The technical side of selling your classes and apps made with Real Studio

As presented at the Real Studio Conference
on May 22, 2011 in Essen, Germany

What this session covers

- Selling your code as encrypted classes
  - Understanding REAL Studio's encryption
  - Providing a clean API to your users
- Selling your applications
  - Preventing unauthorized use of your app
  - Using AquaticPrime for licensing
  - Payment processing
  - Enforcing the license
  - License installation
  - Foiling attempts to circumvent the license enforcement
  - Update mechanisms for your app

Selling your code as encrypted classes

The technical side

How safe is RS' encryption?

- The compiler must be able to decrypt it without knowing the password, which means that a good cracker can get to the encrypted source code.
- No public cracking tools known.
- If you are really worried about exposing your code, consider writing your code in C, then create a library or plugin from it.

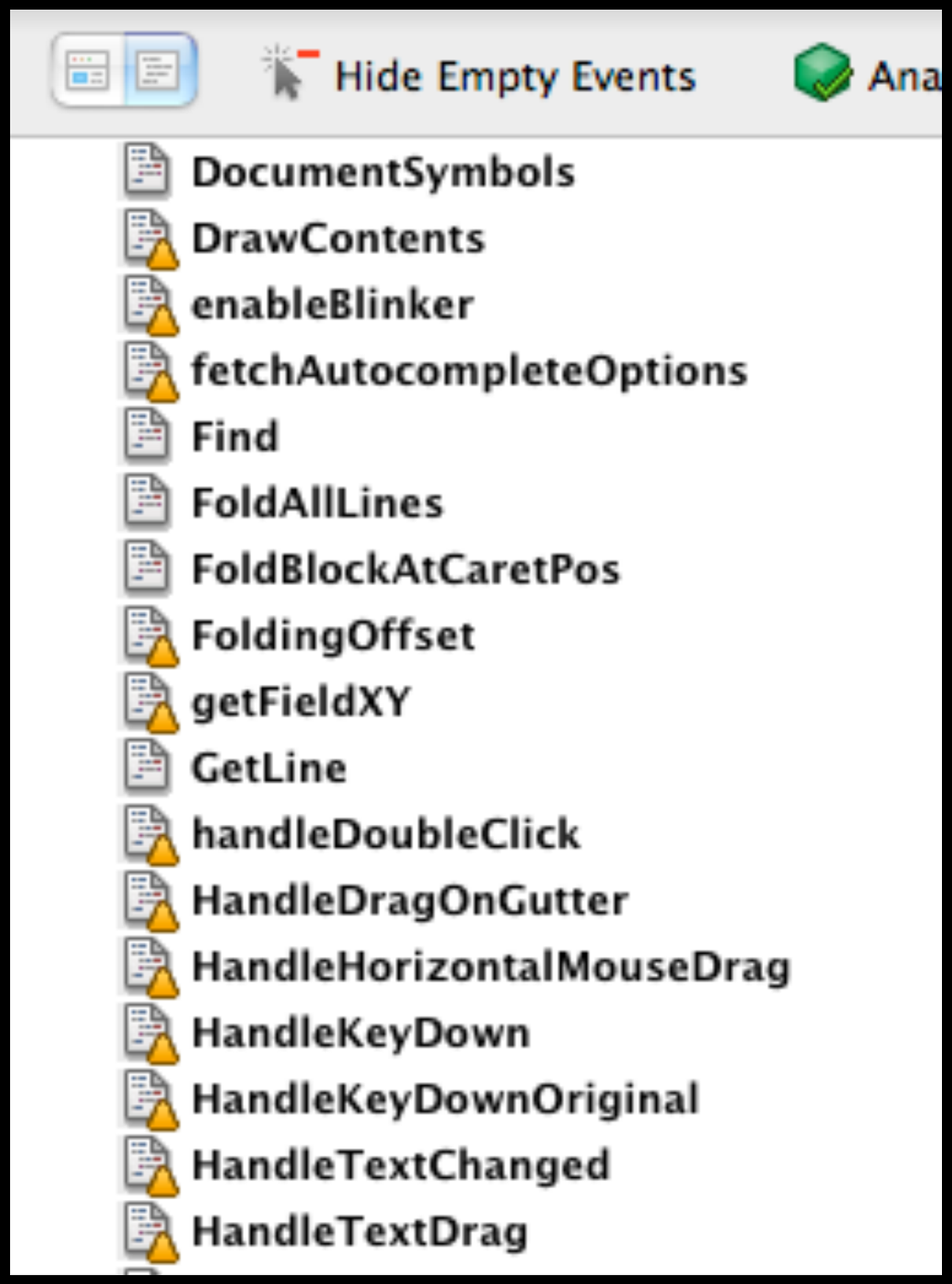
Caveats with encypted classes

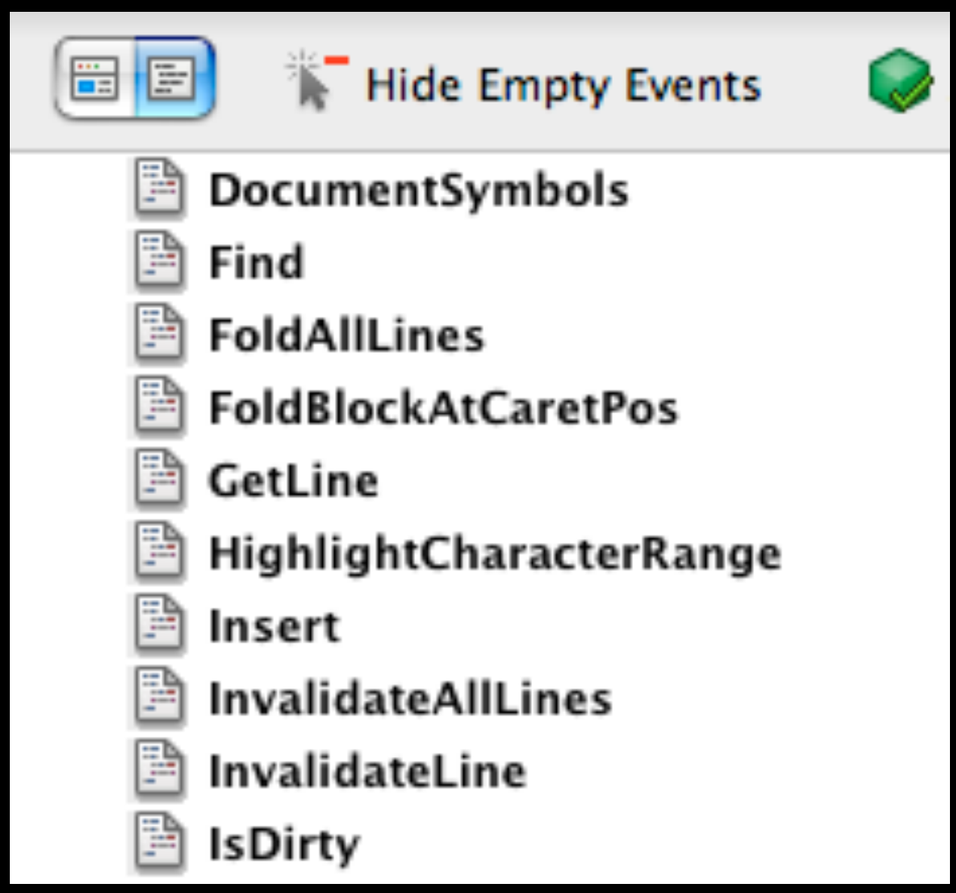
- While the compiler can read the source code, the IDE won’t show it to you when it reports a compile error.
  - If your code uses code that is not compilable with earlier or later REAL Studio versions, the user can’t really tell what’s going on.
  - Similarly, if your code conflicts with the user’s code, causing a name conflict, the user won’t get to see which name causes it.
- If there is an exception in the encrypted code, the debugger will show misleading information, making it difficult for the user to figure out that the problem is within the encrypted code and not in his own.
- The user can not even see the functions the classes are making available, meaning he has to rely on your documentation.

Best practises encrypting classes

- Consider giving the paying customer the decryption password so that he can avoid all of the aforementioned caveats.
- In any case, create a two-layer API:
  - 1. Put all your secret code into a private class, with no public but only protected and private members. Encrypt this class.
  - 2. Create a subclass of the encrypted class. This one shall only contain the exposed API, i.e. the functions the user can call, which in turn call the protected functions of the encrypted class. That way, the user can look at the method declarations of your class, even use code completion. This automatically makes your API self-documenting, too.
  - As an example, see my Zip classes at http://www.tempel.org/RB/ZipPackage

Best practises encrypting classes





Selling your applications

The technical side

The basics of selling an application

- To raise interest in your customer to buy your app, you have to offer a demo or a good promise that the user will be satisfied if he purchases it.
- If the customer wants to buy it, you need to be able to take his money. Usually this means that you will need a payment processor to handle the financial transaction.
- Lastly, the customer needs to get the product he paid for. Two ways:
  - The full-featured App is separate from the Demo version and is available only to paying customers, as a “secret” download that no one else gets to.
  - Alternatively, your App has two modes: A demo and a licensed mode. Meaning the user has to let the app know that he has purchased the right to the product features.

Payment processors

- All processors want a share of your sales. Here's a few popular ones I had a look at:
- Paypal
  - Lowest fees of all if you use just the simple "Pay now" buttons. No invoicing, limited post-processing.
- Kagi
  - Oldest processor. Has gotten a bit behind. Solid.
- eSellerate
  - Has processing plugin for REALbasic, though a bit outdated.
- FastSpring
  - Seems to be the youngest and freshest of them. Easy-to-use store setup.
- Apple's App Store. Expensive. Best exposure.

Using a licensing key scheme

- Having separate demo and full-featured apps is an option that I won't go into further here, as it is rather straight-forward from a technical standpoint.
- Here, I will show how to handle license keys, enabling features in a single application that can run as a demo if no key is added.
- A license key may be made up of a small set of character that the user has to enter, or be delivered as a file that the user has to then made known to the application.

Properties of a license key

- A key may contain the following information:
  - The buyer's name, company and/or e-mail address. If this appears in clear text either in the license file directly or is shown by the application, this might act as a deterrent for the person to freely share the key with others when he's not supposed to.

    (Consider the possibility that the key gets stolen from the customer without his knowledge. I suggest that you let the customer know in advance that his name will be on the license when they make the purchase.)
  - A code to unlock all or a set of features of the app.
  - A purchase date.
  - An expiration date.
  - A signature that proves that this license is valid.

The meaning of the license signature

- It is necesary to ensure that no one can generate their own license, enabling all features, without paying for them, of course.
- There are two ways to do this:
  - Either the entire license data gets encrypted in a way that is only known to you as the license generator, and to your program verifying it. REAL Studio does this, for instance.
  - Or the plainly visible data gets just signed with a digital signature, using public-key cryptography.
- This latter option, using a signature, based on an asymmetric key, is safer than the first option in the way that it can’t reveal to a cracker how to recreate a fake but valid license.

How asymmetric encryption works

- You, the publisher, keep a secret code, the “private key”, to create the signature. You will keep this key to yourself.
- The application contains another code, the “public key”, which it will use to check if the signature matches the data it has signed.

How asymmetric encryption works

- If someone wanted to create a fake license for your app, he would need to know how to sign the license so that the app validates it.
  - But for that to work, he'd have to know the "private key", which you keep secret and which won't appear in the application.
  - Therefore, as long as your own computers do not get comprimised, this is a pretty safe method.
- Of course, the cracker can modify your code to accept even a bad signature, but he can't produce new licenses to take over or damage your business that way.

The AquaticPrime license system

- AquaticPrime (AQ) is an open source implementation for generating, managing and testing digitally signed license key files.
- AQ comes with an application to create the public and private keys for a product, along with creating and archiving invidual license keys per purchase.
- This application is currently Mac-only, but there are also solutions for Windows and even PHP and Python, even though not as comfortable to use (yet).
- The other half of AQ is the code that validates a license. This is available for many languages, including REALbasic, both for OS X and Windows applications.

Generating the licenses

- While AQ offers ways to generate license files for your customers, there is an even easier way:
- Some payment processors offer AQ license generation on their servers, saving you from the need of setting up your own server to provide the licenses.
- Of all the ones I looked at, FastSpring offered the most flexible way to generate licenses, letting me add name, company, e-mail, purchase date and even a free form value per product. All I need to do is upload the once-generated private and public keys, and the service takes care of the rest.
- eSellerate offers a similar AQ generation service, but I didn’t follow through with it as it seemed less capable.
- Kagi announced AQ support, too, but wasn’t ready yet.

Getting your app to accept a license

- Assuming you deliver a license key to the customer after his purchase, the customer needs to “enter” it into the application. In case of AQ files, the best way is to let the app open the file and then copy its data to a safe place.
- I recommend to use a dedicated file extension for your license files so that the user can simply double click the file, which will then launch your application so that it can process it. It can be long, e.g. “.programname-license”.
- Registering a file extension on both OSX and Windows is challenging since REAL Studio doesn’t automate this yet, even though it supposedly does.
- First off, you need to add a FileType to the project, setting an icon, the extension and a UTI (*com.yourdomain.appname.license*).

Registering an extension on OS X

- For OS X to recognize the file extension mapping for your application, you need to add some data to the Info.plist file until REAL Studio does this on its own (<feedback://showreport?report_id=15933>).
- To add the information, first use a text editor to add it directly to a built version of your app's Info.plist.
- To test if your modification to Info.plist worked, use my free tool "Launch Services": Drop your modified app onto the tool (this updates the LS database), then enter your extension into the lower field and click "Find App". If this reveals your app, you're set.
- Once it works, you can use /usr/libexec/PlistBuddy (installed with Xcode) to add the data automatically, e.g. by using IDE Scripting or Build Automation.

Registering an extension on OS X

```
<key>UTExportedTypeDeclarations</key>
  <array>
     <dict>
        <key>UTTypeConformsTo</key>
        <array>
           <string>public.data</string>
        </array>
        <key>UTTypeIdentifier</key>
        <string>com.yourdomain.appname.licensefile</string>
        <key>UTTypeTagSpecification</key>
        <dict>
           <key>public.filename-extension</key>
           <array>
              <string>the-extension-name</string>
           </array>
        </dict>
    </dict>
</array>
```

Associate an extension on Windows

- On Windows, you either need to use an Installer that sets up the file extension mapping or your can do this in your own app - which requires that the user runs your app once before trying to double click the license file, though.
- To associate the file extension, you have to add it to the Registry. To get this right for both Windows XP and Windows 7, I suggest to try to add it first under HKEY_CLASSES_ROOT, and if that fails (which is likely on Win7), under HKEY_CURRENT_USER/SOFTWARE/Classes.
- Sample code for this is a bit long, download link to follow at end of session.

Accepting the license in your app

- Once your app can open the license file (make sure the user can also simply drop the file onto the app, or even use the File -> Open menu command for this), it should be validated and copied to a safe place.
- Validation is handled on the upcoming slides.
- After validation, I recommend copying the license file to a folder that is owned by the user. For example, get the folder for application data by calling *SpecialFolder.ApplicationData*, then create a folder named after your application in it, and use that as the location for the license and other data the app wants to store. I.e. do not place such files into the Documents folder where only files the user explicitly handles are supposed to be stored.
- When your app starts, look for the license there to validate it.

Validating the license

- The AquaticPrime class has an easy-to-use API:
  - Create a new AQ instance by passing the public key to it.
  - If you have any blacklisted keys, you can supply them via its the AddToBlacklist method.
  - Then call its DictionaryForLicenseFile function, passing it the FolderItem reference of the license file.
  - If the license is valid, the function returns a dictionary containing all the values from the license file, minus the signature. You can then read the customer's name and/or e-mail address to show that in a window as necessary, and read the purchase date and other data from it as you see fit.
  - If the function returns nil, it means that the license is not valid and the app can enter demo mode or tell the user that the license he has installed is not valid.

Blacklisting

- Individual licenses may still get compromised:
  - Someone could deliberately make a purchase under a false identity and then spread the license, not caring about detection.
  - A honest customer could get his license stolen and abused.
- In these cases, you actively need to block these licenses from being accepted by your app. Your options:
  - Add these license keys to your new releases of your software, passing them as Blacklisted to the AquaticPrime class when validating a license so that they get rejected.
  - Consider letting your app “call home”, getting the blacklist regularly form your server, so that even current versions of your app will learn about compromised keys and stop accepting them.

Security measures against cracking

- While there's little chance someone can create fake licenses that your app would accept, it's rather easy for a skilled programmer to patch your application so that it will accept any license as valid. A few scenarios:
  - Since the license data is not hidden, all he has to do is find the code that checks if the signature is valid, and get that result ignored, returning always the dictionary with the values from the license.
  - If he finds your hard coded blacklist, he can erase the blacklist without even having to have programming skills if the blacklisted keys are found by a search with a hex editor.
  - Similarly, if your public key is easily found, it can be replaced by his own key so that he can then create valid licenses for this patched version of your app.

Security measures (cont.)

- Fact: You can not prevent a good and dedicated cracker from disabling your license checks.
- You can, however, make this not too easy for the casual cracker. And if you're lucky, he gives up as soon as he sees you're not making the simplest mistakes, and so he has no idea how sophisticated it'll get and how much time he may have to waste on it, with the possibility to not even ever succeed because it's above his skills.
- Do not be tempted to tease the cracker should you discover that he's patching your code. That will only tackle his pride, making it less likely that he gives up.
- Be subtle. Add code that detects modification to your app and then destroy small but important things in the app so that the app does react only later to the crack, misguiding the cracker and frustrating him.

Security measures (cont.)

- Here are a few ideas:
  - Do not store the blacklist nor the public key in their original value. Make an effort to obscure and fragment them so that a simple replace of the entire key is not possible.
  - Invoke the AquaticPrime class with different licenses that you know to be both valid and invalid, and test if they turn out that way. If a cracker just changes the code so that it always tests positive, a false test will reveal his hack. You can even go so far as to use another public key to see how that behaves.
  - Basically, always perform multiple tests, both positive and negative, from different places in your code.
  - If you detect that something is wrong, change the state of your app, possibly something that will soon lead to crash or lock it up.

Further considerations

- Consider that some markets require localization, e.g. Japan. German, French, Spanish markets also expect translated applications.
- For Mac OS X, the App Store may be a good deal despite its high fees. The upside is that is give an app good exposure and that Apple takes care of the financial and licensing tasks. All you need to do is some code to validate the license, which can be found in the CertTools module of the open source MacOSLib.
- Your application should have a way to tell the user if an update is available. Consider Sparkle, with good support for OS X and basic support for Windows and Linux. There's also the MBS Update Kit, which offers a more complete solution for Windows including installer invocation.

References

http://www.tempel.org/RB/Realcon2011

- AquaticPrime: http://github.com/bdrister/AquaticPrime
- “Launch Services” tool (with source code):
  http://files.tempel.org/RB/LaunchServices.rbp.zip
- Windows code to associate a file extension:
  http://files.tempel.org/RB/WindowsOS.rbbas.zip
- Localization: http://www.tempel.org/RB/Localization
- Mac App Store: http://www.tempel.org/RB/AppStoreGuide
- Updaters:
  - Sparkle:
    http://www.declaresub.com/article/67/sparkle-for-realbasic
  - RBSparkle: http://www.tempel.org/RB/Sparkle
  - MBS Kit:
    http://www.monkeybreadsoftware.de/realbasic/UpdaterKit/

The END

Thomas Tempelmann
tempelmann@gmail.com
Twitter: tempelorg
http://www.tempel.org/RB

More up-to-date (Sep 2011) info here:
http://www.tempel.org/UsingAquaticPrime