Bottlenose developer’s guide

Version 0.1a, 2005-09-15.

Introduction

Bottlenose is a system to enable HTML/CSS/Javascript (”JHC”) designers to create the user interface (UI) for native applications.

A **native application** (as opposed to a web app, ASP, applet, widget, etc.) is just a regular application that runs on the local machine, such as a word processor or a graphics program. It is written in a native programming system such as C++, Objective-C, Java, C, etc.

Traditionally the native developer would write the UI using some kind of integrated development environment (IDE) such as Visual C++, Interface Builder, Eclipse, etc. (with a proprietary visual design tool), or even write the UI programmatically using native code.

- In the traditional workflow, a graphic designer creates a “mock up” of the interface in a program like PhotoShop. Then the native developer implements that UI by hand in the IDE visual tool or by hand programming. Sometimes the developer creates a UI without the aid of a graphic designer.
- Some new tools like Konfabulator and Apple’s Dashboard allow a “JHC” designer to create small applications using just HTML, CSS and Javascript. There is usually a significant overlap between graphic designers and JHC designers. Bottlenose extends that vision to include native applications written in native languages.
- In the “new style” workflow using Bottlenose the native developer and the JHC designer split the work. The native developer is responsible for the **native view**, which takes over a portion of the screen. The native view displays fixed and real-time graphics, data visualizations, and the like.

Then, the native programmer passes off the work to the JHC designer, who is responsible for the **web view**, which occupies the rest of the screen. All of the controls and widgets for the application are done in the web view, as well as any text, help information, status messages, etc. These are all written in standard HTML, CSS, and Javascript.

Since most graphic designers are familiar with HTML tools, this new style allows the graphic designer to not only “mock up” the interface but also implement most of it themselves.

The work flow can even be split three ways: a native developer, a Javascript developer, and an HTML/CSS/graphic designer.

A little aside: bottlenose dolphins

Do you know much about bottlenose dolphins? Here are some facts about these amazingly speedy and intelligent creatures, who might be intelligent enough to communicate with us, if only we knew how.

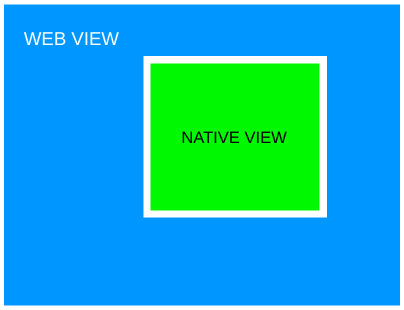
- **Long childhood, long life** - dolphins don't mature until age 10, and live in the wild to an average age of 25 years, and on up to 40 or 50 (Wells 1990)
- **Speed** - cruising speed of 10 km/h all day long, with burst speed of 40 km/h (Rohr 1998)
- **Names** - dolphins have "signature whistles" which function as names (Sayigh 1990)
- **Mirror self-recognition** - dolphins can tell who they are in a mirror, unlike most animals (SciAm 2001)
- **Superior communications** - although no one knows yet if it's a language, dolphins communicate with a very complex system of chirps, whistles and clicks which they generate using four separate sound generators in their head (Howlett 1997, etc).

Bottlenose dolphin *(tursiops truncatus)*
(Photo Viviane Frost, WikiMedia Commons, GFDL 1.2)

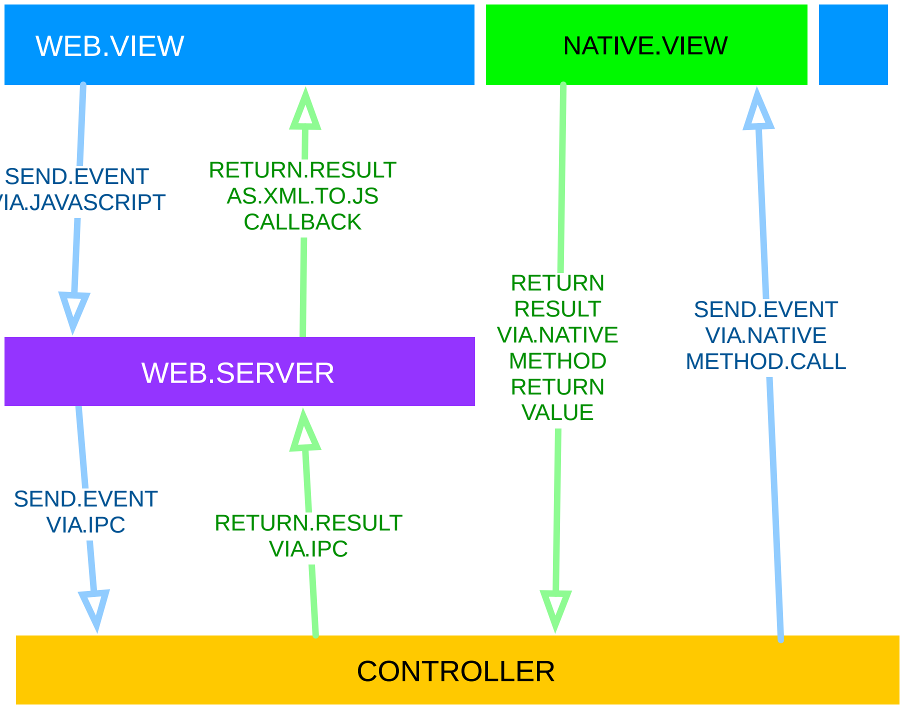
Diagrams

The following diagrams show the relationship between the different components of a Bottlenose-based application.

Note: even though it says “web server” all of these components are inside a single application bundle. Bottlenose uses web tools and conventions but it doesn’t require an internet connection. Everything runs within the application.



THE SCREEN OF A BOTTLENOSE APPLICATION


WEB.VIEW
NATIVE.VIEW
SEND.EVENT VIA.JAVASCRIPT
RETURN.RESULT AS.XML.TO.JS CALLBACK
RETURN RESULT VIA.NATIVE METHOD RETURN VALUE
SEND.EVENT VIA.NATIVE METHOD.CALL
WEB.SERVER
SEND.EVENT VIA.IPC
RETURN.RESULT VIA.IPC
CONTROLLER

Note:.All.of.these.components.are.inside.a.single.application.bundle.
COMMUNICATIONS.FLOW.CHART.FOR.BOTTLENOSE.DEVELOPERS

HTML / CSS / Javascript designers

It's easy for HTML/CSS/Javascript ("JHC") designers to develop the user interface (UI) for a Bottlenose-based application. All of the HTML/CSS/JS components of the UI are located inside the application bundle. You can access them by control-clicking on the application and choosing `Show Package Contents`. Then open the folders `Contents` > `Resources`.

The `index.html` file is the main user interface for the app. It's a regular HTML file that you can edit in any HTML editor. In addition you can use javascript and CSS to design the interface.

Use iframe to display the native view

The only special part of the .html file is that it must contain an `<iframe>` element with the `src` set to a `*.bottleview` file. The native application view will appear inside the iframe when you run the application. You can use HTML and CSS to style the iframe as you like, although the contents will be replaced with the native view, so setting a background colour, for example, will be ignored. There should be only one iframe.

Once you define the iframe, the application will automatically fill it with the native view when you run it. The inside of the iframe is the only part of the application's UI that you, as the JHC designer, don't control. You should coordinate with the native view developer to ensure that the look and feel of the native view contents match what you are developing for the HTML interface.

Sending events to the native view

Use javascript to send an event to the native view, and optionally to receive back a response or return value.

Bottlenose uses asynchronous javascript calls to send events to the native view. Asynchronous means that your call and the result will take place independently of the normal flow of HTML file loads (as you might say, in the background). There is plenty of documentation available on the web about asynchronous javascript, and it's often called the "AJAX" model.

To make this work, when the user manipulates an HTML widget, your `onblur` handler (or other javascript handler) calls a javascript function which sends an asynchronous request to the native server. The request contains the name of the event, and any parameters in the form of a "dictionary" or hashmap. A dictionary is simply a list of key/value pairs. Each key is the name of a parameter that you need to send with the event, and each value is the quantity for that key. Bottlenose makes it easy to send events with the send_event() function. Provide the event name as the first parameter. Then, provide a pair of parameters for each key/value that you need to send in.

For example, `send_event( “startGame”, “player”, “Alfonso” )`would send an event named "startGame" to the native view, telling it that the "player" parameter should be "Alfonso".

Receiving an event response and updating the UI
Then you write a "callback" function for each event. After sending the event, some small amount of time will pass, and then the native server will pass back the results to your javascript code in the form of an asynchronous callback. The result comes back as XML but Bottlenose (bottlenose.js) provides convenience functions to access the results easily without needing to understand XML.

Your callback function can do anything you like but it must be named exactly as follows: the word "`event`", and underscore character ("`_`"), and then the name of the event. And it must take a single parameter, which you should call "`r`" (for response).

For example, `function event_startGame(r)`

Inside the function you can do whatever you like to update the UI based on the response. Commonly the first thing you would do is to verify that the event succeeded. You can always test this by calling `get_key(r, kBottleKeyResponseCode).` A value of `1` (`kBottleEventOK`) tells you that the event was successful. You can also get other return values using the `get_key()` function, even ones that are custom to the specific event. Each return value has it's own name, and you get it by calling `get_key(r, "myReturnValueName")`.

Once you have written the skeleton of your callback function and tested that the response code is successful, you should use javascript to update the user interface as appropriate. This is how your user gets feedback after they manipulate a widget to know if it worked or not, and what happened.

The index.bottleview file
Also inside the resources directory of the application bundle, there must be a `index.bottleview` file. This file contains some configuration data for Bottlenose to use when it sets up the native view. Currently, the file just contains the viewName, and it's formatted as a Mac OS X PLIST file. Don't change the view name unless you're instructed to do so by the native programmer. The view name will determine which view class Bottlenose loads on the native side.

The name of the file can be anything, but it has to have the extension `.bottleview` and match the `src` link in the HTML document's `<iframe>` tag.

The bottlenose.js file
This file contains critical Javascript code and provides you with easy-to-use functions such as `send_event()` and `get_key()` (described above). Normally you won't want to change this file. It has inline documentation (in comments in the source) which you can read for more details.

The *.js and *-callbacks.js files
Normally you would create at least these two javascript files to control your UI. The `*.js` file would contain your functions and variables. It's a good idea to collect all of the

callback functions into a file called `*-callbacks.js` (although this is not strictly necessary).

Other files
Feel free to add more files to the document root/resources directory inside the application bundle. Images, CSS files, etc., that you need to link into your HTML are all fine. You can use relative links or absolute links — the resources directory is `/` - the document root.