LiveCode 7.0.0-dp-5 Release Notes

Table of contents
Overview
Known issues
Platform support
  Windows
  Linux
  Mac
Setup
  Installation
  Uninstallation
Reporting installer issues
Activation
Multi-user and network install support (4.5.3)
Command-line installation
Command-line activation
Proposed changes
Engine changes
  Some filesystem entries in the root of a volume on Mac report as files when they are really folders.
  Memory leak in bitmap effects with spread of non-zero radius (e.g. spread 100%, radius 1; spread 50%, radius 2).
  Password protected stacks are corrupted by LiveCode 7
Unicode Support
Unicode and LiveCode
Creating Unicode Apps
  New & Existing apps - things to look out for
New Commands, Functions & Syntax
  Chunk expressions: byte, char, codepoint, codeunit
  Chunk expressions: paragraph, sentence and trueWord
  Synonym: segment
  Property: the formSensitive
  Command: open file/process/socket ... for <encoding> text
  Functions: textEncode, textDecode
  Functions: numToCodepoint, codepointToNum
  Functions: numToNativeChar, nativeCharToNum
  Function: normalizeText
  Function: codepointProperty
Updated Functions
  Function: binaryEncode
  Function: binaryDecode
Deprecated Features
  Functions: numToChar, charToNum
  Property: useUnicode
  Functions: uniEncode, uniDecode
  Function: measureUnicodeText
  Properties: unicodeText, unicodeLabel, unicodeTitle, unicodeTooltip, unicodePlainText,
Specific bug fixes (7.0.0-dp-5)
Specific bug fixes (7.0.0-dp-4)
Specific bug fixes (7.0.0-dp-3)
Specific bug fixes (7.0.0-dp-2)
Dictionary additions
Dictionary changes
Previous Release Notes
Overview

The LiveCode engine has undergone a large quantity of changes for the 7.0 release. The way values of variables are stored internally has been changed - in particular where before the engine used C-strings, it now uses a reference counted MCStringRef type. Every bit of code that displays text in LiveCode has been updated, and all the platform-specific API functions that manipulate characters now use the Unicode versions; as a result LiveCode is now fully Unicode compatible.

The other significant change to engine internals is the work done on syntax refactoring. The code that deals with statement execution, function evaluation and property access has been cleaned up and separated out from the parsing code, and moved into distinct modules based on functionality. This represents a major first step towards being able to implement Open Language.

Known issues

Every effort has been made to ensure that externally, the engine behaviour is identical to the current unrefactored release. In other words, users should not notice any difference in functionality in their existing stacks. However, users will notice a general slow-down caused by lack of optimisation in this release - this will be addressed for DP 2.

- 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 engine files are much larger than previous versions due to inclusion of ICU data
- LiveCode does not run correctly when installed to Unicode paths on OSX
- On Windows, executing LiveCode from the installer fails as it cannot find the IDE
- Android app label is not yet Unicode compatible
- Auto-updater process doesn't terminate when dismissed

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

The engine supports the following Windows OSes:

- 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 platforms the engine still runs as a 32-bit application through the WoW layer.

Linux

The linux engine requires the following:
• 32-bit installation, or a 64-bit linux distribution that has a 32-bit compatibility layer
• 2.4.x or later kernel
• X11R5 capable Xserver running locally on a 24-bit display
• glibc 2.3.2 or later
• gtk/gdk/glib (optional – required for native theme support)
• pango/xft
• lcms (optional – required for color profile support in JPEGs and PNGs)
• gksu (optional – required for elevate process support)

**Note:** The optional requirements (except for gksu and lcms) are also required by Firefox and Chrome, so if your linux distribution runs one of those, it will run the engine.

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

**Note:** LiveCode and standalones it builds may work on remote Xservers and in other bit-depths, however this mode of operation is not currently 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

**Note:** The engine runs as a 32-bit application regardless of the capabilities of the underlying processor.
Setup

Installation

Each distinct version has its own complete folder – multiple versions will no longer install side-by-side: on Windows (and Linux), each distinct version will gain its own start menu (application menu) entry; on Mac, each distinct version will have its own app bundle.

The default location for the install on the different platforms when installing for 'all users' are:

- Windows: `<x86 program files folder>/RunRev/ LiveCode 7.0.0-dp-5`
- Linux: `/opt/runrev/livecode-7.0.0-dp-5`
- Mac: `/Applications/ LiveCode 7.0.0-dp-5.app`

The default location for the install on the different platforms when installing for 'this user' are:

- Windows: `<user roaming app data folder>/RunRev/Components/LiveCode 7.0.0-dp-5`
- Linux: `~/runrev/components/livecode-7.0.0-dp-5`
- Mac: `~/Applications/ LiveCode 7.0.0-dp-5.app`

Note: If your linux distribution does not have the necessary support for authentication (gksu) then the installer will run without admin privileges so you will have to manually run it from an admin account to install into a privileged location.

Uninstallation

On Windows, the installer hooks into the standard Windows uninstall mechanism. This is accessible from the appropriate pane in the control panel.

On Mac, simply drag the app bundle to the Trash.

On Linux, the situation is currently less than ideal:

- open a terminal
- `cd` to the folder containing your rev install. e.g.

  ```
  cd /opt/runrev/livecode-7.0.0-dp-5
  ```

- execute the `.setup.x86` file. i.e.

  ```
  ./setup.x86
  ```

- follow the on-screen instructions.

Reporting installer issues

If you find that the installer fails to work for you then please file a bug report in the RQCC or email support@runrev.com so we can look into the problem.

In the case of failed install it is vitally important that you include the following information:

- Your platform and operating system version
- The location of your home/user folder
- The type of user account you are using (guest, restricted, admin etc.)
- The installer log file located as follows:
  - Windows 2000/XP: `<documents and settings folder>/<user>/Local Settings/`
Windows Vista/7: \<users folder>\<user>\AppData\Local\RunRev\Logs
Linux: \<home>\.runrev\logs
Mac: \<home>\Library\Application Support\Logs\RunRev

Activation

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.

Multi-user and network install support (4.5.3)

In order to better support institutions needing to both deploy the IDE to many machines and to license them for all users on a given machine, a number of facilities have been added which are accessible by using the command-line.

Note: These features are intended for use by IT administrators for the purposes of deploying LiveCode in multi-user situations. They are not supported for general use.

Command-line installation

It is possible to invoke the installer from the command-line on both Mac and Windows. When invoked in this fashion, no GUI will be displayed, configuration being supplied by arguments passed to the installer. On both platforms, the command is of the following form:

\<exe> install noui options

Here options is optional and consists of one or more of the following:

- allusers: Install the IDE for all users. If not specified, the install will be done for the current user only.
- desktopshortcut: Place a shortcut on the Desktop (Windows-only).
- startmenu: Place shortcuts in the Start Menu (Windows-only).
- location: The location to install into. If not specified, the location defaults to those described in the Layout section above.
- log logfile: A file to place a log of all actions in. If not specified, no log is generated.

Note that the command-line variant of the installer does not do any authentication. Thus, if you wish to install to an admin-only location you will need to be running as administrator before executing the command. As the installer is actually a GUI application, it needs to be run slightly differently from other command-line programs.
In what follows <installerexe> should be replaced with the path of the installer executable or app (inside the DMG) that has been downloaded.
On Windows, you need to do:

start /wait <installerexe> install noui options
On Mac, you need to do:

```
"<installerexe>/Contents/MacOS/installer" install noui options
```

On both platforms, the result of the installation will be written to the console.

**Command-line activation**

In a similar vein to installation, it is possible to activate an installation of LiveCode for all-users of that machine by using the command-line. When invoked in this fashion, no GUI will be displayed, activation being controlled by any arguments passed.

On both platforms, the command is of the form:

```
<exe> activate -file license -passphrase phrase
```

This command will load the manual activation file from `license`, decrypt it using the given `passphrase` and then install a license file for all users of the computer. Manual activation files can be downloaded from the 'My Products' section of the RunRev customer accounts area.

This action can be undone using the following command:

```
<exe> deactivate
```

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

In what follows `<livecodeexe>` should be replaced with the path to the installed LiveCode executable or app that has been previously installed.

On Windows, you need to do:

```
start /wait <livecodeexe> activate -file license -passphrase phrase
start /wait <livecodeexe> deactivate
```

On Mac, you need to do:

```
"<livecodeexe>/Contents/MacOS/LiveCode" activate -file license -passphrase phrase
"<livecodeexe>/Contents/MacOS/LiveCode" deactivate
```

On both platforms, the result of the activation will be written to the console.

**Proposed changes**

The following changes are likely to occur in the next or subsequent non-maintenance release:

- The engine (both IDE and standalone) **will require** gtk, gdk and glib on Linux
Engine changes

[[ Bugfix 12502 ]] Fix a null-pointer deref in PDF printin (7.0.0-dp-5)

Some filesystem entries in the root of a volume on Mac report as files when they are really folders. (7.0.0-dp-4)

The 'net', 'home' and 'dev' folders do not report as folders in when using 'the folders' - they appear as files instead.

Memory leak in bitmap effects with spread of non-zero radius (e.g. spread 100%, radius 1; spread 50%, radius 2). (7.0.0-dp-4)

Password protected stacks are corrupted by LiveCode 7 (7.0.0-dp-2)

Unicode Support (7.0.0-dp-1)

Unicode and LiveCode

Traditionally, computer systems have stored text as 8-bit bytes, with each byte representing a single character (for example, the letter 'A' might be stored as 65). This has the advantage of being very simple and space efficient whilst providing enough (256) different values to represent all the symbols that might be provided on a typewriter.

The flaw in this scheme becomes obvious fairly quickly: there are far more than 256 different characters in use in all the writing systems of the world, especially when East Asian ideographic languages are considered. But, in the pre-internet days, this was not a big problem.

LiveCode, as a product first created before the rise of the internet, also adopted the 8-bit character sets of the platforms it ran on (which also meant that each platform used a different character set: MacRoman on Apple devices, CP1252 on Windows and ISO-8859-1 on Linux and Solaris). LiveCode terms these character encodings "native" encodings.

In order to overcome the limitations of 8-bit character sets, the Unicode Consortium was formed. This group aims to assign a unique numerical value ("codepoint") to each symbol used in every written language in use (and in a number that are no longer used!). Unfortunately, this means that a single byte cannot represent any possible character.

The solution to this is to use multiple bytes to encode Unicode characters and there are a number of schemes for doing so. Some of these schemes can be quite complex, requiring a varying number of bytes for each character, depending on its codepoint.

LiveCode previously added support for the UTF-16 encoding for text stored in fields but this could be cumbersome to manipulate as the variable-length aspects of it were not handled transparently and it could only be used in limited contexts. Unicode could not be used in control names, directly in scripts or in many other places where it might be useful.

In LiveCode 7.0, the engine has been extensively re-written to be able to handle Unicode text transparently throughout. The standard text manipulation operations work on Unicode text without any additional effort on your part; Unicode text can now be used to name controls, stacks and other objects; menus containing Unicode selections no longer require tags to be usable - anywhere text is used, Unicode should work.
Adding this support has required some changes but these should be minor. Existing apps should continue to run with no changes but some tweaking may be required in order to adapt them for full Unicode support - this is described in the next section - Creating Unicode Apps.

Creating Unicode Apps

Creating stacks that support Unicode is no more difficult than creating any other stack but there are a few things that should be borne in mind when developing with Unicode. The most important of these is the difference between text and binary data - in previous versions of LiveCode, these could be used interchangeably; doing this with Unicode may not work as you expect (but it will continue to work for non-Unicode text).

When text is treated as binary data (i.e when it is written to a file, process, socket or other object outside of the LiveCode engine) it will lose its Unicode-ness: it will automatically be converted into the platform's 8-bit native character set and any Unicode characters that cannot be correctly represented will be converted into question mark '?' characters.

Similarly, treating binary data as text will interpret it as native text and won't support Unicode.

To avoid this loss of data, text should be explicitly encoded into binary data and decoded from binary data at these boundaries - this is done using the textEncode and textDecode functions (or its equivalents, such as opening a file using a specific encoding).

Unfortunately, the correct text encoding depends on the other programs that will be processing your data and cannot be automatically detected by the LiveCode engine. If in doubt, UTF-8 is often a good choice as it is widely supported by a number of text processing tools and is sometimes considered to be the “default” Unicode encoding.

New & Existing apps - things to look out for

- When dealing with binary data, you should use the byte chunk expression rather than char - char is intended for use with textual data and represents a single graphical character rather than an 8-bit unit.
- Try to avoid hard-coding assumptions based on your native language - the formatting of numbers or the correct direction for text layout, for example. LiveCode provides utilities to assist you with this.
- Regardless of visual direction, text in LiveCode is always in logical order - word 1 is always the first word; it does not depend on whether it appears at the left or the right.
- Even English text can contain Unicode characters - curly quotation marks, long and short dashes, accents on loanwords, currency symbols...

New Commands, Functions & Syntax

Chunk expressions: byte, char, codepoint, codeunit

byte x to y of text -- Returns bytes from a binary string
char x to y of text -- As a series of graphical units
codepoint x to y of text -- As a series of Unicode codepoints
codeunit x to y of text -- As a series of encoded units

A variety of new chunk types have been added to the LiveCode syntax to support the various methods of referring to the components of text. This set is only important to those implementing low-level functions and can be safely ignored by the majority of users.

The key change is that byte and char are no longer synonyms - a byte is strictly an 8-bit unit and can only be reliably used with binary data. For backwards compatibility, it returns the corresponding native character
from Unicode text (or a ‘?’ if not representable) but this behaviour is deprecated and should not be used in new code.

The **char** chunk type no longer means an 8-bit unit but instead refers to what would naturally be thought of as a single graphical character (even if it is composed of multiple sub-units, as in some accented text or Korean ideographs). Because of this change, it is inappropriate to use this type of chunk expression on binary data.

The **codepoint** chunk type allows access to the sequence of Unicode codepoints which make up the string. This allows direct access to the components that make up a character. For example, á can be encoded as (a,combining-acute-accent) so it is one character, but two codepoints (the two codepoints being a and combining-acute-accent).

The **codeunit** chunk type allows direct access to the UTF-16 code-units which notionally make up the internal storage of strings. The codeunit and codepoint chunk are the same if a string only contains unicode codepoints from the Basic Multilingual Plane. If, however, the string contains unicode codepoints from the Supplementary Planes, then such codepoints are represented as two codeunits (via the surrogate pair mechanism). The most important feature of the ‘codeunit’ chunk is that it guarantees constant time indexed access into a string (just as char did in previous engines) however it is not of general utility and should be reserved for use in scripts which need greater speed but do not need to process Supplementary Plane characters, or are able to do such processing themselves.

The hierarchy of these new and altered chunk types is as follows: byte w of codeunit x of codepoint y of char z of word...

**Chunk expressions: paragraph, sentence and trueWord**

The **sentence** and **trueWord** chunk expressions have been added to facilitate the processing of text, taking into account the different character sets and conventions used by various languages. They use the ICU library, which uses a large database of rules for its boundary analysis, to determine sentence and word breaks. ICU word breaks delimit not only whitespace but also individual punctuation characters; as a result the LiveCode **trueWord** chunk disregards any such substrings that contain no alphabetic or numeric characters.

The **paragraph** chunk is identical to the existing **line** chunk, except that it is also delimited by the Unicode paragraph separator (0x2029), which reflects paragraph breaking in LiveCode fields.

The hierarchy of these new chunk types is as follows: trueword v of word w of item x of sentence y of paragraph z of line...

**Synonym: segment**

The **segment** chunk type has been added as a synonym to the existing **word** chunk. This in order to allow you to update your scripts to use the newer syntax in anticipation of a future change to make the behaviour of the **word** chunk match the new **trueWord** behaviour.

We would anticipate changing the meaning of **word** with our ‘Open Language’ project. It requires us to create a highly accurate script translation system to allow old scripts to be rewritten in new revised and cleaner syntax. It is at this point we can seriously think about changing the meaning of existing tokens, including **word**. Existing scripts will continue to run using the existing parser, and they can be converted (by the user) over time to use the newer syntax.

**Property: the formSensitive**

set the **formSensitive** to false -- Default value
This property is similar to the `caseSensitive` property in its behaviour - it controls how text with minor differences is treated in comparison operations.

Normalization is a process defined by the Unicode standard for removing minor encoding differences for a small set of characters and is more fully described in the `normalizeText` function.

**Command:** open file/process/socket ... for <encoding> text

`open file "log.txt" for utf-8 text read` -- Opens a file as UTF-8

Opens a file, process or socket for text I/O using the specified encoding. The encodings supported by this command are the same as those for the `textEncode` / `textDecode` functions. All text written to or read from the object will undergo the appropriate encoding/decoding operation automatically.

**Functions:** `textEncode`, `textDecode`

`textEncode(string, encoding)` -- Converts from text to binary data

`textDecode(binary, encoding)` -- Converts from binary data to text

Supported encodings are (currently):

- "ASCII"
- "ISO-8859-1" (Linux only)
- "MacRoman" (OSX only)
- "Native" (ISO-8859-1 on Linux, MacRoman on OSX, CP1252 Windows)
- "UTF-16"
- "UTF-16BE"
- "UTF-16LE"
- "UTF-32"
- "UTF-32BE"
- "UTF-32LE"
- "UTF-8"
- "CP1252" (Windows only)

Spelling variations are ignored when matching encoding strings (i.e all characters other than [a-zA-z0-9] are ignored in matches as are case differences).

It is very highly recommended that any time you interface with things outside LiveCode (files, network sockets, processes, etc) that you explicitly `textEncode` any text you send outside LiveCode and `textDecode` all text received into LiveCode. If this doesn't happen, a platform-dependent encoding will be used (which normally does not support Unicode text).

It is not, in general, possible to reliably auto-detect text encodings so please check the documentation for the programme you are communicating with to find out what it expects. If in doubt, try "UTF-8".

**Functions:** `numToCodepoint`, `codepointToNum`

`numToCodepoint(number)` -- Converts a Unicode codepoint to text

`codepointToNum(codepoint)` -- Converts a codepoint to an integer

These functions convert between the textual form of a Unicode character and its numerical identifier ("codepoint"). Codepoints are integers in the range 0x000000 to 0x1FFFF that identify Unicode characters. For example, the space (" ") character is 0x20 and "A" is 0x41.
The codepointToNum function raises an exception if the argument contains multiple codepoints; it should generally be used in the form:

    codepointToNum(codepoint x of string)

The numToCodepoint function raises an exception if the given integer is out of range for Unicode codepoints (i.e. if it is negative or if it is greater than 0x10FFFF). Codepoints that are not currently assigned to characters by the latest Unicode standard are not considered to be invalid in order to ensure compatibility with future standards.

Functions: numToNativeChar, nativeCharToNum

numToNativeChar(number) -- Converts an 8-bit value to text
nativeCharToNum(character) -- Converts a character to an 8-bit value

These functions convert between text and native characters and are replacements for the deprecated numToChar and charToNum functions.

As the "native" character sets for each platform have a limited and different repertoire, these functions should not be used when preservation of Unicode text is desired. Any characters that cannot be mapped to the native character set are replaced with a question mark character ('?').

Unless needed for compatibility reasons, it is recommended that you use the numToCodepoint and codepointToNum functions instead.

Function: normalizeText

normalizeText(text, normalForm) -- Normalizes to the given form

The normalizeText function converts a text string into a specific 'normal form'.

Use the normalizeText function when you require a specific normal form of text.

In Unicode text, the same visual string can be represented by different character sequences. A prime example of this is precomposed characters and decomposed characters: an 'e' followed by a combining acute character is visually indistinguishable from a precombined 'é' character. Because of the confusion that can result, Unicode defined a number of "normal forms" that ensure that character representations are consistent.

The normal forms supported by this function are:

- "NFC" - precomposed
- "NFD" - decomposed
- "NFKC" - compatibility precomposed
- "NFKD" - compatibility decomposed

The "compatibility" normal forms are designed by the Unicode Consortium for dealing with certain legacy encodings and are not generally useful otherwise.

It should be noted that normalization does not avoid all problems with visually-identical characters; Unicode contains a number of characters that will (in the majority of fonts) be indistinguishable but are nonetheless completely different characters (a prime example of this is "M" and U+2164 "Ⅿ" ROMAN NUMERAL ONE THOUSAND).

Unless the formSensitive handler property is set to true, LiveCode ignores text normalization when
performing comparisons (is, <>, etc).

Returns: the text normalized into the given form.

    set the formSensitive to true
    put "e" & numToCodepoint("0x301") into tExample -- Acute accent
    put tExample is "é"      -- Returns false
    put normalizeText(tExample, "NFC") is "é"  -- Returns true

Function: codepointProperty

codepointProperty("A", "Script") -- "Latin"
codepointProperty("β", "Uppercase") -- false
codepointProperty("ơ", "Name") -- GREEK SMALL LETTER SIGMA

Retrieves a UCD character property of a Unicode codepoint.

The Unicode standard and the associated Unicode Character Database (UCD) define a series of properties
for each codepoint in the Unicode standard. A number of these properties are used internally by the engine
during text processing but it is also possible to query these properties directly using this function.

This function is not intended for general-purpose use; please use functions such as toUpper or the "is"
operators instead.

There are many properties available; please see the version 6.3.0 of the Unicode standard, Chapter 4 and
Section 5 of Unicode Technical Report (TR)#44 for details on the names and values of properties. Property
names may be specified with either spaces or underscores and are not case-sensitive.

Examples of supported properties are:

- "Name" - Unique name for this codepoint
- "Numeric_Value" - Numerical value, e.g. 4 for "4"
- "Quotation_Mark" - True if the codepoint is a quotation mark
- "Uppercase_Mapping" - Uppercase equivalent of the character
- "Lowercase" - True if the codepoint is lower-case

Updated Functions

Function: binaryEncode

A new letter has been introduced to allow one to binary encode unicode strings.
Following the dictionary definitions, it consists of:

u{<encoding>}: convert the input string to the encoding specified in the curly braces, and output up to
amount bytes of the string created - stopping at the last encoded character fitting in the amount - padding
with \0.

U{<encoding>}: convert the input string to the encoding specified in the curly braces, and output up to
amount bytes of the string created - stopping at the last encoded character fitting in the amount - padding
with encoded spaces, and then \0 if the last encoded space cannot fit within the amount specified.
The encoding, surrounded by curly braces, is optional - no one specified would default to the behaviour of 'a' - and must match one of those applicable to textEncode.

**Function: binaryDecode**

A new letter has been introduced to allow one to binary decode unicode strings. Following the dictionary definitions, it consists of:

\[ u{<\text{encoding}>} \]: convert amount bytes of the input string to the specified encoding, padding with '0'.

\[ U{<\text{encoding}>} \]: converts amount bytes of the input to the specified encoding, skipping trailing spaces.

The encoding, surrounded by curly braces, is optional - no one specified would default to the behaviour of 'a' - and must match one of those applicable to textEncode.

**Deprecated Features**

**Functions: numToChar, charToNum**

These functions should not be used in new code as they cannot correctly handle Unicode text.

**Property: useUnicode**

This property should not be used in new code, as it only affects the behaviour of numToChar and charToNum, which are themselves deprecated.

**Functions: uniEncode, uniDecode**

These functions should not be used in new code as their existing behaviour is incompatible with the new, transparent Unicode handling (the resulting value will be treated as binary data rather than text). These functions are only useful in combination with the also-deprecated unicode properties described below.

**Function: measureUnicodeText**

This function should not be used in new code. measureUnicodeText(tText) is equivalent to measureText(textDecode(tText, "UTF16")).

**Properties: unicodeText, unicodeLabel, unicodeTitle, unicodeTooltip, unicodePlainText, unicodeFormattedText**

These properties should not be used in new code; simply set the text, label, title etc. as normal. Assigning values other than those returned from uniEncode to these properties will not produce the desired results.

The following are now equivalent:

```
set the unicodeText of field 1 to tText
```

```
set the text of field 1 to textDecode(tText, "UTF16")
```

and similarly for the other unicode-prefixed properties.
Specific bug fixes (7.0.0-dp-5)
(bug fixes specific to the current build are highlighted in bold, reverted bug fixes are stricken through)

12502  [[ Bugfix 12502 ]] Fix a null-pointer deref in PDF printing
12499  trueWord n + m of tText for n the number of trueWords of tText always returns trueWord n
12497  pageRanges property missing from LiveCode 7.0
12496  [[ Bugfix 12496 ]] Set the clipping rectangle for text blocks correctly
12494  Setting the randomSeed to large number fails in 7.0
12491  "Go to Definition" doesn't work in script editor
12489  filter/replace difference in 7.0
12486  [[ Bugfix 12486 ]] Add missing MovieControllerID property to the Player property table
12483  Graphic effects not working in 7.0 DP4
12482  replace does not work
12074  Answer dialog messages should be aligned to the right

Specific bug fixes (7.0.0-dp-4)

12459  Setting any graphic effects to "none" crashes LC 7 dp3
12457  sorting marked cards with single unmarked card crashes LiveCode
12432  clickchunk and click text are not identical
12428  Lc 7.0 DP3 does not sanitize data when setting points of polygon
12423  If you choose the browse tool (run) after Editing a group - Livecode crashes.
12422  Sort puts a "p" after the last character and foreign letters is not sorted correct
12409  Fields in LC 7 fail to display binfile url imagesource
12407  'Garbage' with read from socket
12360  open file as utf-8 mode doesn't work exactly as documented
12351  Crash on write then read until EOF on driver
12350  The fontStyles is incorrect on iOS
12345  AVD's appear in the list but can't be selected for testing.
12344  Can't open recent file
12309  Build for Windows fails with i/o error
12294  Crash and flaky behavior at certain points.
12288  Prevent User Samples stack hanging due to resize error
12286  Maximizing a window where only the title-bar is on-screen causes a crash on Mac.
12265  Rounded rects are drawn incorrectly when using the image editing tools with a linesize 1
12246  Serial I/O fails on write
12239  Magnification window shows corrupted image.
12237  Attempt to attach to the launching console on Windows (if any) in standalones.
12236  Dropdown menus are clipped on Windows when text scale > 100%
12235  Tooltips clipped on Windows when text scaling > 100%
12227  When Windows screen display is set to 125% popups sometimes break
12223  Windows backdrop doesn't cover the full desktop area when displayed on a high-dpi screen.
12210  revBrowserSnapshot not working on Windows with IE9+
12206  Buttons of menu type can't be inspected if first created object
12200  Some filesystem entries in the root of a volume on Mac report as files when they are really
folders.

- linux uninstaller needs execute permission
- Standalone engine crashes when -ui specified on Linux.
- Clicking in the scrollbar well doesn't work if the click is too short.
- 'the pageRanges' doesn't work on fields with more than 64K chars.
- Setting the usePixelScaling property doesn't update all windows on Mac.
- Styling does not work for certain iOS fonts
- Non-existant command line parameter variables ($) behave strangely with split.
- setting tabstops to 2 equal numbers and then turning vGrid on hangs LC
- put the executionContexts crashes LiveCode server
- exit causes livecode server to crash
- Livecode server crashed if you call paramCount()
- Graphics missing from imported Hypercard stack
- On awakening Android device from sleep
- The script editor doesn't scroll horizontally as text is entered
- Can't test an app on Android
- The backdrop on Windows is always black
- Opaque groups do not completely draw their backgrounds when acceleratedRendering is enabled
- Slow-down in setting contents of fields on Windows since 6.1.3.
- On Retina Mac's scrollbars with small thumbs render smaller than they should.
- Caret is too thin on Retina displays.
- Windows engine hangs after multiple stack redraws.
- import screen snapshot on iOS creates image of incorrect size.
- HTTP (HTML) URLs encoded with anything other than a native character set are returned incorrectly
- arrayDecode on a file containing the result of arrayEncode on an empty array causes execution error
- "import snapshot from rect ..." only imports part of the screen on Windows
- Spacing is incorrect for Windows scaled text
- effective textColor returns empty value for styled text
- Memory leak in bitmap effects with spread of non-zero radius (e.g. spread 100%)
- Italic characters with underhang are clipped on windows
- Stoked graphics clipped when printed
- uuid and randomBytes functions don't work on iOS when Encryption support is not included
- Crash when putting an empty string into an XML node using PutIntoXMLNode.
- Android apps only partially drawn after rotating device during lock sreen
- Once large scripts start scrolling the script editor
- ResizeControl is not sent when resizing images
- Round buttons are drawn incorrectly
- Backdrop not displayed on Linux
- Anti-aliasing inconsistent for 1 pixel lines and curves
- magnify and edit image crashes LC
- Only allow interaction with scrollbars on groups in browse mode.
- On Mac
- The machine() function returns "unknown" under Mac OSX
Specific bug fixes (7.0.0-dp-3)

- Saving 2.7 file format stack causes crash
- Case sensitive does not work
- TextEncode ASCII support is actually native
- Equality testing is slow
- ‘char/byte/codepoint 1 of s’ is slow
- ‘repeat for each byte b in empty’ crashes
- ‘the number of bytes of ...’ is slow
- Fetching byte chunks does not clamp the range to the bounds of the input data.
- Sometimes length() and number or chars are wrong
- Put after/before on an uninitialised
- LiveCode crashes when changing the window kind
- Create button in group command fails
- The mousechunk end index is one larger than it ought to be
- Erroneous Socket Timeout Error
- The drawer command crashes Livecode 7.0 when using ‘...at position’ variant.
- Fix wrong application title displaying on Linux
- Update GTK icon cache post-install
- RevExecuteSQL writes incomplete data into SQLite BLOB columns
- Scrambled word order for label field with Hebrew and English Text
- Buttons that contain Hebrew Text is in wrong order
- Linux Standalone does not run. Segmentation fault.
- "save stack" corrupt password protected stacks
- IDE fails to launch when installed to a Unicode path
- Char 1 of (e + combining acute accent) returns e
- Split command causes IDE to stop responding
- IDE takes 8 seconds when adding a new line in Script Editor
- Repeat loop is very slow in 7.0 DP1
- Opening the TestFramework stack crashes LiveCode

Specific bug fixes (7.0.0-dp-2)

- Convert command fails with invalid date since 7.0
- Setting acceleratorModifiers of button causes crash
- OSX picking wrong file extension for filenames with two '.' characters
- HilitColor and borderColor is not working in 7.0DP1
- HGrid
- Group with label can't be saved in 5.5 file format
- Formatting hex string crashes LiveCode 7.0
- New chunk types (paragraph
- 'lock screen for visual effect in rect...' not working
- NumToByte works differently form numToChar in 6.6
- Put does not populate the result on iOS
- Calling mobileControlTarget () crashes the application
- Password protected stacks are corrupted by LiveCode 7
11963  Dotted border of selection in List control is incorrectly aligned
11960  LC crashes when selecting wrapped text in Contents pane
11958  Text wrapping improperly breaks text mid-word
11954  sort field does not work
11953  sort card of stack crashes
11950  mark card does not work
11949  find string in field does not work
11948  Export snaphot crashes LiveCode when it should return empty rect error
11947  Vertical tabulation in a field causes the engine to hang
11945  The number of paragraphs reported value is incorrect
11943  Script Editor does not resize correctly with the resize handle
11940  Variables not being resolved in the script debugger.

**Dictionary additions**

- `byteOffset (function)` has been added to the dictionary.
- `codepointOffset (function)` has been added to the dictionary.
- `codepointProperty (function)` has been added to the dictionary.
- `codepointToNum (function)` has been added to the dictionary.
- `codeunitOffset (function)` has been added to the dictionary.
- `nativeCharToNum (function)` has been added to the dictionary.
- `normalizeText (function)` has been added to the dictionary.
- `numToCodepoint (function)` has been added to the dictionary.
- `numToNativeChar (function)` has been added to the dictionary.
- `paragraphOffset (function)` has been added to the dictionary.
- `sentenceOffset (function)` has been added to the dictionary.
- `textDecode (function)` has been added to the dictionary.
- `textEncode (function)` has been added to the dictionary.
- `tokenOffset (function)` has been added to the dictionary.
- `truewordOffset (function)` has been added to the dictionary.
- `codepoint (keyword)` has been added to the dictionary.
- `codepoints (keyword)` has been added to the dictionary.
- `codeunit (keyword)` has been added to the dictionary.
- `codeunits (keyword)` has been added to the dictionary.
- `paragraph (keyword)` has been added to the dictionary.
- `paragraph (keyword)` has been added to the dictionary.
- `segment (keyword)` has been added to the dictionary.
- `segments (keyword)` has been added to the dictionary.
- `sentence (keyword)` has been added to the dictionary.
- `sentences (keyword)` has been added to the dictionary.
- `trueWord (keyword)` has been added to the dictionary.
- `trueWords (keyword)` has been added to the dictionary.
- `formSensitive (property)` has been added to the dictionary.
- `tabAlign (property)` has been added to the dictionary.

**Dictionary changes**
The entry for create (command) has been updated.
The entry for do (command) has been updated.
The entry for export snapshot (command) has been updated.
The entry for find (command) has been updated.
The entry for open driver (command) has been updated.
The entry for open file (command) has been updated.
The entry for open process (command) has been updated.
The entry for sort container (command) has been updated.
The entry for sort (command) has been updated.
The entry for repeat (control structure) has been updated.
The entry for charToNum (function) has been updated.
The entry for measureUnicodeText (function) has been updated.
The entry for numToChar (function) has been updated.
The entry for uniDecode (function) has been updated.
The entry for uniEncode (function) has been updated.
The entry for byte (keyword) has been updated.
The entry for character (keyword) has been updated.
The entry for plain (keyword) has been updated.
The entry for word (keyword) has been updated.
The entry for words (keyword) has been updated.
The entry for is among (operator) has been updated.
The entry for is not among (operator) has been updated.
The entry for HTMLText (property) has been updated.
The entry for pageIncrement (property) has been updated.
The entry for textStyle (property) has been updated.
The entry for unicodeFormattedText (property) has been updated.
The entry for unicodeLabel (property) has been updated.
The entry for unicodePlainText (property) has been updated.
The entry for unicodeText (property) has been updated.
The entry for unicodeTitle (property) has been updated.
The entry for unicodeTooltip (property) has been updated.
The entry for useUnicode (property) has been updated.
## Previous Release Notes

<table>
<thead>
<tr>
<th>Version</th>
<th>Release Notes</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.1</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_6_1/LiveCodeNotes-6_6_1.pdf" alt="6.6.1 Release Notes" /></td>
</tr>
<tr>
<td>6.6.0</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_6_0/LiveCodeNotes-6_6_0.pdf" alt="6.6.0 Release Notes" /></td>
</tr>
<tr>
<td>6.5.2</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_5_2/LiveCodeNotes-6_5_2.pdf" alt="6.5.2 Release Notes" /></td>
</tr>
<tr>
<td>6.5.1</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_5_1/LiveCodeNotes-6_5_1.pdf" alt="6.5.1 Release Notes" /></td>
</tr>
<tr>
<td>6.5.0</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_5_0/LiveCodeNotes-6_5_0.pdf" alt="6.5.0 Release Notes" /></td>
</tr>
<tr>
<td>6.1.3</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_1_3/LiveCodeNotes-6_1_3.pdf" alt="6.1.3 Release Notes" /></td>
</tr>
<tr>
<td>6.1.2</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_1_2/LiveCodeNotes-6_1_2.pdf" alt="6.1.2 Release Notes" /></td>
</tr>
<tr>
<td>6.1.1</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_1_1/LiveCodeNotes-6_1_1.pdf" alt="6.1.1 Release Notes" /></td>
</tr>
<tr>
<td>6.1.0</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_1_0/LiveCodeNotes-6_1_0.pdf" alt="6.1.0 Release Notes" /></td>
</tr>
<tr>
<td>6.0.2</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_0_2/LiveCodeNotes-6_0_2.pdf" alt="6.0.2 Release Notes" /></td>
</tr>
<tr>
<td>6.0.1</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_0_1/LiveCodeNotes-6_0_1.pdf" alt="6.0.1 Release Notes" /></td>
</tr>
<tr>
<td>6.0.0</td>
<td>Release Notes</td>
<td><img src="http://downloads.livecode.com/livecode/6_0_0/LiveCodeNotes-6_0_0.pdf" alt="6.0.0 Release Notes" /></td>
</tr>
</tbody>
</table>