Appcelerator Titanium

Create an explosive iPad game

Appcelerator Titanium is a fantastically versatile tool for creating mobile apps, as **Anton Mills** demonstrates with this game for the iPad

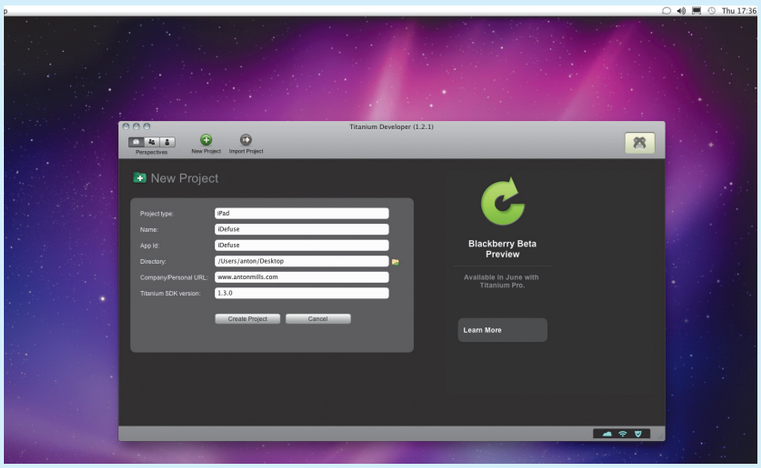
When creating a mobile application, it's not always easy to decide which platform to choose. Do you select Apple's iPhone or iPad? A BlackBerry? Or maybe Google's Android platform? Why not make it for all of them? Appcelerator's Titanium software is so versatile it enables us to create applications rapidly using JavaScript, which can then be published to any of the above platforms. Featuring geo-location/mapping, accelerometer, multi-touch and a whole lot more, Appcelerator's a great asset to have in our arsenal for the next client pitch.

In this tutorial we will create a small game for the hottest new mobile technology, the iPad. We will learn how to set up a Titanium project, the fundamentals of mobile app creation and how we test our projects in an iPad simulator as we build a basic bomb-defusing game. Download the files from this issue's disc, then once you have the iPhone SDK (free but requires registration: http://developer.apple.com/iphone) and Appcelerator Titanium (free from www.appcelerator.com) installed, we're ready to go!

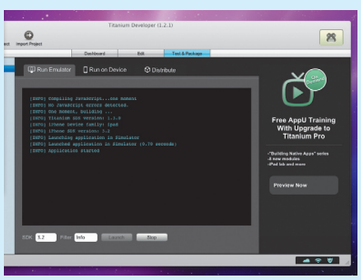
Anton Mills
A freelance interactive developer based in Cardiff, Anton Mills works primarily with Flash, indulging his new love of mobile development for iPhone, iPad and Android. Find out more about him and his work at www.antonmills.com

Time needed
1 hour

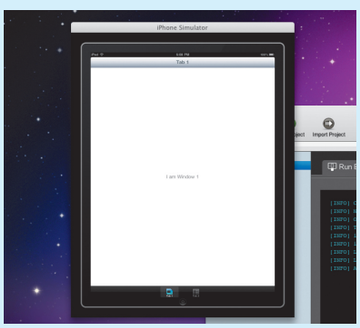
Skills
- Create a Titanium project
- Learn the fundamentals of the Titanium workflow
- Basic JavaScript usage
- Compile and test Titanium apps
- Simulate an iPad



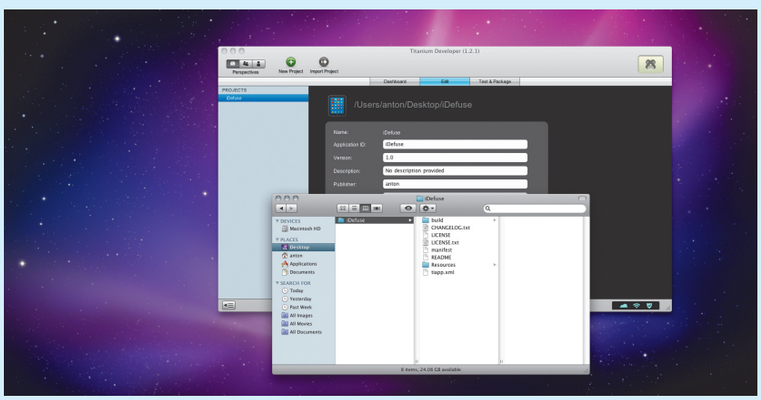
01 With Titanium running, we will need to create our project, which we'll call iDefuse. Do this by clicking the New Project button, which is shown as a green plus sign in the title bar. Set the Project Type to 'iPad' and enter 'iDefuse' in the Name and App Id fields.



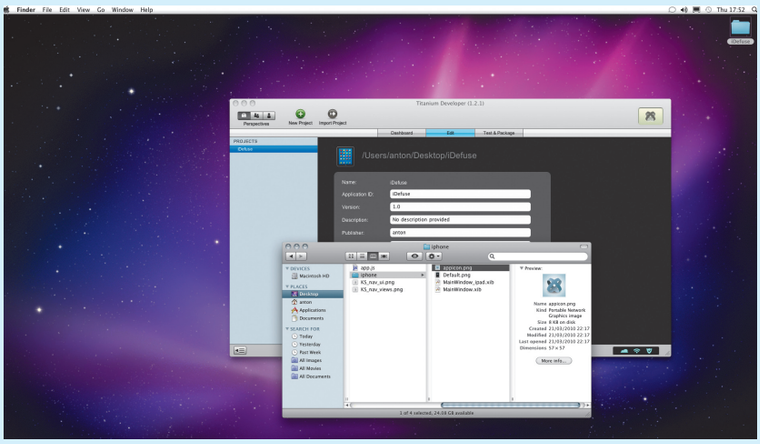
02 With the project successfully created, our next step is to test our application in the iPad simulator. To do this we have to press the Test and Package button just below the title bar, and click Launch. The iPad simulator will launch and start our application as it currently stands.



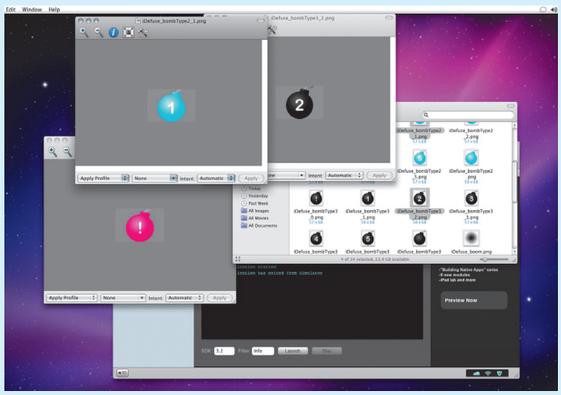
03 If all that went smoothly, you should have the iPad simulator up and running with the default Titanium project. It's a simple demo that has two tab groups, which you can switch between. We will change this later but, for now, let's have a look at Titanium's folder structure.



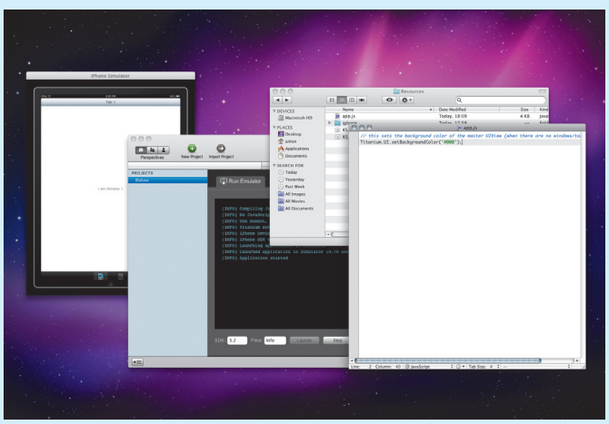
04 A quick inspection of our newly created iDefuse folder shows that the standard Titanium project structure contains a Resources folder and a Build folder. The Build folder will hold the compiled application and the Resources folder is where all of our work will be housed.



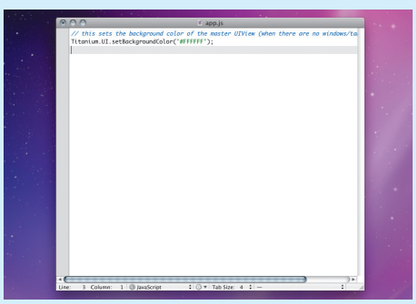
05____Inside the 'Resources' folder is a file named app.js – which is the main JavaScript file for Titanium applications – and a folder named 'iphone'. The 'iphone' folder contains resources for the iPhone/iPad platform, and in this folder you can see an app icon and a start-up screen called Default.png.



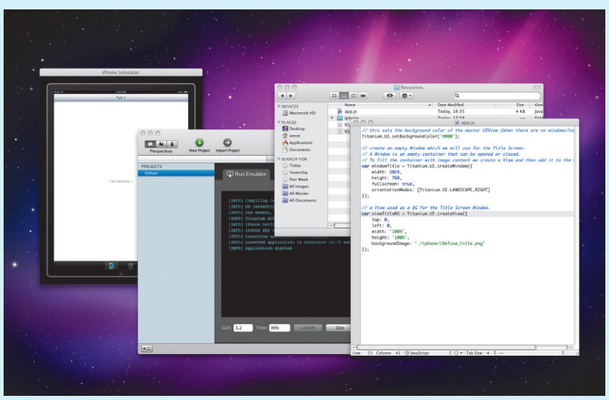
06____I've already created the game graphics using Photoshop, but you can use any design package that supports JPEG or PNG. The images are located inside the 'Step 6 images' folder on this issue's disc. Copy all of these images and paste them into the Resources\iphone folder of your project.



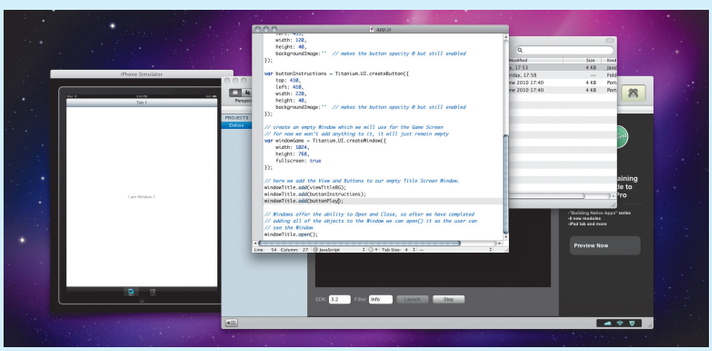
07____With the images now included in the project, it's time to begin working with JavaScript. Open app.js in your favourite JavaScript editor. You'll notice that it contains all of the default code for the default application we saw in Step 3. Delete everything apart from the top two lines.



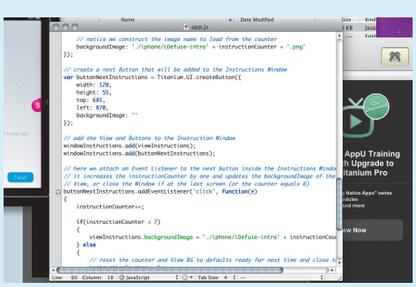
08____To save a lot of typing, we have created an app.js for each significant step. You can also find these inside the 'Step 6 images' folder on the disc. The comments provide a detailed explanation of each line and are worth reading as we piece together the game.



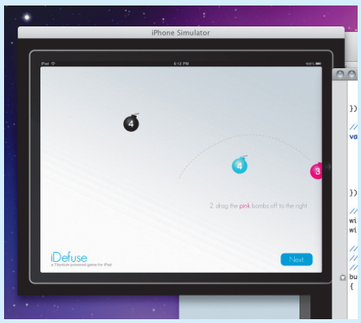
09____Copy this step's app.js and overwrite the old one. We begin by creating a window (an empty screen that we'll populate with images and interface elements). We then create a view, which is a visual object that we can style – in this case our Title Screen image.



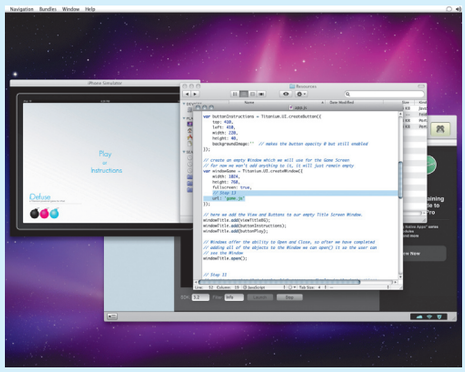
10____Now create two buttons on the Title Screen window that will link to the Instructions window and the Game window. Create another window for the Game screen, which we will work with later. Finally we set about adding the views and buttons to the Title Screen window.



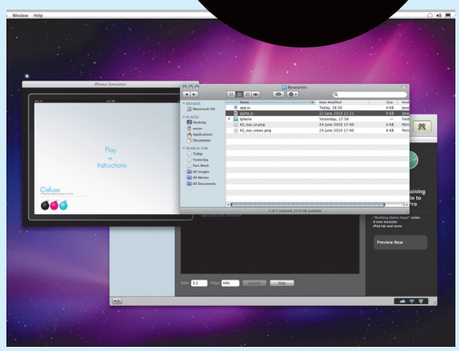
11____Copy this step's app.js and have a look at the code. I've created an Instructions screen and attached event listeners to the buttons so that, when they are clicked, they either open the Game window or the Instructions window. The comments here explain how this works. →



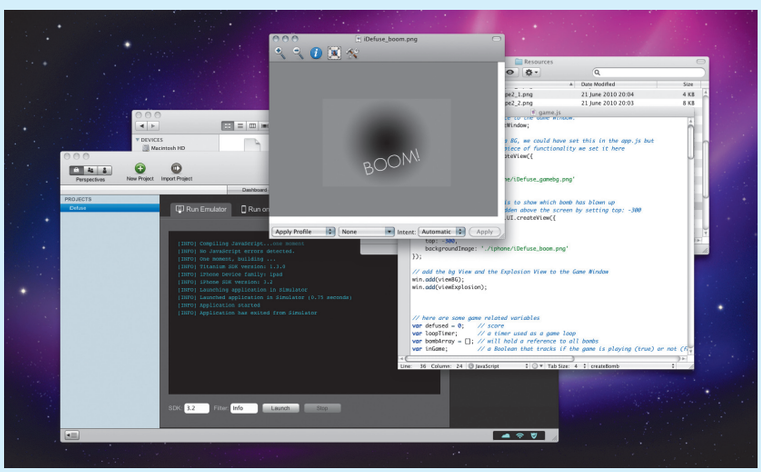
12____Compile your application as you did previously by pressing 'Launch' at the Test and Compile screen. You should have a fully functioning Title screen that enables you to test the Instructions screen. In the remaining steps, we'll explore how to get the game itself up and running.



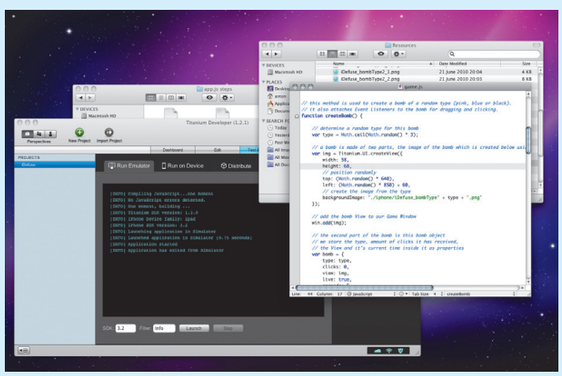
13____As there is a lot of logic for the game, it could clutter up our main app.js. To avoid this, we can separate different parts of an application to different JavaScript files using the URL property. See the revised app.js to see how to add this property to a window.



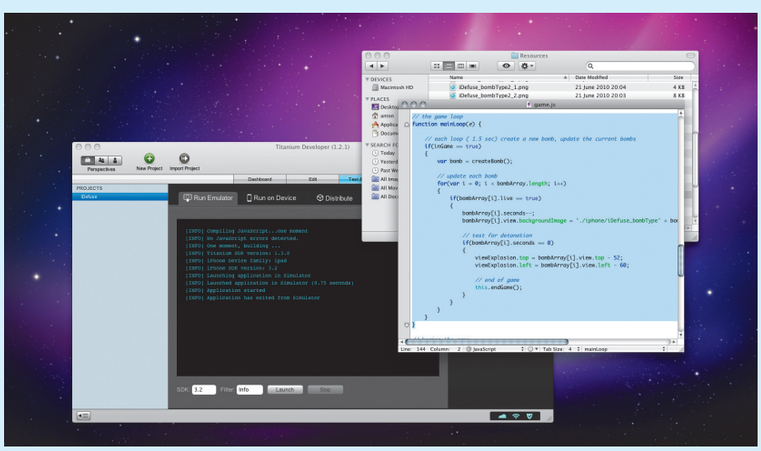
14____Be sure to copy the game.js from the 'Step 6 images' folder on the *Computer Arts* disc to the folder of your project containing the app.js. The game.js file is read when we create the Game window, but only begins the game when the window is opened.



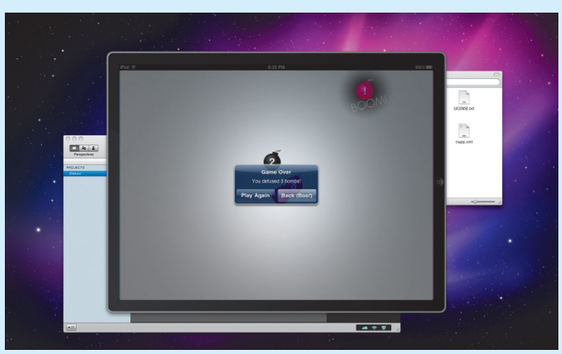
15____Inside the game.js, the first few lines deal with creating another view for the background image, creating an explosion graphic and some miscellaneous variables for the game. They are then added to the reference to the Game window called 'win'.



16____This step adds a createBomb() function. Calling it gives us a random type of bomb added to the screen. The bomb also has event listeners automatically attached to it that perform certain game functionality, such as clicking three times to defuse, or dragging it off the screen.



17____Now we have a way to create bombs, we need to create a game loop. Each time this loop runs it will need to create a new bomb by calling the function from the previous step. The main loop also needs to update all current bombs on screen.



18____The final step in our iPad game is create a Begin function that resets the game data and uses JavaScript's setInterval() to run the main loop from Step 17. Lastly it needs an End Game function to show an alert to the player with the score – and that's it!