs, tBCCs, tBody, tAttachments
```

Note: *There are hard limits imposed by the OS of the size of attachments that can be made. This isn't precisely specified anywhere but appears to be around 16Mb based on forum threads.*

Upon completion of a compose request, **the result** will be set to one of the following:

- *sent* – the email was sent successfully
- *failed* – the email failed to send
- *saved* – the email was not sent, but the user elected to save it for later
- *cancel* – the email was not sent, and the user elected not to save it for later
- *not configured* – the device is not configured to send email

Some devices will not be configured with email sending capability. To determine if the current device is, use the **iphoneCanSendMail()** function. This returns **true** if the mail client is configured.

File and folder handling

In general handling files and folders in the iPhone engine is the same as that on the desktop. All the usual syntax associated with such operations will work. Including:

- **open file**/**read**/**write**/**seek**/**close file**
- **delete file**
- **create folder**/**delete folder**
- setting and getting **the folder**
- listing files and folders using **the [detailed] files** and **the [detailed] folders**
- storing and fetching *binfile:* and *file:* urls

However, it is important to be aware that the iPhoneOS imposes strict controls over what you can and cannot access. Each application in iPhoneOS is stored in its own 'sandbox' folder (referred to as the *home* folder. An application is free to read and write files within this folder and its descendants, but is not allowed to access anything outside of this.

When an application is installed on a phone (or in the simulator) a number of initial folders are created for use by the application. You can locate the paths to these folders using the **specialFolderPath**() function with the following selectors:

- *home* – the (unique) folder containing the application bundle and its associated data and folders
- *documents* – the folder in which the application should store any document data (this folder is backed up by iTunes on sync)
- *cache* – the folder in which the application should store any transient data that needs to be preserved between launches (this folder is **not** backed up by iTunes on sync)
- *temporary* – the folder in which the application should store any temporary data that is not needed between launches (this folder is **not** backed up by iTunes on sync)
- *engine* – the folder containing the built standalone engine (i.e. the bundle). This is useful for constructing paths to resources that have been copied into the bundle at build time.

In general you should only create files within the documents, cache, and temporary folders. Indeed, be careful not to change or add any files within the application bundle. The application bundle is digitally signed when it is built, and any changes to it after this point will invalidate the signature and prevent it from launching.

Note: *Unlike (most) Mac OS X installs, the iPhoneOS filesystem is* ***case-sensitive*** *so take care to ensure that you consistently use the same casing for filenames when constructing them. Also note that the Simulator has the same case-sensitivity as the host system and* ***not*** *the device.*

System alert support

Support has been added for **the beepSound** and **beep** commands. These hook into iPhoneOS's standard *PlayPlayerSound* support.

To specify a sound to be played as the system sound, use **the beepSound** global property. This should be set to the filename of the sound to use when **beep** is executed. If you want no sound to play when using **beep**, simply set **the beepSound** to **empty**.

To perform a system alert, use the **beep** command. If no sound has been specified via **the beepSound** global property, the engine will request a vibration alert.

Note: *The iPhone has no default system alert sound so if a sound is required one must be specified by using* ***the beepSound****. The action of* ***beep*** *is controlled by the system and depends on the user's preference settings. In particular, a beep will only cause a vibration if the user has enabled that feature. Similarly, a beep will only cause a sound if the phone is not in silent mode.*

Basic sound playback support

Basic support for playing sounds has been added using a variant of the **play** command. A single sound can be played at once by using:

play *soundFile* [**looping**]

Executing such a command will first stop any currently playing sound, and then attempt to load the given sound file. If **looping** is specified the sound will repeat forever, or until another sound is played.

If the sound playback could not be started, the command will return "could not play sound" in **the result**.

To stop a sound that is currently playing, simply use:

play empty

The volume at which a sound is played can be controlled via **the playLoudness** global property.

The overall volume of sound playback depends on the current volume setting the user has on their device.

This feature uses the built-in sound playback facilities on the iPhone (AVAudioPlayer, to be specific) and as such has support for a variety of formats including AIFF and MP3's.

You can monitor the current sound being played by using **the sound** global property. This will either return the filename of the sound currently being played, or "done" if there is no sound currently playing.

Multi-channel sound support

In addition to basic sound playback support, there is also support for playing sounds on different channels. This feature uses the iOS *AVAudioPlayer* object, which allows many concurrent sounds to be played simultaneously.

Example: *You can find a simple stack using the multi-channel soundl features in the IDE resources folder (open using the* ***Help > Example Stacks and Resources*** *menu item). The stack can be found at:* ***Mobile Examples/Sound Example.livecode***

Playing Sounds

To play a sound on a given channel use the following command:

iphonePlaySoundOnChannel *sound*, *channel*, *type*

Where *sound* is the sound file you wish to play, *channel* is the name of the channel to play it on and *type* is one of:

- *now* – play the sound immediately, replacing any current sound (and queued sound) on the channel.
- *next* – queue the sound to play immediately after the current sound, replacing any previously queued sound. If no sound is playing the sound is prepared to play now, but the channel is immediately paused – this case allows a sound to be prepared in advance of it being needed.
- *looping* – play the sound immediately, replacing any current sound (and queued sound) on the channel, and make it loop indefinitely.

If a sound channel with the given name doesn't exist, a new one is created. When queuing a sound using *next*, the engine will 'pre-prepare' the sound long before the current sound is played, this ensures minimal latency between the current sound ending and the next one beginning.

If an empty string is passed as the sound parameter, the current and scheduled sound on the given channel will be stopped and cleared.

When a sound has finished playing naturally (not stopped/replaced) on a given channel, a ***soundFinishedOnChannel*** message is sent to the object which played the sound:

soundFinishedOnChannel *channel*, *sound*

The message is sent after the switch has occurred between a current and next sound on the given channel. This makes it is an ideal opportunity to schedule the next sound on the channel, thus allowing continuous and seamless playback of sounds.

To stop the currently playing sound, and to clear any scheduled sound, on a given channel use:

iphoneStopPlayingOnChannel *channel*

To pause the currently playing sound on a given channel use:

iphonePausePlayingOnChannel *channel*

To resume the current sound's playback on a given channel use:

iphoneResumePlayingOnChannel *channel*

Channel Properties

To control the volume of a given sound channel use the following:

iphoneSetSoundChannelVolume *channel*, *volume*

iphoneSoundChannelVolume(*channel***)**

Here *channel* is the channel to affect, and *volume* is an integer between 0 and 100 where 0 is no volume, 100 is full volume.

Changing the volume affects the currently playing sound and any sounds played subsequently on that channel.

Note that you can set the volume of a non-existant channel and this will result in it being created. This allows you to set the volume *before* any sounds are played. If you attempt to get the volume of a non-existent channel, however, empty will be retuned.

To find out what sounds (if any) are currently playing and are scheduled for playing next on a given channel use:

iphoneSoundOnChannel(*channel***)**

iphoneNextSoundOnChannel(*channel***)**

These will return empty if no sound is currently (scheduled for) playing (or the channel doesn't exist).

To query a channel's current status use **iphoneSoundChannelStatus**(). This returns one of the following:

- *stopped* – there is no sound currently playing, nor any sound scheduled to be playing
- *paused* – there are sounds scheduled to be played, but the channel is currently paused
- *playing* – a sound is currently playing on the channel

Managing Channels

To get a list of the sound channels that currently exist use:

iphoneSoundChannels()

This returns a return-delimited list of the channel names.

Sound channels persist after any sounds have finished playing on them, retaining the last set volume setting. To remove a channel from memory completely use:

iphoneDeleteSoundChannel *channel*

Sound channels only consume system resources when they are playing sounds, thus you don't need to be concerned about having many around at once (assuming most are inactive!).

Video playback support

Basic support for playing videos has been added using a variant of the **play** command. A video file can be played by using:

play (*video-file* | *video-url*)

The video will be played fullscreen, and the command will not return until it is complete, or the user dismisses it.

If a path is specified it will be interpreted as a local file. If a url is specified, then it must be either an 'http', or 'https' url. In this case, the content will be streamed.

The playback uses iOS's built-in video playback support (MPMoviePlayer) and as such can use any video files supported by that, including mp4's.

On iPhoneOS 3.1.3, the video will always play with landscape orientation (there is no 'legal' way to change this). On iOS 3.2 and later, however, the orientation of the video will be tied to the current interface orientation.

Appearance of the controller is tied to **the showController of the templatePlayer**. Changing this property to true or false, will cause the controller to either be shown, or hidden.

When a movie is played without controller, any touch on the screen will result in a ***movieTouched*** message being sent to the object's whose script started the video. The principal purpose of this message is allow the **play stop** command to be used to stop the movie. e.g.

on movieTouched

    play stop

end movieTouched

Note: *The movieTouched message is* ***not*** *sent if the video is played with* ***showController*** *set to true.*

Playing a video can be made to loop by setting **the looping of the templatePlayer** to true before executing the play video command. Note that looping video is only supported on iOS 3.2 and higher.

A section of a video can be played by setting **the playSelection of the templatePlayer** to true before executing the play video command. This will then use **the startTime** and **the endTime** properties of **the templatePlayer** to determine what section to play. The values of these properties will be interpreted as the number of milliseconds from the beginning of the video.

URL launching support

Support for launching URLs has been added. The **launch url** command can now be used to request the opening of a given url:

launch url *urlToOpen*

When such a command is executed, the engine first checks to see if an application is available to handle the URL. If no such application exists, the command returns "no association" in **the result**. If an application is available, the engine requests that it launches with the given url.

Using this syntax it is possible to do things such as:

- open Safari with a given *http:* url
- open the dialer with a given phone number using a *tel:* url

Important: *Successfully launching a url will cause another application to open and the requesting application to be quit. The application will receive a* ***shutdown*** *message before this happens, however.*

Font querying support

The list of available fonts can now be queried by using **the fontNames** function. This returns a return-delimited list of all the available font families.

The list of available styles can be queried by using the **fontStyles** function:

fontStyles(*fontFamily*, 0)

This will return the list of all font names in the given family. It is these names which should be used as the value of **the textFont** property.

Note**:** *Strictly speaking the list returned by* ***fontStyles*** *isn't the font styles, but the font names and the list returned by* ***fontNames*** *isn't the font names but the font families.*

Visual effect support

The iOS engine now has support for a range of visual effects – including some specific to iOS. The following effects are available:

- scroll (up | left | down | right)
- reveal (up | left | down | right)
- push (up | left | down | right)
- dissolve
- curl (up | down)
- flip (left | right)

Speed can be controlled via the usual adjectives *very slow*, *slow*, *normal*, *fast* or *very fast*.

For the *flip* visual effect, the background behind the flip will be taken from the background color of the current stack – i.e. the card is cut out and flipped over the stack.

Status bar configuration support

You can now configure the status bar that appears at the top of the iOS screen.

To control the visibility of the status bar use the following commands:

iphoneShowStatusBar

iphoneHideStatusBar

To control the style of the status bar use the following command:

iphoneSetStatusBarStyle *style*

Where *style* is one of:

- *default* – the default mode for the device
- *translucent* – a semi-transparent status bar (in this case the stack will appear underneath it)
- *opaque* – a black status bar (in this case the stack will appear below it).

On iPad devices, anything other that *default* has no effect.

Locale and system language query support

You can query the list of preferred languages using the **iphonePreferredLanguages()** function. This returns a return-delimited list of standard language tags in order of user preference (for example "en", "fr", "de", etc.)

You can query the currently configured locale using the **iphoneCurrentLocale()** function. This returns a standard locale tag (for example "en_GB", "en_US", "fr_FR", etc.)

Hardware and system version query support

You can fetch information about the current hardware and system version using the standard LiveCode syntax in the following ways.

To determine what processor an application is running on use **the processor**. In the simulator this will return *i386* and on a real device this will return *ARM*.

To determine the type of device an application is running on use **the machine**. This will return one of:

- *iPod Touch* – the device is one of the iPod Touch models
- *iPhone* – the device is one of the iPhone models
- *iPhone Simulator* – the device is a simulated iPhone
- *iPad* – the device is the iPad
- *iPad Simulator* – the device is a simulator iPad

To determine the version of iPhoneOS the application is running on, use **the systemVersion**. For example, if the device has iPhoneOS 3.2 installed, this property will return *3.2*; if the device has iPhoneOS 3.1.3 installed, this property will return *3.1.3*.

You can fetch the current device's unique system identifier with the **iphoneSystemIdentifier()** function. This returns a string in the standard UUID/GUID format.

Modal Pick-Wheel support

You can present the user with a list of choices to pick from using standard iOS interface elements

using:

iphonePick *optionList*, *initialIndex*

Where *optionList* is a return-delimited list to choose from, and *initialIndex* is the (1-based) index of the item to be initially highlighted. The item the user chooses is returned in **the result**. If the user does not choose an item, 0 is returned.

On the iPhone, a standard Action Sheet pops up containing the standard pick-wheel user interface element.

On the iPad, a standard pop-over is presented with a list to choose from. The currently selected entry being marked with a check. In this case, if *initialIndex* is 0, then no item is checked when displayed.

Idle Timer configuration

By default, iOS will dim the screen and eventually lock the device after periods of no user-interaction.

To control this behavior, use the following commands:

iphoneLockIdleTimer

iphoneUnlockIdleTimer

Locking the idle timer increments an internal lock count, while unlocking the idle timer decrements the lock count. When the lock count goes from 0 to 1, the idleTimer is turned off; when the lock count goes from 1 to 0, the idleTimer is turned on.

To determine whether the idleTimer is currently locked (i.e. turned off) use **iphoneIdleTimerLocked()**.

This feature wraps the UIApplication class's *setIdleTimerDisabled* method.

Querying camera capabilities

To find out the capabilities of the current devices camera(s), use the following function:

iphoneCameraFeatures([*camera*])

The *camera* parameter is a string containing either "rear" or "front". In this case, the capabilities of that camera are returned. These take the form of a comma-delimited list of one or more of the following:

- *photo* – the camera is capable of taking photos
- *video* – the camera is capable of recording videos
- *flash* – the camera has a flash that can be turned on or off

If the returned string is empty it means the device does not have that type of camera.

If no camera parameter is specified (or is empty), then a comma-delimited list of one or more of the following is returned:

- *front photo* – the front camera can take photos
- *front video* – the front camera can record video

- *front flash* – the front camera has a flash
- *rear photo* – the rear camera can take photos
- *rear video* – the rear camera can record video
- *rear flash* – the rear camera has a flash

If the returned string is empty it means the device has no cameras.

Clearing pending interactions

As interaction events (touch and mouse messages) are queued, it is possible for such messages to accumulate when they aren't needed. In particular, when executing 'waits', 'moves' or during card transitions.

To handle this case, the **iphoneClearTouches** command has been added. At the point of calling, this will collect all pending touch interactions and remove them from the event queue.

Note that this also cancels any existing mouse or touch sequences, meaning that you (nor the engine) will not receive a mouseUp, mouseRelease, touchEnd or touchCancel message for any current interactions.

A good example of when this command might be useful is when playing an instructional sound:

```
on tellUserInstructions
    play specialFolderPath("engine") & slash & "Instruction_1.mp3"
    wait until the sound is "done"
    iphoneClearTouches
end tellUserInstructions
```

Here, if the *iphoneClearTouches* call was not made, any tough events the user created while the sound was playing would be queued and then be delivered immediately afterwards potentially causing unwanted effects.

Network reachability checking (experimental)

The network connection on iOS devices is generally more transient than normal network connections and can change between wireless and wide-area wireless (GPRS, 3G, EDGE etc.) transport as it moves, and indeed be lost entirely.

As the behavior of an application may vary depending on what kind of network connection is present it is useful to be able to monitor a given server for the type of connection the device currently has to it.

To start monitoring a specific server for reachability via the network use:

```
iphoneSetReachabilityTarget hostNameOrAddress
```

Where *hostNameOrAddress* is the server to start monitoring, or **empty** to stop monitoring.

The server currently being monitored can be determined by using the **iphoneReachabilityTarget()** function. This returns empty if no server is currently being monitored.

While a server is being monitored, any changes to network connectivity that affect access to it will

cause a **reachabilityChanged** message to be delivered to this card of the defaultStack:

reachabilityChanged *hostNameOrAddress*, *reachabilityInfo*

Here *hostNameOrAddress* will be the server that is being monitored (the same string as passed to **iphoneSetReachabilityTarget**), and *reachabilityInfo* will be a comma-delimited list of zero or more of the following items:

- *transient* – the specified server can be reached via a transient connection
- *reachable* – the specified server can be reached via the current network configuration.
- *connection required* – the specified server can be reached via the current network configuration, but a connection needs to be established before it can.
- *connection on traffic* – the specified server can be reached via the current network configuration, but a connection needs to be established before it can. Any traffic directed to the server will initiate the connection.
- *intervention required* – the specified server can be reached via the current network configuration, but some form of user intervention will be required to establish this connection.
- *is local* – the specified server is associated with a network interface on the current system.
- *is direct* – network traffic to the given server will not go through a gateway, but is routed directly to one of the interfaces in the system.
- *is cell* – the specified server can be reached via an EDGE, GPRS or other 'cell' connection.

If no items are specified then it means the given server is not currently reachable.

The current reachability facilities are a direct wrapper around the SCNetworkReachability functions of the OS, thus the *reachabilityInfo* flags are a direct mapping of what that provides.

Our testing indicates the following are reasonable guidelines for checking for various states:

- To determine if there is no network connection at all (e.g. flight-mode or no cell nor wireless signal) use:

  reachabilityInfo **is empty**

- To determine if a network connection should succeed use:

  "reachable" **is among the items of** reachabilityInfo

- To determine if a network connection should succeed but would use a cell network use:

  "is cell" **is among the items of** reachabilityInfo

- To determine if a network connection should succeed and would use a wireless network use:

  "reachable" **is among the items of** reachabilityInfo **and** \

  "is cell" **is not among the items of** reachabilityInfo

Feedback: *Please let us know if you find any other useful combinations of flags, or indeed find cases where the above guidelines do not work. This feature is currently is lower-level than we would like, and will improve/replace it when we have better set of common empirical use-cases and scenarios to work from.*

Important: This feature is currently experimental. This means that it may not be complete, or may fail in some circumstances that you would expect it to work. Please do not be afraid to try it out as we need feedback to develop it further.

iOS Native Controls

Low-level support has been added for creating and manipulating some native iOS controls (views). There are generic set of commands and functions for creating and configuring certain UIView sub-classes which then layer above the currently displayed stack.

At present, there is an implementation for the UIWebView control (*browser*) and for the UIScrollView control (*scroller*).

To create a native control use:

iphoneControlCreate *controlType,* [*name*]

Where *controlType* is the type of control you wish to create – either "browser" or "scroller" and *name* is an optional string to use to identify the control in the other functions. The *name* must be unique amongst all existing controls and cannot be an integer. The unique (numeric) id for the new control is returned in **the result**.

To destroy a native control use:

iphoneControlDelete *idOrName*

Where *idOrName* is the numeric id returned by *iphoneControlCreate,* or the *name* of the control if provided.

A list of all native controls currently in existence can be fetched using the **iphoneControls()** function. This returns a return-delimited list of control names or ids. Where a control has a name that is used, otherwise its id is used.

Once such a control has been created, you can configure it using:

iphoneControlSet *idOrName*, *property*, *value*

Where

- *idOrName* is the numeric id returned by *iphoneControlCreate,* or the name of the control if provided.
- *property* is the name of the property to change
- *value* is the value of the property to change to

Properties can also be read by using **iphoneControlGet**(*id, property).*

Control specific behavior can be invoked by using:

iphoneControlDo *idOrName, action, ...*

Where *action* is what is to be done, and the parameters are action/control specific.

While in the context of a message that has been dispatched from a native control, you can use the **iphoneControlTarget()** function to fetch the name (or id, if no name is set) of the control that sent the message.

In general, any messages dispatched by the native control will be sent to the object containing the script which created it, this also works correctly with behaviors – messages being sent to the object

referring to the behavior, and not the behavior's object.

All controls (UIView)

All native controls are descendants of the UIView class and therefore inherit a common set of properties and actions.

Properties

id	read-only	The unique (integer) id of the control.
name	read-only	The unique name of the control if one was provided at creation time, empty otherwise.
rect	read/write	The bounds of the control, relative to the top-left of the card.
visible	read/write	Set to true or false to determine whether the control should be displayed.
opaque	read/write	Set to false if the control should be rendered with transparency. In particular, set this control to true if you set a backgroundColor that is not fully opaque.
alpha	read/write	Set to a value between 0 and 255 to blend the control with any controls underneath it.
backgroundColor	read/write	Set to either a standard color name, or a string of the form *red,green,blue* or *red,green,blue,alpha*. Where the components are integers in the range 0 to 255.

Browser control – UIWebView

A UIWebView control is created using a control type of "browser". For full details of what the UIWebView control is capable of, and background about it see the iOS reference document.

Example: *You can find a simple stack using the native browser control features in the IDE resources folder (open using the* ***Help > Example Stacks and Resources*** *menu item). The stack can be found at:* ***Mobile Examples/Browser Example.livecode***

Properties

url	read/write	The url to be loaded into the web-view.
autoFit	read/write	Set to true or false to determine whether the page will be scaled to fit the rect of the control (wraps the *scalesPageToFit* property of UIWebView).
canAdvance	read-only	Returns true if there is a next page in the history (wraps the *canGoForward* property of UIWebView).
canRetreat	read-only	Returns true if there is a previous page in the history (wraps the *canGoBack* property of UIWebView).
delayRequests	read/write	Set to true to cause the *loadRequest* message to be sent.

		Note that in this mode, web-pages that trigger sub-document loads (such as those containing iframes) will not load correctly.
dataDetectorTypes	read/write	Use this property to specify the types of data that should be automatically converted to clickable URLs in the web-view. It is specified as a comma-delimited list of one or more of the following values: • phone number • calendar event (iOS4.0+) • link • address (iOS4.0+) (this property wraps the *dataDetectorTypes* property of UIWebView).
allowsInlinePlayback	read/write	Set to true if the web-view should allow media files to be played 'inline' in the page (wraps the *allowsInlineMediaPlayback* property of the UIWebView). Note that this property is only available on iOS4.0 and later
mediaPlaybackRequiresUserAction	read/write	Set to false to allow media files to play automatically in the web-view (wraps the *mediaPlaybackRequiresUserAction* property of the UIWebView). Note that this property is only available on iOS4.0 and later.

Actions

iphoneControlDo *id,* "advance"

Move forward through the history (wraps the *goForward* method of UIWebView).

iphoneControlDo *id,* "retreat"

Move backward through the history (wraps the *goBack* method of UIWebView).

iphoneControlDo *id,* "reload"

Reload the current page (wraps the *reload* method of UIWebView).

iphoneControlDo *id,* "stop"

Stop loading the current page (wraps the *stopLoading* method of UIWebView).

iphoneControlDo *id,* "load", *baseUrl, htmlText*

Loads as page consisting of the given *htmlText* with the given *baseUrl* (wraps the *loadHtmlString* method of UIWebView).

iphoneControlDo *id,* "execute", *script*

Evaluates the given JavaScript *script* in the context of the current page (wraps the *stringByEvaluationJavaScriptFromString* method of UIWebView).

Messages

browserStartedLoading *url*

Sent when the given url has started to load (sent in response to the *webViewDidFinishLoad* delegate method).

browserFinishedLoading *url*

Sent when the given url has finished loading (sent in response to the *webViewDidStartLoad* delegate method).

browserLoadRequest *url*, *type*

Sent when the given url has been requested. The reason for the request is specified in *type* which can be one of *click, submit*, *navigate*, *reload*, *resubmit* or *other.*

Not passing the message will cause the load request to not go ahead.

This message is only sent if *delayRequests* has been set to true. Note that delaying requests can cause web-pages that load pages into sub-documents to not work correctly.

(This message is sent in response to the *webView:shouldStartLoadWithRequest:* delegate method).

browserLoadFailed *url*, *error*

Sent when the given url fails to load (sent in response to the *webView:didFailLoadWithError:* delegate method).

Scroller control – UIScrollView

A UIScrollView control is created using a control type of "scroller". For full details of what the UIScrollView control is capable of, and background about it see the iOS reference document.

Rather than act as a container for other controls, the 'scroller' is intended to be used as an overlay on part of the screen you wish to interact with the proper iOS scrollbars. By responding to the various scroller messages, you can move LiveCode controls or set the appropriate scroll properties of group and fields to get a native scrolling effect.

Example: *You can find a simple stack using the native scroller control features in the IDE resources folder (open using the* ***Help > Example Stacks and Resources*** *menu item). The stack can be found at:* ***Mobile Examples/Scroller Example.livecode***

Properties

contentRect	read/write	The rectangle over which the scroller scrolls. This is distinct from the scroller's rect, and is essentially the minimum/maximum values of the scroll properties (adjusted for the size of the scroller). This is a comma-separated list of four integers, describing a rectangle.
hScroll	read/write	The horizontal scroll offset. This is an integer value ranging between the left and right of the contentRect, adjusting appropriately for the size of the scroller (i.e.

		contentRect.left to contectRect.right – rect.width).
vScroll	read/write	The vertical scroll offset. This is an integer value ranging between the top and bottom of the contentRect, adjusting appropriately for the size of the scroller (i.e. contentRect.top to contectRect.bottom – rect.height).
canBounce	read/write	Determines whether the scroller will 'bounce' when it hits the edge of the contentRect (maps to the UIScrollView *bounces* property). This is a boolean value.
canScrollToTop	read/write	Determines whether a touch on the status bar causes the scroll to scroll to the top (maps to the UIScrollView *scrollsToTop* property). This is a boolean value.
canCancelTouches	read/write	Determines whether the scroller is allowed to cancel an touch that has been passed through to the underlying controls when it thinks its a scroll gesture (maps to the UIScrollView *canCancelContentTouches* property). This is a boolean value.
delayTouches	read/write	Determines whether the scroller delays passing on touch-down events until it has determined whether it is the start of a scroll gesture or not (maps to the UIScrollView *delaysContentTouches* property). This is a boolean value.
pagingEnabled	read/write	Determines whether scrolling stops on multiples of the scroller's bounds (maps to the UIScrollView *pagingEnabled* property). This is a boolean value.
decelerationRate	read/write	Determines the rate at which scrolling decelerates when a finger is lifted (maps to the UIScrollView *decelerationRate* property). This can be either *normal*, *fast* or a real number.
indicatorStyle	read/write	Determines the style of indicators to display (maps to the UIScrollView *indicatorStyle* property). This can be one of *default*, *white* or *black*.
indicatorInsets	read/write	Determines how far from the edge of the scrollers bounds, the indicators are inset (maps to the UIScrollView *scrollIndicatorInsets* property). This is a comma-separated list of four integers, describing the left, top, right and bottom inset distances.
scrollingEnabled	read/write	Determines whether touches on the scroller cause scrolling (maps to the UIScrollView *scrollEnabled* property). This is a boolean value.
hIndicator	read/write	Determines whether the horizontal indicator should be displayed when scrolling (maps to the UIScrollView

		showsHorizontalScrollIndicator property). This is a boolean value.
vIndicator	read/write	Determines whether the vertical indicator should be displayed when scrolling (maps to the UIScrollView *showsVerticalScrollIndicator* property). This is a boolean value.
lockDirection	read/write	Determines whether scrolling is locked to the initial direction a drag occurs in (maps to the UIScrollView *directionalLockEnabled* property). This is a boolean value.
tracking	read-only	Returns true if the scroller is monitoring a touch for the start of a scroll action (maps to the UIScrollView *tracking* property). This is a boolean value.
dragging	read-only	Returns true if the scroller is currently performing a scroll action (maps to the UIScrollView *dragging* property). This is a boolean value.
decelerating	read-only	Returns true if the scroll is currently decelerating after a scroll action (maps to the UIScrollView *decelerating* property). This is a boolean value.

Actions

iphoneControlDo *id,* "flashScrollIndicators"

Makes the scroll indicators flash momentarily.

Messages

scrollerBeginDecelerate

This message is sent when scrolling is about to start decelerating.

scrollerEndDecelerate

This message is sent when scrolling has finished decelerating.

scrollerScrollToTop

This message is sent when the scroller is scrolled to top by touching the status bar.

scrollerBeginDrag

This message is sent when a scroll initiating drag is started.

scrollerEndDrag *didDecelerate*

This message is sent when a scroll initiating drag is finished.

scrollerDidScroll *hScroll, vScroll*

This message is sent when the scroll properties of the scroller have changed.

Player control – MPMoviePlayerController

An MPMoviePlayerController control is created using a control type of "player". For full details of what the MPMoviePlayerController control is capable of, and background about it see the iOS Reference Document.

On iOS versions < 4.2 you can only have a single MPMoviePlayerController instance in existance at once. Therefore, on these iOS versions you can only create a single native player control at any one time, and while one is present you cannot use the **play video** command to play fullscreen videos.

On iOS version >= 4.2, while you can have multiple MPMoviePlayerController instances (and thus multiple native player controls) simultaneously, only a single one can be playing at any one time.

Note: *The player control is only available on iOS 4.0 and later.*

Properties

filename	read/write	The filename of URL of the media to play. Setting the filename of the player automatically 'prepares' the movie for playback.
fullscreen	read/write	Determines whether the player's content is played fullscreen. This is a boolean value.
preserveAspect	read/write	Determines whether the player's content should preserve its aspect ratio when scaled to fit within the control's bounds. This is a boolean value.
showController	read/write	Determines whether the controller will be displayed over the content. This is a boolean value.
useApplicationAudioSession	read/write	Determines whether the movie uses a system-supplied audio session or not (maps to the native useApplicationAudioSession property). This is a boolean value.
shouldAutoplay	read/write	Determines whether the playback of network-based content begins automatically when there is enough buffered data to ensure uninterrupted playback (maps to the native shouldAutoplay property). This is a boolean value.
looping	read/write	Determines whether the playback of the movie should loop indefinitely. This is a boolean value.
allowsAirPlay	read/write	Determines whether a control should be presented to allow the user to choose AirPlay-enabled hardward for playback (maps to the native allowsAirPlay property). This is a boolean value.

Note: *This property is only supported on iOS 4.3 and later.*

duration	read-only	The duration of the movie, measured in milliseconds (maps to the native duration property). If the duration of the movie is not yet known, 0 is returned. If the duration is subsequently determined, an appropriate *playerPropertyAvailable* message is sent and the property updated. This is an integer value.
playableDuration	read-only	The amount of currently playable content, measured in milliseconds (maps to the native playableDuration property). This is an integer value.
currentTime	read/write	The current position of the playhead, measured in milliseconds (maps to the native currentPlaybackTime property). This is an integer value.
startTime	read/write	The position at which playback should start, measured in milliseconds (maps to the native initialPlaybackTime property). This is an integer value.
endTime	read/write	The position at which playback should end, measured in milliseconds (maps to the native endPlaybackTime property). This is an integer value.
playRate	read/write	The current playback rate for the player (maps to the native currentPlaybackRate property). This represents a multiplier for the default playback rate of the current content. A value of 0.0 indicates playback is stopped, while a value of 1.0 indicates normal speed. Positive values indicate forward playback, while negative values indicate reverse playback. This is real value.
loadState	read-only	The network load state of the player (maps to the native loadState property). This is a comma-delimited list of zero or more of the following: • *playable* – enough data is available to start playing, but it may run out before playback finishes • *playthrough* – enough data has been buffered for playback to continue uninterrupted • *stalled* – buffer of data has stalled and playback may pause automatically if the player runs out of data. This is a string value.
playbackState	read-only	The current playback state of the player (maps to the native playbackState property).

This is one of the following values:

- *stopped* – playback is stopped and will commence from the beginning when started.
- *playing* – playback is current underway
- *paused* – playback is paused and will resume from the point it was paused
- *interrupted* – playback is temporarily interrupted, perhaps because the buffer ran out of content
- *seeking forward* – the player is currently seeking towards the end of the movie
- *seeking backward* – the player is currently seeking towards the beginning of the movie

naturalSize — read-only — The raw size of a video frame in pixels (maps to the native naturalSize property).

This is a comma-separated list of two integers, the first is the width, the second is the height.

Actions

iphoneControlDo *id,* "play"

Start playing the content of the player.

iphoneControlDo *id,* "pause"

Pause the content at the current position.

iphoneControlDo *id,* "prepareToPlay"

Make the content ready to play, but don't actually commence playback.

iphoneControlDo *id,* "stop"

Stop playing the content of the player.

iphoneControlDo *id,* "begin seeking forward"

Start seeking forward through the content of the player.

iphoneControlDo *id,* "begin seeking backward"

Start seeking backward through the content of the player.

iphoneControlDo *id,* "end seeking"

Stop seeking through the content of the player.

iphoneControlDo *id,* "snapshot" | "snapshot exactly", *time*, [*maxWidth*, *maxHeight*]

Take a snapshot of the movie at *time* milliseconds from the beginning. If the exactly form is specified the frame produced will be as close as possible to *time*, otherwise the nearest key-frame will be used.

If *maxWidth* and *maxHeight* are specified, the snapshot will be scaled to fit within a rectangle

of that size but preserving the frame's aspect ratio.

The snapshot is made available as a new image object cloned from **the templateImage**, with data in the format as specified by the global **paintCompression** property.

Messages

playerPropertyAvailable *propertyName*

Enough data has become available to make the given *propertyName* available. Properties that might not be available immediately are *duration* and *naturalSize*.

playerProgressChanged

The *loadState* property has changed value.

playerEnterFullscreen

The player has entered full screen mode.

playerLeaveFullscreen

The player has left full screen mode.

playerMovieChanged

The content of the player has changed.

playerFinished

The content has finished playing through.

playerStopped

The content finished playing through due to a user exit.

playerError

The content finished playing due to an error.

Input control – UITextField

A UITextField control is created using a control type of "input". For full details of what the UITextField control is capable of, and background about it see the iOS reference document.

The input control allows the editing of a single line of text, with the 'return' key ending editing and allowing the application to perform an appropriate action.

Properties

text	read/write	The content of the control (maps to the native text property). This is a string value.
unicodeText	read/write	The content of control encoded as UTF-16 (maps to the native text property). This is a binary value.
textColor	read/write	The color to use for the text in control (maps to the native textColor

		property). This is either a standard color name, or a string of the form *red,green,blue* or *red,green,blue,alpha*. Where the components are integers in the range 0 to 255.
fontName	read/write	The name of the font to use for text in the control. This is a string value.
fontSize	read/write	The size of the font to use for text in the control. This is an integer value.
textAlign	read/write	The alignment to use for text in the control (maps to the native textAlignment property). This is one of *left*, *center* or *right*.
autoFit	read/write	Determines whether the size of the text is scaled so that it fits within the width of the control down to the size specified by the minimumFontSize property (maps to the native adjustsFontSizeToFitWidth property). This is a boolean value.
minimumFontSize	read/write	The minimum size text should be shrunk to to satisfy autoFit requirements (maps to the native minimumFontSize property). This is an integer value,
autoClear	read/write	Determines whether the control is emptied automatically when editing begins (maps to the native clearsOnBeginEditing property).
clearButtonMode	read/write	The display mode of the standard 'clear' button overlay (maps to the native clearButtonMode property). This is one of the following: • *never* – never display the clear button • *while editing* – only display the clear button while editing • *unless editing* – only display the clear button when not editing • *always* – always display the clear button
borderStyle	read/write	The type of border to draw around the control (maps to the native borderStyle property). This is one of the following: • *none* – do not draw a border • *line* – draw a thin line around the control • *bezel* – draw a bezel-style border around the control • *rounded* – draw a rounded rectangle style border around the control

editing	read-only	Indicates whether the control is currently being edited or not (maps to the native editing property). This is a boolean value.
autoCapitalizationType	read/write	Determines when the shift-key is automatically enabled (maps to the native autocapitalizationType property). This is one of the following: • *none* – the shift-key is never automatically enabled • *words* – the shift-key is enabled at the start of words • *sentences* – the shift-key is enabled at the start of sentences • *all characters* – the shift-key is enabled at the start of each character
autoCorrectionType	read/write	Determines whether auto-correct behavior should be enabled (maps to the native autocorrectionType property). This is one of the following: • *default* – use the appropraite auto-correct behavior for the current script system. • *no* – disable auto-correct behavior • *yes* – enable auto-correct behavior
manageReturnKey	read/write	Determines whether the return key should be automatically enabled or disabled based on whether the control has content or not (maps to the native enablesReturnKeyAutomatically property). This is a boolean value.
keyboardStyle	read/write	Determines what kind of appearance the keyboard has (maps to the native keyboardAppearance property). This is one of the following: • *default* – the standard keyboard appearance • *alert* – the keyboard that is suitable for an alert panel (iPhone/iPod only)
keyboardType	read/write	Determines what kind of keyboard should be displayed (maps to the native keyboardType property). This is one of the following: • *default* – the normal keyboard • *alphabet* – the alphabetic keyboard • *numeric* – the numeric keyboard with punctuation • *url* – the url entry keyboard • *number* – the number pad keyboard

- *phone* – the phone number pad keyboard
- *contact* – the phone contact pad keyboard
- *email* – the email keyboard
- *decimal* – the decimal numeric pad keyboard (iOS 4.1+)

returnKeyType — read/write — Determines what kind of return-key the keyboard should have (maps to the native returnKeyType property).

This is one of the following:

- *default* – the normal return key
- *go* – the 'Go' return key
- *google* – the 'Google' return key
- *join* – the 'Join' return key
- *next* – the 'Next' return key
- *route* – the 'Route' return key
- *search* – the 'Seach' return key
- *send* – the 'Send' return key
- *yahoo* – the 'Yahoo' return key
- *done* – the 'Done' return key
- *emergency call* – the 'emergency call' return key

contentType — read/write — Determines what kind of content the control contains.

This is one of the following:

- *plain* – plain, unstyled text
- *password* – plain text displayed in the standard iOS password style.

enabled — read/write — Determines whether the control is enabled or not.

This is a boolean value.

Actions

iphoneControlDo *id,* "focus"

Focus on the control, displaying the keyboard if necessary.

Messages

inputBeginEditing

The control has become focused and editing has commenced.

inputEndEditing

The control has lost focus and editing has ceased.

inputTextChanged

An editing operation has taken place and the content of the control has changed.

inputReturnKey

The return key has been pressed and focus removed from the input control.

Miscellaneous Information

Encryption Compliance – HTTPS

Any applications using encryption need to declare this fact to Apple when binaries are submitted in iTunesConnect.

Use of the HTTPS protocol counts as using encryption and as such you must answer 'Yes' to the relevant question when prompted to do so. This may result in you needing to go through the CCATS approval process for your application depending on how the encryption is used.

A useful blog post on this matter can be found here: http://blog.theanimail.com/iphone-encryption-export-compliance-for-apps.

For reference, the iOS engine currently utilizes the iOS built-in APIs for HTTPS.

Note: *It is your responsibility to ensure that you comply with any and all requirements with regards encryption usage – it is not something that can be done 'once' for the iOS engine as it (alone) does not constitute an application nor does it (alone) actually utilize encryption, it merely provides the means to do so.*

Noteworthy Changes

Scrolling problems (R18)

In previous builds, the browser, scroll and photo-picker features suffered a serious bug which caused scrolling and related actions to stick/not-decelerate and generally not work correctly. These problems have been resolved in this release.

Browser loadRequest changes (R18)

Due to technical limitations it has been necessary to change the *loadRequest* callback feature of the browser native control.

The *loadRequest* message is now only sent if the *delayRequests* property has been set to true (it is false by default).

When using the browser with *delayRequests* set to true, please bear in mind that any loads into sub-documents will end up being loaded into the main view meaning many websites will not function correctly.

With *delayRequests* set to false, websites will load as they should however you will not be able to prevent load requests from taking place.

URL progress parameter order (R18)

In previous builds the parameter order of the *urlProgress* message was inconsistent with both other callbacks in the iOS engine, and the libUrl status callback.

This has been rectified in this build. The urlProgress message now has the url as the *first* parameter rather than the second.

Scripts that make use of this feature will need to be updated and swap the first two parameters around.

Initial orientation handling (R20)

The engine will now read the initial orientation and supported orientations from the plist on startup, and uses these values to initialize the 'allowed orientations' and to ensure the initial orientation is as expected.

For more details see the orientation handling section.

Font metrics (R20)

The vertical metrics (ascent/descent) have been adjusted slightly in this release to make them more consistent with those on the Mac desktop. Assuming the chosen fonts match, then text layout on screen in the IDE will now be much closer, if not identical, to that in the simulator or on a device.

Out-of-bounds scrolling (R20)

The properties *unboundedHScroll* and *unboundedVScroll* have been added to control the 'out-of-

bounds' scrolling feature of group objects. If you are using this feature to provide the 'bounce' effect when using a scroller control in conjunction with a group, then you must now explicitly set the appropriate properties on the group. (Previously the behavior was always on).

Screen metrics (R25)

The *screenRect* and *working screenRect* properties have been changed to reflect the current orientation of the device.

Examples:

- on an iPad for which the current orientation is landscape and the status bar hidden the properties will both return (0, 0, 1024, 768).
- on a non-Retina iPhone in portrait with the status bar displayed, screenRect will be (0, 0, 320, 480) and working screenRect will be (0, 20, 320, 480).
- on a Retina iPhone in landscape with the status bar displayed and device resolution turned on, screenRect will be (0, 0, 640, 960) and working screenRect will be (0, 40, 640, 960).

This change has been made as it is more consistent with transparent handling of orientation and improves consistency with the now implemented screen snapshot feature.

Multi-channel sound playback (R29)

The behavior of **iphonePlaySoundOnChannel** has been adjusted in the case where there is no sound currently scheduled, and the *next* is specified. This usage will now cause the sound to be scheduled to play 'now', but instead of playing the channel will be prepared and then immediately paused.

The previous behavior can be obtained by doing **iphoneResumePlayingOnChannel** immediately after the play command, as this has no effect if the channel is already playing.

'Exits on Suspend' support (R30)

The support for changing the value of the 'Exits on Suspend' plist flag in standalone settings has been removed and the property will always be YES. This flag was never directly supported and having it set to NO causes the LiveCode engine to work incorrectly after it is resumed. Support will be reintroduced in due course, when the issues related to its use are resolved.

Change Logs and History

Engine Change History

pre-alpha-3 (2010-03-04)	MW	Initial version.
pre-alpha-4 (2010-03-05)	MW	Bold and italic font styles now honoured in font selection Image picker no longer 'sticks' after selection GIF images now display Max width and height parameters added to iphonePickPhoto Import snapshot no longer crashes
pre-alpha-5 (2010-03-11)	MW	Unicode text will now display Umlauts and other non-ASCII characters will now display Return key now causes a newline in fields Crashes when changing image content have been fixed Export snapshot now makes images with the correct colors Rotating a non-masked images no longer causes corruption of the image
pre-alpha-6 (2010-03-18)	MW	Answer command now returns its the chosen button in 'it' Added support for detecting device orientation Added support for setting interface orientation Added basic support for CoreLocation Refined control hit-test for touch handling so disabled controls are not targetted. mouseLoc now reports the correct y-coordinate Added support for mail composition/sending Corrected file handling functions interpretation of '/' Added support for specialFolderPath() Fixed problem with incorrect display of animated GIFs
pre-alpha-7 (2010-03-29)	MW	Added basic support for 'play <soundfile>' Added support for 'beep' system alert Added support for 'launch url' Added support for 'the fontNames' and 'the fontStyles' Added support for 'uniEncode' and 'uniDecode' Added support for system date/time Fixed issue with engine not picking up 'Oblique' fonts for italic style Fixed issue with unicode text not displaying in fields on load Added support for 'engine' in 'specialFolderPath'
pre-alpha-8 (2010-04-12)	MW	Added support for targetting iPad Added support for 'the systemVersion' Added support for 'the machine' Added support for 'the processor' Fixed problem with orientation returning portrait mispelt Improved responsiveness of image picker Added support for iphonePickPhoto on the iPad
pre-alpha-10 (2010-08-12)	MW	Added support for 'play video <filename/url>' Fixed issues with support for environment properties (the

		systemVersion, the processor, etc.)
		Added support for 'the sound'
		Fixed issue with garbage being returned from specialFolderPath in some cases.
		Added support for 'libUrlDownloadToFile'
pre-release-14 (2010-11-10)	MW	Added support for 'load url'
		Added support for 'post url'
		Added support for status bar configuration
		Added support for building for appstore/ad-hoc
		Added support for visual effects
		Fixed issue with iphonePickPhoto crashing on iOS4
		Fixed issue with some PNGs not displaying correctly
		Fixed issue with graphic effects have inverted colors
		Fixed issue with black screen appearing on startup
		Fixed issues with landscape orientation mode
release-17 (2010-12-01)	MW	Added support for browser native control
		Added support for scroller native control
		Added support for querying current locale and preferred languages
		Added support for 'movieTouched' message while movie playing
		Added supoprt for 'play stop' command while movie playing
		Added support for building iOS apps with evaluation licenses
		Added support for modal pickwheel, and hooked into option menus
		Changed support for orientation handling to leverage built-in iOS mechanism
		Fixed various glitches with movie playback
		Fixed issue with entering accented characters with the iOS keyboard
		Fixed issue with visual effects not working correctly in non-portrait orientation
		Fixed issue with 'the mouseColor' causing a crash
release-18 (2010-12-10)	MW	Added support for opaque, alpha and backgroundColor properties to all native controls.
		Added ability to upload to FTP.
		Added support for ask question/ask password
		Added delayRequests property to browser to control loadRequest message.
		Changed urlProgress callback parameter order for consistency
		Fixed bug with movie controller not working
		Fixed bug with the browser load action not working
		Fixed bug with scroller, browser and photo-picker not scrolling correctly.
		Fixed decelerating property name
		Fixed common native control property getting (rect, visible etc.)
		Fixed bug with browser not loading some pages correctly
		Fixed bug with iphonePickPhoto not returning correct value
release-19 (2010-12-16)	MW	Added support for keyboardActivated and keyboardDeactivated.

		Improved 'pick' and option menu display for iPad Hooked URL operation timeouts to *the socketTimeout* Fixed bug with pick views not respecting orientation properly
release-20 (2010-12-19)	MW	Added support for setting keyboard type and return style Added support for *unboundedHScroll* and *unboundedVScroll* propertie. Fixed return value of iphonePick command Fixed alignment of ask dialog when no title Fixed return value of ask dialog when nothing is entered Fixed answer and other such commands not working in initial preOpenStack/preOpenCard/openStack/openCard Fixed issue with nested answer/ask dialogs Fixed issue with nested orientationChanged messages Revised and improved initial orientation handling Made nomenclature for device and interface orientation consistent Improved vertical font metrics consistency with the desktop (Mac) engine. Made round rectangle corner radius consistent with desktop engine.
release-21 (2010-12-21)	MW	Added support for *the httpHeaders*. Added experimental support for multi-channel sound. Fixed issue with iphoneControl commands not working with a string id. Fixed issue with pick-wheel not scrolling correctly. Fixed issue with pick-wheel crashing after multiple shows. Fixed issue with option menu popup not sending menuPick.
release-22 (2011-01-07)	MW	Added support for looping video (iOS 3.2 and later) Fixed issue with stackfiles not saving on iOS Fixed crash on startup when running on iOS 3.1.3 device Fixed movieTouched message handling on iOS 3.1.3 Further improved vertical font metrics consistency with the desktop (Mac) engine
release-23 (2011-01-14)	MW	Added support for remaining UIWebView properties Added ability to determine display scale (retina, or non-retina) Added support for visual effects in sub-regions Fixed issue with type command and accented chars (9294) Fixed issue with pick-wheel causing anomalous keyboard behavior (9295) Multi-channel sound is no longer considered experimental
release-24 (2011-01-24)	MW	Added support for revZip, revXML and dbSQlite externals Added support for configuring the iOS 'idleTimer' Changed iPad popup lists to dismiss after select Fixed issue with iphonePickPhoto not working correctly with camera import (9303) Fixed issue with export/import snapshot not working correctly (9307) Fixed issue with interface orientation on startup (9314)

		Fixed issue with 'the files' and 'the folders' being urlEncoded
release-25 (2011-01-30)	MW	Added support for import/export snapshot from rect (9343) Added support fetching camera info: *iphoneCameraFeatures()* Added support for picking from specific camera Added support for different background colors to flip visual effect Added support for listing native controls: *iphoneControls()* Added support for creating named native controls Added support for clearing pending interactions with *iphoneClearTouches* Change screenRect properties to take into account orientation.
release-26 (2011-02-04)	MW	Fixed issue with movie playback with controller on 3.2 (9319) Fixed issue with topLeft of stack being incorrect (9371)
release-27 (2011-02-08)	MW	Fixed issue with sound when movie 'shrunk' on 3.2 (9319)
release-28 (2011-03-04)	MW	Added support for system identifier: *iphoneSystemIdentifier()* Improved ask dialog implementation (9379) Fixed issue with pixel properties requiring open stack (9419) Fixed issue with non-breaking spaces causing compile issues
release-29 (2011-03-09)	MW	Added experimental support for 'player' native control Added experimental support for 'input' native control Added experimental support for network reachability tracking Added experimental support for playing sections of video Added support for preparing a sound on a channel without playing Added support for pausing and resuming a sound channel Added support for querying a sound channel's status Fixed issue with rotation and movie playback (9409)
release-30 (2011-03-13)	MW	Added experimental support for improved email composition Fixed issue with stack positioning after movie playback (9409)
release-31 (2011-03-15)	MW	After further testing and consideration, removed 'experimental' status from native player control. After further testing and consideration, removed 'experimental' from improved email composition support. Improved interoperability of play video and native player control on iOS < 4.2.
release-32 (2011-03-16)	MW	Made all iphoneComposeMail arguments are optional. Tweaked iphonePick to use checkmark on iPad. Fixed bug with iphonePick not returning 0 if nothing selected.
release-33 (2011-03-17)	MW	Support for pattern fills has been implemented.
release-34 (2011-03-18)	MW	After further testing and consideration, removed 'experimental' status from native input control. Improved iphonePickPhoto on iPad to make it display relative to **the target** and allow orientation changes. Added 'inputReturnKey' message to native input control. Added 'enabled' property to native input control. Fixed issue with orientation lock when displaying fullscreen movies from UIWebViews and native player controls. Fixed issue with stack view not being placed corrected after

		returning from fullscreen movies. Fixed issue with opaque property not taking effect properly. Fixed issue with streaming playback not working player control. Fixed issue with player control not preparing movies after changing filename. Changed player loadState property to a set, as it should be.
release-35 (2011-03-21)	MW	Fixed memory leak in backgroundPattern support. Fixed crash when doing 'ask' without a 'titled' clause. Fixed file access mode when doing 'open file' with no mode.
release-36 (2011-03-25)	MW	Fixed issue with iphonePick initial index not working on iPhone

iOS Deployment Change History

pre-alpha-3 (2010-03-04)	MW	Initial version.
pre-alpha-4 (2010-03-05)	MW	Bundle identifier setting no longer lost on reload
pre-alpha-5 (2010-03-11)	MW	Project settings are no longer lost when adding/removing files
pre-alpha-10 (2010-08-12)	MW	Added support for configuring SDK roots Added support for adding folders of files to the app bundle.
pre-release-14 (2010-11-10)	MW	Added support for ad-hoc and store profiles Added support for specifying a splash screen Added support for copying in icons of different resolutions Added support for plist configuration Fixed issue with app bundle not being deleted before rebuilding
release-17 (2010-12-01)	MW	Integrated plugin's functionality into IDE Simulate start/stop buttons replaced by single menubar 'Simulate' button Deploy button replaced by standard 'Save as Standalone Application' action Plist editor integrated as new Standalone Builder pane Simulator selection moved to Simulator Version menu item of Development menu SDK configuration moved to 'Mobile Support' pane of preferences Added support UIFileSharingEnabled plist tag Added ability to choose device type for simulator (iPad/iPhone) Fixed issue with launch image filenames not being correctly computed (and thus failing to copy into the bundle)
release-18 (2010-12-10)	MW	Added 'Simulate' menu item to menubar
release-19 (2010-12-16)	MW	Simulator options now remembered as global preferences Simulator options filtered by minimum version and device family in Standalone Settings. Appropriate warnings and messages added when invalid folders are included in the app bundle via 'Copy Files'
release-20 (2010-12-18)	MW	Improved UI for orientation settings in standalone builder. Fixed issue with wrong provisioning profile being included when more than 9 are installed. Fixed issue with Simulate getting confused with some stack names.
release-21 (2010-12-21)	MW	MinimumOSVersion now correctly included in plist

release-22 (2011-01-07)	MW	Added support for DataGrid Added support for UIPrerenderedIcon plist tag
release-23 (2011-01-14)	MW	No changes
release-24 (2011-01-24)	MW	Added support for including revZip, revXML and dbSqlite externals (to be improved later).
release-25 (2011-01-30)	MW	No changes
release-26 (2011-02-04)	MW	No changes
release-27 (2011-02-08)	MW	No changes
release-28 (2011-03-04)	MW	Added support for iOS 4.3 simulator and device builds Improved code-signing identity detection and use
release-29 (2011-03-09)	MW	Added support for including custom fonts (iOS 3.2+) Added support for including custom externals Fixed issue with deploying to 4.2 simulator
release-30 (2011-03-13)	MW	Removed 'Exits on Suspend' option from settings pane and forced to always be YES.
release-31 (2011-03-15)	MW	No changes.
release-32 (2011-03-16)	MW	Fixed issue where not setting the bundle version explicitly would block the Application Loader from uploading an app.
release-33 (2011-03-17)	MW	Fixed issue with identities containing non-ASCII characters not working, Fixed issue with splash image requiring a relative path in educational and personal editions.
release-34 (2011-03-18)	MW	No changes.
release-35 (2011-03-21)	MW	No changes.
release-36 (2011-03-21)	MW	No changes.

Document History

Revision 1 (2010-03-04)	MW	Initial version.
Revision 2 (2010-03-05)	MW	Added documentation for new iphonePhotoPick parameters
Revision 3 (2010-03-11)	MW	Improved consistency of syntax specifications and use Refined documentation for touch events Added new section about mouse events Added new section on restrictions to dynamic features Restructured headings to make sure PDF index works
Revision 4 (2010-03-18)	MW	Added section on orientation handling Added section on location handling Refined statements about the mouseLoc Refined description of touch handling with regard to hit-testing Clarified support for dynamic chunks Added section on email composition Added section file handling Clarified blocking behavior of non-file url's
Revision 5 (2010-03-29)	MW	Added section on system alerts Added section on sound support Added section on url launching Added section on font querying Added description of *engine* parameter to specialFolderPath
Revision 6 (2010-04-12)	MW	Revised setting up your system with regard iPad support

		Added section on hardware and system version querying
Revision 7 (2010-08-12)	MW	Revising initial sections to include details of SDK configuration and requirements. Revised 'Sound file support' section to include 'the sound'. Added 'Video file support' section. Revised 'Non-file url' section to include 'libUrlDownloadToFile' Revised 'The revMobile Plugin' section to include changes to UI. Revised 'Projects and Files' section to include details about adding folders.
Revision 8 (2010-09-16)	MW	Rebranded from revMobile to iOS Deployment Rebranded from rev* to LiveCode Removed section on dynamic language features as no longer relevant.
Revision 9 (2010-11-10)	MW	Various edits to improve language. Expanded section on url commands Added section on visual effects Added section on status bar configuration Revised 'The Deployment Plugin' section Revised the non-test deployment section Added a section on the plist editor Added a section on launch images Added a section on splash images
Revision 10 (2010-12-01)	MW	Rewrote and reorganised initial sections to reflect new integration into the IDE. Rewrote section on orientation handling. Added section on native controls and further sub-sections on browser and scroller controls. Added section of locale and system language query support. Revised the play video section.
Revision 11 (2010-12-04)	MW	Added section on 'Engine Version'
Revision 12 (2010-12-10)	MW	Corrected iphoneControlDestroy to iphoneControlDelete Corrected declerationRate to decelerationRate Corrected description of flashScrollIndicators Added a section on common control properties and actions Added section on out-of-bounds group scrolling Added section on noteworthy changes Updated browser control section Updated movie playback section Updated non-file URL section Updated orientation handling section to describe how to lock orientations to a specific set on startup
Revision 13 (2010-12-16)	MW	Correct description of 'the sound' Added section on 'copy files restrictions' Added section on 'encryption compliance' Updated keyboard section with new messages
Revision 14 (2010-12-19)	MW	Revised section on orientation handling Added a note about URL format to non-file URL access section Added note about nesting to ask/answer dialog section

		Added a note about 'go in window' to stacks/windows section Added a note about changes to out-of-bounds scrolling Updated keyboard section with new commands Updated out-of-bounds scroll section with details of new properties.
Revision 15 (2010-12-22)	MW	Added section on multi-channel sound support. Updated non-file URL access with details of *the httpHeaders*.
Revision 16 (2011-01-07)	MW	Updated video playback section to describe how to loop Updated 'what doesn't work' to mention painting tools
Revision 17 (2011-01-14)	MW	Updated browser control section with new properties Updated resolution handling section to mention *iphoneDeviceScale()*
Revision 18 (2011-01-24)	MW	Added section of configuring the idle timer. Added section on externals. Updated section on what does/what doesn't work to include externals correctly.
Revision 19 (2011-01-30)	MW	Added a section on snapshot capabilities Added a section on querying camera capabilities Added a section on handling pending interactions Updated native controls section to mention control listing and naming Updated section on visual effects to mention new flip behavior
Release 20 (2011-02-04)	MW	Corrected description of *mediaPlaybackRequiresUserAction*
Release 21 (2011-02-08)	MW	No changes
Release 22 (2011-03-04)	MW	Added mention of *iphoneSystemIdentifer()* to hardware querying section. Added reference to 4.3 SDK in installation section.
Release 23 (2011-03-09)	MW	Added section on native 'player' control Added section on native 'input' control Added section on network reachability tracking Added section on including custom fonts Added section on including custom externals Updated section on video playback to mention playing sections of video Updated section on multi-channel sound to mention pause, resume, status and preparing.
Revision 24 (2011-03-13)	MW	Updated section on email composition Updated noteworthy changes section Replaced all curly double quotes with plain double quotes Corrected borderStyle 'round' to be 'rounded'
Revision 25 (2011-03-15)	MW	Improved notes on iOS version compatibility of native player control.
Revision 26 (2011-03-16)	MW	Clarified iphonePick command behavior.
Revision 27 (2011-03-17)	MW	Corrected iphoneComposeMail syntax.
Revision 28 (2011-03-18)	MW	Corrected iphoneComposeEmail to iphoneComposeMail Corrected *content* to *filename* in native player control. Updated section on iphonePickPhoto to mention iPad behavior. Updated section on native input control to mention changes.

		Updated section on native player control to mention changes. Clarified that iphonePick indices are 1-based. Changed support for native player control to be iOS 4.0+.
Revision 29 (2011-03-21)	MW	Updated screenshots as appropriate.
Revision 30 (2011-03-25)	MW	No changes.