



## Zygodact 2.0

Automated registration key system

---

*Zygodactyl: A toe arrangement in perching birds where digits 1 and 4 are reversed.*

---

Zygodact provides an easy, automated way to add a complete registration serial key system to your LiveCode standalone. With only a single line of code, Zygodact will check to see whether your standalone has been registered with a valid serial key. If not, it displays a dialog box requesting a name and registration number. Zygodact does all the work for you—it gives you the tools you need to generate any number of unique serial keys for your standalone, and handles every aspect of requesting and tracking a valid registration on your user's computer.

Zygodact creates a matched set of stacks: a registration dialog, a serial key generator, and optionally, a CGI stack for use on a server. Every set of stacks is unique, and the registration dialog and key generator you produce will never match any others. In fact, you will need to generate all the stacks at once, since you can't mix and match between sets. This gives you the security of knowing that other developers who use Zygodact can never generate the same serial keys and registration data as yours.

### Features

- One-click generation of a registration substack and serial key generator
- Optional matched CGI stack
- Complete solution with only one line of script
- Cross-platform use on Windows, OS X, Linux, Android, and iOS\*

\* Apple, Amazon, and the Android Play Store do not allow custom registration schemes, and will reject an app submitted to their on-line stores if it uses Zygodact. Zygodact can, however, be used for in-house distribution with Apple's Enterprise or Ad Hoc provisioning profiles, or with Android apps distributed privately or in-house.

## Contents

What's new in Zygodact 2.0 .....	4
Setting up Zygodact .....	5
How Zygodact works .....	5
<i>Zygodact's default preference paths</i> .....	6
<i>Register behavior during development</i> .....	6
Invalidating a registration .....	6
Building a standalone with Zygodact .....	7
For more advanced work .....	7
<i>Using your own Preferences stack</i> .....	7
<i>Data returned to your script</i> .....	8
<i>Internationalizing the error text</i> .....	8
<i>Using Zygodact in combination with a free trial period</i> .....	9
Customizing the stacks .....	12
<i>Customizing the Preferences stack</i> .....	12
<i>Customizing the Register stack</i> .....	12
Using the Key Generator .....	12
<i>Generating a single key</i> .....	12
<i>Generating multiple keys</i> .....	13

Using a custom identifier .....	13
<i>Serial key differences when using custom identifiers.....</i>	<i>13</i>
<i>Retrieving an identifier from a key .....</i>	<i>13</i>
Emailing a key in a form letter .....	14
<i>Template format .....</i>	<i>14</i>
Compiling the Key Generator for use on a mobile device .....	15
Using the CGI Generator.....	15
<i>The Zygodact universal script.....</i>	<i>16</i>

## **What's new in Zygodact 2.0**

1. Custom identifiers allow you to tag serial keys in order to differentiate them. Now your app can distinguish trial versions from full versions, or determine what features to unlock.
2. Compatibility with Android and iOS mobile platforms (subject to OS restrictions.)
3. Auto-generation of emails from the Key Generator using your own text templates.
4. Key Generators can be compiled for mobile devices so you can provide fulfillment on the go.
5. Improved cross-platform compatibility when generating keys from non-ASCII text.
6. Automatic version checking notifies you of new updates.
7. Bug fixes and internal optimizations.

## Setting up Zygodact

You can set up a Zygodact installation in minutes with only three steps.

1. *Generate a set of stacks with one click.*

In Zygodact Setup, click the “Create Set” button. Zygodact will create a Key Generator stack and a Register stack. These must be used in tandem. The Key Generator creates serial keys that will work with the Register stack.

2. *Make Register a substack of your main stack.*

To embed the Register stack in your project, open both the Register stack and your main stack in LiveCode. Open the Property Inspector for the Register stack. In the General pane of the stack inspector, choose your main stack from the “Main stack” popup menu. Then save your main stack.

3. *Add one line of script to a preOpenStack or preOpenCard handler in your main stack.*

All of Zygodact’s functionality is automatic. The only thing you need to do is call it with a single line of script whenever you want to check the validity of a registration. Generally this will be immediately when your standalone opens, in a `preOpenStack` or `preOpenCard` handler, but it can be done any time you need to check the registration status.

To tell Zygodact to check registration, place this line in an appropriate handler in the main stack:

```
send "zygodact" to stack "register"
```

That's it. Everything else is taken care of for you.

## How Zygodact works

When Zygodact activates, it looks for a preferences stack. If one exists and contains a valid registration, Zygodact exits quietly and your mainstack script continues. Your users will never know it ran. If there is no preferences stack, or an existing one does not contain a valid registration, Zygodact displays a registration dialog.

When the user enters the correct name and serial key, the registration dialog closes and Zygodact writes valid registration data to the Preferences stack. If Preferences does not exist yet, Zygodact creates it. Alternately, you can tell Zygodact to use any existing preferences stack you have already created.

The registration dialog allows unlimited attempts to register, but will not allow your standalone to continue without a valid serial key. Without a valid key, the only option your user has is to click the Quit button, and your standalone will immediately terminate.

Zygodact strips out extra spaces and carriage returns before it processes the user entries, so you don’t need to worry about white space accidentally pasted into the dialog.

Zygodact does not force you to use its own Preferences. If you already have a preferences stack, you can pass an optional file path as a parameter when calling Zygodact. There's more on that below. If you do allow Zygodact to create its default Preferences, the stack will be named identically to your mainstack, with an extension or identifier added. Preferences will be saved to the OS-appropriate location on the hard drive.

### *Zygodact's default preference paths*

Assuming a mainstack named "MyApp", Zygodact's default names and preference paths are:

OS X:	~/Library/Preferences/MyApp Preferences
WinXP:	C:/Documents and Settings/<user>/Application Data/MyApp.pref
Vista, Win 7:	C:/Users/<user>/AppData/Roaming/MyApp.pref
Linux:	\$HOME/MyAppPref
Android:	documents/MyApp.pref
iOS:	documents/MyApp.pref

In all cases, the actual stack name (the stack's "short name") will be the name of your mainstack followed by "prefs" without any spaces: MyAppPrefs.

### *Register behavior during development*

When Register is part of a standalone, the Quit button will immediately shut down your application. During development you probably don't want this behavior. Register checks the environment when Quit is clicked, and during development it will simply close itself and return "Invalid registration" in the `result`.

This will not happen when Register is part of a standalone. Your users cannot bypass the registration requirement.

If you want to use Zygodact in a stack rather than in a standalone, your script should check the `result` after calling "zygodact" and take appropriate action when it receives notice of an invalid registration.

## **Invalidating a registration**

Moving or deleting Preferences, or attempting to manually edit it in a text editor, will cause Zygodact to assume there is no registration information, and it will display its registration dialog the next time your standalone launches. Copying Preferences to a different computer will also invalidate the registration, which means your users cannot share the preferences file in an attempt to bypass the registration requirement.

Unfortunately, like most other software, there is little you can do if a user decides to give a friend their name and serial key. If you want to track user registrations by having your standalone "call home" to an online database, you can implement that by using the

information that is returned to your script after a successful registration. See “For More Advanced Work” below for more about the data that is returned to your handler.

**IMPORTANT:** *Zygodact does not provide full protection on mobile devices. On Android and iOS, it is not possible to reliably lock the registration to a specific device. In general it will still work because by default the preferences file is not accessible on mobile devices. But if users have root privileges (a “jailbroken” device) the file can be discovered and shared with other rooted users. In some cases the shared file can unlock your app. Zygodact will be a good deterrent for the majority of mobile users but is not suitable for strong security on mobile devices.*

## Building a standalone with Zygodact

LiveCode’s standalone builder can’t check for automatic inclusions if a password-protected stack is part of the stack file. When you include Zygodact in your main stack, you will need to open Standalone Application Settings from the File menu and select the checkbox that allows you to select inclusions manually. If you leave the default auto-checking turned on, the standalone builder will exit with this error:

```
Unable to search for required inclusions because stack
Register is password protected.
```

## For more advanced work

### *Using your own Preferences stack*

Zygodact allows more control if you want it. An optional parameter allows you to pass a file path for a new or existing preferences stack, like this:

```
put "/Hard Drive/Folder/SubFolder/myApp.prf" into prefsPath
send "zygodact prefsPath" to stack "register"
```

Zygodact will create new preferences at this location if one does not yet exist. If there is already a stack at that location, Zygodact will add a single custom property to it and re-save it.

**NOTE:** *You must pass a complete path to the file in the parameter. If you pass only a folder path, the default location will be used instead.*

### *Data returned to your script*

After a successful registration, Zygodact returns three lines of information in the `result`: the user name, the serial key, and the internet date, like this:

```
Merry User
093364AA3F57D485D4753404
Tue, 12 Nov 2014 14:31:06 -0600
```

If you are using a custom identifier in your key (see “Using a custom identifier” in the Key Generator section) then the identifying character will be on the fourth line of the returned data, like this:

```
Merry User
093364AA3F57D485D4753404
Tue, 12 Nov 2020 14:31:06 -0600
B
```

To retrieve this information, check the `result` immediately after calling “zygodact”. For example:

```
send "zygodact" to stack "register"
put the result into userInfo
```

You can then use the information any way you want. For example, you may want to store the user name in Preferences so you can retrieve it later and show it in an About box. This is also a good time to grab the serial key if your standalone will need it later, since Zygodact can’t retrieve it from the preferences file. You’ll need to handle that yourself.

If your standalone contacts an online database for serial key verification, this will be the only opportunity to gather the data, so you should retrieve and store it at this time.

### *Internationalizing the error text*

It is easy to customize the Register labels for any language by simply changing their text. However, Zygodact displays an error to the user if an incorrect registration is entered. By default, the error text is “Incorrect name or registration key.”

If you prefer a different error message, you can customize it by passing a second parameter when calling Zygodact. (If you are not using a custom preferences stack, pass empty in the first parameter.) For example, the following will tell Zygodact to use its default preferences, and display Spanish error text to your users:

```
put "Nombre o número de registro es incorrecto." into prompt
send "zygodact empty,prompt" to stack "register"
```

Or to use your own preferences stack:

```
put "Nombre o número de registro es incorrecto." into prompt
put "/Hard Drive/Folder/SubFolder/myApp.prf" into prefsPath
send "zygodact prefsPath,prompt" to stack "register"
```



If the user enters an incorrect registration, your custom error will display.

### *Using Zygodact in combination with a free trial period*

Allowing a free trial period for your software is easy—simply do not call “zygodact” in your application until the free trial period has expired. One method for tracking a free trial period is to check for a preferences stack or a text file on launch. If no file exists yet, this is likely the first launch of the software and your script should create the file and store the current date in it. If the file does exist, read the stored date and calculate the number of days that have passed. If that number has been exceeded, call “zygodact” to show the registration dialog. If the user cannot provide a valid registration, Zygodact will exit any pending handlers and quit your software.

Some developers prefer to store the date file in a non-obvious location, such as in a hidden file inside one of the system folders. Others simply store the date in a preferences stack, often in encrypted form, where they store other user preferences as well. It’s up to you how you want to store and manage the date information.

The following example script looks for a hidden stack that has been saved in an obscure location. If no stack exists yet, the handler creates one and initializes it with the current time in seconds.

```
on checkTrialStatus
    put 30 into tTrialDays -- adjust trial days here
    put hiddenFilePath() into tHiddenStack -- where hidden data
    is
    if there is no stack tHiddenStack then
        create stack "HiddenInfo"
        -- probably a first launch; store the time:
        set the cStartSecs of stack "HiddenInfo" to the seconds
        set the filename of stack "HiddenInfo" to tHiddenStack
        set the destroystack of stack "HiddenInfo" to true
        save stack "HiddenInfo"
        if the platform = "win32" then -- hide it
            get shell("attrib +h" &&"& tHiddenStack &"&)
        end if
        close stack tHiddenStack
    end if
    -- get the first launch seconds:
    put the cStartSecs of stack tHiddenStack into tStartSecs
    -- convert to days; there are 86400 seconds in a day:
```

```

    put (the seconds - tStartSecs)/86400 into tDays
    if tDays > tTrialDays then
        -- trial is expired, call Zygodact for registration:
        send "zygodact tHiddenStack" to stack "register"
    end if
end checkTrialStatus

function hiddenFilePath
-- returns path to a hidden stack based on OS:
switch the platform
    case "MacOS"
        put specialFolderPath("preferences") & slash & \
            ".MyAppTrial" into tPath
        break
    case "win32"
        put specialFolderPath(26) & slash & \
            "MyAppTrial.dat" into tPath
        break
    default
        put $HOME & slash & ".MyAppTrial" into tPath
end switch
return tPath
end hiddenFilePath

```

### *Using Zygodact with a software expiration date*

Zygodact can accommodate software that is licensed for a period of time and then expires. In this case, your software should call Zygodact for a registration on first launch and store the registration date in the preferences stack, or in any other file of your choice. On each subsequent launch, your scripts should check the current date, calculate the number of days that have passed since registration, and call Zygodact again when the license period is expired. This is similar to the free trial example above.

When using Zygodact with software expiration schemes, you will need to provide a different user name for each new serial key if the key does not include a custom identifier. With no custom identifier, the same user name will generate the same key each time. One way to vary the user name is to provide customized IDs for each registration, rather than simply using their names. For example, rather than registering a user as “John Smith”, provide a randomized set of characters as a user ID, or add an identifier to the

name such as “John Smith 082”. Then each time he re-registers, he will need to enter the revised ID and the new serial key that was generated from that ID.

If your keys do include a custom identifier, the same name will produce different keys each time it is generated and you don't need to change the user name. For this reason, we recommend including a custom identifier in the serial key for software that expires.

Your software should check to make sure that a successful registration does not match a previous registration. To do this, your scripts need to store the original registration key in the preferences stack or another file of your choice. After the user re-registers, cross-check the new registration key with the old one. If they do not match, the user has successfully re-registered.

The following handler shows one way to manage a renewed registration:

```
on checkReReg
    put prefsPath() into tPrefsStack
    put the cOldReg of stack tPrefsStack into tOldReg
    put tOldReg into tNewReg -- match them temporarily
    repeat until tOldReg <> tNewReg
        send "zygodact tPrefsStack" to stack "register"
        put line 2 of the result into tNewReg
    end repeat
    -- if we get this far, new reg was entered successfully.
    -- store for next time:
    set the cOldReg of stack tPrefsStack to tNewReg
    save stack tPrefsStack
end checkReReg

function prefsPath
    -- returns path to preferences based on OS:
    switch the platform
        case "MacOS"
            put specialFolderPath("preferences") & slash & \
                "MyApp" && "Preferences" into tPath
            break
        case "win32"
            put specialFolderPath(26) & slash & \
                "MyApp" & ".pref" into tPath
            break
        default
```

```
        put $HOME & slash & "MyApp" & "Prefs" into tPath
    end switch
    return tPath
end prefsPath
```

You'll also want to reset the original stored date and time to the current date, to reflect the start of a new licensing period.

## Customizing the stacks

### *Customizing the Preferences stack*

Zygodact's default Preferences is created invisibly. The first time you open it you may need to choose it from LiveCode's Window menu to make it visible, or issue a command from the message box.

You can add to or customize Preferences however you like, and include whatever you need to support your main stack. Zygodact only uses Preferences to store a single custom property. The rest of the stack is yours to work with in any way.

### *Customizing the Register stack*

You can do any visual customization you like, though of course you cannot edit the scripts, which are locked for security. You can add images and logos, re-color objects, add informational text fields, or change the fonts and styling. Any change that does not require script access will work.

Protected stacks do not allow you to copy their objects. It's a good idea to design your layout first in a new, unprotected stack and use that as a backup. You can copy objects from the backup and paste them into future Register stacks if you ever need to generate a new set. It's also the only way to add new buttons or other scripted objects, since you can't edit scripts in the Register stack.

## Using the Key Generator

The Serial Key Generator is for use by the software distributor, and creates registration serial keys that work with its matching Register dialog. The keys it generates can be sent to customers so they can unlock their copy of your standalone. You can generate keys individually, or a whole list at once.

### *Generating a single key*

To create a single serial key, enter a user name in the top field. (A "user name" can be any

data you choose. Some developers prefer to use an email address since that is less likely to be shared indiscriminately.) Click the "Make One Key" button. The name and serial key are displayed in the bottom field, ready to copy and paste into an email. Users must enter this information exactly to complete a successful registration.

*NOTE: When registering, the name is case sensitive. The serial key is not.*

### *Generating multiple keys*

If you need to generate many keys at once, save the names in a text file, one name per line, and click the "Make Keys From List" button. The Key Generator will read the file, create a unique key for each name in the list, and store the information in a new text file at a location you choose.

You can customize the appearance of the Key Generator in the same ways you can customize Register. Only the scripts are locked.

## **Using a custom identifier**

Zygodact can create serial keys that contain a custom identifier which your script can use in any way you like, such as to unlock certain features or to distinguish a trial version from a paid version. The identifier can be any single character or integer from A-Z or 1-9. Enter the identifying character into the Custom field in the Key Generator before creating a key.

An identifier of zero, or an empty identifier, indicates there is no identifier in the serial key.

### *Serial key differences when using custom identifiers*

There are differences between the keys that Zygodact creates with and without a custom identifier. Keys with no identifier will match those created by 1.x versions of Zygodact and the same user name will always generate the same key. Keys containing a custom identifier will vary slightly each time the same name is regenerated. Keys with custom identifiers contain a random element that is rarely duplicated.

***IMPORTANT:** If you use custom identifiers, keep a record of the user's serial key because you won't be able to regenerate the same one twice.*

### *Retrieving an identifier from a key*

When a user registers your app, the identifier is returned as the last line in the result. Your app should store the identifier somewhere, such as in the preferences stack, so your scripts can retrieve it and use it to unlock features or perform other custom behaviors.

On the desktop, you can manually determine what identifier a serial key contains from within the Key Generator. Paste the key into the Name field in the Key Generator. The

identifier will appear in the Custom field.

The key must have been created with Zygodact 2.0. Keys created in earlier versions will not have an identifier and the Custom field will yield random results.

## Emailing a key in a form letter

The Key Generator can set up an email that contains the user name and serial key in the message body, based on a template file you provide. After you have created a serial key, click the Email Key button to generate a form letter. Zygodact will ask you to choose a template file to use. You can have any number of template files.

Since user names and email addresses are rarely the same, no attempt is made to fill out the email address and you'll need to add that manually. The generated form letter will open in your email client, ready to edit and send.

### *Template format*

Templates are plain text files stored on disk. Zygodact does a simple LiveCode merge to move the user name and key into the text of the template. The template should include these two merge variables:

```
[ [USERNAME] ]
```

```
[ [SERIAL] ]
```

Be sure to include the square brackets around the variable names. Here is a sample template:

```
Dear [ [USERNAME] ],

Thank you for your purchase. Enclosed is your serial key to
unlock the software.

Please copy and paste your username and license code into the
"Register" dialog which appears the first time you open the
program:

Username: [ [USERNAME] ]
Serial: [ [SERIAL] ]

Your user name is case sensitive.

If you have any questions please contact me@mydomain.com.
```

We appreciate your patronage!

The Management

## Compiling the Key Generator for use on a mobile device

You may find it convenient to compile the Key Generator stack into a standalone app for use on a mobile device. This lets you generate and email user keys while you are away from your desktop machine. Compiling requires LiveCode 7.0 or higher.

All the normal provisions apply when creating an iOS standalone: you must have a developer account with Apple and the correct certifications in place, and the device must be listed in the developer portal as a test device.

For Android, you can build with either the generic development key, or with your own signed key.

Some things to keep in mind when creating a Key Generator standalone:

1. Native (mobile) input fields will use the same textsize as their same-named LiveCode field. Changing the textsize of the field in the IDE lets you adjust how large the text will be on mobile. Android handles text sizing differently, so you may need to set the LiveCode field text to a smaller number, generally about half the normal size.
2. If you use email templates, they must be in a folder named “templates” (lower case) that is included in the Copy Files pane of the standalone builder. The Key Generator will look for this folder at the engine level of the standalone.
3. The mobile picker that allows you to choose a template also includes the option to use no template at all. If you choose “Key Code Only,” the user name and serial key will appear in the email without any other body text.
4. Emails are generated as plain text messages.

## Using the CGI Generator

Zygodact can optionally create a stack that works as a CGI on a server. To do that, be sure “Include CGI stack and script” is checked in Zygodact Setup before creating your registration stacks. The CGI Generator will create the same serial keys as the manual Key Generator. You can provide compatible serial keys for your standalone using either or both methods. You may want to generate a CGI stack even if you don’t have an immediate need for one. Since all stacks are created as a matched set, you won’t be able to make a separate, compatible one later; you’ll have to create a whole new set instead.

The CGI Generator (named “gen.lc”) and Zygodact’s universal CGI script are ready to use, and can be dropped into your server’s CGI directory without any alterations. You will need to set their access permissions on your server to 755. You will also need to have a

copy of the LiveCode engine version 6 or higher installed on your server (you don't need the IDE, only the Linux executable,) and an appropriate storefront, PHP script, or web interface that can send a query and receive a response.

The PHP script should send a request to the CGI script via stdout, with the user name supplied as the first parameter. If you are using a custom identifier, include it as the second parameter. The CGI script uses the parameters to calculate and return a serial key. The Zygodact CGI script will be suitable for most purposes, but if you want to write your own, your script must do at least these things:

1. Declare at least two global variables: `gName` and `gSerial`. If you use a custom identifier, also declare the global `gCustomChar`.
2. Parse the incoming parameters and put the user name into the `gName` global and, if needed, put the custom identifier into `gCustomChar`.
3. Put the `gen.lc` stack in use. When that happens, `gen.lc` calculates a serial key, and puts it into the global `gSerial`.
4. Read the global variable `gSerial` to retrieve the serial key.
5. put the serial key, which sends it to stdout.

The PHP script then reads the reply in stdin, which will contain the generated key.

### *The Zygodact universal script*

When you generate a CGI stack, Zygodact also creates a CGI script for use on your server. It works with PHP (or any method that interacts with stdin and stdout), as well as with HTTP POST or GET requests:

```
#!/livecode -ui

global gName,gSerial,gCustomChar
on startup
  put $REQUEST_METHOD into tMethod
  if tMethod is "GET" then
    put urlDecode($QUERY_STRING) into tStr
    set the itemDelimiter to "="
    put last item of tStr into gName
  else if tMethod is "POST" then
    read from stdin until empty
    put urlDecode(it) into gName
  else -- stdin
    put $1 into gName
```



```

end if
# If using a custom character, you can either pass it in
the above
# params and change the parsing, or hard-code one here:
# put "A" into gCustomChar
if tMethod is in "post,get"
then put "Content-Type: text/plain" & cr & cr -- add a
header
if gName = "" then
put "Error: name is missing."
exit startup
end if
if gCustomChar = "" then put "0" into gCustomChar --
default
start using stack "gen.lc" -- creates the serial key
if gSerial = ""
then put "Error: no data from generator." into gSerial
put gSerial
end startup

```

HyperActive Software has an online tutorial that explains how to use scripts and LiveCode stacks as CGIs. You can read more here:

<http://www.hyperactivesw.com/resources/cgitutorial/>

---

*Disclaimer:* Zygodact provides good registration protection for all LiveCode software; however, it is subject to the same limitations as any software protection scheme, and a determined hacker may be able to break into it. Zygodact will protect your software distribution for the majority of computer users, but if you require an extremely high level of protection or a customized registration scheme, do not use Zygodact. HyperActive Software is not responsible for any damages or loss you may incur through the use of Zygodact.

*License information:* Purchasers of Zygodact are allowed to use Zygodact to create any number of serial keys, key generators, CGI stacks, and registration stacks for all LiveCode software they develop. Users are not permitted to distribute the Zygodact utility itself to others, in any manner. Zygodact cannot be given away or shared; only one user is allowed per purchase.

*In short:* no decompiling, no hacking, no copying, no peeking, no distributing, or we let Guido out. Guido doesn't like you.