LiveCode 8.0.0-dp-13 Release Notes

Overview

LiveCode 8.0 brings important new capabilities to all LiveCode developers:

- Use new **custom controls and libraries** in your applications, including a new **browser widget**.
- Extend LiveCode with the new **LiveCode Builder language**

- **Deploy to HTML5** and run your application in a web browser
- Many other improvements!

Simplify design with widgets

With LiveCode 8.0, your apps are set free from using the small range of user interface controls that were previously available in LiveCode. The LiveCode engine now lets you load custom controls, called **widgets**, and use them in your apps just like any other control.

LiveCode comes with a selection of widgets that simplify creating many commonly-needed sets of controls in mobile apps, and if they aren't enough, you can download and install widgets created by members of the LiveCode development community and third-party vendors.

One of the most exciting new widgets introduced in LiveCode 8.0 is the **browser widget**. It replaces the revBrowser external with a much more powerful and flexible browser control that's easier to use and more reliable.

The LiveCode IDE has also been extended and revised in order to support widgets and other extensions. Widgets are now available in the "Tools" palette, and installed extensions can be viewed in the "Extension Manager". The dictionary has been extended to include extension documentation.

Extend Livecode with LiveCode Builder

In LiveCode 8.0, the well-known LiveCode scripting language is joined by a brand new programming language called **LiveCode Builder**. LiveCode Builder looks a lot like LiveCode script, and should be easy to learn for any experienced LiveCode developer.

Using LiveCode Builder, it is now possible to extend LiveCode with new controls and libraries without any need to program in C or C++. The IDE has a new "Extension Builder" tool that helps developers test, debug and package their extensions.

For more information, please refer to the "Extending LiveCode" guide and the "LiveCode Builder" section of the dictionary.

Note: LiveCode Builder is a new and experimental language. There is **no stability guarantee** for the language or its standard libraries. Be aware that the language syntax or features may change incompatibly in future versions of LiveCode!

Deploy to the browser with HTML5

The LiveCode 8.0 engine now runs on a new platform: the web browser. The LiveCode engine now runs as a JavaScript library in an HTML page, allowing users to run your application without having to install anything.

For more information, please refer to the "HTML5 Deployment" guide.

Note: The HTML5 platform is very different to the other platforms that LiveCode supports, and many engine features are either unsupported or work differently.

More!

LiveCode 8.0 includes many other enhancements, including:

- more powerfuland complete **clipboard access**, sponsored by FMProMigrator
- **SSL support for PostgreSQL connections**, sponsored by the community Feature Exchange
- **Unicode printing** on Linux
- a new **JSON library** extension.

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.
- revBrowser for 32-bit Linux fails to run (causing the dictionary to be blank).

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 XP SP2 and above
- Windows Server 2003
- Windows Vista SP1 and above (both 32-bit and 64-bit)
- Windows 7 (both 32-bit and 64-bit)
- Windows Server 2008
- Windows 8.x (Desktop)

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

Linux

LiveCode supports Linux installations which meet the following requirements:

- Supported CPU architectures:
    - 32-bit or 64-bit Intel/AMD or compatible processor
    - 32-bit ARMv6 with hardware floating-point (e.g. RaspberryPi)
- Common requirements for GUI functionality:

    - GTK/GDK/Glib 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)

- Requirements for 32-bit Intel/AMD:
    - glibc 2.11 or later
- Requirements for 64-bit Intel/AMD:
    - glibc 2.13 or later
- Requirements for ARMv6:
    - glibc 2.7 or later

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:

- 10.6.x (Snow Leopard) on Intel
- 10.7.x (Lion) on Intel
- 10.8.x (Mountain Lion) on Intel
- 10.9.x (Mavericks) on Intel
- 10.10.x (Yosemite) on Intel
- 10.11.x (El Capitan) on Intel

Note: LiveCode runs as a 32-bit application regardless of the capabilities of the underlying processor.

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 4.6 on MacOS X 10.7
- Xcode 5.1 on MacOS X 10.8
- Xcode 6.2 on MacOS X 10.9
- Xcode 6.2, 6.4 and 7.1 on Mac OS X 10.10

- Xcode 7.1 on MacOS X 10.11

It is also possible to set other versions of Xcode, to allow testing on a wider range of iOS simulators. For instance, on Yosemite, you can add *Xcode 5.1* in the *Mobile Support* preferences, to let you test your stack on the *iOS Simulator 7.1*.

We currently support the following iOS Simulators:

- 6.1
- 7.1
- 8.2
- 8.4
- 9.1

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.

To enable the deployment to an Android device, you need to download the Android SDK and to install the following component, using the *Android SDK Manager*:

- Android SDK Platform for Android 4.0.3 (API 15)
- Android SDK Platform Tools
- Android SDK Build Tools

You also need to have Java *JDK 1.6* installed on your machine (or *openjdk-6* on Linux).

Once you have set the path of your Android SDK in the *LiveCode Preferences > Mobile Support*, you can deploy your stack to Android devices running Android >= 2.3.3.

Android Watch is not officially supported at the moment.

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 to its own, separate folder, to 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, and on Mac OS X, each version has its own app bundle.

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>`
Mac OS X	`/Applications/LiveCode <version>.app`

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>`
Mac OS X	`~/Applications/LiveCode <version>.app`

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`

Platform	Path
Mac OS X	`<home>/Library/Application Support/Logs/RunRev`

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 Mac, 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 noui [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 noui [OPTION ...]
```

On Mac OS X, you need to do:

```
<installer>/Contents/MacOS/installer install noui *options*
```

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 noui
```

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
```

Engine changes

Packaged extension name changes (8.0.0-dp-13)

Earlier versions of the widgets and libraries which are bundled with the IDE were named inconsistently.

Now all LiveCode extensions are named either **com.livecode.widget.** or **com.livecode.library.**.

Their directories in the installed LiveCode bundle have also been updated to match.

Note: This change will break some stacks that have widgets saved on them, or scripts which refer to a widget by its kind.

Improvements for saving in old stack file formats (8.0.0-dp-13)

It is now possible to specify a version of the stack file format to be used when saving directly with the `save` command, by using the `save STACK with format VERSION` form of the command.

You can request the latest supported file format using the `save STACK with newest format` form.

The `stackFileVersion` global property is now deprecated, and should not be used in new stacks.

Use more accurate measurements when laying out text. (8.0.0-dp-12)

The measurements used by LiveCode to lay out text (the text "metrics") in versions prior to 8

were optimised for low-resolution bitmap fonts. In LiveCode 8, more accurate measurements are now being used by the field and other controls.

For many existing stacks, the most visible change is that lines in the field are now closer together and better match the text display in other programs. If this is not desirable, a fixed line height can be set on the field to emulate the old behaviour.

Improve printing on Linux (8.0.0-dp-11)

The Linux printing commands now use the revpdfprinter external to generate a PDF to send through the system printing process.

Among other things, this means that there is now full support for Unicode text.

Note: To use printing on Linux in standalones you must now make sure you include revpdfprinter. Make sure the appropriate checkbox is highlighted in the standalone builder.

Make odd numbers of dashes work properly. (8.0.0-dp-11)

If the **dashes** property of a graphic was set to an odd number of items, then the pattern would not replicate correctly.

Correct lineOffset / itemOffset mode in wholeMatches mode. (8.0.0-dp-11)

The lineOffset and itemOffset functions now work correctly in wholeMatches mode in that they will not stop after the first occurrance of needle.

Consistent value union and intersect semantics (8.0.0-dp-10)

The array **union** and **intersect** commands and their recursive counterparts now follow a consistent recipe to produce their result.

`union <left> with <right>` now has the semantics of the following LiveCode function:

```
function ArrayUnion(pLeft, pRight, pRecursive)
    repeat for each key tKey in pRight
        if tKey is not among the keys of pLeft then
            put pRight[tKey] into pLeft[tKey]
        else if pRecursive then
            put ArrayUnion(pLeft[tKey], pRight[tKey], true) into pLeft[tKey]
        end if
    end repeat

    return pLeft
end ArrayUnion
```

and `intersect <left> with <right>` the following:

```
function ArrayIntersect(pLeft, pRight, pRecursive)
    repeat for each key tKey in pLeft
        if tKey is not among the keys of pRight then
            delete variable pLeft[tKey]
        else if pRecursive then
            put ArrayIntersect(pLeft[tKey], pRight[tKey], true) into
pLeft[tKey]
        end if
    end repeat

    return pLeft
end ArrayIntersect
```

Previously the semantics for **intersect** were different depending on whether the intersect was in a recursive step or not. The two affected forms are:

```
intersect <string> with <value>
```

- in LiveCode 6.7, `<string>` becomes `empty`.
- in LiveCode 8.0, `<string>` is unchanged.

```
put "a" into tLeftArray[1][1]
put "b" into tRightArray[1]
intersect tLeftArray with tRightArray recursively
```

- in LiveCode 6.7 `tLeftArray` is unchanged.
- in LiveCode 8.0, `tLeftArray[1]` becomes `empty`

Native string operations are much faster compared to LiveCode 7 (8.0.0-dp-10)

The performance of native string operations has been vastly improved, with many achieving similar speeds to those in LiveCode 6.7.

New "documentFilename" stack property (8.0.0-dp-9)

A new property has been added to specify the file path to the file that a stack represents.

On Mac OS X, setting the **documentFilename** property will set the represented filename of the window. The window will show an icon for the file next to the window title.

On other platforms there is no visual representation of the association between the stack and the

document file, but the property may still be used to manage the association.

Update the Mac OS X printer code for 64-bit compatibility (8.0.0-dp-9)

Due to changes in Apple's APIs for printing, there are some (very small) differences in printing behaviour. In 64-bit engines, no dialogue will be displayed while spooling to the printer as Apple no longer provides it.

Additionally, the API previously used to implement the **printerSettings** property was deprecated by Apple. The engine has been updated to use the new API, and compatibility between the binary strings returned by the two APIs cannot be guaranteed.

Make "the effective rect of stack" more accurate on Linux (8.0.0-dp-9)

Because the engine began using GDK on Linux in LiveCode 7.0, there is now a better method for computing the **effective rect** of a window. The engine has been updated to use this method, rather than the heuristic which was there before.

DataGrid added to the Standalone Settings script libraries list (8.0.0-dp-8)

Formerly, the DataGrid library was only included in a standalone application if the stack saved as a standalone was using a DataGrid. When the stack saved did not use DataGrid, but loaded a stack which used it, the DataGrid library was not saved with the standalone, and the DataGrid would not work in the loaded stack.

To correct this, "DataGrid" has been added to the list of "Script Libraries" in the "Standalone Settings" dialog. You can now force the inclusion of the DataGrid library, to ensure that any stack loaded by the standalone can use DataGrids.

Position resize handles outside object rectangle (8.0.0-dp-8)

The resize handles of a selected object are now outside the rect of the object. There is no longer a minimum width and height of an object when resizing via handles.

SSL Support for PostgreSQL Connections (8.0.0-dp-8)

The PostgreSQL database driver has been updated to support secure connections. The desired SSL connection options can be specified as key value pairs in the additional parameters of **revOpenDatabase**.

The syntax for connecting to PostgreSQL databases is now as follows:

```
revOpenDatabase("postgresql", host[:port], databasename,
                [username], [password], [ssloption=*ssloptionvalue],
                ...)
```

For full information on the new parameters see the dictionary entry for **revOpenDatabase**.

This feature was sponsored by the community Feature Exchange.

Improved clipboard and drag-and-drop support (8.0.0-dp-8)

It is now possible to put multiple types of data on the clipboard at the same time. For example, you can put an image on the clipboard along with text describing the image and some private data associated with the image.

For more information, see the **fullClipboardData** and **fullDragData** entries in the dictionary.

For apps requiring more sophisticated clipboard functionality, new **rawClipboardData** and **rawDragData** properties have also been added. These provide low-level access to the system clipboard.

This feature was sponsored by FMProMigrator.

Changed behaviour of submenu menu items (8.0.0-dp-8)

Previously, clicks to menu items that open submenus would dismiss the menu and send a **mouseRelease** message. Unfortunately, this did not match the conventions of modern user interfaces where these clicks are ignored. This can be seen on OSX where the system's menus are used instead of being emulated by LiveCode.

Now, clicking on the item that anchors a submenu will send the **mouseRelease** message as before but, if that message is not handled or is passed, the click will be ignored and the menu will remain on-screen.

To restore the old behaviour, handle the **mouseRelease** message without passing. To add the new behaviour to existing code that handles the **mouseRelease** message, add a **pass** command to the end of the handler.

New "scriptOnly" stack property (8.0.0-dp-5)

A new boolean stack property **scriptOnly** has been added. If its value is "true", the stack will be saved as script only. A script only stack does not retain any objects or custom properties when it is saved.

New "popup widget" command (8.0.0-dp-5)

A new **popup widget** command has been added which opens a widget within a popup window. The syntax is:

```
popup widget <kind> [ at <location> ] [ with properties <propertyArray> ]
```

For example, this command can be used to show a color picker widget as a popup:

```
local tProps
-- Set the size of the popup
put "0,0,120,50" into tProps["rect"]
-- Set the initial color value
put "1,1,0.5" into tProps["initialColor"]

-- Show the widget in a popup window
popup widget "com.example.mycolorpicker" at the mouseloc with properties
tProps
```

New HTML5 deployment platform for LiveCode apps (8.0.0-dp-4 - experimental)

The LiveCode engine will now run in web browsers that support HTML5. This means that you can now deploy simple LiveCode apps to users without any installation required.

To deploy a stack as an HTML5 application, enable the "Build for HTML5" checkbox on the "HTML5" page of the standalone settings window, and then generate the standalone in the normal way.

For more information on HTML5 deployment, including options for embedding LiveCode standalones in web pages, please see the "HTML5 Deployment" guide in the IDE dictionary.

Important: This feature is currently experimental. This means 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.

New "metadata" image property (8.0.0-dp-3)

A new read-only **metadata** property of image controls has been added. It provides access to the metadata in the image file. The returned array is in the same format as that used for the export command.

If no metadata is found then the property returns **empty** rather than an array with empty elements. Currently the only metadata key that is implemented is `density`, which can be used to determine pixel density in pixels per inch. Metadata is currently only parsed from JPEG and PNG file formats.

Usage example:

```
put the metadata of image 1 into metadataArray
set the width of image 1 to the width of image 1 div
(medatadaArray["density"] / 72)
set the height of image 1 to the height of image 1 div
(medatadaArray["density"] / 72)
```

Extensions can be loaded from raw data (8.0.0-dp-3)

LiveCode Builder extensions can now be loaded from data values by using the new `load extension from data <data>` syntax.

LiveCode Builder (8.0.0-dp-1)

LiveCode Builder Language

LiveCode Builder (LCB) is a variant of the current LiveCode scripting language (LiveCode Script) which has been designed for "systems" building. It is statically compiled with optional static typing and direct foreign code interconnect (allowing easy access to APIs written in other languages such as C). The compiled bytecode can then be packaged together with any required resources (icons, documentation, images, etc) into a `.lce` extension package.

Unlike most languages, LiveCode Builder (LCB) has been designed around the idea of extensible syntax. The core language is very small — comprising declarations and control structures — with the majority of the language syntax and functionality being defined in modules written in LCB itself.

Note: It is an eventual aim that control structures will also be extensible. However, this has not yet been implemented.

The syntax will be familiar to anyone who has coded with LiveCode Script. However, LCB is a great deal more strict. The increased strictness is because LCB is intended to eventually be compilable to machine code, with the performance and efficiency you'd expect from any 'traditional' programming language like C or C++. Over time we hope to move the majority of implementation of the whole LiveCode system over to being written in LCB.

Note: One of the principal differences is that type conversion is strict. There is no automatic conversion between different types, such as between numbers and strings. conversion must be explicitly specified using syntax (currently this is done using syntax like `<expression> parsed as number` and `<expression> formatted as string`.

Extensions

There are two types of extensions which can be written in LiveCode Builder: widgets and libraries. All installed extensions appear in the new "Extension Manager" stack, which can be opened from the "Tools" menu.

An LCB **library** is a new way of adding functions to the LiveCode message path. Public handlers in loaded LCB libraries are available to call from LiveCode Script.

A **widget** is a new type of custom control which, once compiled and packaged, can be loaded into the IDE. Using the widget is no different from any of the classic LiveCode controls you've been used to. Simply drag it onto a stack and start interacting with it as you would any another control.

You can reference the widget in script both as a control:

```
set the name of the last control to "clock"
```

and more specifically as a widget:

```
set the tooltip of widget 1 to "This is my nice new clock widget"
```

Getting Started

To get started with LiveCode Builder, click on the "Dictionary" icon in the IDE toolbar, select the "Guide" tab and then "Extending LiveCode" from the drop-down menu. This will show you the user-guide on getting started with writing widgets and libraries in LCB.

Alternatively, you can start by looking at some of the extensions shipped with LiveCode 8.0. The source and other resources for these are located in the `extensions` sub-folder of your LiveCode installation directory.

LCB source files are named `*.lcb` and can be edited with a text editor of your choice. For example, there is a language-livecode package available for the Atom editor which provides syntax highlighting and autocompletion when editing LCB source code.

Specific bug fixes

- **Bugs fixed in 8.0.0-dp-13: 15970, 16267, 16351, 16415, 16533, 16539, 16615, 16642, 16644, 16666, 16667, 16686, 16691, 16696, 16699, 16703.**
- Bugs fixed in 8.0.0-dp-12: 15231, 15811, 16156, 16186, 16448, 16493, 16524, 16543, 16577, 16584, 16599.
- Bugs fixed in 8.0.0-dp-11: 14388, 15862, 16076, 16294, 16410, 16411, 16452, 16460, 16467, 16474, 16476, 16497, 16500, 16501, 16504, 16512, 16515, 16522, 16528, 16529, 16530.
- Bugs fixed in 8.0.0-dp-10: 15819, 15855, 16008, 16219, 16223, 16278, 16288, 16308, 16360, 16372, 16395, 16405, 16407, 16417, 16434, 16440, 16445, 16450, 16451, 16458.
- Bugs fixed in 8.0.0-dp-9: 11923, 15097, 15617, 15866, 15984, 16162, 16177, 16203, 16210, 16218, 16221, 16228, 16239, 16250, 16279, 16280, 16283, 16292, 16306, 16313, 16345, 16363, 16365, 16367, 16368, 16391.
- Bugs fixed in 8.0.0-dp-8: 11854, 15689, 16161, 16193, 16202, 16220, 16238, 16256, 16259, 16272, 16276, 16323, 16324, 7414.
- Bugs fixed in 8.0.0-dp-7: 16073, 16089, 16092, 16094, 16096, 16118, 16124.
- Bugs fixed in 8.0.0-dp-6: 15197, 15981, 16032.
- Bugs fixed in 8.0.0-dp-5: 13694, 14837, 14970, 15129, 15345, 15798, 15805, 15808, 15836, 15845, 15846, 15848, 15868, 15870, 15897, 15908.
- Bugs fixed in 8.0.0-dp-4: 15752.
- Bugs fixed in 8.0.0-dp-3: 14961, 14978, 15056, 15156, 15214, 15286, 15358, 15378,

15405, 15509, 15605, 15618, 15620, 15630.
- Bugs fixed in 8.0.0-dp-1: 14538, 14602, 14851.

IDE changes

Dictionary data build location (8.0.0-dp-13)

Previously the IDE would try to build dictionary data internally, which was a problem for the installed application on Windows and Linux (as the location was not necessarily writable).

Now the data is built and modified in an appropriate 'Application Support' folder, or equivalent. This also allows direct navigation to specific dictionary entries via the 'launch documentation' button / 'Find in Docs' contextual menu item in the script editor, and the 'Show Documentation' contextual object menu item, in an external browser window on platforms where the browser widget is not available.

Documentation API (8.0.0-dp-12)

There is now a documentation API used in the IDE, and available for use by plugin authors and toolmakers.

It consists of the following functions:

- `ideDocsFetchLCSData pEntryName` : Fetch the data for entries with name `pEntryName` in the LiveCode Script dictionary.
- `ideDocsFetchLCSDataOfType pEntryName, pType`: Fetch the data for the entry with name `pEntryName` and type `pType` in the LiveCode Script dictionary.
- `ideDocsFetchLCSElementOfType pEntryName, pType, pElement`: Fetch the data for the docs element `pElement` with entry name `pEntryName` and type `pType` in the LiveCode Script dictionary.
- `ideDocsFetchLCSEntries`: Fetch all the data for entries in the LiveCode Script dictionary.
- `ideDocsFetchLCBData pEntryName` : Fetch the data for entries with name `pEntryName` in the LiveCode Builder dictionary.
- `ideDocsFetchLCBDataOfType pEntryName, pType`: Fetch the data for the entry with name `pEntryName` and type `pType` in the LiveCode Builder dictionary.
- `ideDocsFetchLCBElementOfType pEntryName, pType, pElement`: Fetch the data for the docs element `pElement` with entry name `pEntryName` and type `pType` in the LiveCode Builder dictionary.
- `ideDocsFetchLCBEntries`: Fetch all the data for entries in the LiveCode Builder dictionary.
- `ideDocsFetchExtensionData pID, pEntryName` : Fetch the data for entries with name `pEntryName` in the API for the extension with id `pID`.

- `ideDocsFetchExtensionDataOfType pID, pEntryName, pType`: Fetch the data for the entry with name `pEntryName` and type `pType` in the API for the extension with id `pID`.
- `ideDocsFetchExtensionElementOfType pID, pEntryName, pType, pElement`: Fetch the data for the docs element `pElement` with entry name `pEntryName` and type `pType` in the API for the extension with id `pID`.
- `ideDocsFetchExtensionEntries pID`: Fetch all the data for entries in the API for the extension with id `pID`.

`ideDocsFetch...Data` returns a numerically keyed array, each element of which is the array of data about a docs entry with the given name in the specified library.

`ideDocsFetch...DataOfType` returns the array of data about the unique docs entry with given name and type in the specified library.

`ideDocsFetch...ElementOfType` returns a specific element of the array of data about the unique docs entry with given name and type in the specified library.

`ideDocsFetch...Entries`: returns all the data for entries in the specified library.

Extension folder locations within the IDE (8.0.0-dp-9)

If you write plugins, or have code that relies on the location of extensions then please ensure you use the following access functions to locate them:

- `revEnvironmentExtensionsPath`: returns the path to the folder where the IDE looks for packaged extensions to load (by default, this is `My Livecode/Extensions`)
- `revEnvironmentUserExtensionsPath`: returns the path to the folder where the IDE looks for user-installed extensions to load.

When building from source, `revEnvironmentExtensionsPath` returns the location in the binaries folder to which the packaged extensions have been extracted.

Property Inspector editorList editor (8.0.0-dp-8)

A new editor has been added to the list of property inspector editors which lays out a series of copies of a specified subeditor to control a string property.

For example, the segmented control widget has a property `segmentIcons`, which is a comma delimited list of names of icons.

In the widget metadata, we have

```
metadata segmentIcons.editor    is "com.livecode.pi.editorlist"
metadata segmentIcons.subeditor is "com.livecode.pi.svgicon"
metadata segmentIcons.delimiter is ","
```

This tells the property inspector that each icon in the sequence should be controlled by the

svgicon editor, and the property as a whole is delimited by ",".

Property Inspector tabs display icons or labels (8.0.0-dp-8)

The tabs in the property inspector can now display icons or labels according to user preference.

Property Inspector tabs have large icons by default (8.0.0-dp-8)

The default palette header size is now "large". Moreover, a new option "largest" has been added to the palette header size options.

Widget default scripts (8.0.0-dp-6)

When opening the code editor for a widget, it will now be prepopulated by the script specified in the widget's `defaultScript` metadata.

Per-version IDE stack filenames (8.0.0-dp-6)

When a binary stackfile is rewritten in the IDE for a new version, it now has a (major) version in the filename to prevent unwanted IDE merging between versions. This can also be used to ensure incompatible stacks are not loaded if present - the IDE will only load stacks with a version less than or equal to its version.

For example, from 8.0 onwards, the "revIdeLibrary" stack has filename `/Toolset/libraries/revidelibrary.8.livecodescript`.

Variable viewer resizable columns (8.0.0-dp-6)

The variable viewer's tree view widget display now has its `showSeparator` property set to true, which allows the user to resize the columns of its display.

General IDE improvements (8.0.0-dp-3)

Menu bar

The menubar has been made a script-only stack to facilitate bugfixes and community contributions. Users should not notice much difference in terms of its appearance. Some of the menu items have been changed, however.

The "New Mainstack" item now has a submenu with a range of size choices, as well as the option to create a script-only stack. Selecting script-only stack will prompt a choice of name, and subsequently open the stack in the script editor.

We have centralised the building and handling of contextual menus in the menubar script, thereby making per-object contextual menus display and behave consistently throughout the IDE.

The "Object → New Control" submenu is now generated based on the property information present for each object type, and the newly added "Object → New Widget" submenu is generated based on the currently loaded widget extensions.

Widget metadata and the IDE

Widget metadata now controls a number of additional features with respect to how the widget interacts with the IDE.

Firstly, the `preferredSize` attribute controls the initial size of the widget when dragged out from the tools palette. For example, the navbar widget now has

```
metadata preferredSize is "320,49"
```

so that when dragged out, it is created at the correct size for an original iPhone screen.

Secondly, the `userVisible` attribute controls whether the widget appears at all in the tools palette of the IDE.

A number of widgets have been declared user invisible for this release, either because they are not meant to be draggable objects at all (eg the icon picker widget, which is designed to be popped up) or are not quite refined to the point where they are suitable for user stacks, but are included because they are being used in the IDE (for example the tree view widget).

Finally, if present, the `SvgIcon` attribute will be used to display an icon for the widget in the tools palette, taking precedence over the included icon resources. All of the widgets included by default in the tools palette now use svg icon paths.

Standalone Settings

A field has been added to the "Copy Files" tab of the standalone settings which is populated with the list of currently installed extensions. All selected extensions from this list are included in standalones and loaded when the standalone is launched. Dependencies between extensions declared in the LiveCode Builder source code using `use` are automatically calculated and included along with the top-level widget.

Updated Property Inspector with support for widgets (8.0.0-dp-3)

The property inspector has been rewritten to allow properties of widgets to be inspected and edited. It has been implemented with flexibility and extensibility in mind, since it must be able to control the values of widget properties in any way required by the widget developer. Each property now has a number of attributes which affect how it appears in the inspector.

Property Attributes

The following property attributes are supported:

- `default`: The default value of the property. If there is no default value (for example the **loc** property does not have one), the string "no_default" can be used. The property inspector

pops up a contextual menu when editors are right-clicked allowing the user to set the property back to a default value.

- `editor`: The editor that will be used to display the value of the property and allow it to be edited. See the dedicated section below for details on property inspector editors.
- `group`: Properties are grouped by themselves in the inspector by default. If a particular group name is specified for a set of properties, their editors are placed next to each other in the inspector.
- `label`: The label to use for this property.
- `options`: For properties whose value is a choice from a set of options, that set should be specified as a comma delimited list for the options attribute. Default editors are provided for "enum" type properties (choice of exactly one from a set) and "set" type properties (choice of zero or more from a set).
- `section`: The section attribute controls which tab of the property inspector contains the property in question. The following sections are currently supported:
    - "Basic"
    - "Data Grid"
    - "Custom"
    - "Table"
    - "Colors"
    - "Effects"
    - "Icons"
    - "Position"
    - "Text"
- `user_visible`: Properties are visible in the property inspector by default. Set the `user_visible` attribute to false to hide a given property from the user.
- `read_only`: Read only properties will be displayed in the property inspector but the corresponding editor will have its **editorEnabled** property set to false. See the "Editors" section below for more details on enabled/disabled editors.

The `default` and `options` attributes can have their values generated dynamically at runtime using LiveCode Script, by using the `execute` keyword in the attribute value. Whatever remains in the **it** variable after executing the specified script is used as the list of options.

For example, the options for the **textFont** property are generated by setting the `options` property to `execute: get the fontNames; sort it`.

Widget Properties

Widget metadata is used to control the display of widget properties in the inspector. Items of metadata which determine property attributes are of the form:

```
metadata <property>.<attribute> is "<value>"
```

These are stored as property data for the widget at load time. The `<attribute>` can be any of those specified in the "Property Attributes" section above. If the attributes are not specified, their values are as follows:

- `default`: "no_default"
- `editor`: "com.livecode.pi.number" for `Integer`/`Real` properties, "com.livecode.pi." for properties of type "".
- `group`: the name of the property
- `label`: the name of the property
- `options`: empty
- `section`: "Basic"
- `user_visible`: true
- `read_only`: true if there is no specified "set" handler or variable for the property, false otherwise.

Script Object Properties

Script-level properties of objects (including widgets) are specified in files in the `Toolset/resources/supporting_files/property_definitions` folder. The `propertyInfo.txt` file specifies the default values for all the property attributes. Each object type then has a specification of which properties should be displayed in the inspector when it is the selected object, and any `options`/`default`/`group` values which override the defaults.

Editors

Currently an editor must be a stack consisting of a group named "template" and a button named "behavior". The property inspector looks up the specified editor for a given property, clones the template group, and sets its behavior to the long id of the button.

The behavior script must at a minimum implement the following three handlers:

```
on editorInitialize
on editorUpdate
on editorResize
```

There are a number of properties available to any editor:

- **editorMinWidth**
- **editorMaxWidth**
- **editorEnabled**
- **editorEffective**
- **editorValue**

These should be set or got appropriately. For example, if an editor consists of a text field, the `editorUpdate` handler should update the value of the field with `the editorValue of me`. Similarly, if the content of the field changes, the field should call a function in the behavior which sets `the editorValue of me` to the content of the field.

The **editorEnabled** and **editorEffective** properties are set by the generic behavior depending on the property info and the values of the properties. The **editorEffective** is true if the value of the property in question is empty but there is an effective value in play. The editor should alter the display of its value accordingly.

Editors can specify their min and max width if required.

The following editors are built-in, and available to use for widget properties with common types:

- **com.livecode.pi.array**: a Tree View widget
- **com.livecode.pi.boolean**: a check box
- **com.livecode.pi.color**: a color swatch and dialog
- **com.livecode.pi.colorwithalpha**: a color swatch and dialog, and alpha value slider
- **com.livecode.pi.enum**: an option menu
- **com.livecode.pi.file**: a file selector
- **com.livecode.pi.number**: a single-line field, with a slider if the property has an associated min/max and an increment/decrement twiddle if it has a step value
- **com.livecode.pi.pattern**: a pattern selector
- **com.livecode.pi.set**: a field with multi-select list behavior
- **com.livecode.pi.string**: a single-line field
- **com.livecode.pi.text**: a multi-line field
- **com.livecode.pi.point**: an editor for a point (e.g. the **hotspot** and **loc** properties)
- **com.livecode.pi.svgicon**: an editor for an SVG icon; the property using the editor gets set to the name of the icon picked
- **com.livecode.pi.timezone**: contains a drop-down list of time zones

There are also some bespoke editors for particular object properties:

- **com.livecode.pi.customprops**
- **com.livecode.pi.datagrid**
- **com.livecode.pi.textalign**
- **com.livecode.pi.textstyle**
- **com.livecode.pi.graphiceffect**
- **com.livecode.pi.gradientramp**
- **com.livecode.pi.script**

It is our intention that ultimately a widget alone will be able to function as a property editor, however currently this feature is not available.

Specific bug fixes

- **Bugs fixed in 8.0.0-dp-13: 16605, 16615, 16640, 16641, 16651, 16672, 16674, 16675.**
- Bugs fixed in 8.0.0-dp-12: 14510, 14511, 14745, 15915, 16106, 16549, 16575, 16585.
- Bugs fixed in 8.0.0-dp-11: 14741, 15540, 15995, 16063, 16165, 16304, 16311, 16312, 16369, 16422, 16464, 16483, 16503, 16510.
- Bugs fixed in 8.0.0-dp-10: 14486, 15784, 15974, 16423, 16429, 16432, 16433.
- Bugs fixed in 8.0.0-dp-9: 15554, 16060, 16100, 16175, 16190, 16281, 16327, 16336, 16361, 16371, 16398.
- Bugs fixed in 8.0.0-dp-8: 15603, 16052, 16088, 16136, 16140, 16144, 16149, 16158, 16169, 16176, 16213, 16242, 16248, 16266, 16271, 9159.
- Bugs fixed in 8.0.0-dp-7-live-3: 16127, 16132.
- Bugs fixed in 8.0.0-dp-7-live-2: 16016, 16065, 16071, 16099, 16101, 16105, 16129, 16138.
- Bugs fixed in 8.0.0-dp-6: 11755, 12880, 13159, 13191, 13215, 13343, 13362, 13398, 13417, 13447, 13475, 13646, 14561, 14738, 14822, 14831, 14852, 14873, 14890, 14971, 15001, 15180, 15216, 15220, 15227, 15235, 15247, 15285, 15293, 15323, 15427, 15430,

15464, 15542, 15600, 15601, 15739, 15744, 15745, 15755, 15757, 15759, 15769, 15770, 15776, 15786, 15787, 15788, 15791, 15792, 15793, 15794, 15795, 15796, 15801, 15803, 15824, 15825, 15827, 15832, 15838, 15841, 15842, 15843, 15847, 15854, 15859, 15882, 15893, 15926, 15947, 15961, 15962, 15972, 15975, 15985, 15998, 16000, 16010, 16011, 16013, 16023, 16025, 16028.

- Bugs fixed in 8.0.0-dp-1: 14627.

LiveCode Builder changes

LiveCode Builder Language

Expressions

- It is now possible to use array constants and array expressions in LiveCode Builder programs. For example:

```
variable tArray as Array
put {"key": true} into tArray
```

  Keys in array literals must be strings.

Syntax

- The precedence of LCB syntax operators has been reworked to use a defined set of 23 named precedence classes. This is to ensure predictable and consistent interactions when multiple LCB operators are used in a single expression.

  There only change that is likely to significantly affect pre-existing LCB code is the reduction in precedence of the logical `not` operator to below that of comparison operators. This may allow removal of parentheses in the condition clauses of `if` statements.

- `use` declarations may now occur anywhere in a module's top-level context.
- Syntax keywords are no longer permitted to match `[A-Z0-9_.]`.
- `metadata` definitions may now occur anywhere a module's top-level context.

Type definitions

- It is now possible define foreign handler types:

```
foreign handler type MyCallback(in pContext as optional pointer, in
pValue as any) as CBool
```

  When used in the context of a foreign handler definition, a foreign handler type will cause automatic bridging of the LCB handler to a C function pointer which can be called directly by the native code.

The function pointers created in this fashion have lifetime equivalent to that of the calling context. In particular, for widgets they will last as long as the widget does, for all other module types they will last as long as the module is loaded.

Core types

- The following deprecated core type names have been removed:
    - `boolean` (replaced by `Boolean`)
    - `integer` (replaced by `Integer`)
    - `real` (replaced by `Real`)
    - `number` (replaced by `Number`)
    - `string` (replaced by `String`)
    - `data` (replaced by `Data`)
    - `array` (replaced by `Array`)
    - `list` (replaced by `List`)
- The use of the keyword `undefined` is now deprecated. Use `nothing` instead.
    - Use `returns nothing` when defining a handler which returns no value.
    - Use `nothing` to indicate no value when manipulating optionally type variables
- The `is defined`, `is undefined`, `is not defined`, and `is not undefined` syntax is now deprecated. Use `is` and `is not` with `nothing` instead.
    - Use `<expr> is nothing` and `<expr> is not nothing` to test whether an expression has a value or not
    - The expression `<left> is <right>` will now evaluate to `true` if `<left>` and `<right>` are both nothing
    - The expression `<left> is not <right>` will now evaluate to `true` if one of `<left>` or `<right>` are nothing (but not both).

Identifiers

- All identifiers are now case-insensitive - i.e. a handler `Main` can be called as `mAin`, `MAIN` and `main`.
- Identifiers are now expected to match `[A-Z0-9_.]`.

Handler definitions

- The syntax for declaring the type of the return value from a handler (or handler type) is now `[ 'returns' 'nothing' | 'returns' <Type> ]`.
- Foreign handler definitions now require explicit typing.
    - A foreign handler definition must declare an explicit return type.
    - Each parameter in a foreign handler definition must declare an explicit type.

LiveCode Builder Tools

lc-compile

Errors

- List expressions with more than 254 elements will now generate an error.
- Array expressions with more than 127 key-value pairs will now generate an error.
- Array constants with non-string keys will now generate an error.
- Integer literals that are too big to fit into an unsigned 32-bit integer representation will now generate a compilation error.

Command-line interface

- The new `-Werror` command line flag converts all compilation warnings into errors.
- A new `--verbose` command line flag has been added. If it is specified, **lc-compile** will output additional debugging information.

Manifest

- A module now outputs its type to the manifest XML.

Warnings

- A new warning has been added for identifiers that may conflict with syntax keywords.
- `metadata` definitions that occur before module imports no longer trigger a warning.

LiveCode Builder Host Library

Local date

The date items are now adjusted so that the year and month are absolute rather than offsets from 1900 and January respectively.

Widget library

- Improved tools for detecting repeated clicks on widgets
    - The `OnClick` event is sent every time a mouseDown/mouseUp sequence is detected by the engine on a widget.
    - `the click count` expression evaluates to the number of successive clicks which happened close together and within a certain time of each other.
- Widgets can pop up a menu constructed from a provided menu text using the `popup menu <MenuText> at <Point>` statement.

- Widgets now have access to their effective font.
    - The **textFont**, **textSize** and **textStyle** properties have been reserved for use by the widget host.
    - A new `my font` expression has been added which evaluates to a `Font` matching the current effective values of the text properties that have been set on the widget.
- Widgets now print along with other controls.
    - Widgets will be rasterized at screen resolution and then printed as an image.
    - Higher-fidelity printing of widgets will be implemented at a later date.
- Added the ability to use widgets within popup dialog windows.
    - `popup widget <Kind> at <Position> [ with properties <Properties> ]`: Launches the named widget as a popup. The popup can return a value in the result.
    - `currently popped up`: Evaluates to true if this widget is part of a popup.
    - `close popup [ returning <Result> ]`: Closes this widget's popup, setting the result of the calling popup statement to `<Result>`.
- Widgets now have access to their enable state.
    - The `my enabled` property returns `true` if the widget is currently enabled
    - The `my disabled` property returns `true` if the widget is currently disabled
    - If script changes the **enabled** (or **disabled**) property of the widget then an `OnParentPropChanged` message will be sent.

Canvas library

- "Outer shadow" effects now have a **knockout** (*Boolean*) property. It controls whether or not the alpha channel of the source image is applied to the blurred shadow created by the effect, and defaults to `true`.
- "Inner glow" effects now have a **source** (*String*) property, which determines the location from which the glow originates. It can be either "center" or or "edge", and defaults to "edge".
- The **spread** property of a canvas effect is now clamped to the range **0–1**.
- The unused **opacity** canvas effect property has been removed. The opacity can be set by specifying a color with alpha value less then 1.
- You can now create a new canvas `Effect` object without setting up an array of properties. For example:

  ```
  variable tEffect as Effect
  put outer shadow effect into tEffect
  ```

  Default values will be assumed for unspecified properties:

    - **size**: 5
    - **spread**: 0

- **distance**: 5
- **angle**: 60

Composed widgets

The ability to compose widget objects has been added. Widgets can either be 'host' widgets, created when a widget is directly embedded in a stack, or 'child' widgets which are created when a widget is used as a child widget within another widget.

- A `Widget` type has been added, so that variables can contain references to child widget objects. Variables that hold widget references can be defined in the usual way, e.g. `variable tWidget as Widget`.
- New widget syntax has been added to create, place, unplace and manipulate child widgets.
  - `a new widget <kind>`: Create a widget object of the specified kind.
  - `place <widget> [at (bottom|top) | (below|above) <other widget>]`: Add a child widget to this widget on the specified layer.
  - `unplace <widget>`: Remove a child widget from this widget.
  - `the target`: Return the child widget that started the current execution.
  - `my children`: Return a list of the currently placed child widgets of this widget.
  - `property <property> of <widget>`: Get or set a property implemented by a child widget.
  - `the rectangle of <widget>`: Get or set the **rectangle** property of a child widget.
  - `the width of <widget>`: Get or set the **width** property of a child widget.
  - `the height of <widget>`: Get or set the **height** property of a child widget.
  - `the location of <widget>`: Get or set the **location** property of a child widget.
  - `the enabled of <widget>`: Get or set the **enabled** property of a child widget.
  - `the disabled of <widget>`: Get or set the **disabled** property of a child widget.
  - `annotation <name> of <widget>`: Tag child widgets with named values.
- Events triggered on child widgets (such as `OnMouseUp`) are automatically passed up to the parent, as long as the child's event handler returns nothing. If any event handler returns something, the event is considered handled and is not passed to the parent.
- Messages posted by the child widget can be handled by the parent in an `On<message name>` handler.

  For example, if the child executes the statement: `post "dataChanged" with [mDataArray]`, this can be handled in the parent by adding a handler `public handler OnDataChanged(in pArray as Array)`. Posted messages can only be handled by a direct parent, and a widget's script object will only receive messages posted by host widget, i.e. the topmost parent.
- The Simple Composed Widget provides an example of how the host/child relationship can be used.

Native code in extensions

LiveCode Builder extensions can now contain native code libraries which LCB will use to resolve foreign handler references.

The foreign handler binding string should be of the form `libname>function` to use this feature. In this case, the engine will look for a library `libname` on a per-platform basis when the foreign handler needs to be resolved.

Native code libraries should be present inside the `resources` folder inside the extension archive. The engine derives the appropriate path from the requested library name and current platform. The structure is as follows:

```
<extension>/
  resources/
    code/
      mac/
        <library>.dylib
      linux-x86/
        <library>.so
      linux-x86_64/
        <library>.so
      win-x86/
        <library>.dll
```

Note: At present, only the desktop platforms are supported.

Note: The above structure is likely to change in a future release. In particular the `code` folder will sit at the same level as resources rather than within it.

LiveCode Builder Standard Library

Mathematical functions

- Several mathematical functions now throw "domain errors" when applied to values that the function is not defined for, including `log10()`, `ln()`, `asin()` and `acos()`, and `x ^ y`.

Date and time

- `the local date` now returns a list with an additional 7th element. This is the local time zone's offset from UTC, in seconds.
- The new `the universal date` expression provides the date in the UTC+00:00 timezone. It returns a list in the same format as `the local date`.

Foreign function interface

- The following deprecated foreign type names have been removed from the `com.livecode.foreign` module:

    - `pointer` (replaced by `Pointer`)
    - `bool` (replaced by `CBool`)
    - `uint` (replaced by `UInt32` or `CUInt`)
    - `int` (replaced by `Int32` or `CInt`)
    - `float` (replaced by `Float32` or `CFloat`)
    - `double` (replaced by `Float64` or `CDouble`)
    - `NativeCString` (replaced by `ZStringNative`)

- An `IntSize` foreign type has been added, which should be used to represent a signed memory extent. It maps to the `ssize_t` C library type.

Sorting

- Lists can now be sorted using an arbitrary comparison handler.
    - A new `SortCompare` handler type has been added to the sort module. A handler that conforms with `SortCompare` might be declared like:

```
MyComparisonHandler(in pLeft as any, in pRight as any) returns Integer
```

    - To sort using a `SortCompare` handler, use the new `sort <List> using handler <SortCompare>` statement.

Sequence operations

- New syntax has been added for searching partial contents of sequence types (`List`, `String` and `Data`) based on the `offset` operation.
    - `the offset of <Needle> before <Position> in <Haystack>`
    - `the offset of <Needle> after <Position> in <Haystack>`
    - Equivalent syntax has been added for the `index` operation.

Specific bug fixes

- **Bugs fixed in 8.0.0-dp-13: 16681.**
- Bugs fixed in 8.0.0-dp-12: 14963, 15429, 15440.
- Bugs fixed in 8.0.0-dp-11: 14546, 14930, 16400, 16465.
- Bugs fixed in 8.0.0-dp-9: 14929, 15435.
- Bugs fixed in 8.0.0-dp-7: 15502, 16053.
- Bugs fixed in 8.0.0-dp-6: 15916, 15966, 16006.
- Bugs fixed in 8.0.0-dp-5: 14809, 14809, 15833, 15941.
- Bugs fixed in 8.0.0-dp-4: 15773.
- Bugs fixed in 8.0.0-dp-3: 14500, 14541, 14678, 14679, 14681, 14805, 14806, 14846, 14889, 14893, 14898, 14906, 14926, 14933, 14938, 14939, 14950, 14956, 14964, 14991,

14993, 14996, 14997, 15005, 15012, 15029, 15035, 15060, 15115, 15128, 15138, 15152, 15276, 15284, 15331, 15410, 15426, 15489, 15529, 15628, 15681, 15768.

- Bugs fixed in 8.0.0-dp-1: 14599, 14871.

LiveCode extension changes

Navigation Bar widget

Widget Theme

The theme of the navigation bar widget can now be set to iOS, Android(Dark) or Android(Light).

The following properties has been added:

- **widgetTheme**: either "iOS", "Android(Dark)" or "Android(Light)"
- **backgroundOpacity**: either "Opaque", "Translucent" or "Transparent"

The **opaqueBackground** property has been removed. Use the **backgroundOpacity** property instead.

The default theme is "iOS" and the background is opaque by default.

Various bug fixes

- Selecting "names" as the **itemStyle** does not work
- Changing the **navIcons** via script / property inspector does not work
- The **navSelectedIcons** property is missing.
- **editMode** should default to "false"
- add **navigate** message to widget docs

Header widget

Widget Theme

The theme of the header bar widget can now be set to iOS or Android.

The following property has been added:

- **widgetTheme**: either "iOS" or "Android"

The default theme is "iOS".

The following properties have also been added, which can only be set if the theme of the widget is "Android":

- **colorScheme**: this can be selected from the list of Android color schemes
- **leftAction**: the icon displayed on the left of the header
- **titleVisibility**: whether the title in the header is displayed or not
- **distinctTitle**: whether the title in the header is distinct or not

The following properties of the widget can only be set if the theme of the widget is "iOS":

- **headerSubtitle**
- **leftLabel**
- **actionStyle**
- **showBackIcon**
- **showSearchIcon**
- **actionColor**
- **showDivide**

The following properties of the widget can be set regardless of the theme:

- **widgetTheme**
- **headerTitle**
- **headerActions**
- **backgroundOpacity**

Documentation

The messages posted by the header bar widget have been documented.

SVG Icon widget

iconPathPreset Editor

The property inspector editor for the **iconPathPreset** property is now an instance of the SVG Icon widget itself, which pops up the Icon Picker widget when clicked.

Push Button widget

New push button widget

The **widgetTheme** of the push button can be:

- iOS
- Android(Raised)
- Android(Flat)

The default theme is "iOS".

Palette Actions widget

Palette actions order

The palette actions widget now uses the sort using handler syntax to sort the elements of its data array, in order to sort numerically.

Browser widget

New Browser Widget Implementation on OSX

This feature adds an additional implementation of the browser widget based on the WebView component provided as part of OSX's WebKit framework.

Due to a number of stability issues in CEF on OSX, we have decided to make this new implementation the default for OSX browser widgets.

New browserType property

A new **browserType** property has been added to the browser widget to allow switching to an alternative implementation if required. To revert to using the CEF-based browser on OSX, set this property to "CEF".

New Browser Widget

A new embeddable browser browser widget has been added, making it much easier to use than the old revbrowser external.

To add a browser to your application, simply drag and drop the browser widget onto your stack and set the properties you need.

Properties

- **url** - The URL of the page displayed in the browser.
- **htmltext** - The HTML source of the content displayed in the browser
- **vscrollbar** - Whether or not the browser displays a vertical scrollbar
- **hscrollbar** - Whether or not the browser displays a horizontal scrollbar
- **userAgent** - The identifier sent by the browser when fetching web content.
- **javascriptHandlers** - A list of object script handlers that can be called by javascript code in the page loaded in the browser.

Messages

- **browserDocumentLoadBegin pUrl** - sent when a new document has completed loading in the browser.
- **browserDocumentLoadComplete pUrl** - sent when a new document has completed loading in the browser.
- **browserDocumentLoadFailed pUrl, pError** - sent when a new document has failed to load in the browser.
- **browserFrameDocumentLoadBegin pUrl** - sent when a new document has completed loading in a frame of the browser.
- **browserFrameDocumentLoadComplete pUrl** - sent when a new document has completed loading in a frame of the browser.
- **browserFrameDocumentLoadFailed pUrl, pError** - sent when a new document has failed to load in a frame of the browser.
- **browserNavigateBegin pUrl** - sent when the browser begins navigation to a new page.
- **browserNavigateComplete pUrl** - sent when the browser successfully navigates to a new page.
- **browserNavigateFailed pUrl, pError** - sent when the browser has failed to navigate to a new page.

Browser widget kind

The browser widget kind has been changed from **com.livecode.extensions.livecode.browser** to **com.livecode.widget.browser**.

This means users will be required to replace any existing browser widgets on their stacks with a new copy.

Tree View widget

Double Click

When a leaf node of the tree is double-clicked, an `actionDoubleClick` message is sent to the widget's script object. It has one parameter: the path associated with the clicked row.

Column View

Two new properties have been added in order to allow the key-value pairs of the tree view to be displayed in separate columns: **showSeparator** and **separatorRatio**.

The **showSeparator** property controls whether the tree view is in columns or not. The **separatorRatio** property controls where the separator is. More specifically, it is the proportion of the view space that is taken up by the key column.

When the **showSeparator** property is true, the separator can be dragged by the user to resize the columns.

Inspect Icon

When in read only mode, the tree widget will now display an 'open in new window' icon if the value of the array at the specified path contains a newline character, or is too large to display in the widget.

If this icon is clicked, an `actionInspect` message is sent to the widget's script object. It has one parameter: the path associated with the clicked row.

Sorting Options

The tree view widget now has the ability to sort the keys of its **arrayData** in different ways. Two properties have been added to achieve this:

- **sortOrder**: either "ascending" or "descending"
- **sortType**: either "text" or "numeric".

The default sort order of the widget is ascending numeric.

Segmented widget

Segment widths

When the segmented control's `segmentDisplay` property is set to "text", the widths of the

segments are calculated using the following rules:

- Measure the text, and assign widths accordingly
- Increase the size of any segments that are smaller than the corresponding `segmentMinWidth`
- If there is any space left in the widget bounds, increase each segment's size to fill the space

If after measuring and increasing to take min widths into account the segments will not fit, the segmented control will clip the content rather than shrink it.

Segment Colors

Four new properties have been added to the segmented control:

- `segmentColor`: the background color of the unselected segments
- `segmentSelectedColor`: the background color of the selected segments.
- `segmentLabelColor`: the label color of the unselected segments.
- `segmentSelectedLabelColor`: the label color of the selected segments.

Where any of these are empty, the control instead uses its associated default color.

This feature was sponsored by FMProMigrator.

Segment Count

A `segmentCount` property has been added to the segmented control.

This controls the number of segments. If it is less than the current number of segments, segments will be deleted from the right. If it is more, segments will be added with default icons (circle), labels ("Title") and names ("segment").

JSON Library

JSON Library Added

An LCB library, **com.livecode.library.json**, has been written to provide support for generating and parsing JavaScript Object Notation (JSON) data. See also http://json.org.

Functions

The library has two public handlers, `JsonImport` and `JsonExport`. `JsonImport` takes a string containing JSON-formatted text and parses it into a LiveCode value. `JsonExport` takes a LiveCode value and returns the equivalent value as a string in JSON format.

Using the library

The library is automatically loaded into the IDE, and the `JsonImport` and `JsonExport` handlers placed in the message path where they are available to call from any object.

In LiveCode Script, `JsonExport` takes any value and converts it to a string representing a JSON

encoded value.

To use the library from a LiveCode Builder widget or library, simply add it to the list of use clauses:

```
use com.livecode.library.json
```

When using the `JsonExport` handler in LCB, an error is thrown if the value passed is not of one of the following types:

- String
- Number
- List
- Array
- Boolean
- nothing

Examples

From LiveCode Script:

```
local tData, tJSON
put "a,b,c,d" into tData
split tData by comma
put JsonExport(tData) into tJSON -- contains {"1": "a","2": "b","3": "c","4":
"d"}
```

From LiveCode Builder:

```
variable tJSON as String
put "[1,1,2,3,5,8]" into tJSON

variable tData as List
put JsonImport(tJSON) into tData -- contains [1,1,2,3,5,8]
```

Widget Utilities module

Utility module added

A module, **com.livecode.widgetutils**, has been written to provide support for functions commonly needed by widgets.

The functions are:

- `constrainPathToRect`: Scales and translates a Path value to fit within a rectangle
- `intToString`: Formats an integer as a string
- `stripZeroes`: Removes any superfluous zeros and decimal places from a string

representation of a number.
- `colorToString`: Converts a value of type Color to an RGB or RGBA string representing the color
- `stringToColor`: Converts a comma-delimited string representing an RGB or RGBA color to a value of type Color.

See the documentation for more details on the individual handlers and their syntax.

Specific bug fixes

- **Bugs fixed in 8.0.0-dp-13: 16517, 16656.**
- Bugs fixed in 8.0.0-dp-11: 16399, 16404, 16553.
- Bugs fixed in 8.0.0-dp-9: 16021, 16155, 16320, 16335, 16341, 16354.
- Bugs fixed in 8.0.0-dp-8: 16135, 16241.
- Bugs fixed in 8.0.0-dp-6: 16009.
- Bugs fixed in 8.0.0-dp-5: 15815, 15840, 15850, 15851.
- Bugs fixed in 8.0.0-dp-3: 15224.

Dictionary additions

- **create widget** (*command*) has been added to the dictionary.
- **do in widget** (*command*) has been added to the dictionary.
- **documentFilename** (*property*) has been added to the dictionary.
- **export widget** (*command*) has been added to the dictionary.
- **fullClipboardData** (*property*) has been added to the dictionary.
- **fullDragData** (*property*) has been added to the dictionary.
- **go in widget** (*command*) has been added to the dictionary.
- **import widget** (*command*) has been added to the dictionary.
- **is not really** (*operator*) has been added to the dictionary.
- **is really** (*operator*) has been added to the dictionary.
- **kind** (*property*) has been added to the dictionary.
- **launch url in widget** (*command*) has been added to the dictionary.
- **load extension** (*command*) has been added to the dictionary.
- **loadedExtensions** (*function*) has been added to the dictionary.
- **lock clipboard** (*command*) has been added to the dictionary.
- **metadata of image** (*property*) has been added to the dictionary.
- **newWidget** (*message*) has been added to the dictionary.
- **popup** (*command*) has been added to the dictionary.
- **rawClipboardData** (*property*) has been added to the dictionary.
- **rawDragData** (*property*) has been added to the dictionary.
- **scriptOnly** (*property*) has been added to the dictionary.
- **unload extension** (*command*) has been added to the dictionary.
- **unlock clipboard** (*command*) has been added to the dictionary.
- **widget** (*object*) has been added to the dictionary.

Previous 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 6.7.8 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 6.7.1 Release Notes
- LiveCode 6.7.0 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