Discovering a Hidden iPhone URL Scheme

Adrian Kosmaczewski

2009-08-04

As an iPhone developer, one of the simplest and easiest mechanisms you have to interact with other applications is through the use of iPhone URL Schemes. These are so important that I've created a wiki page where I keep track of those I come across, including code samples that help me exchange data with them.

However, not all editors document the URL schemes they support in their apps, and this blocks reuse and collaboration. I recently came into such a problem, trying to use TwitterFon from my own apps, to post messages to Twitter. The TwitterFon site only specifies the following iPhone URL scheme:

twitterfon:///post?this%20is%20a%20test

The problem is, this URL scheme does not perform an URL-decoding on the message parameter, which means that a phrase like "this is a test" will appear in TwitterFon URL-encoded, that is, as "this%20is%20a%20test". Clearly not acceptable.

However, thanks to Ashley Mills, I learnt that the USA Today iPhone app is able to use TwitterFon to share articles via Twitter, and does this properly, without URL-encoded characters. How do they do that? Obviously, they are using an URL scheme exported by TwitterFon, but not documented anywhere (*). I finally discovered that the URL scheme sought is the following ("message" instead of "post"!):

twitterfon:///message?some%20text%20here

This is how I found out: I impersonated TwitterFon in my own iPhone with an ad-hoc app created in Xcode, that shows me the URL used by USA Today to launch TwitterFon.

These are the steps required:

- Open iTunes and look for the application whose URL schemes you're interested in (in my case, TwitterFon Pro); right click on it and select "Show in Finder";
- Duplicate the .ipa file in the Finder and change its extension to .zip
  - yes, .ipa applications are simply compressed .zip files;

- Uncompress the .zip file and open the folder; inside, navigate to the "Payload" folder, and right-click on the .app file inside; select "Show Package Contents";
- Browse inside the package and open the Info.plist file; inside, you'll find two keys that are interesting for us: CFBundleURLTypes ("URL types") and CFBundleIdentifier ("Bundle identifier"). Select them and copy them to your clipboard;
- Create a new Xcode application, using the "iPhone OS / View-based application" template;
- Open the Info.plist file corresponding to the default target and remove the existing CFBundleIdentifier ("Bundle identifier") key; paste the two items you've copied in the previous step - this means we're creating an application that will "impersonate" itself as the "real" one;
- Modify the new Xcode project's app delegate adding the following method:

```
- (BOOL)application:(UIApplication *)application
      handleOpenURL:(NSURL *)url
{
    viewController.viewer.text = [url absoluteString];
    return YES;
}
```

- Add a UITextView to your viewController (you can do this easily in Interface Builder, editing the .xib file), and expose it through a public property (called "viewer" above) so that the app delegate can access it;
- Prepare your Xcode project for ad-hoc deployment: add a "distribution" configuration, an entitlements.plist file, etc;
- Plug your iPhone, select "Distribution / iPhone OS Device", and "Run" your application; the application will build and Xcode will install it into your device. ATTENTION: this application will overwrite the original one! This is because it has the same bundle identifier. Be sure to backup your data before doing this, as it will be lost completely;
- Now run the application calling the one you impersonate (in this case, the USA Today one) and force it to call the "impersonated" application (in this case, by "sharing" an article via Twitter). This will trigger the launch of the impersonated application, the call to application:handleOpenURL:, which itself will display the calling URL on the iPhone screen.

You can download a zip file with the Xcode project created above if you want. Be careful if you run it on your own device!

Voilà! Finally, delete your own impersonated app, go to the App Store, reinstall the application you've impersonated (normally it's a free download, even for non-free apps), and you are done. The same mechanism could be used to find out similar, hidden URL mechanisms in other apps.

(*) Actually this URL scheme is only shortly mentioned in the changelog of the application…