Fox on the Run

Developing Native Mobile Apps that use your existing data and code

Eric Selje
Geeks and Gurus, Inc.
Madison, WI
Voice: 608/213-9567
Email: ejselje@geeksandgurus.com
Twitter: EricSelje

You've got a large investment in your Visual FoxPro infrastructure, and you're getting pressure to make that available on mobile devices. How do you get started? This session will show you one path to getting a native mobile application up and running fairly quickly and cheaply, without rewriting everything or migrating your data.

Fox on the Run

Those of us still using Visual FoxPro have accumulated a massive amount of code and data using our beloved product. From its origins as FoxBase in 16-bit DOS all the way through 64-bit Windows 7, our applications continue to perform. As long as Microsoft's operating systems continue to be the primary choice for businesses, we have no worries.

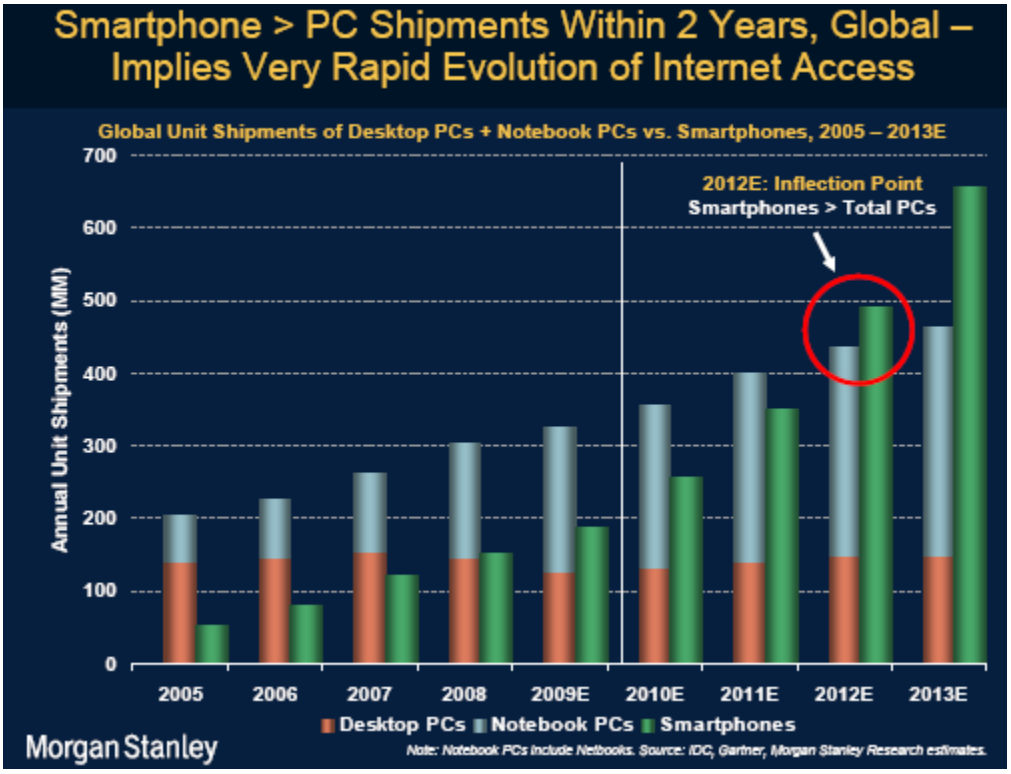
Alas, that time is upon us. Increasingly end users are turning to tablet PCs and mobile phones as their information device of choice, and Windows is not the premiere O/S on those devices. Even the new "Windows Phones" are not compatible with Visual FoxPro applications.

So what are we to do?

This whitepaper lays out one possible path for leveraging as much of our current infrastructure as possible. Along the way down that path, I'll point out alternative routes that may have been chosen instead so that you can choose which route is best for you as you try to get "Fox on the Run."[i]

Why Go Mobile?

Simply put, next year for the first time smartphones will outsell traditional PCs (laptops and desktops combined). By 2014 and the widespread available of "4G" networks, mobile internet access is expected to overtake desktop internet access.



That statistic doesn't mean your users will be using smartphones to input all the data your application collects, but they *will* want to see the results of that data on their smartphone.

So you need to create a mobile application that complements your existing applications and ideally reuses as much of your existing code as possible.

Isn't a Web Application Enough?

It's tempting to think that all you have to do for your mobile strategy is throw up a web application. In fact as we'll see, a well-written web application gets you well on your way, but it's not enough. Even web applications that are smart enough to recognize the capabilities of the client devices that's accessing it still cannot take advantage of all the features that smartphones and tablets have to offer. For example, simply alerting your user when an event is about to happen by vibrating the phone is beyond the capabilities of web applications.

You need your application to take advantage of the *native functionality* of the device: the camera, GPS, speaker, vibration, touch controls, contact lists, etc. For that, you need more than a web application

Which Device Do I Target?

"But wait a minute," you may be thinking. "If I have to write a native application doesn't that mean I have to learn the native language that each phone uses?"

It would be a daunting task to try to learn each device's native language, and the nuances for each one. Perhaps it'd be better if we focused on one device. But which one? *Figure 1* shows the market shares for each O/S. Even if you decided to specifically target the O/S that's currently most popular, Android, you'd still be leaving over 50% of the market behind! And there's no guarantee that Android will retain its popularity.

So perhaps you should learn Android and Apple's iOS. That would give you nice chunk of the market share. All you'd have to do is learn Java for the Android version of your application, and Objective-C for the Apple version of your application.

Some developers may have the inclination and wherewithal to be able to pick up these new languages, but wouldn't it be preferable if there was a "universal" language that would allow you to write one version of your application, possible in a language that you already know, and allow you to compile it into native applications on all the popular mobile devices?

It turns out there are tools that allow you to do exactly that - they can compile HTML5/JavaScript/CSS applications! This session will take a look at a couple of alternative programs for doing that.

Worldwide smartphone operating system (OS) market share in 2009-2015, according to Gartner					Smartphone OS market share and compound annual growth rate 2011-2015, according to IDC		
OS (listed alphabetically)	**2009 market share**	**2010 market share**	**2011 market share**	**2015 market Share**	**2011 market share**	**2015 market Share**	**2011-2015 CAGR**
Android	3.9%	22.7%	38.5%	48.8%	39.5%	45.4%	23.8%
BlackBerry	19.9%	16.0%	13.4%	11.1%	14.9%	13.7%	17.1%
iOS	14.4%	15.7%	19.4%	17.2%	15.7%	15.3%	18.8%
Symbian	46.9%	37.6%	19.2%	0.1%	20.9%	0.2%	-65.0%
Windows Phone/Mobile	8.7%	4.2%	5.6%	19.5%	5.5%	20.9%	67.1%
Others	6.1%	3.8%	3.9%	3.3%	3.5%	4.6%	28.0%
Total smartphones sold	172 million	297 million	468 million	631 million	450 million	N/A	19.6%
Source: Gartner (April 2011)					**Source:** IDC (March 2011)		**via:**mobiThinking

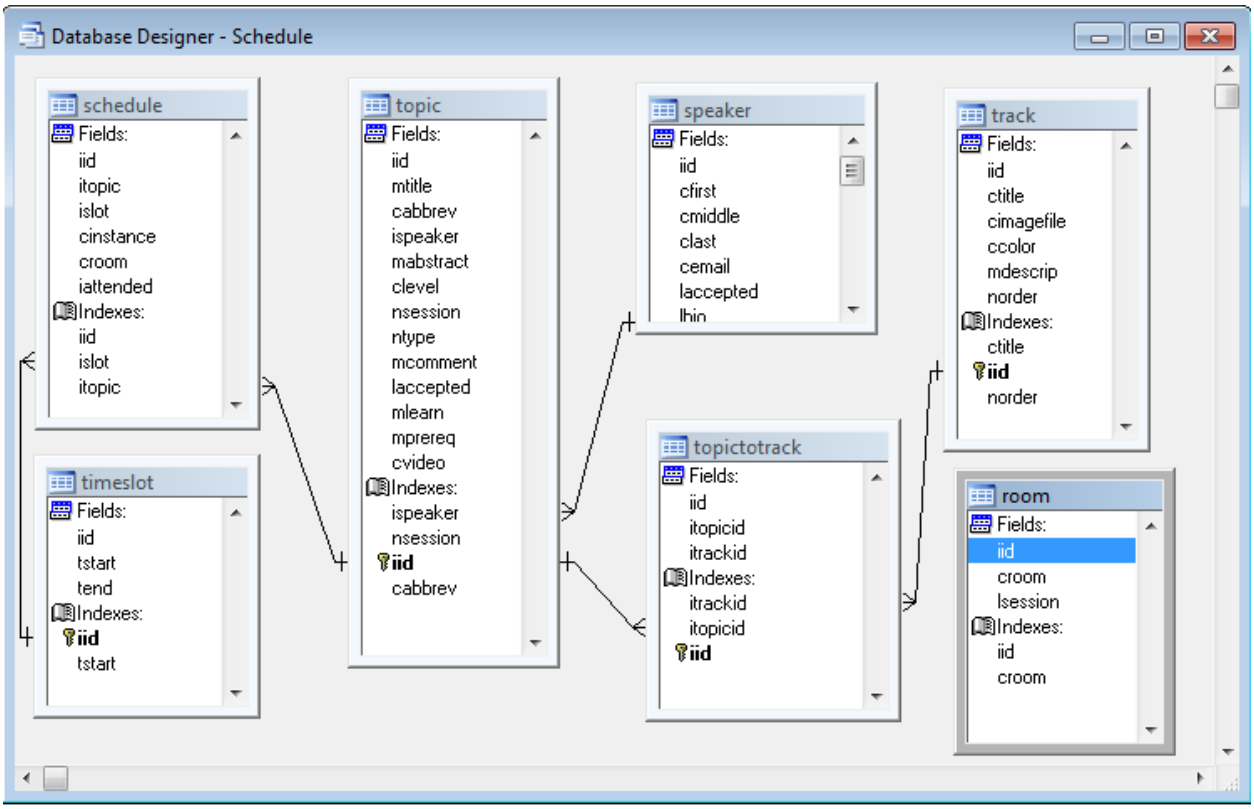
Figure 1: OS Market Share by Year

Getting Started

Our Sample Data and Application

For this session, I needed some sample data that would be universally (well, at least internationally) understood. I had considered using my baseball league's statistics database, but it occurred to me that many people may not know RBIs from ERAs. It was a little difficult for me to think of a good example that conference attendees might all understand, and then it hit me – a conference application!

Thankfully Rick Schummer agreed to share the schema of the Southwest Fox attendee database. And I'm going to share it with you, right here[ii]



As you can see, this database tracks who's speaking, what they're speaking about, and when and where they're speaking.

Our sample application will make all of this speaker information available on any mobile device. Here are the requirements:

- View all courses by day, sorted by time, room, or speaker name
- View detailed courses descriptions
- View speaker bios
- Select which courses we're interested in, and get an alert when that course is about to begin. This will demonstrate using the native features of a device.

We want to keep our data in this Visual FoxPro DBC so we can continue to use our investment in our existing infrastructure.

Making our Data Available

The first step to getting this FoxPro data available on a mobile app is freeing it from the confines of Windows. We could literally migrate the data out of the DBC and into a more Internet-enabled database such as MySQL or SQL Server, but that defeats the goal of allowing us to keep our data available to our existing app without rewriting it.

Another alternative, and the one we're going to take, is to write a piece of "middleware" that goes between our web server and our FoxPro data, accepting requests for the data, retrieving it, and sending it back in a way that other applications can use.

We could write this middleware ourselves in any language. For example, Ruby is very popular at the moment and is used all over the web. But using Ruby would again void our ability to use our FoxPro classes that we've already invested in and were hoping to re-use. We'd also need a DBF database driver that Ruby understands. The popular DBI module for Ruby speaks ODBC (as well as MySQL, Postgres, and many more), but adding all these levels of indirection really discourages me from taking this path (let alone learning Ruby itself). The same argument goes for any solution that's not written in FoxPro itself.

In this example we're going to use Rick Strahl's West Wind Web Connection, mostly because it's popular and I've used it for a long time and so I am familiar with it. FoxWeb[iii] is another solution that many people use, and FoxTrails[iv] looked interesting but I can't tell if it's being actively developed and supported. So, let's go with Web Connection.

Getting Web Connection to Expose our Data

If you don't have West Wind Web Connection already installed, download it from www.west-wind.com and follow the installation instructions. You can install it right on your local machine for development purposes. The excellent documentation makes this very straightforward.

Now that Web Connection is installed, we'll create a project for our application. In this project we'll create subclasses of West Wind's "BizObject" class for each of the tables in our database, and then write methods that allow any web page to request the data.

Step 1: Create the new project in Web Connection

Fire up the Web Connection Management Console and select the first choice, Create New Project.

After the project has been successfully created and configured, Web Connection will automatically start your default web browser and show you the results. It will look something like Figure 2.

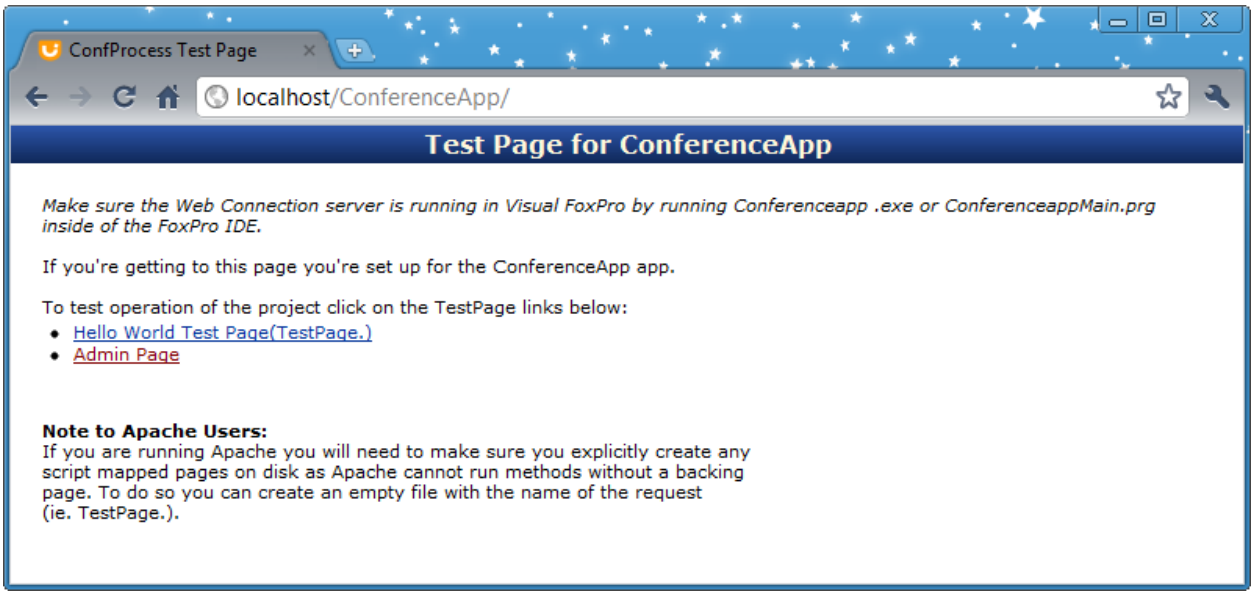


Figure 2 Default Page after Creating Project in Web Connection

It's not quite ready to deploy to your mobile device yet, but this was a big first step!

Step 2: Create business objects for your data

A business object is a wrapper that allows you to perform the common tasks surrounding data, such as adding new records, deleting records, or retrieving a specific record in a consistent and secure way. You may already have business objects for your tables. If so, you can completely skip this step.

Web Connection includes a wizard that allows you to easily create "business objects" for your data. Fire up the Management Console again and select that option. We'll start by creating a business object class for the "Speaker" table and storing it in its own, new, class library, "ConferenceBizObjects.vcx." We have to tweak the defaults a little, and the resulting class looks like this:

```
DEFINE CLASS cstspeaker AS wwbusiness
    cfilename = "speaker"
    calias = "speaker"
    cdatapath = "c:\users\eric\documents\dev\swfox\foxonrun\data\"
    cpkfield = "iId"   && I had to change this from the default
    ndatamode = 0
    Name = "cstspeaker"
ENDDEFINE
```

There's not much there, but let's see what we can do with that. Let's try loading my speaker record, whose ID is *36*.

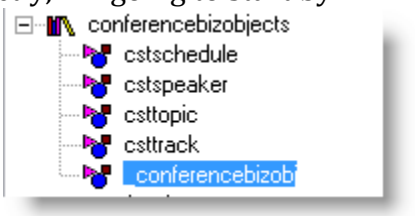
```
SET CLASSLIB TO conferencebizobjects.vcx
oSpeaker=CREATEOBJECT("cstSpeaker")
oSpeaker.load(36)
? oSpeaker.odata.cFirst-(" "+oSpeaker.oData.cLast)
```

That last line returns "Eric Selje," just as it should. Simple, but excellent! Let's go ahead and create these business objects for the remainder of our tables.

Step 3: Create a method to get some of the data in a format the web understands

Simply creating these business object classes doesn't magically make the data show up on the web of course. We need to do that manually. Ideally, the data will come back in JSON format, which is very easy for pages to deal with. Alternatively we could use XML (or an XML variant such as SOAP), or even plain old text.

The first method we need would be something that simply returns all the records from the table. Since I know right now I'm going to need this for every one of these business object classes, and I don't want to modify the wwBusiness class directly, I'm going to start by creating a subclass of wwBusiness and using the class browser to redefine the classes created in the previous step to our new subclass. We end up with this:



Now, by adding a *getList ()* method to the base *_ConferenceBizObj* abstract class and allowing its subclasses to inherit it, we'll have a method in each business object that exports all the data for the base table. Here's that method's simple code, which takes advantage of West Wind's *wwJsonSerializer* class:

```
* Return the data from this table in JSON format

This.open()
DO wwJSONSerializer && Load the classes
loSerializer = CREATEOBJECT("wwJsonSerializer")
lcJSON = loSerializer.Serialize("cursor:"+This.cAlias)
RETURN lcJSON
```

Now we can ask for any of our business objects to return their data in a format the web understands, JSON. JSON standards for JavaScript Object Notation and has become the standard for slinging serialized objects around the net. It's easily read and is terser than XML, which is quite appealing. And it's helpful that JavaScript (and most languages) have the ability to easily deserialize the JSON back into objects.

Let's see how that works with our *Tracks* class, the wrapper around the table that holds the tracks for the Southwest Fox conference:

```
oTopic=CREATEOBJECT("cstTopic")
? oTopic.getList()
```

Returns:

```
{
    "Rows": [
        {
            "iid": 1,
            "ctitle": "Solidifying VFP Development",
            "cimagefile": "Solidifying.png",
            "ccolor": "B4B4B4",
            "mdescrip": "This track appeals to all VFP developers. Sessions cover
either VFP fundamentals, aimed at those newer to VFP, whether moving up from FP2.x or
coming to VFP from other languages; VFP best practices, aimed at helping all VFP
developers improve; or in-depth looks at aspects of VFP that some developers may not
have mastered yet.",
            "norder": 3        },
        {
            "iid": 2,
            "ctitle": "Web Development",
            "cimagefile": "WebDev.png",
            "ccolor": "4D6DF3",
            "mdescrip": "This track covers topics related to developing websites and
web applications.",
            "norder": 6        },
        {
            "iid": 3,
            "ctitle": "Extending VFP",
            "cimagefile": "Extending.png",
            "ccolor": "22B14C",
            "mdescrip": "The sessions in this track look at ways to extend VFP's
reach, by using it together with other products or by taking advantage of native
extension capabilities.",
            "norder": 2        },
        {
            "iid": 7,
            "ctitle": "Pre-Conference",
            "cimagefile": "Precon.png",
            "ccolor": "643198",
            "mdescrip": "These half-day sessions are offered on Wednesday, October
26, and are available for an additional cost; see the <a
href=\"register.aspx\">Register</a> page for details.",
            "norder": 1        },
       {
             "iid": 8,
            "ctitle": "Taking Advantage of VFPX",
"cimagefile": "VFPX.png",
            "ccolor": "ED1C24",
```

```
            "mdescrip": "The sessions in this track showcase the projects in VFPX,
providing attendees with enough information to put those projects to work for them.
These sessions are deep dives into specific VFPX projects.",
            "norder": 4        },
        {
            "iid": 11,
            "ctitle": "Technology for VFP Developers",
            "cimagefile": "Technology.png",
            "ccolor": "990030",
            "mdescrip": "This track looks at tools, technologies and techniques to
make life as a developer easier and more productive.",
            "norder": 5        },
        {
            "iid": 12,
            "ctitle": "Post-Conference",
            "cimagefile": "Postcon.png",
            "ccolor": "FFC20E",
            "mdescrip": "These sessions are offered on Saturday, October 29 and
Sunday, October 30; see the <a href=\"workshops.aspx\">Workshops</a> page for
details.",
            "norder": 7        }
    ]
}
```

All right, this looks good and is easy to read. We now have our business objects that return JSON on demand. Let's get that data out the door and onto the web!

Step 4: Show the data on the web

One thing I like to do for my Web Connection apps is to create a custom scriptmap for each application, rather than just use the default .WC extension. This makes it easier for me to keep everything straight in my head, and adds flexibility to my applications by allowing me to have a different request extension for each of my processing modules.

For our example, I'm going to create a handler mapping in IIS that sends any requests with a '.swfox' extension to our ConferenceApp processor that we created back in Step 1. This is done in the IIS Manager's "Handler Mappings" section of our app, as in figure 4 below. We also have to add a bit of code to our main program, as in figure 3. And as long as you're editing the main program, add

```
*** Add any SET CLASSLIB or SET PROCEDURE code here
SET CLASSLIB TO ConferenceBizObjects ADDITIVE
```

at the bottom of the OnLoad() function, in order to have those classes available to our process.

```
OTHERWISE
    *** Check for Script Mapped files for: .WC, .WCS, .FXP
    lcExtension = Upper( JustExt(THIS.oRequest.GetPhysicalPath() ) )

    DO CASE
    CASE lcExtension == "SWFOX"
        DO ConfProcess with THIS

    *** ADD SCRIPTMAP EXTENSIONS ABOVE - DO NOT MOVE THIS LINE ***

    *** Default Web Connection script handling
    CASE lcExtension == "WC" OR lcExtension == "WCS" OR lcExtension == "FXP"
```

Figure 3: Add this code to your main program

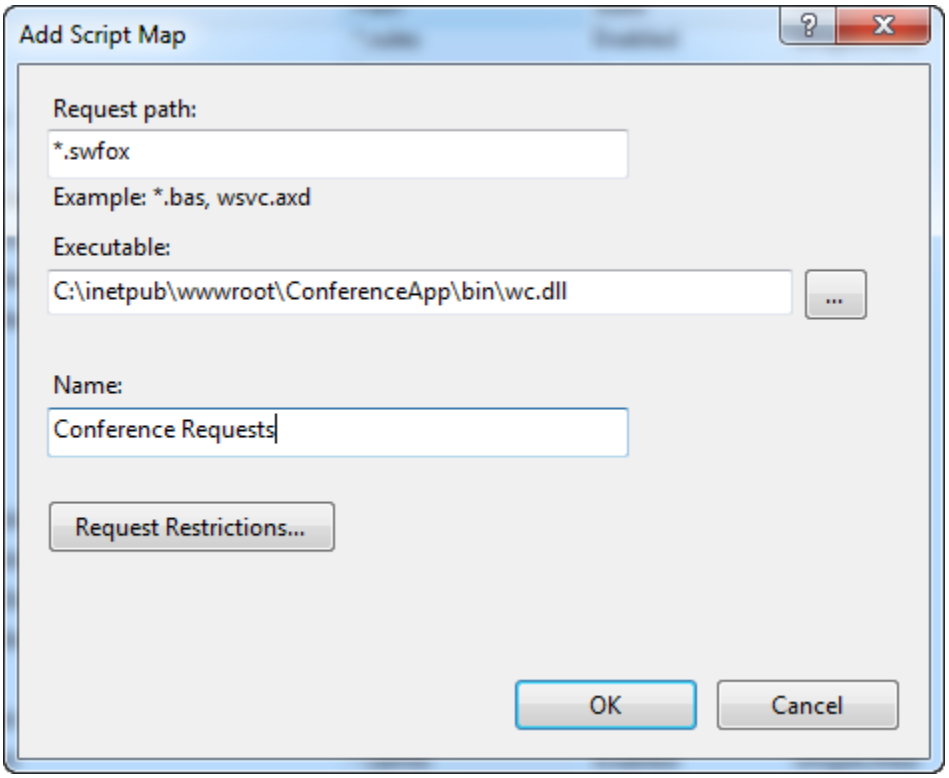
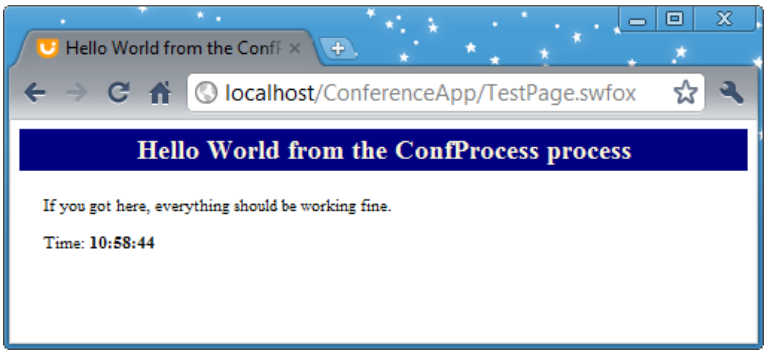


Figure 4: Adding a Scriptmap for our app

Now we can make requests into our application like this:



Hello world is fine, but what we really want to see is some of our conference data, such as the listing of tracks, sessions, or speakers. Let's create a new method in our ConfProcess

class that will return the whole list of whatever you ask for. We'll call it 'getList' to mimic the name of the method in the business object that it will be using:

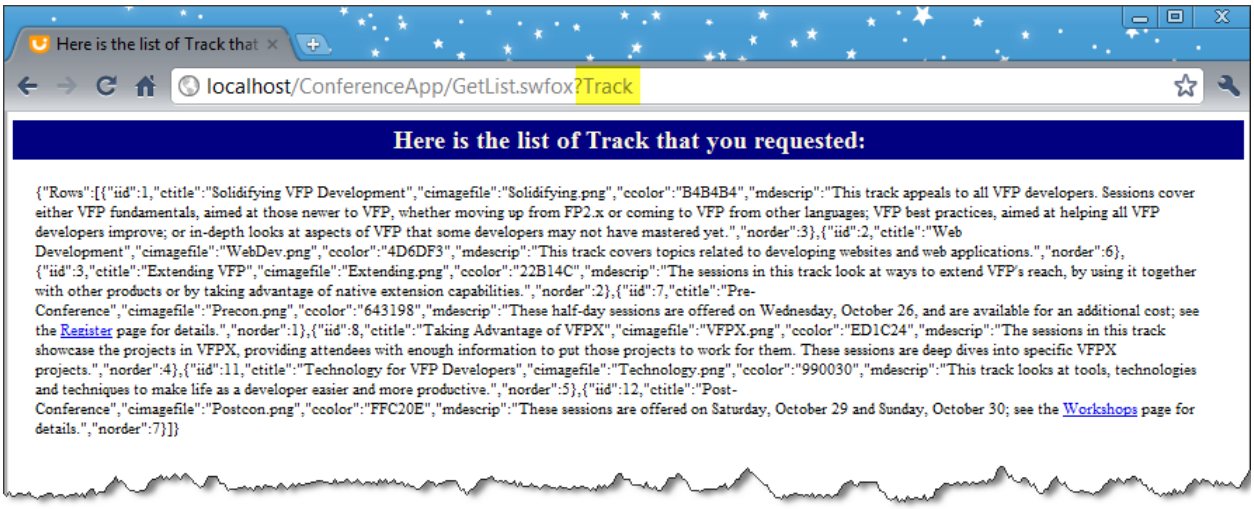
```
FUNCTION GetList(cTable)

* Our convention was the biz object name was 'cst'+ the table name
cTable = EVL(cTable, Request.cQueryString)

LOCAL oBizObj
oBizObj = CREATEOBJECT("cst"+cTable)
cJSONData = oBizObj.getList()
THIS.StandardPage("Here is the list of " + cTable + " that you requested:",
cJsonData)
```

Note: Eventually we're going to need to add security and error handling to our application, but in the interest of making this example as easy to follow as possibly I'm going to forego that for now. We'll also eventually remove the StandardPage call from this method and just have it return the JSON data and let the calling page do with it what it wants to, but that's for the next step.

With that method now in place, if we form our request correctly, being sure to use the name of the table we want the data from, we'll get back the data, albeit in a rudimentary form:



This is the same data as returned in the previous step, but now it's on the web!

We could make the same request for any of our tables. For example:
Localhost/ConferenceApp/GetList.swfox?Speaker
Localhost/ConferenceApp/GetList.swfox?Topic

Pretty cool, and we haven't even done much coding yet.

Step 5: Making a Conference Menu Web Page

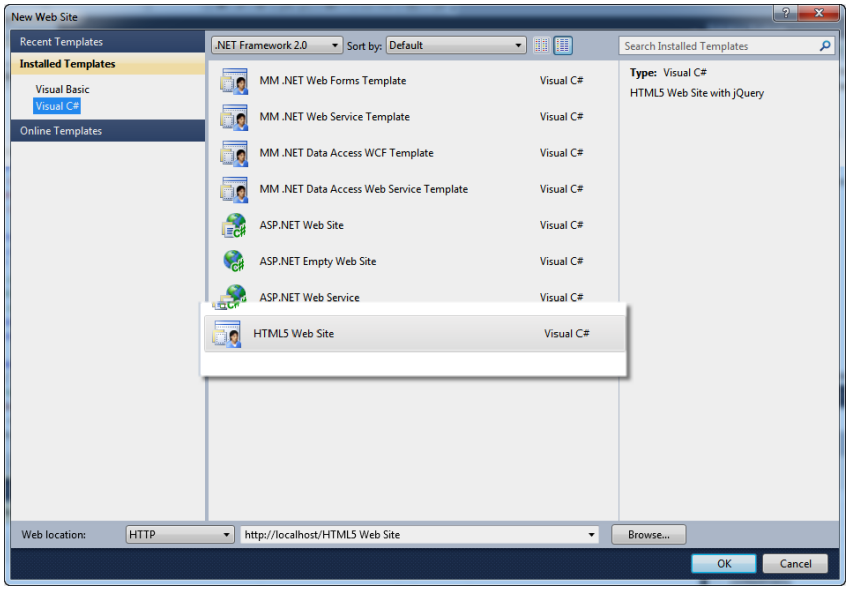
Now that we've get our web service in place, we need to create our actual web site. Here again were are at a point where we can choose any number of technologies: ASP.Net, Silverlight, Adobe AIR, Flash/Flex, and many many more.

In my opinion HTML5 is emerging as the clear winner in the client wars. It's especially attractive because it's an industry standard and it's not controlled by one particular company. It runs on all platforms, from Android, Windows, Blackberry, and Apple smartphones on up. Additionally, it's a technology that most developers already know – no major relearning required.

What's more, there are technologies available that can take an HTML5 web application and recompile it into native apps for smartphones! More on that in the next step but suffice it to say we'll be doing our web site for this session using HTML5.

Note: I've been doing web development in Visual Studio 2010, so I'll be using that in my examples. Visual Studio is absolutely not required. Feel free to substitute your chosen development environment.

To start, select File, New, Website in Visual Studio. Choose "HTML5 Web Site" from the list of templates. (I believe you have to have the latest Service Pack for Visual Studio for this choice to be present).



When I develop locally, I use the HTTP protocol for the Web Location and localhost for the sitename. Be sure to rename your site from the default!

After you hit Ok, you'll have a site that you can immediately browse to. Granted, it'll be a blank page, but it should be there. If you can't see it, check your IIS settings and your local machine's firewall to see if something's preventing it from coming up.

Let's take a look at the source of that HTML5 page, because if your environment doesn't include the same markup in its page things aren't going to work and you're going to wonder why.

The first line is the beautifully simply new DOCTYPE declaration for HTML5 pages:

```
<!doctype html>
```

No longer do you have to specify some DTD with an incredibly long URL looking thing after it as in pages past. This one is so simple even I can remember it.

```
<html lang="en">
```

I've never noticed the *Lang* attribute on the html tag before, but apparently it's been in there a while and is helpful for speech synthesizers and search engines.

```
 <meta name="viewport" content="width=device-width; initial-scale=1.0" />
```

This 'viewport' meta tag is a fairly recent addition, and is not actually part of the HTML5 web standard. It is supported by most mobile browsers however. What this is saying is that the width of the page is the device's maximum width, and the screen is scaled to fix exactly one the screen. If you decrease the scale to 0.5, the browser will render your content as if your screen was twice as wide as it is, causing only half of your content to show. If you increase your scale to 2.0, everything on the page will show up twice as big within the width of the screen.

```
<!-- !CSS -->
<link href="css/html5reset.css" rel="stylesheet" />
<link href="css/style.css" rel="stylesheet" />
```

The next tags are links to the stylesheets. *Html5reset.css* explicitly sets properties that tend to render differently in different browsers. If you don't have it you can download it from http://html5doctor.com/html-5-reset-stylesheet/.

Style.css is completely blank. I suspect that's where Visual Studio would like me to put my custom CSS, rather than cluttering up html5reset.css. That makes good sense.

```
<!-- !Modernizr - All other JS at bottom
<script src="js/modernizr-1.7.min.js"></script> -->
```

This bit of code pulls in Modernizr. Among other things, this handy script library will gracefully degrade any code that you use that isn't supported by the browser. For example if you try to HTML5's native drag and drop on a browser that doesn't support it, Modernizr will substitute its own implementation that should work fine. Other features may just be ignored without and remnants on the screen. In a nutshell, Modernizr removes worrying about whether your client will support new features, allowing you to code for the future without worrying too much about the past.

```
<!-- Grab Microsoft's or Google's CDN'd jQuery. fall back to local if necessary -->
<!-- <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-
1.5.1.min.js " type="text/javascript"></script> -->
<!--<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.min.js">
</script> -->
<script> !window.jQuery && document.write('<script src="js/jquery-
1.5.1.min.js"><\/script>') </script>
```

This block of code is interesting, as it reflects Microsoft's recent embrace of jQuery as the standard client-side scripting library. Savvy coders have long known about jQuery's excellence, and Microsoft attempted to write their own competitor before giving in to jQuery's superiority and including that with the web pages.

If you want to have your clients download jQuery from a 3rd party "content-delivery network" rather than your own server, uncomment out one (but not both!) of the first two <script> tags.

It's interesting that these <script> tags point to a specific version of jQuery rather than using the generic pointer to the "latest" version of jQuery, which looks like:

http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js

The last bit of script tells the browser to load a local copy if no other copy was loaded yet.

That's the end of the <head> section of our blank HTML5 page, and the body is, well, blank.

```
<body>
    <div id="container">
    </div>
</body>
```

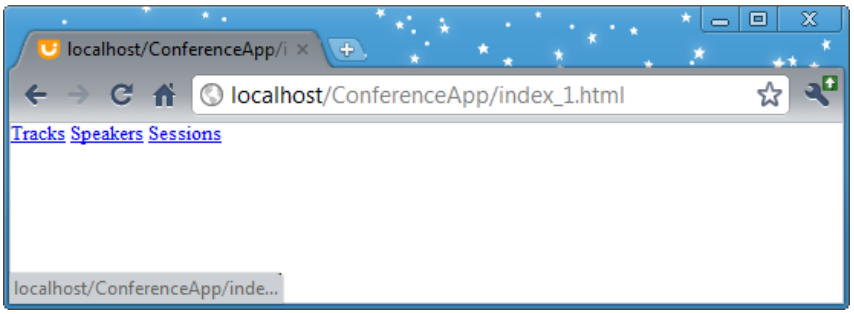
Let's start adding some content to get our conference data showing.

Navigation

HTML5 features a new <nav> tag to aid in navigation. It's helpful for screen readers to find the *primary* navigation links of web pages, and is part of the movement towards more semantic markup. We'll add it between the s to put the links to our page:

```
<div data-role="content" id="menu">
       <nav>
         <a href="#Tracks">Tracks</a>
         <a href="#Speakers">Speakers</a>
         <a href="#Sessions">Sessions</a>
       </nav>
</div>
```

This gives us a very basic screen.



Step 6: Adding jQuery Mobile

At this point I could start walking you down the path where we create a perfectly usable website that has decent looking navigation and brings up our data just like we'd expect. But since we have our eyes on developing a *mobile* application, there are actions we can take now that will make our mobile development easier. One choice that should be made is implementing the *jQuery Mobile* framework. Alternatives such as *Sencha Touch* and *XUI* are also out there, but I'm going to go with jQuery Mobile since I've had such good experiences with jQuery up to this point.

The jQuery Mobile framework is merely a set of CSS files and a JavaScript library that styles your HTML so that it looks great on mobile devices. It is in fact so good that once we've applied it to our application we may feel like it's good enough to deploy even though it's not a true native app, but merely a website.

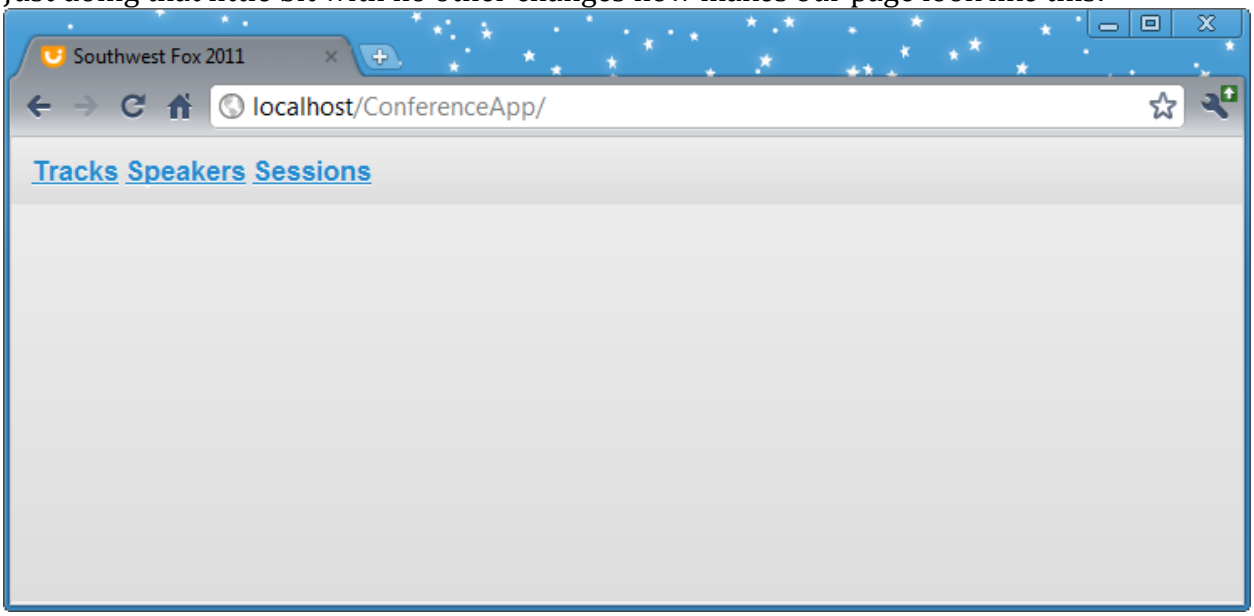
To get started using the jQuery Mobile framework, we have to uncomment one of the lines that loads the main jQuery library that we saw in the previous step, and then add in the new references to jQuery Mobile JavaScript libraries.

```
<script type="text/javascript" src="http://code.jquery.com/jquery-1.6.2.min.js">
</script>
<script type="text/javascript" src="http://code.jquery.com/mobile/1.0b2/jquery.mobile
-1.0b2.min.js"> </script>
```

And we need to add our references to the jQuery Mobile CSS library:

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.0b2/jquery.mobile-
1.0b2.min.css" />
```

Just doing that little bit with no other changes now makes our page look like this:

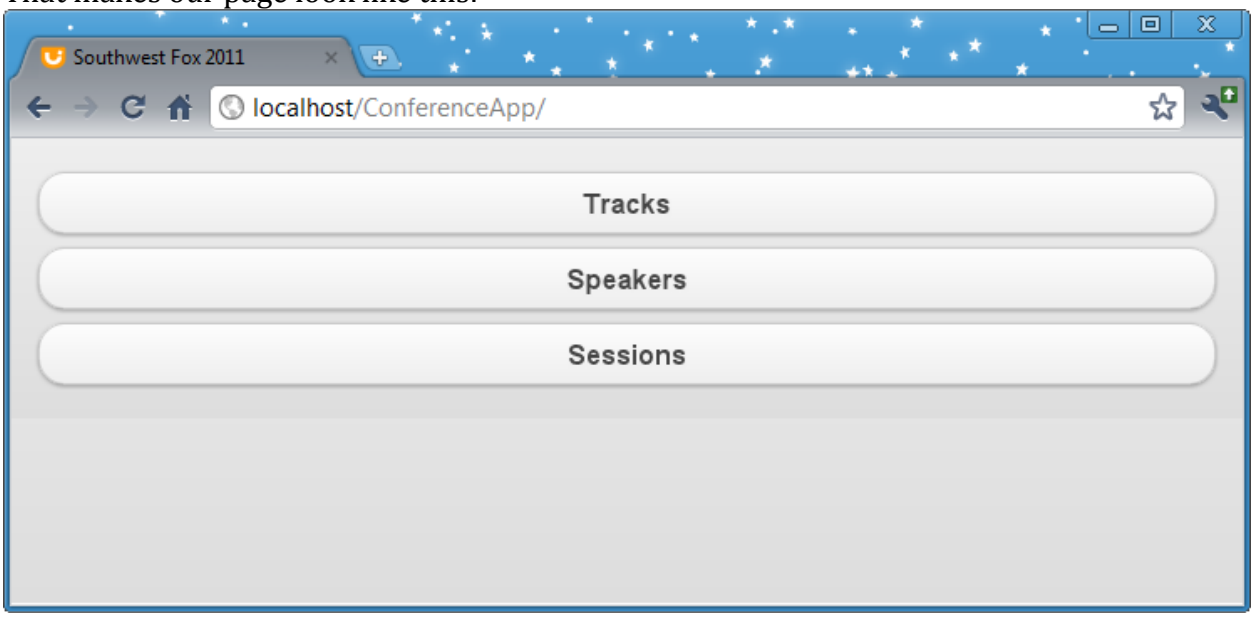


Very iPhone-ish!

I want the links to look more like buttons, however, so I'm going to add a *data-role* attribute to the links, like this:

```
<a href="#Tracks" data-role="button">Tracks</a>
<a href="#Speakers" data-role="button">Speakers</a>
<a href="#Sessions" data-role="button">Sessions</a>
```

That makes our page look like this:



Nice! Let's add a header, like this…

```
<div data-role="header">
    <h1>SW Fox 2011</h1>
</div>
```

…and actually see how it looks **on the phone** (figure 5).

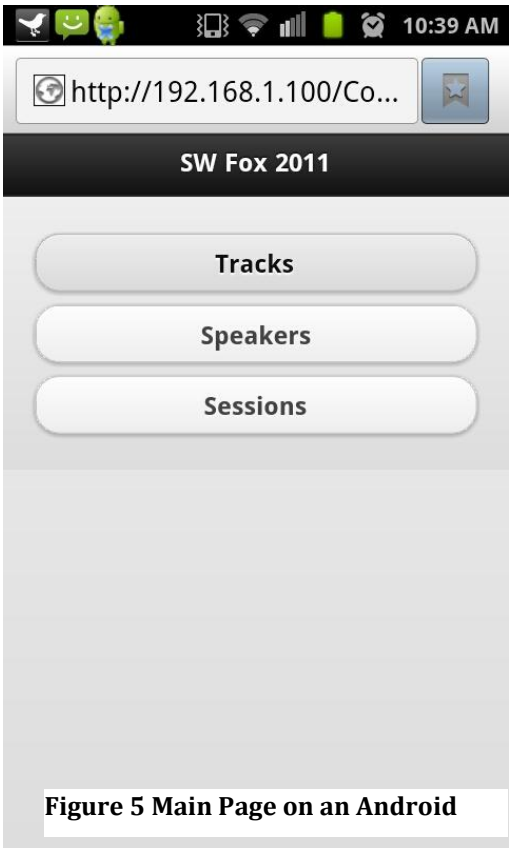


Figure 5 Main Page on an Android

Hey, not too bad! Now let's create the target Divs for each of these links. (I *could* create separate pages for them, but jQuery Mobile makes it very easy to put the content into Div tags which gives the appearance of a separate page to the user, but has the ease of developing on a single page.)

Our target "pages" then are nothing more than divs that have been given a data-role of *page*, like this:

```
<div data-role="page" id="Tracks”>
```

This causes jQuery Mobile to hide the div initially, but show it when the link is clicked. We can also add

a header for each "page", another Div where we can insert out content, and a "Back" button to allow the user to return to the main div, and our entire completed page looks like this:

```
<div data-role="page" id="Tracks" data-theme="a">
    <div data-role="header">
    <h1>
        Tracks</h1>
    </div>
    <div data-role="content" id="TrackList">
    </div>
    <p><a href="#main" data-direction="reverse" data-role="button" data-
theme="b">Back to Menu</a>
    </p>
</div>
```

Repeat this for the other links, making sure to give the Div an id that corresponds with the *href* attribute of the <a> tag, and we now have a rudimentary web application. Now we just need to call the services that we set up in Steps 3 and 4 to our page, and we'll have some live FoxPro data on our phone.

Step 7: Getting and Display the Data with Ajax

Remember all the way back in Step 4 when we made calls to our Web Connection service and got JSON data back? Now we're going to put that to use by making AJAX calls to the very same service, and then format that JSON data nice and pretty for our users.

JQuery makes AJAX calls very easy, by wrapping everything in one simple function, like this call for retrieving our list of the Session Tracks at Southwest Fox:

```
$.getJSON('./GetList.swfox?Track', function (data) {});
```

If that call is successful, the function that we send as the second parameter will get called, and the results will be in the 'data' parameter, in JSON format. We want our page to enumerate the results in a list, so we'll use jQuery's succinct syntax and easy of handling JSON data to append the information inside of the div with the Id of *TrackList*, in this example. So our fleshed out function now looks like this:

```
$.getJSON('./GetList.swfox?Track', function (data) {
  var items = [];
  var aTracks = data.Rows;
  $.each(aTracks, function (key, oTrack) {
    // Put the track's title in a div
    items.push('<div>' + oTrack.ctitle + '</div>');

  });

  $('<div />', {
    'class': 'my-new-list',
       html: items.join('')
  }).appendTo('#TrackList').trigger('create');
})
.error(function () { alert("error"); });
```

In case you're not familiar with JavaScript, we're essentially creating an array (items) and adding code (items.push) that has all the HTML content. After the array is created, we 'join' the array (combine all rows together) and insert the resulting string into the element that has the id *TrackList*. `trigger('create')` makes JavaScript reapply the CSS for our new element.

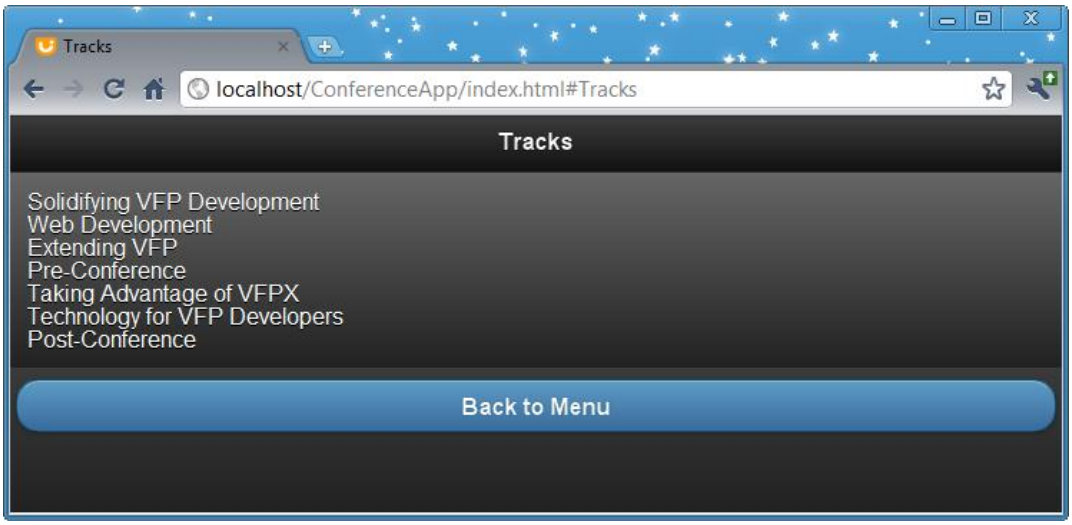
Note that I appended an error handler to the end of our getJSON() request, just in case there's a problem with our server. Right now it doesn't do much, but we can certainly flesh that out later.

But where do we put this AJAX call, which we want to invoke when we hit the link for the page? JQuery Mobile allows you to bind methods to events with *live()* much the way BINDEVENT() does in Visual FoxPro. So if we put the entire getJSON() call inside one of these binding like this…

```
$('#Speakers').live('pagecreate', function (event) {
  // Load the speakers into the DOM before showing the page
  $.getJSON('./GetList.swfox?speaker', function (data) {
    var items = [];
    var aResults = data.Rows;
    $.each(aResults, function (key, oRow) {
    items.push('<div><h3>'
+ oRow.cfirst + ' ' + oRow.clast + '</h3></div>');
    });
    $('<div />', {
        'class': 'my-new-list',
        html: items.join('')
    }).appendTo('#SpeakerList').trigger('create');
  })
  .error(function () { alert("error"); });
});
```

…then that AJAX call will occur when the Speakers page is created. *(Notice I switched it up on you here, using the Speakers rather than the Tracks I used before).*

Now do a similar thing with the Tracks and Topics. And if we hit the page now and click on the Tracks list, it looks like this:



Excellent, we now have our data coming out of a FoxPro table and showing nicely on our web page.

Let's spiff this up. Wouldn't it be nice if you could see the description of the track if you clicked on it? There are a variety of ways you could accomplish this, but I'll show you the 'collapsible list' way which looks pretty good, and is trivial to accomplish. All you have to do is change the line of code that puts the name of the track into your content.

```
items.push('<div data-role="collapsible" data-collapsed="true"><h3>'
 + oTrack.ctitle + '</h3><p>' + oTrack.mdescrip + '</p></div>');
```

We've added the `data-role="collapsible" data-collapsed="true"` to let jQuery Mobile know this is collapsible and should be collapsed, put the title into an `<h3>` element (it could have been any `<h>` tag), and added the description into a subsequent `<p>` tag. And, voila, it now looks really good on the desktop, iPhone, and Android. And it has hardly taken any effort at all.

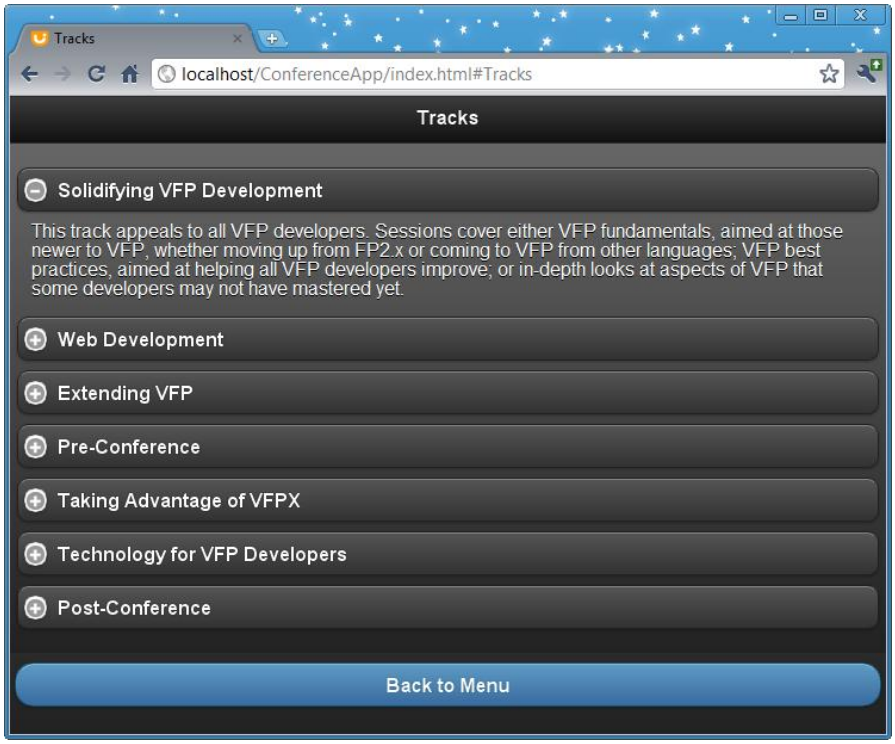
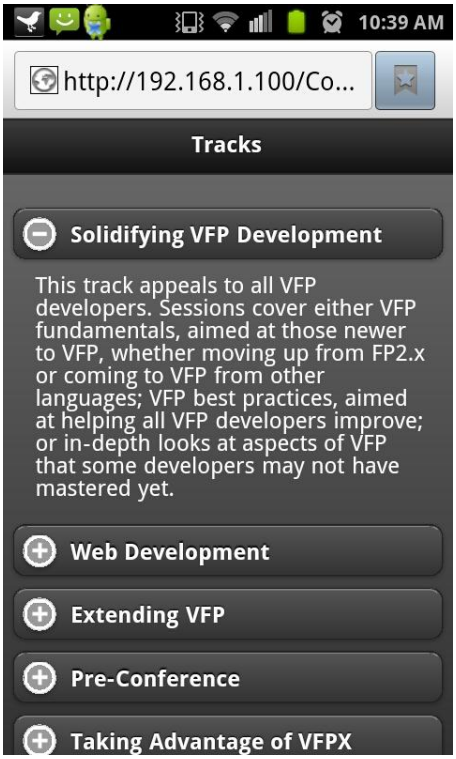


Figure 6 Collapsible content on the Desktop, iPhone, and Android


10:39 AM
http://192.168.1.100/Co...
Tracks
Solidifying VFP Development
This track appeals to all VFP developers. Sessions cover either VFP fundamentals, aimed at those newer to VFP, whether moving up from FP2.x or coming to VFP from other languages; VFP best practices, aimed at helping all VFP developers improve; or in-depth looks at aspects of VFP that some developers may not have mastered yet.
Web Development
Extending VFP
Pre-Conference
Taking Advantage of VFPX


No SIM
11:38 AM
Tracks
Solidifying VFP Development
This track appeals to all VFP developers. Sessions cover either VFP fundamentals, aimed at those newer to VFP, whether moving up from FP2.x or coming to VFP from other languages; VFP best practices, aimed at helping all VFP developers improve; or in-depth looks at aspects of VFP that some developers may not have mastered yet.
Web Development
Extending VFP
Pre-Conference

Rather than walk you through each similar step for the remaining pages, I encourage you to look at the source code that comes with this session to see the rest. It all follows a similar pattern.

Adding the Bookmark to our Home Screen

At this point we actually have a web application that looks really good and could even be deployed. On the iPhone, you can simply click on the middle icon on the bottom of Safari to add a bookmark of this website to your Home Screen, making it appear much like any other application. On Android phones, you click on the menu and select "Add shortcut to Home" to accomplish this.

Web applications are great, and with HTML5 they are overcoming many of their shortcomings. For example we now have reliable local storage and a cache manifest, which makes are application usable when disconnected from the Internet. We also have a geolocation API that lets us know more or less where we physically are in the world.

But there are still features on the mobile device that HTML5 cannot overcome, such as setting alerts, using the accelerometer or camera, and accessing the phone's contact list. For that we need a Native App, which we'll do in the next step.

Step 8: Moving from an HTML5 Web Application into a Native App

This next step may seem a bit like magic, but since we created our web application in standard HTML5 code, along with JavaScript and CSS, we can take advantage of tools that can "compile" this into a native web application. You would really only need to do this if you had to take advantage of native features of the phone, such as the camera or contacts list (see Step 9), or if you wanted to distribute your app through the various markets.

There are a couple of tools that I'm aware of that can do this. The first one that I became interested in was **PhoneGap**, and the second was **Titanium.** Both products can easily deploy to Android and Apple's iPhone, but in order to deploy to that you have to have paid the $99 to be part of have Apple's "Developer Certification," which gives you the tools you need (Xcode) to compile the application. Oh yeah, and you also have to have a Mac. Since this paper is geared towards Visual FoxPro developers I'm going to skip talking about iPhone deployment except to say that it's pretty similar.

So, focusing merely on Android development…

PhoneGap

What I like about PhoneGap is that it is Open Source and uses the Eclipse IDE, which I'm familiar with and already have installed on my development box. PhoneGap also targets Blackberry and WebOS, and Symbian, though those two platforms are dying a quick death (refer back to Figure 1).

Getting started with PhoneGap is really easy.[v] If you don't already have Eclipse installed, download and install that. You also need to install the Android SDK (a set of Java libraries, but don't worry, you'll never have to see them) and ADT plugin so Eclipse can talk to the Android emulator and phone.

You have to download and install PhoneGap as well, which turns out to be a Java class library , a JavaScript library, a configuration file, and a sample application.

You then Launch Eclipse and start a new Android application, set a few properties of the application, and click Finish. Don't create the application on top of the Visual Studio web project you created earlier – keep them separate.

Then follow these steps (taken right from the Getting Started page) to tell Eclipse to include the PhoneGap features in your app:

1. In the root directory of the project, create two new directories:
    - /libs
    - /assets/www
2. Copy phonegap.js from your PhoneGap download earlier to /assets/www
3. Copy phonegap.jar from your PhoneGap download earlier to /libs
4. Copy xml folder from your PhoneGap download earlier to /res

5. Make a few adjustments to the project's main Java file found in the src folder in Eclipse: (see Figure 7)
6. Change the class's extend from **Activity** to **DroidGap**
7. Replace the **setContentView()** line with**super.loadUrl("file:///android_asset/www/index.html");**
8. Add **import com.phonegap.*;**
9. Remove **import android.app.Activity;**

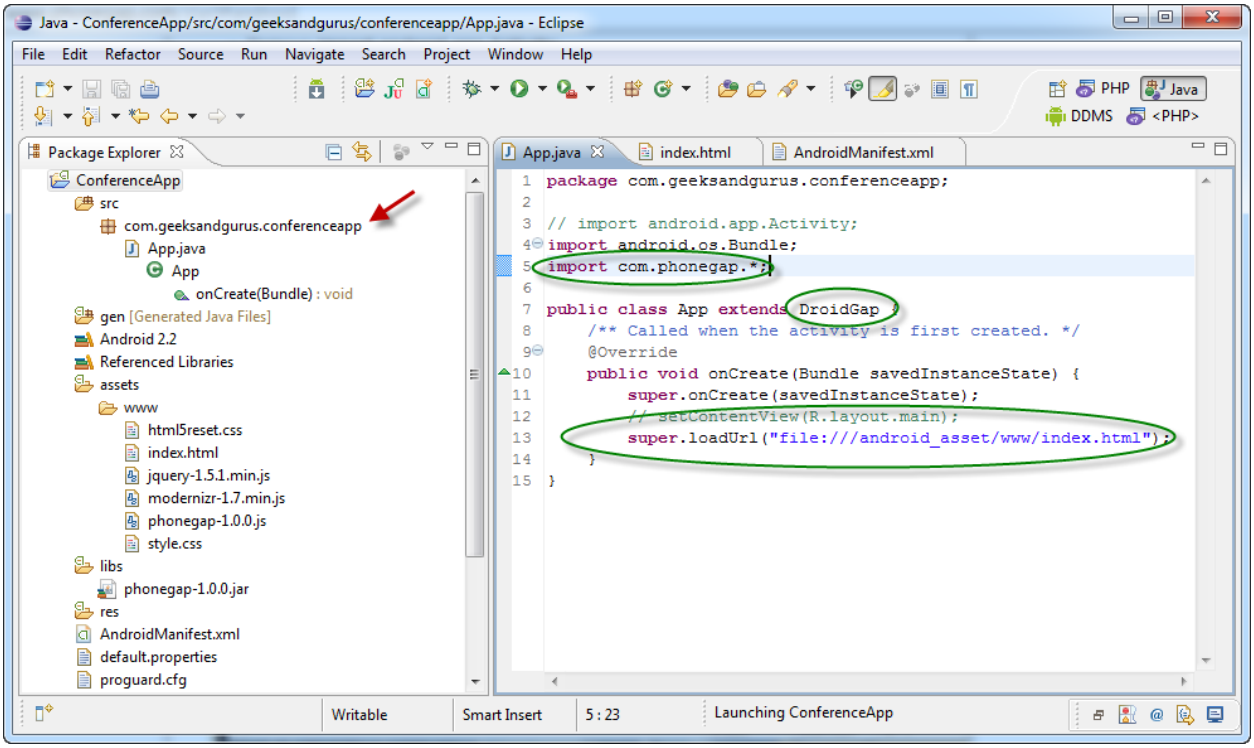


Figure 7Setting up your Java environment

Note: You might experience an error here (I did), where Eclipse can't find phonegap-1.0.0.jar. In this case, right click on the /libs folder and go to Build Paths/ > Configure Build Paths. Then, in the Libraries tab, add phonegap-1.0.0.jar to the Project. If Eclipse is being temperamental, you might need to refresh (F5) the project once again.

To tell our application which features are required for this application, we can modify the *AndroidManifest.xml* file.

1. Right click on AndroidManifest.xml and select **Open With > Text Editor**
2. Paste whichever of the following permissions you're application requires under the versionName tag:

```
<supports-screens
android:largeScreens="true"
android:normalScreens="true"
android:smallScreens="true"
android:resizeable="true"
android:anyDensity="true"
/>
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" /> <uses-
permission android:name="android.permission.GET_ACCOUNTS" />
```

3. Add `android:configChanges="orientation|keyboardHidden"` to the activity tag in AndroidManifest (see Figure 8).
4. Add a second activity under you application tag in AndroidManifest (see Figure 8).

```
<activity android:name="com.phonegap.DroidGap"
android:label="@string/app_name"
android:configChanges="orientation|keyboardHidden"> <intent-
filter> </intent-filter> </activity>
```

Your *AndroidManifest.xml* file will now look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:windowSoftInputMode="adjustPan"
      package="com.phonegap" android:versionName="1.1" android:versionCode="5">
    <supports-screens
        android:largeScreens="true"
        android:normalScreens="true"
        android:smallScreens="true"
        android:xlargeScreens="true"
        android:resizeable="true"
        android:anyDensity="true"
        />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.RECORD_VIDEO"/>
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.BROADCAST_STICKY" />

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />

    <application android:icon="@drawable/icon" android:label="@string/app_name"
        android:debuggable="true">
        <activity android:name=".test"
                  android:label="@string/app_name" android:configChanges="orientation|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.phonegap.DroidGap" android:label="@string/app_name"
                  android:configChanges="orientation|keyboardHidden">
            <intent-filter>
            </intent-filter>
        </activity>
    </application>

    <uses-sdk android:minSdkVersion="2" />
</manifest>
```

Figure 8 ApplicationManfiest.xml after tweaks for PhoneGap

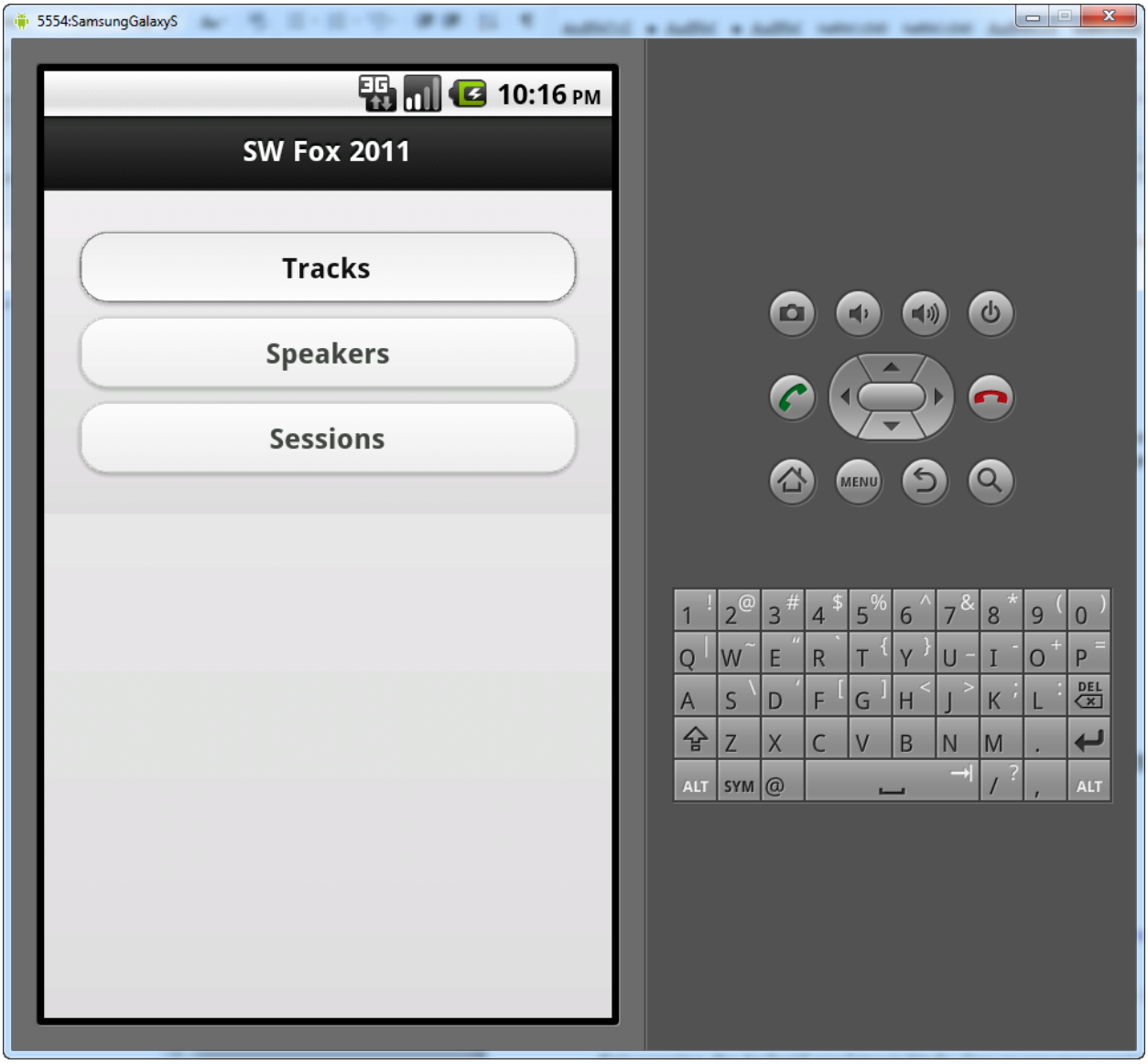
Now copy the files from our web application from c:\inetpub\wwwroot\ConferenceApp to the */assets/www* folder under our Eclipse project root. Be sure to get all the js files and css files too:

- Html5reset.css
- Style.css
- Jquery*.js
- Mondernizr*.js
- Index.html

Open *Index.html* and add a reference to the PhoneGap library

```
<script type="text/javascript" charset="utf-8" src="phonegap-
1.0.0.js"></script>
```

And we're all set! Simple right+click the Project name, click Run As, then Android Application and the emulator will appear, showing our app.



Fair warning, the Android emulator is kinda slow so you may want to grab a cup of coffee at this point. However if you've got an actual Android phone handy, unplug the USB cord, put your phone in "Debug" mode, plug the USB cable back in, and then when you Run As Android Application the application will install on your phone and run just like a native app you downloaded from the Market!

Titanium

An alternative to PhoneGap is Appcelerator's *Titanium*[vi] . Unlike PhoneGap, Titanium comes with its own development environment which was derived from Aptana Studio. It uses the same Android SDK and emulator as PhoneGap however.

Because PhoneGap worked so well for me I didn't dive into Titanium too deeply. It appears that Titanium is better documented and has a much deeper API than PhoneGap. For example, Titanium has a barcode reader API, which I think would be handy for the many warehouse applications that are written in Visual FoxPro.

Titanium is free for a basic edition, which doesn't have all of the APIs that the Premium edition does. The premium edition costs $49/mo. for the average Visual FoxPro developer. I think most of us might shy away from it for that reason, but as I mentioned with the cost comes good support and documentation.

Step 9: Adding Native Features with PhoneGap

Now that we've gone through the trouble of compiling our application to a native application, we'd better take advantage of some of the functionality of the phone, otherwise we could have just left it as a web application.

A few ideas I thought about for this app were:

- Shut off the phone's ringer automatically while I'm in a session that I told to mute
- Vibrate the phone when giving JavaScript Alerts
- Click a link on Speaker's profile to automatically ad that speaker's information to your device's contact list.
- Disable the "Back" button, forcing the user to use the onscreen navigation.

Muting the Ringer

Unfortunately I couldn't figure out how to mute the phone, so we're going to have to forego that one for now. I mention this rather than omitting this section entirely in order to illustrate that not every feature of every smarthone is accessible through PhoneGap's API. If you were doing true native development, there are actually ways to mute the phone.

Vibration

But getting the phone to vibrate was trivial. If I want my phone to buzz when it can't reach the web service that sends back my data, I can append this to my getJson() call:

```
.error(function () {
                        navigator.notification.vibrate(2000);
                        navigator.notification.alert("Web Service
Unavailable", alertDismissed, 'Alert', 'Ok');
```

Notice I commented out my previous, standard JavaScript altert and simply replaced it with a call to PhoneGap's alert() method instead, which is in the navigator.notification namespace.

I figured out how to do this by perusing PhoneGap's excellent API documentation at http://docs.phonegap.com/.

Adding Information to Contacts

This isn't very hard either, though it does require putting the data that you get back from your web service into a new object.

```
// create a new contact object
var contact = navigator.contacts.create();
contact.displayName = oRow.cfirst + ' ' + oRow.clast;
//specify both to support all devices
contact.nickname = oRow.cfirst + ' ' + oRow.clast;

// populate some fields
var name = new ContactName();
name.givenName = oRow.crirst;
name.familyName = oRow.clast;
contact.name = name;
contact.emails = new ContactField('email', oRow.cemail)

// save to device
contact.save(onSuccess, onError);
```

Disabling the Back Button

In our sample app, we've added our own "Back" button to the bottom of each page, so we don't want the device's back button to override this. We can disable that button by binding an eventlistener to it, like this:

```
document.addEventListener("backbutton", btnBackClicked, false);
```

Now whenever the user clicks the device's back button, our own 'backbutton' method will be invoked:

```
function btnBackClicked() {
   navigator.notification.alert("Please use the navigation on the screen",
alertDismissed, 'Alert', 'Ok');

}
```

Distributing Your Application

Distributing a native application isn't quite as easy as just putting a web app up on a web site and having your users point their phone's browser application at it. On the other hand, you can't make any money selling web applications: for that you need to distribute the application through one of the "marketplaces" that the platforms provide.

Beta Testers and Free Users

Android applications are .APK files, and you actually don't need to place them in the marketplace. You can manually give the APK file to users and they can put their phones in debugger mode and copy the APK file to their phone and it should appear on their home screen. The application won't auto-update this way, and it's not as easy as downloading from the marketplace, but it is possible and handy when you want to give the app to a few users to try out before you put it up on the market. Nothing will kill a promising app faster than negative reviews from a version that was released before proper testing.

iPhone apps, even non-free apps, can also be distributed to up to 100 users at no charge. You have to create a "provisioning list" for which users are going to be allowed to run the application, which can be a bit onerous.

Paid Users and the Market/AppStore

Once your Android application is ready to go, you sign it with your "private release key," which you obtain by registering on the Android Market. Then you can simply log into the market and upload the application. When a new version is ready, you change the android:versionName in the manifest file and re-upload it, and the market will notify your users that an update is available.

It's a similar process with iPhone apps, but of course you'll have to have laid out the $99 developer's registration fee. iPhone apps are subject to Apple's approval as well, so make

sure your app conforms to their Human Interface Guidelines, isn't too derivative, trivial, pornographic, or uses too much bandwidth.

Advertising

A lot of "Free" apps, such as Angry Birds, still make a ton of money from in-application advertising. This can be more lucrative than paid apps because the revenue continues to generate after the intial purchase of the application. Brilliant!

For most of our business applications, we probably just want to get it into the hands of our users and aren't concerned with selling the app or generating ad revenue. But don't rule out the possibility of putting apps in even your business apps. With advertising networks such as AdMob, MoPub, and AdSense, your app can include fairly unobtrusive advertisements that just might generate enough revenue to cover your development costs, or at least buy you some beer. Integrating these into your app usually involves a simple SDK and enabling your app for Internet access so it can grab the app on demand. Each network is different so I won't attempt to show them here, but you can find good instructions for each one on their site.

Summary

As you can now see, it's not terribly difficult to get an application running on a mobile phone that uses Visual FoxPro data. You need a way to expose that data through web services, know how to consume that data on the web client, and transform your application with the right tools.

In retrospect however, getting this all to work is only easy because of the groundwork laid out by a great number of people and projects, to whom I owe a great deal:

- Rick Strahl, West Wind Web Connection (www.west-wind.com) *and* the wwJSONSerializer class which makes web services with Visual FoxPro easy.
- The jQuery and JQuery Mobile projects
- Modernizr
- Richard Clark, and HTML5Reset.css
- The PhoneGap project
- Appcelerator

---

[i] If you're unfamiliar with it, "Fox on the Run" was a big hit in 1975 for the band Sweet. It's a catchy number, and I apologize if it gets stuck in your head after seeing this session or reading this whitepaper. The title was just too good for me to not reappropriate it for the title of my session.

[ii] It's also in the session materials in a digital format.
[iii] http://www.foxweb.com/
[iv] http://www.foxtrails.org/
[v] See http://www.phonegap.com/start#android for just how easy.
[vi] www.Appcelerator.com