HyperActive software

Zygodact

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 Revolution 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 registeration dialog and 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

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 registration 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 Revolution. 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 an 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:	C:/Users/<user>/AppData/Roaming/MyApp.pref
Linux:	$HOME/MyAppPref

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 the 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. Consider it a compliment to your work. If you want to track user registrations by having your standalone "call home" to an online database, you can implement that by making use of the information that is returned to your script after a successful registration. See "For More Advanced Work" below for information about the data that is returned to your handler.

Building a standalone containing Zygodact

Revolution's standalone builder can't do automatic inclusion checking 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 check the box 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 file path in the parameter. If you pass only a folder path, the default locations 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
Mon, 15 Dec 2008 22:15:25 -0500
```

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.

Using Zygodact in combination with a free trial period

Allowing a free trial period for your software is easy—simply do not 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
      save stack "HiddenInfo"
      if the platform = "win32" then -- hide it
        get shell("attrib +h" &&quote& tHiddenStack &quote)
     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 accomodate 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 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. Since the same user name will generate the same key as before, the easiest way to do this is to provide customized IDs to your users 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 identifer 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.

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 re-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 Preferences stack

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

You can add your own data to Preferences at any time, including objects, images, custom properties, scripts, 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.

Preferences can display a custom application icon in the OS X Finder or in Windows Explorer. To do that on Windows, pass a preferences file path that includes a file name with an appropriate extension (your custom application extension will need to be registered with the Windows registry before it is recognized.) On Mac OS X, set the `stackfiletype` property in your script before calling "zygodact".

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 or 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.

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” does not have to be an actual name. It can be any data you choose.) Click the "Make One Key" button. The name and serial key are displayed in the bottom field, ready for you 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 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.rev”) 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 Revolution installed on your server, 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 a parameter. The CGI script uses the parameter to calculate and return a serial key.

The Zygodact CGI script will be suitable for almost all purposes, but if you want to write your own, your script must do at least these things:

1. Declare two global variables: `gName` and `gSerial`.
2. Parse the incoming parameters and put the user name into the `gName` global.
3. Put the gen.rev stack in use. When that happens, gen.rev 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:

```
#!revolution -ui

global gName,gSerial
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 tMethod is in "post,get"
   then put "Content-Type: text/plain" & cr & cr
   if gName = "" then
      put "Error: name is missing."
      exit startup
   end if
   start using stack "gen.rev"
   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

Revolution stacks as CGIs. You can read more here:

http://www.hyperactivesw.com/cgitutorial/index.html

Disclaimer: Zygodact provides solid registration protection for all Revolution 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 to 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 any Revolution 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.