LiveCode 9.5.0-dp-1 Release Notes

Overview

LiveCode 9.0 enables access to libraries and platform APIs written in many other languages thanks to the community-funded 'Infinite LiveCode' project.

This includes a greatly improved LiveCode Builder virtual machine.

LiveCode 9.0 contains many additional improvements to support LiveCode app developers, including:

- A new "spinner" widget
- OAuth2 authentication library for use with web APIs (e.g. Facebook, Google and GitHub)
- A command argument parser library for building command-line standalones
- Updates and performance improvements for existing widgets

Known issues

- The installer will currently fail if you run it from a network share on Windows. Please copy the installer to a local disk before launching on this platform.
- The browser widget does not work on 32-bit Linux.
- 64-bit standalones for Mac OS X do not have support for audio recording.

Breaking changes

Boolean constants

In this release, boolean constants `true` and `false` have been changed so that they resolve to values of boolean type (rather than string). This will affect any uses of the `is strictly` operator on such values, i.e. previously the following were true:

true is strictly a string false is strictly a string

Now, they are both false, and the following are true:

true is strictly a boolean false is strictly a boolean

Boolean constants passed as elements of arrays to LCB handlers will not require conversion to boolean values in LCB - in fact any attempt to do so assuming they are strings will cause an error. Any array elements which are intended to be booleans in LCB should be checked for their type before conversion. For example, any of the following could be done by an LCB library user:

```
put true into tArray["enabled"]
put "true" into tArray["enabled"]
put (tVar is not "enabled") into tArray["enabled"]
```

An LCB handler to which `tArray` is passed should do the following:

```
variable tEnabled as Boolean
if tArray["enabled"] is a boolean then
    put tAction["enabled"] into tEnabled
else
    put tAction["enabled"] parsed as boolean into tEnabled
end if
```

Infinity constant

The constant `infinity` has been added to the language in this release. As a result, the unquoted literal `infinity` is now reserved. Any existing uses of it should be quoted, as otherwise it will resolve to the floating point value representing infinity, rather than the string "infinity".

Implicit object

A number of LCB commands use an implicit object to provide context for their execution. Some of these commands also allow specifying an explicit object. These commands are:

- `execute script`
- `send`
- `post`
- `image from file`
- `resolve file` - new in this version

In previous releases `execute script` and `image from file` would use `this card of the defaultStack` as the implicit object even if called from a widget. The `send` and `post` commands, however, used `this card of the defaultStack` when in a library module handler and the host widget when in a widget module handler. This release changes `execute script` and `image from file` to also use the host widget as the implicit object. This means, for example, that `image from file` will resolve a relative file path relative to the `stackFile` the host widget is on rather than the `stackFile` of the `defaultStack`.

Platform support

The engine supports a variety of operating systems and versions. This section describes the platforms that we ensure the engine runs on without issue (although in some cases with reduced functionality).

Windows

LiveCode supports the following versions of Windows:

- Windows 7 (both 32-bit and 64-bit)
- Windows Server 2008
- Windows 8.x (Desktop)
- Windows 10

Note: On 64-bit Windows installations, LiveCode runs as a 32-bit application through the WoW layer.

Linux

LiveCode supports the following Linux distributions, on 32-bit or 64-bit Intel/AMD or compatible processors:

- Ubuntu 14.04 and 16.04
- Fedora 23 & 24
- Debian 7 (Wheezy) and 8 (Jessie) [server]
- CentOS 7 [server]

LiveCode may also run on Linux installations which meet the following requirements:

- Required dependencies for core functionality:

    - glibc 2.13 or later
    - glib 2.0 or later

- Optional requirements for GUI functionality:

    - GTK/GDK 2.24 or later
    - Pango with Xft support
    - esd (optional, needed for audio output)
    - mplayer (optional, needed for media player functionality)
    - lcms (optional, required for color profile support in images)
    - gksu (optional, required for privilege elevation support)

Note: If the optional requirements are not present then LiveCode will still run but the specified features will be disabled.

Note: The requirements for GUI functionality are also required by Firefox and Chrome, so if your Linux distribution runs one of those, it will run LiveCode.

Note: It may be possible to compile and run LiveCode Community for Linux on other architectures but this is not officially supported.

Mac

The Mac engine supports:

- 10.9.x (Mavericks)
- 10.10.x (Yosemite)
- 10.11.x (El Capitan)
- 10.12.x (Sierra)
- 10.13.x (High Sierra)
- 10.14.x (Mojave)

iOS

iOS deployment is possible when running LiveCode IDE on a Mac, and provided Xcode is installed and has been set in LiveCode *Preferences* (in the *Mobile Support* pane).

Currently, the supported versions of Xcode are:

- Xcode 6.2 on MacOS X 10.9
- Xcode 6.2 and 7.2 on Mac OS X 10.10
- Xcode 8.2 on MacOS X 10.11
- Xcode 9.2 on MacOS 10.12 (Note: You need to upgrade to 10.12.6)
- Xcode 10.1 on MacOS 10.13 (Note: You need to upgrade to 10.13.4)

It is also possible to set other versions of Xcode, to allow testing on a wider range of iOS simulators. For instance, on MacOS 10.12 (Sierra), you can add *Xcode 6.2* in the *Mobile Support* preferences, to let you test your stack on the *iOS Simulator 8.2*.

We currently support deployment for the following versions of iOS:

- 8.2 [simulator]

- 9.2
- 10.2
- 11.2
- 12.1

Android

LiveCode allows you to save your stack as an Android application, and also to deploy it on an Android device or simulator from the IDE.

Android deployment is possible from Windows, Linux and Mac OSX.

The Android engine supports devices using ARMv7 or ARMv8 processors. It will run on the following versions of Android:

- 4.1-4.3 (Jelly Bean)
- 4.4 (KitKat)
- 5.0-5.1 (Lollipop)
- 6.0 (Marshmallow)
- 7.x (Nougat)
- 8.x (Oreo)

To enable deployment to Android devices, you need to download the Android SDK, and then use the 'Android SDK Manager' to install:

- the latest "Android SDK Tools"
- the latest "Android SDK Platform Tools"

You also need to install the Java Development Kit (JDK). On Linux, this usually packaged as "openjdk". LiveCode requires JDK version 1.6 or later.

Once you have set the path of your Android SDK in the "Mobile Support" section of the LiveCode IDE's preferences, you can deploy your stack to Android devices.

Some users have reported successful Android Watch deployment, but it is not officially supported.

HTML5

LiveCode applications can be deployed to run in a web browser, by running the LiveCode engine in JavaScript and using modern HTML5 JavaScript APIs.

HTML5 deployment does not require any additional development tools to be installed.

LiveCode HTML5 standalone applications are currently supported for running in recent versions of Mozilla Firefox, Google Chrome or Safari. For more information, please see the "HTML5 Deployment" guide in the LiveCode IDE.

Setup

Installation

Each version of LiveCode installs can be installed to its own, separate folder. This allow multiple versions of LiveCode to be installed side-by-side. On Windows (and Linux), each version of LiveCode has its own Start Menu (or application menu) entry. On Mac OS X, each version has its own app bundle.

On Mac OS X, install LiveCode by mounting the `.dmg` file and dragging the app bundle to the `Applications` folder (or any other suitable location).

For Windows and Linux, the default installation locations when installing for "All Users" are:

Platform	Path
Windows	`<x86 program files folder>/RunRev/LiveCode <version>`
Linux	`/opt/livecode/livecode-<version>`

The installations when installing for "This User" are:

Platform	Path
Windows	`<user roaming app data folder>/RunRev/Components/LiveCode <version>`
Linux	`~/.runrev/components/livecode-<version>`

Note: If installing for "All Users" on Linux, either the **gksu** tool must be available, or you must manually run the LiveCode installer executable as root (e.g. using **sudo** or **su**).

Uninstallation

On Windows, the installer hooks into the standard Windows uninstall mechanism. This is accessible from the "Add or Remove Programs" applet in the windows Control Panel.

On Mac OS X, drag the app bundle to the Trash.

On Linux, LiveCode can be removed using the `setup.x86` or `setup.x86_64` program located in LiveCode's installation directory.

Reporting installer issues

If you find that the installer fails to work for you then please report it using the LiveCode Quality Control Centre or by emailing support@livecode.com.

Please include the following information in your report:

- Your platform and operating system version
- The location of your home or user folder
- The type of user account you are using (guest, restricted, admin etc.)
- The installer log file.

The installer log file can be located as follows:

Platform	Path
Windows 2000/XP	`<documents and settings folder>/<user>/Local Settings/`
Windows Vista/7	`<users folder>/<user>/AppData/Local/RunRev/Logs`
Linux	`<home>/.runrev/logs`

Activating LiveCode Indy or Business edition

The licensing system ties your product licenses to a customer account system, meaning that you no longer have to worry about finding a license key after installing a new copy of LiveCode. Instead, you simply have to enter your email address and password that has been registered with our customer account system and your license key will be retrieved automatically.

Alternatively it is possible to activate the product via the use of a specially encrypted license file. These will be available for download from the customer center after logging into your account. This method will allow the product to be installed on machines that do not have access to the internet.

Command-line installation

It is possible to invoke the installer from the command-line on Linux and Windows. When doing command-line installation, no GUI will be displayed. The installation process is controlled by arguments passed to the installer.

Run the installer using a command in the form:

```
<installer> install -ui [OPTION ...]
```

where `<installer>` should be replaced with the path of the installer executable or app (inside the DMG) that has been downloaded. The result of the installation operation will be written to the console.

The installer understands any of the following `OPTION`s:

Option	Description
`-allusers`	Install the IDE for "All Users". If not specified, LiveCode will be installed for the current user only.
`-desktopshortcut`	Place a shortcut on the Desktop (Windows-only)
`-startmenu`	Place shortcuts in the Start Menu (Windows-only)
`-location LOCATION`	The folder to install into. If not specified, the `LOCATION` defaults to those described in the "Installation" section above.
`-log LOGFILE`	The file to which to log installation actions. If not specified, no log is generated.

Note: the command-line installer does not do any authentication. When installing for "All Users", you will need to run the installer command as an administrator.

As the installer is actually a GUI application, it needs to be run slightly differently from other command-line programs.

On Windows, the command is:

```
start /wait <installer> install -ui [OPTION ...]
```

Command-line uninstallation

It is possible to uninstall LiveCode from the command-line on Windows and Linux. When doing command-line uninstallation, no GUI will be displayed.

Run the uninstaller using a command of the form:

```
<uninstaller> uninstall -ui
```

Where is *.setup.exe* on Windows, and *.setup.x86* on Linux. This executable, for both of the platforms, is located in the folder where LiveCode is installed.

The result of the uninstallation operation will be written to the console.

Note: the command-line uninstaller does not do any authentication. When removing a version of LiveCode installed for "All Users", you will need to run the uninstaller command as an administrator.

Command-line activation for LiveCode Indy or Business edition

It is possible to activate an installation of LiveCode for all users by using the command-line. When performing command-line activation, no GUI is displayed. Activation is controlled by passing command-line arguments to LiveCode.

Activate LiveCode using a command of the form:

```
<livecode> activate -file LICENSEFILE -passphrase SECRET
```

where `<livecode>` should be replaced with the path to the LiveCode executable or app that has been previously installed.

This loads license information from the manual activation file `LICENSEFILE`, decrypts it using the given `SECRET` passphrase, and installs a license file for all users of the computer. Manual activation files can be downloaded from the My Products page in the LiveCode account management site.

It is also possible to deactivate LiveCode with:

```
<livecode> deactivate
```

Since LiveCode is actually a GUI application, it needs to be run slightly differently from other command-line programs.

On Windows, the command is:

```
start /wait <livecode> activate -file LICENSE -passphrase SECRET
start /wait <livecode> deactivate
```

On Mac OS X, you need to do:

```
<livecode>/Contents/MacOS/LiveCode activate -file LICENSE -passphrase SECRET
<livecode>/Contents/MacOS/LiveCode deactivate
```

LiveCode Community engine changes

Math operation refactor

The math operations have been refactored to use common code for error checking. One of three different execution errors can now be thrown:

- "numeric: domain error"
- "numeric: range error (overflow)"
- "numeric: divide by zero"

The error checking depends solely on the finiteness or otherwise of the operation's input(s) and output.

A domain error occurs when a math operation, given finite inputs, results in not-a-number (NaN) - this is the case when the function is not defined for the given inputs, for example `acos(2)`; or the output does not exist in the extended real line (`ℝ ∪ {−∞, +∞}`), for example, `sqrt(-1)`.

A range error occurs when a math operation's output overflows given finite inputs, i.e. when the result is greater than the maximum value of a 64-bit floating point, for example `10^308 * 2`.

A divide by zero error occurs when a math operation causes division by zero either directly, for example `1/0` or `0^-1` or as part of its computation, for example `10 wrap 0`.

Math operations now do not throw execution errors when any of the inputs are non-finite, for example neither of `(1^(-inf) + inf) / 2 = inf` or `sqrt(-inf) = NaN` causes an execution error.

New container layer mode

Container layer mode support has been added to the accelerated rendering architecture.

The container layer mode only has an effect on unadorned groups whose ancestors are also container layer mode unadorned groups.

A container layer mode group provides a container for static and dynamic layers, allowing nested groups to also benefit from being cached for fast re-rendering.

For more information, see the **layerMode** entry in the dictionary.

Progress, isSecure and allowUserInteractin features added to browser widget

The message `browserProgressChanged` has been added to the browser widget to allow monitoring the progress of page loads.

The property `isSecure` has been added to the browser widget to determine if the content of the current URL has been loaded securely.

The property `allowUserInteraction` has been added to the browser widget to control if the browser should respond to user input.

See the dictionary for full documentation.

Deploy 64-bit Windows standalones

You can now deploy 64-bit standalones for Windows. The `Standalone Settings` dialog now has a `Windows x86` and a `Windows x86_64` checkbox allowing you to choose to build either or both 32-bit and 64-bit executables.

New keyboardType field property

A new property has been added to fields to control the keyboard type displayed on the mobile keyboard.

New Android Architectures

Android builds now support four architectures. Previously android was built for `armv6` only. Android is now built for `armv7`, `arm64`, `x86` and `x86_64`.

Checkboxes are included on in the Android standalone settings to choose which architectures to include in the build. When deploying your application via the Test button to a device the device architecture will be detected and used even if not chosen in standalone settings.

New mobileSetKeyboardDisplay and mobileGetKeyboardDisplay handlers

A new command `mobileSetKeyboardDisplay` has been added to support a `pan` mode where the view is panned up if the currently focused field control is not visible when the keyboard is shown. Use `mobileGetKeyboardDisplay` to get the current mode. There are two modes supported:

- `over` - the default where the keyboard displays over the stack
- `pan` - the view is panned up the minimum amount required to ensure the foucused field is visible.

Static linked code libraries for iOS device builds

The standalone builder now supports `.lcext` compiled objects that link static libraries used by a LCB module to the module compiled as C++ using lc-compile's `--forcebuiltins --outputauxc` options. Additionally, the `Using compiled libraries` section of the `Extending LiveCode` guide has been updated to describe the creation of `.lcext` objects.

New layerClipRect control property

A new property 'layerClipRect' has been added to all controls.

Use the layerClipRect property to clip an object's display to a rectangle. The clipping rectangle only changes what part of the object is rendered, it has no effect on interaction; in particular, mouse events will still occur as they would without it being set.

New `log` command and `logMessage` property

A new command (`log`) and global property (`logMessage`) have been added to allow an easy and low-cost method to disable or redirect script logs.

The `log` command invokes the handler named by the `logMessage` as though the `logMessage` were directly written in the script. For backwards compatability the default value of the `logMessage` is `log` so any scripts that currently have a `log` handler will continue to work. To allow this `log` has been special cased as both a command name and a permitted handler name.

If the `logMessage` is set to empty then the `log` command will not invoke any handler or evaluate any of the parameters in the argument list.

In this example the `log` command will not be called with `pInfo` as `loading resources` when the `uBuildMode` of the stack is `release`:

```
on preOpenStack
   -- uBuildMode property set before building standalone
   if the uBuildMode of this stack is "release" then
      set the logMessage to empty
   end if

   loadResources
end preOpenStack

command loadResources
   log "loading resources"
end loadResources

on log pInfo
   -- unhandled put will go to system logs
   put pInfo
end log
```

New returnKeyType field property

A new property has been added to fields to control the return key type displayed on the mobile keyboard.

Implement filter where clause

A new clause has been added to the filter command to filter where an expression evaluates to true. For example:

```
put "foo,bar,baz" into tList
filter items of tList where each begins with "b"
-- tList contains "bar,baz"
```

mobileSetKeyboardReturnKey on android

The mobileSetKeyboardReturnKey is now supported on android and the iphoneSetKeyboardReturnKey synonym is now deprecated

Real boolean constants

The constants `true` and `false` have been changed so that they are represented internally by values of boolean type, where previously they were strings.

This should have no effect on LiveCode applications other than in two cases:

- use of the `is strictly` operator: expressions like `true is strictly a string` now evaluate to false `true is strictly a boolean` to true.
- passing boolean constants as elements of arrays to LCB handlers.

In the latter case, LCB handlers should already be written to accept varying types of values coming in as elements of arrays as they could be generated by boolean expressions. Since a user may well do any of the following in LiveCode:

```
put true into tArray["enabled"]
put "true" into tArray["enabled"]
put (tVar is not "enabled") into tArray["enabled"]
```

any LCB handler to which `tArray` is passed should be doing something like:

```
variable tEnabled as Boolean
if tArray["enabled"] is a boolean then
    put tAction["enabled"] into tEnabled
else
    put tAction["enabled"] parsed as boolean into tEnabled
end if
```

Infinity

The constant `infinity` has been added to the language.

This constant resolves to the floating point value representing positive infinity. When `infinity` is used in math operations, they never throw errors. Usually they will return `infinity` or `NaN` (not a number)

```
(infinity / 0)^2 - 5 = infinity

infinity / infinity = NaN
```

Specific engine bug fixes (9.5.0-dp-1)

14316	**Don't throw error when infinity used innocuously in arithmetic expressions**
14721	**Implement 'go visible ' as an antonym to 'go invisible'**
18309	**Implement filter where clause**
20488	**Ensure the put command throws an error when a preposition is expected and not found**
20951	**Fix bug ignoring garbage values at end of repeat line**
	Add `extension` as a category to the `securityPermissions` property to

21343	**restrict the `load extension` command**
21464	**mobileSetKeyboardReturnKey on android**
21465	**New keyboardType field property**
21466	**New returnKeyType field property**
21467	**New mobileSetKeyboardDisplay and mobileGetKeyboardDisplay handlers**
21915	**Fix memory leak in 'the processor' function**
infinity	**Infinity**

LiveCode Community IDE changes

Accelerated DataGrid

The DataGrid has been updated to use the new container layer mode feature when running in form view mode.

To take advantage of this, the datagrid must be at top-level or contained withing groups all having container layer mode set. It must have showBorder set to false, and the acceleratedRendering property must be enabled on the stack with appropriate compositor property settings.

To get the maximum benefit from accelerated rendering, the behavior script for a row template should changing properties within the template unnecessarily.

A new datagrid property `minimal layout` has been added. When this property is true, a row template will only receive the `LayoutControl` message if its data or its width or height has changed as opposed to every time its rect changes (e.g. due to scrolling).

Specific IDE bug fixes (9.5.0-dp-1)

4382	**Add `Clear Recent Files` option to Open Recent File menu**
17476	**Use numeric sort for variables/keys in SE variables lists**
20121	**Icon Picker: Use a popup stack for icon property setting to allow filtering**
21514	**Automatically select text of fields in PI**

LiveCode Community extension changes

Tree View widget

Properties

The tree view widget now will automatically expand to reveal a row when it is selected. If **scrollHilitedElementIntoView** is not `true` then the scroll position will be adjusted to maintain the currently visible top row.

The tree view widget now has the ability to get and set the fold state of the selected element via the **hilitedElementFoldState** property. Values are `folded`, `unfolded`, and `leaf`.

Setting a value other than `folded` or `unfolded` will throw an error.

Setting a value when nothing is selected or setting a value on a leaf node will have no effect.

The **autoFoldStateReset** boolean property is added to allow the fold state to be reset when the array data is set.

Default value is `false` to match existing behavior.

The tree view widget now has the ability to automatically select a new row when it is added.

The tree view widget now has the ability to scroll the selected row into view. If `true`, this will happen when setting the **arrayData**, setting the **hilitedElement**, and when adding a new row (when **hiliteNewElement** is `true`).

Two properties have been added to achieve these options:

- **hiliteNewElement**: either `true` or `false`
- **scrollHilitedElementIntoView**: either `true` or `false`

The default values for both are `false` to match the behavior of previous versions of the widget.

When selecting a row that is partially visible, the view will be adjusted so that the full row is visible.

When changing the **readOnly** property, the view will only change position if the value is being set to `true` and the `Add new element` row is currently visible.

The tree view widget now has the ability to get and set the fold state.

One property has been added to achieve this option:

- **foldState**: array

The fold state array is structured as follows:

```
   [key1]
       ["folded"]
       ["array"]
           [subkey1]
               ["folded"]
   [key2]
       ["folded"]
```

Mac Status Menu library

Mac Status Menu

An mac status menu library has been added. Use the new library to create, delete and set properties on a status menu. The menu's `items` property uses the familiar menu text format from LiveCode menus.

Android Utilities module

Android Permission Checking

The ability to check Android permissions in LCB has been added to the android utility module.

Use these handlers to check and request permissions before accessing resources (e.g. camera access).

The following handler have been added:

- `AndroidRequestPermission` - Display a dialog requesting a given permission.
- `AndroidPermissionExists` - Check to see if a given permission name is valid.
- `AndroidHasPermission` - Check to see if a given permission has been granted.

See the dictionary for full details.

Specific extension bug fixes (9.5.0-dp-1)

19754 Enhance TreeView numeric sort to sort non-numbers alphabetically
21361 Add option to reset fold state when setting arrayData
21570 Add option to select new elements automatically
21659 Update TreeView to ignore mouseUp when row has changed

LiveCode Indy extension changes

PDF widget

PDF Widget

A PDF widget has been implemented for all platforms with the exception of HTML5 and is available in LiveCode Business. The widget has a wide range of properties allowing you to alter the appearance of the widget, the way the user can interact with the document and providing details about the loaded document.

Android Barcode Scanner widget

Android Barcode Scanning Widget

A new Android widget had been added that allows for scanning of barcodes using the device's camera.

Multiple barcodes can be detected in a single frame. Turning the guide on will set the widget to single detect mode, with only barcodes overlapping the guide being detected.

Callbacks are sent on barcode detection and removal.

See the dictionary for full details.

Properties

The barcode widget has the following properties:

- `device`: The camera device to use.
- `previewWidth`: The width of the camera's resolution.
- `previewHeight`: The height of the camera's resolution.
- `previewFPS`: The camera's frame rate.
- `overlayShowLabels`: If labels should be displayed on top of each detected barcode.
- `overlayShowRects`: If rectangles should be displayed around each detected barcode.
- `overlayShowGuide`: If the scanning guide should be displayed.
- `overlayLabelColor`: The color of any barcode labels .
- `overlayRectColor`: The color of any barcode rectangles.
- `acceptedFormats`: A list of barcode types to detect.
- `snapshotMode`: If and when a snapshot should be taken of the barcode.

Messages

The barcode widget sends the following messages:

- `barcodeDetected`: Sent when a new barcode is detected.
- `barcodeRemoved`: Sent when a previously detected barcode is no longer in the frame.
- `barcodeClicked`: Sent when a detected barcode has been clicked by the user.

Android Barcode Library

Android Barcode Library

A new Android library has been added that provides functionality for detecting barcodes in images.

The library defines the function `barcodeLibraryDetect` which takes an image and returns an array of detected barcodes.

See the dictionary for full details.

LiveCode builder changes

LiveCode Builder Host Library

Canvas library

New syntax has been added to enable getting image metadata and density information.

The image 'metadata' property returns data associated with the image in the form of an array.

The image 'density' property returns an images density in dots per inch (DPI). This allows widgets to scale images appropriately for display.

A new statement `clip to <path> on <canvas>` has been implemented to allow drawing on a canvas to be clipped to the path. Previously the clip region could only be set to a rectangle via the `clip to <rectangle> on <canvas>` statement.

For example to draw an image into a circle:

```
clip to circle path centered at point [my width /2,my height / 2] with radius
my width/2 on this canvas
variable tImage as Image
put image from resource file "foo.png" into tImage
draw tImage into rectangle [0, 0, the width of tImage, the height of tImage]
of this canvas
```

A new statement `begin effect only layer with <effect>` has been implemented to allow drawing shadow & glow effects without rendering the source content

For example to draw a dropshadow of a rectangle without drawing the rectangle itself: variable tEffect as Effect put outer shadow effect into tEffect

```
begin effect only layer with tEffect on this canvas
fill rectangle path of rectangle [50,50,100,100] on this canvas
end layer on this canvas
```

Engine library

- You can now use `resolve file <path expression> [relative to <script object expression>]` statement to resolve a relative file path to an absolute file path in the same way that it would be resolved in LiveCode Script.
- You can now use `the <modifierkey> is [currently] down` expression to determine if the shift, command, control, alt/option or caps lock keys are down. Use the optional

`currently` adverb to differentiate between the shift key being down at the present moment and it being down when the current event occurred.

if the shift key is down then

```
DoShiftKeyThing()
```

end if

LiveCode Builder Tools

lc-compile

Command-line interface

- The new `--forcebuiltins` command line flag can be used in combination with the `--outputauxc OUTFILE` option to generate shims for C foreign bindings. Use the output file to link with any static libraries used by modules on iOS to create a lcext binary and the shims will ensure that required objects from the static libraries are included in the resulting binary.

Specific builder bug fixes (9.5.0-dp-1)

19137 **`execute script` without an explict object now executes in the context of the host widget if called from a widget handler**

21080 **When binding Objective C methods throw an error if the parameter count is incorrect**

Dictionary additions

- **keyboardType** (*property*) has been added to the dictionary.
- **layerClipRect** (*property*) has been added to the dictionary.
- **log** (*command*) has been added to the dictionary.
- **logMessage** (*property*) has been added to the dictionary.
- **mobileGetKeyboardDisplay** (*function*) has been added to the dictionary.
- **mobileSetKeyboardDisplay** (*command*) has been added to the dictionary.
- **returnKeyType** (*property*) has been added to the dictionary.

Previous release notes

- LiveCode 9.0.4 Release Notes
- LiveCode 9.0.3 Release Notes
- LiveCode 9.0.2 Release Notes
- LiveCode 9.0.1 Release Notes
- LiveCode 9.0.0 Release Notes
- LiveCode 8.1.9 Release Notes
- LiveCode 8.1.8 Release Notes
- LiveCode 8.1.7 Release Notes
- LiveCode 8.1.6 Release Notes
- LiveCode 8.1.5 Release Notes
- LiveCode 8.1.4 Release Notes
- LiveCode 8.1.3 Release Notes
- LiveCode 8.1.2 Release Notes
- LiveCode 8.1.10 Release Notes
- LiveCode 8.1.1 Release Notes
- LiveCode 8.1.0 Release Notes
- LiveCode 8.0.2 Release Notes
- LiveCode 8.0.1 Release Notes
- LiveCode 8.0.0 Release Notes
- LiveCode 7.1.4 Release Notes
- LiveCode 7.1.3 Release Notes
- LiveCode 7.1.2 Release Notes
- LiveCode 7.1.1 Release Notes
- LiveCode 7.1.0 Release Notes
- LiveCode 7.0.6 Release Notes
- LiveCode 7.0.4 Release Notes
- LiveCode 7.0.3 Release Notes
- LiveCode 7.0.1 Release Notes
- LiveCode 7.0.0 Release Notes
- LiveCode 6.7.9 Release Notes
- LiveCode 6.7.8 Release Notes
- LiveCode 6.7.7 Release Notes
- LiveCode 6.7.6 Release Notes
- LiveCode 6.7.4 Release Notes
- LiveCode 6.7.2 Release Notes
- LiveCode 6.7.11 Release Notes
- LiveCode 6.7.10 Release Notes
- LiveCode 6.7.1 Release Notes
- LiveCode 6.7.0 Release Notes
- LiveCode 6.6.5 Release Notes
- LiveCode 6.6.4 Release Notes
- LiveCode 6.6.3 Release Notes
- LiveCode 6.6.2 Release Notes
- LiveCode 6.6.1 Release Notes
- LiveCode 6.6.0 Release Notes
- LiveCode 6.5.2 Release Notes
- LiveCode 6.5.1 Release Notes
- LiveCode 6.5.0 Release Notes
- LiveCode 6.1.3 Release Notes
- LiveCode 6.1.2 Release Notes

- LiveCode 6.1.1 Release Notes
- LiveCode 6.1.0 Release Notes
- LiveCode 6.0.2 Release Notes
- LiveCode 6.0.1 Release Notes
- LiveCode 6.0.0 Release Notes