

AdminDev2012

HOSTED BY **THE VIEW**

Mobilizing Your Applications with XPages

Bruce Elgort
Elguji Software

© 2012 Wellesley Information Services. All rights reserved.



In This Session ...

- **Users are expecting, if not demanding, mobile access to their Notes and Domino applications**
 - ♦ **This session will show you how to give your application a nice, native mobile user interface that users will love**
 - ▶ **Mobilizing your apps using open-source custom controls makes this easy**
- **XPages are also a great tool for providing services and resources for “native” mobile apps**
 - ♦ **Providing an API so that iPhone, Android, and other devices can interact with your apps is something you can do with XPages**
 - ♦ **Learn how a popular iPhone app uses XPages to deliver data to iPhones from a server in the cloud**

What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

The XPages Mobile Controls

- Developed by IBM and originally posted to OpenNTF.org as open source
- Originally released in March 2010
 - ♦ Still a relatively “young” set of tools
- Support native UI elements for WebKit-based devices including:
 - ♦ Safari iPhone
 - ♦ Android
 - ♦ BlackBerry OS 6
- Delivered as a set of custom controls that you can connect to your applications
 - ♦ Based on Dojo 1.6.1
 - ♦ <http://dojofoundation.org/mobile>

The XPages Mobile Controls (cont.)

- The XPages Mobile Controls are now part of Upgrade Pack 1 (UP1)
- UP1 is available for the Domino server and for Notes and Domino Designer
 - ♦ The Upgrade Pack is available on the IBM Passport site:
www-01.ibm.com/software/howtobuy/passportadvantage
 - ♦ The IBM part number is CI5GIEN
- Once you install UP1, you will see additional controls in Domino Designer that you will use for your mobile application development projects

The XPages Mobile Controls (cont.)

- Developing mobile apps is as easy as dragging and dropping controls onto an XPage
- In this session, we will look at the “US Airports” application and how it was built
 - ♦ This app contains a list of all the airports in the US, and contains:
 - ▶ IATA codes
 - ▶ ICAO codes
 - ▶ Airport names
 - ▶ Latitude and Longitude
 - ▶ Runway Information
 - ▶ Miscellaneous comments

The Sample/Demo Application

- During this session we will be building an app that adds mobile capabilities to a Notes database
- The database contains documents for all US airports, and includes airport:
 - ♦ IATA codes
 - ♦ ICAO codes
 - ♦ Airport name
 - ♦ State
 - ♦ Country
 - ♦ Runway length
 - ♦ Runway elevation
 - ♦ Comments

The Completed US Airports Application

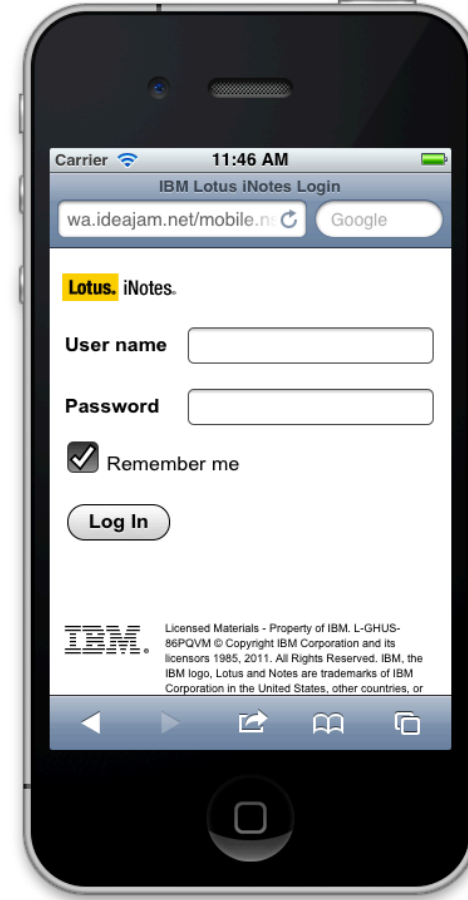
- Here are some of the app's screens (the sample app is available for you to download):



Home screen icon

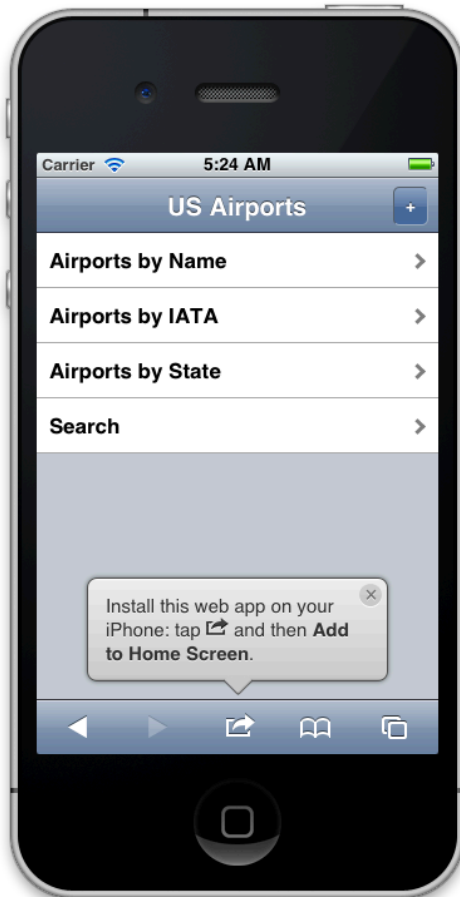


Splash screen

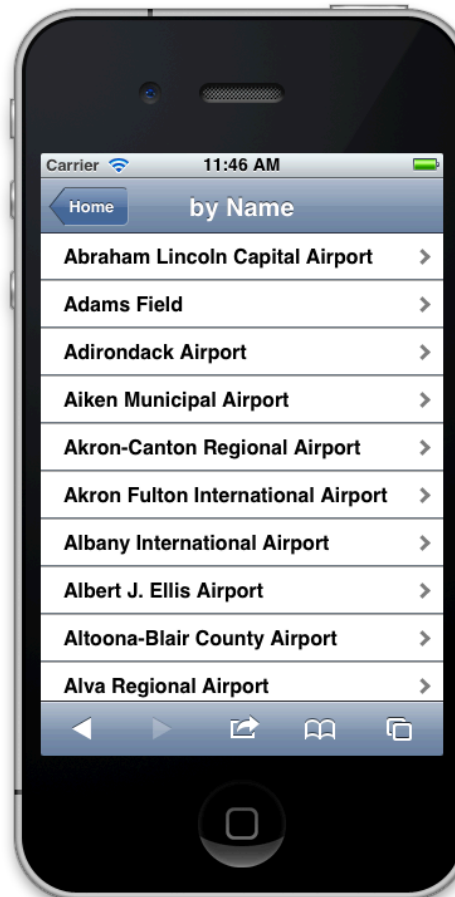


Login page

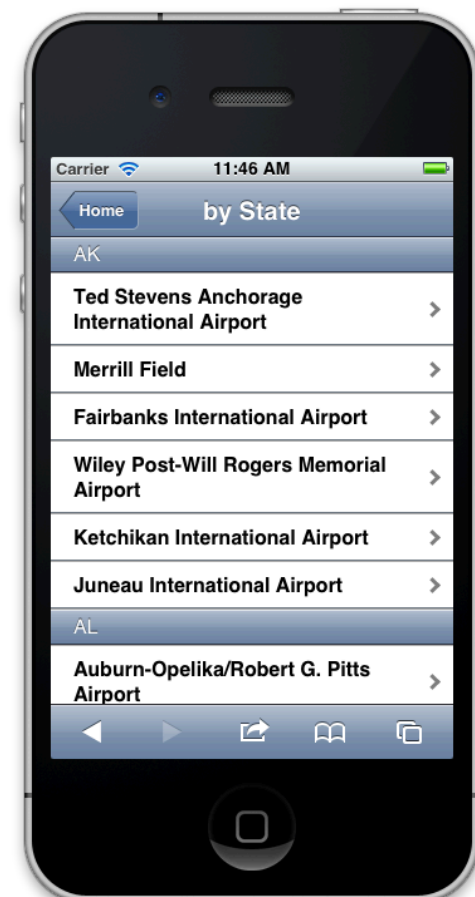
The Completed US Airports Application (cont.)



Main page

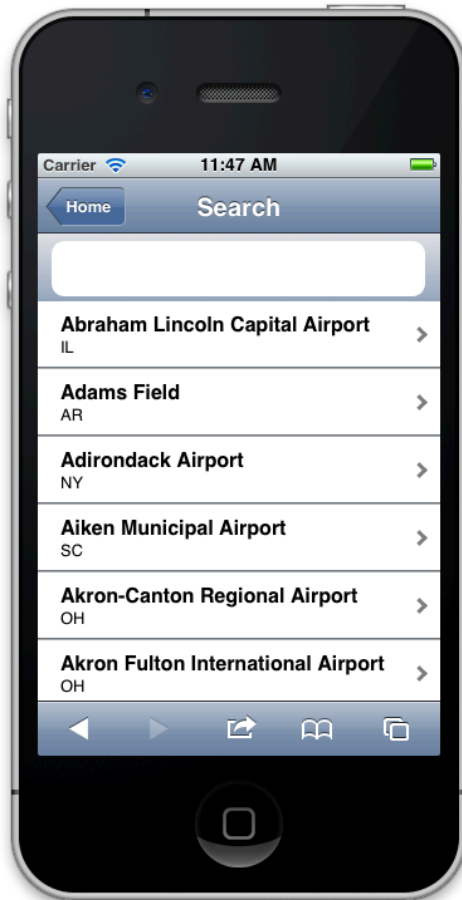


View by Name page

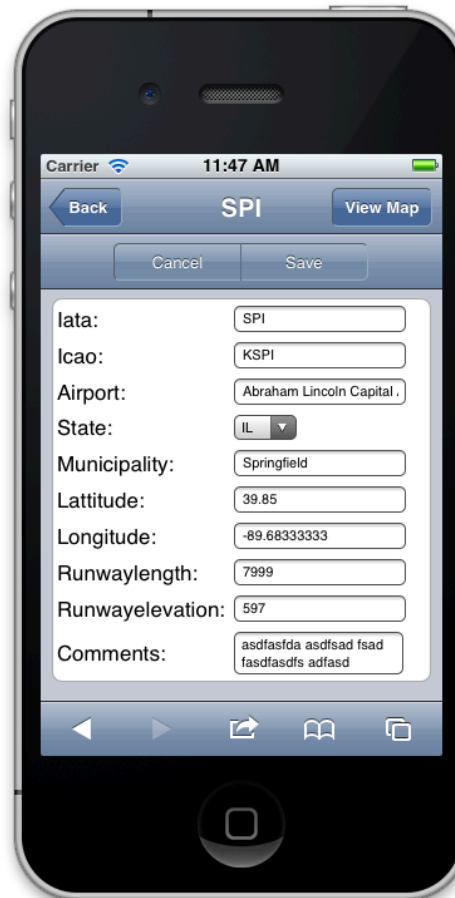


View by IATA page
(categorized view)

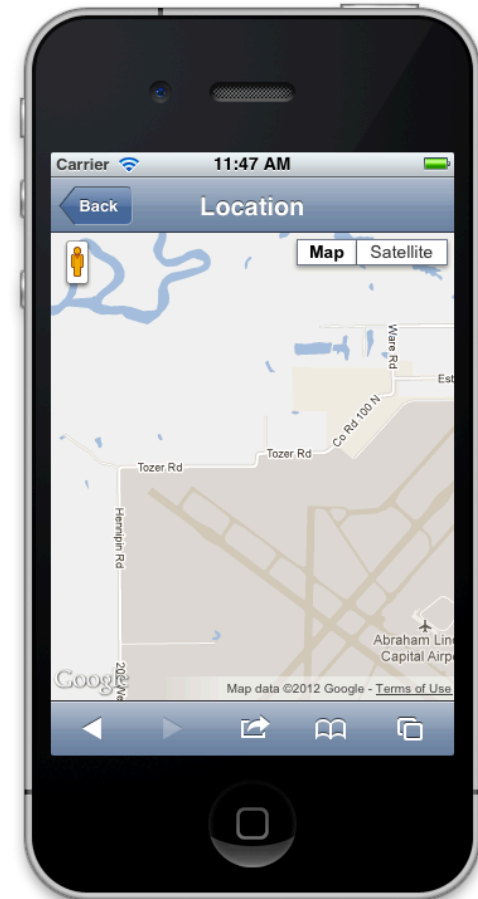
The Completed US Airports Application (cont.)



Search page

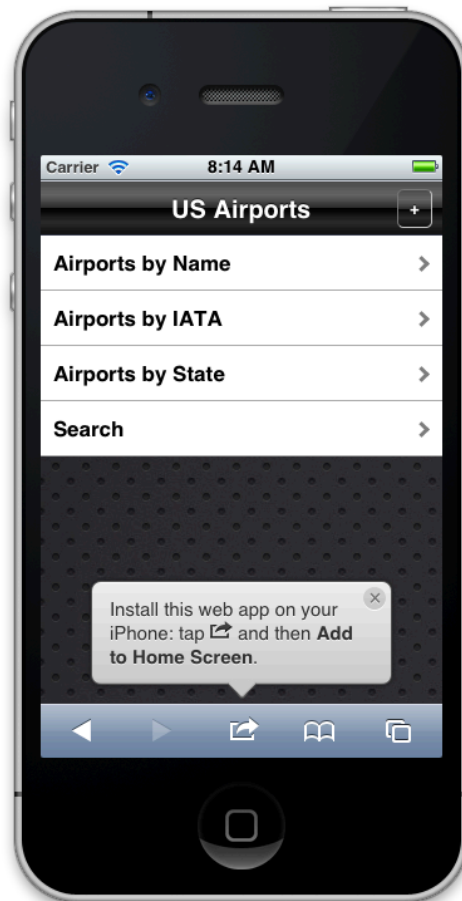


Edit document page

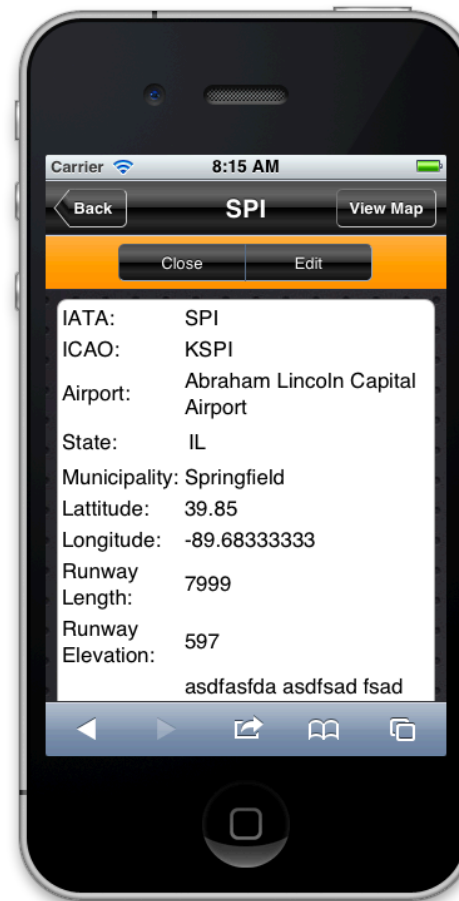


Google map page

The Completed US Airports Application (cont.)



Customized skin



Customized skin

The Completed US Airports Application (cont.)

Download the sample database:

**[https://www.dropbox.com/s/qzeumkm2vbvz701/
ADMIN_DEV_USAirports.nsf](https://www.dropbox.com/s/qzeumkm2vbvz701/ADMIN_DEV_USAirports.nsf)**

Customized skin

Customized skin

What We'll Cover ...

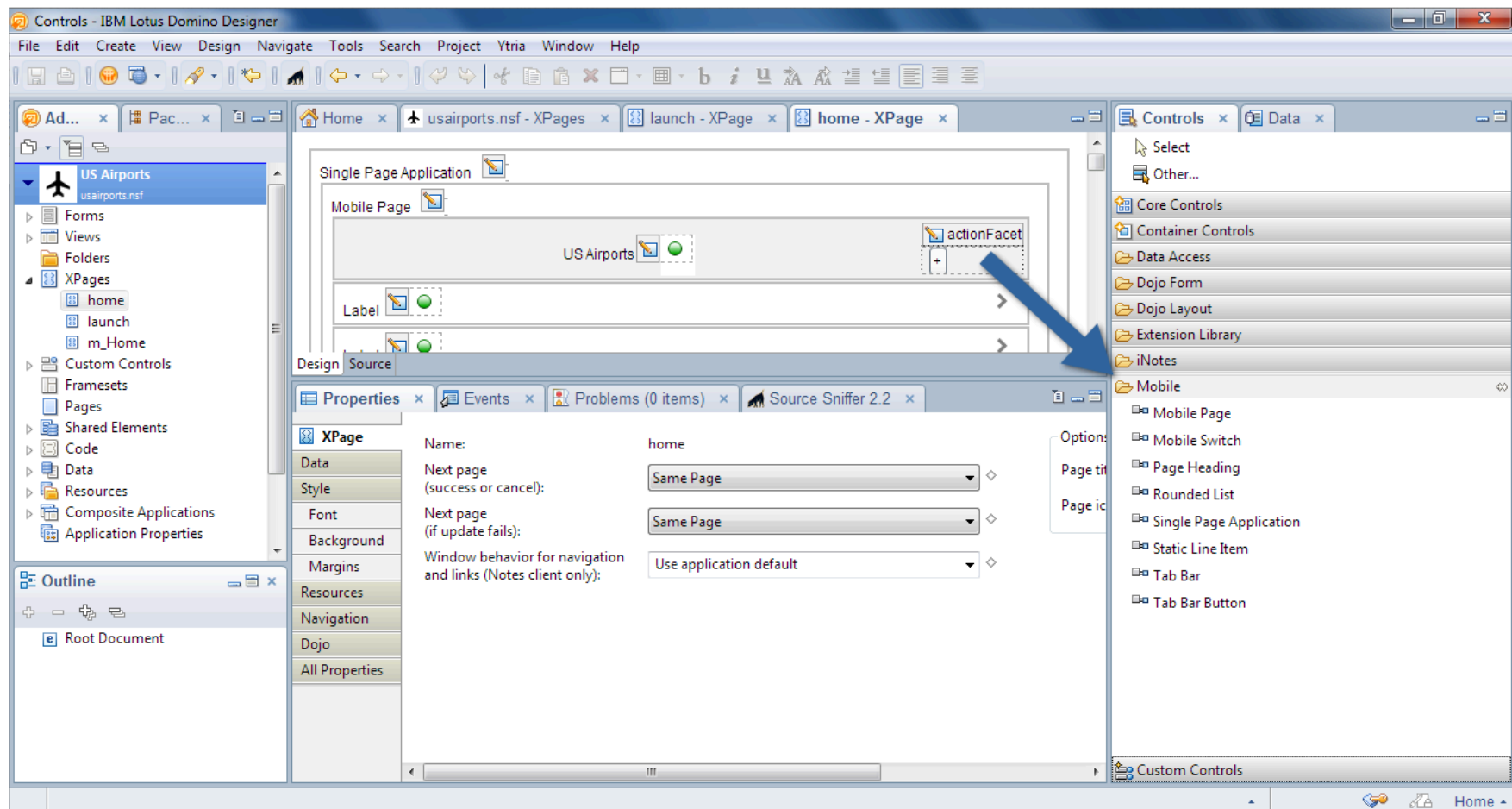
- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

Development Environment Requirements

- In order to get started with the XPages Mobile Controls, you will need to have the following:
 - ♦ Domino Designer 8.5.3 with Upgrade Pack 1 installed
 - ▶ You may also use the XPages Extension Library on OpenNTF.org
 - ♦ Access to a Domino 8.5.3 server with Upgrade Pack 1 installed
 - ♦ The Apple Safari browser to test your apps for iOS devices
 - ♦ The Apple iOS Emulator
 - ▶ Requires membership in the Apple Developer Network
 - ♦ The Chrome browser to test your apps for Android devices
 - ▶ <http://developer.android.com/sdk/index.html>

Meet the Mobile Custom Controls

- Once you have installed Upgrade Pack 1 on top of Domino Designer, you will see a new set of controls added to the controls palette



What We'll Cover ...

- Introducing the XPages Mobile Controls and our demo app
- Development environment requirements
- A review of each of the controls and what they do
- Configuring your application to be mobile
- The basic structure of a mobile page
- Working with views
- Editing data in documents
- Customizing the user interface using CSS
- More tips and tricks to make users love your mobile apps
- Meet TSAzr – an XPages-powered native iPhone app
- Using Appcelerator to build the native front-end user experience
- How we used XPages to build the back-end services
- Wrap-up

The Mobile Controls

- **UP1 adds 8 library controls to the Mobile palette in Domino Designer**
- **While these are the “core” controls that you will use to build apps, there are others from the Core Controls and Extension Library (included with UP1) that you can also use**
 - ♦ **Single Page Application**
 - ♦ **Mobile Page**
 - ♦ **Page Heading**
 - ♦ **Rounded List**
 - ♦ **Static Line Item**
 - ♦ **Tab Bar**
 - ♦ **Tab Bar Button**
 - ♦ **Mobile Switch**

The Mobile Controls (cont.)

- **Single Page Application**

- ♦ This control is used as the overall “container” for your mobile application
- ♦ This control will appear on your XPage markup as `xe:singlePageApp`

- **Mobile Page**

- ♦ This control will be used to display the actual application screens in your application. You most likely will have several Mobile Page controls within a “Single Page Application Control.”
- ♦ This control will appear in your XPage markup as `xe:appPage`
- ♦ Your Mobile Page controls may contain views, documents, and other elements

The Mobile Controls (cont.)

- **Page Heading**

- ♦ This control provides your mobile pages with title bars. Title bars can contain navigation, page titles, and other elements.
- ♦ This control will appear in your XPages markup as `xe:djxmHeading`

- **Rounded List**

- ♦ This control provides a panel that has rounded corners
- ♦ Use this control to give elements in your application a nicer look
- ♦ This control will appear in your XPages markup as `xe:djxmRoundRectList`

The Mobile Controls (cont.)

- **Static Line Item**

- The static line item control provides a way to provide links to other elements in your application
- The home screen in the US Airports app uses 4 Static Line Item controls to provide links to the three views, and another for a link to the Search page
- This control also has additional properties that allow you to include icons and additional text labels
- This control will appear in your markup as `xe:djxmLineItem`

The Mobile Controls (cont.)

- **Mobile Switch**

- ♦ This control provides an on/off switch
- ♦ Use this control to allow users to turn on and off application preferences
- ♦ This control will appear in your XPage markup as `xe:djxmSwitch`

- **Tab Bar**

- ♦ The Tab Bar is a container control and typically holds Tab Bar Buttons
- ♦ This control can be used to hold buttons to Edit and Save documents. It can also be used to switch between panels.
- ♦ This control will appear in your XPage markup as `xe:tabBar`

The Mobile Controls (cont.)

- **Tab Bar Button**

- Tab bar buttons are used within the Tab Bar Control
- They are like the XPage Button control; however, they have styling that matches mobile devices
- This control will be appear in your XPage markup as `xe:tabBarButton`

What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

Configuring Your Application to Be Mobile

- The XPage runtime environment will apply a mobile theme that is specific to the platform your app is rendering on
- There are a few things that you will need to do before creating your first mobile-enabled XPage
 - ♦ In Domino Designer, you will need to ensure that your application is configured to use Upgrade Pack 1 (or the XPages Extension Library)
 - ▶ Under Application Properties in Domino Designer
 - ▶ Locate the XPages Libraries section in the lower right
 - ▶ Check the box next to `com.ibm.xsp.extlib.library`
 - ▶ This will automatically be checked for you when you drag the Single Application Page onto your XPage

Configuring Your Application to Be Mobile (cont.)

- Here is what you will see in Domino Designer:

Advanced Properties

Unread Marks

☒ Maintain unread marks

Replicate:

Space Savers

☐ Compress database design

☐ Compress document data

☐ Use LZ1 compression for attachments

☐ Use Domino Attachment and Object Service

Advanced Options

☐ Optimize document table map

☐ Don't overwrite free space

☐ Maintain LastAccessed property

☐ Disable transaction logging

Limit entries in \$UpdatedBy fields:

Limit entries in \$Revisions fields:

Features

☐ Allow more fields in database

☐ Don't allow simple search

☐ Allow soft deletions

Permanent document deletion interval (hours):

☐ Support Response Thread History

☐ Don't support specialized response hierarchy

☐ Disable automatic updating of views

☐ Disable export of view data

☐ Don't allow headline monitoring

Allow Domino Data Service:

XPage Libraries

Select the libraries of extended XPage controls to use in this application.

Library ID
<input checked="" type="checkbox"/> com.ibm.xsp.extlib.library
<input type="checkbox"/>
<input type="checkbox"/>
<input type="checkbox"/>

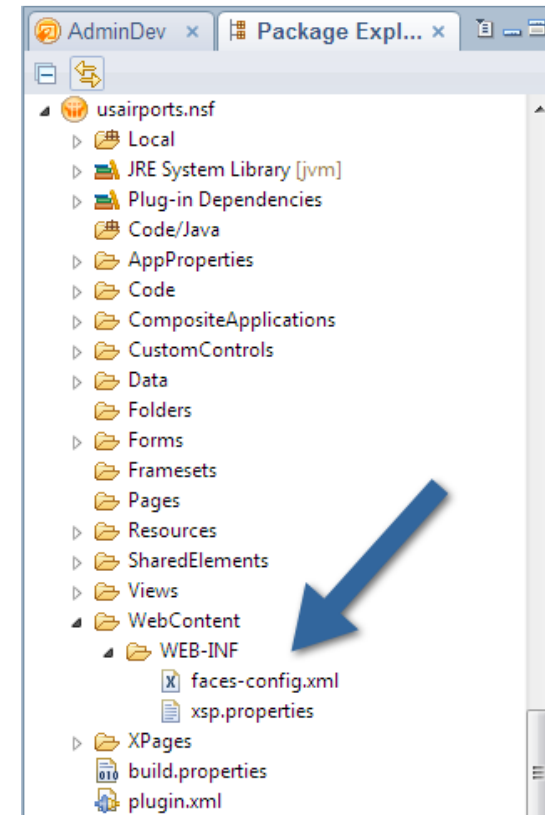
When running on the Web, the libraries must be available on the server. When running on the Notes client, the library plug-ins must be installed on the client.

Configuring Your Application to Be Mobile (cont.)

- The next thing you will need to do is to tell the XPages runtime when it should apply the mobile page themes, and to which pages
- This is done by adding a line to the xsp.properties file
- In order to modify the xsp.properties file, you will need to open up the XPages perspective in Domino Designer
- You can also access the Package Explorer through the “Show Eclipse Views” menu by selecting “Package Explorer”
- In the Package Explorer, navigate to the WebContent\WEB-INF folder
- Here you will see the xsp.properties file

Configuring Your Application to Be Mobile (cont.)

- Double click on the `xsp.properties` file to edit the file
- The XPage Properties page will be displayed
- At the bottom, you will see a tab labeled “Source”
- Click on this tab to open the properties file source
- Here is where we are going to need to add a property entry
- This property entry will tell the XPages runtime which pages to apply mobile theming to



Configuring Your Application to Be Mobile (cont.)

- The `xsp.theme.mobile.pagePrefix` property defines the XPage page prefix that your XPage page names start with
- XPages that do not begin with this prefix will render using the theme defined by the `xsp.theme` property
 - Typically `oneui2.1` or one your company has created
- In the US Airports demo application, we use “`m_`” as the page prefix
 - This means that the `m_Home.xsp` XPage that we will be creating will have a mobile theme rendered for it

```
xsp.ajax.renderwholetree=false  
xsp.library.depends=com.ibm.xsp.extlib.library  
xsp.persistence.mode=file  
xsp.resources.aggregate=true  
xsp.theme=oneui2.1  
xsp.theme.mobile.pagePrefix=m_
```

Configuring Your Application to Be Mobile (cont.)

- One last thing you may have noticed in the previous slide that is important is the `xsp.resource.aggregate=true` entry
- This property is new with Domino 8.5.3, and will combine all of your JavaScript and CSS resources into fewer files
- This reduces the number of GET requests that your application needs to perform
- This option is located in the XPages Properties page under the “Persistence” tab
 - ♦ In the Options section, check the option that says “Use runtime optimized JavaScript and CSS resources”
 - ♦ This will set the value of `xsp.properties.aggregate` to true

Configuring Your Application to Be Mobile (cont.)

- Setting the “Use runtime optimized JavaScript and CSS resources” option

xsp.properties - usairports.nsf

Performance Properties

Persistence Options

Server page persistence: **Keep pages on disk**

Maximum pages in memory:

Page persistence mode: **Server default**

Maximum pages on disk:

☐ GZip persisted files

☒ Persist files asynchronously

Save to memory instead when less than (x) bytes:

Options

☐ Evaluate the entire page on partial refresh

☒ Use runtime optimized JavaScript and CSS resources

☒ Discard JavaScript context after each page

Compiled JavaScript cache size: (expressions)

Compiled XPath cache size: (expressions)

Global resource expiration: (days)

Temporary Locations

Attachment:

Upload:

Pages:

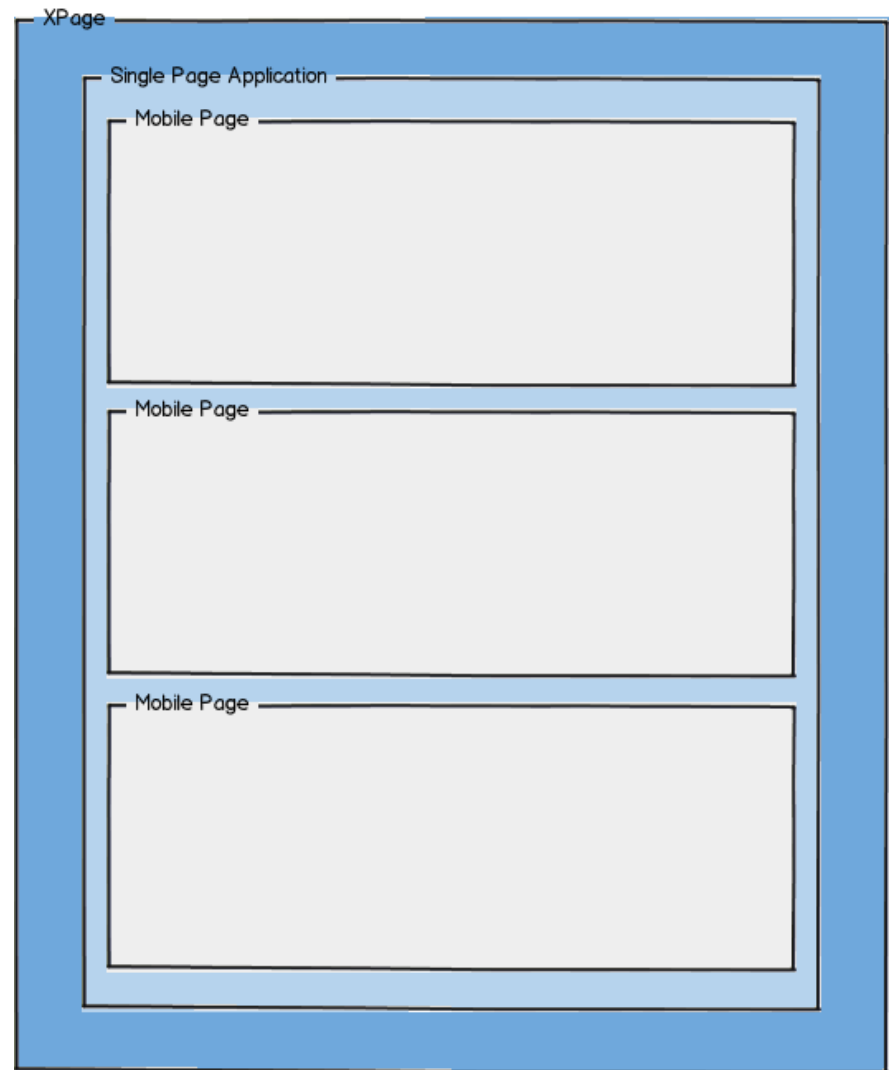
General Persistence Page Generation Source

What We'll Cover ...

- Introducing the XPages Mobile Controls and our demo app
- Development environment requirements
- A review of each of the controls and what they do
- Configuring your application to be mobile
- The basic structure of a mobile page
- Working with views
- Editing data in documents
- Customizing the user interface using CSS
- More tips and tricks to make users love your mobile apps
- Meet TSAzr – an XPages-powered native iPhone app
- Using Appcelerator to build the native front-end user experience
- How we used XPages to build the back-end services
- Wrap-up

The Basic Structure of a Mobile Page

- When we introduced the mobile controls, we met the Single Page Application
- There can only be one of these controls on an XPage
- The US Airports app has an XPage named m_Home
- On the m_Home XPage, there is one Single Page Application control and 7 Mobile Page controls contained within it



The Basic Structure of a Mobile Page (cont.)

- Here is the US Airports app page structure:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core"
3   xmlns:xe="http://www.ibm.com/xsp/coreex"
4   xmlns:xc="http://www.ibm.com/xsp/custom" createForm="true"
5   pageTitle="US Airports">
6+ <xp:this.resources>[]
34 </xp:this.resources>[]
35 <!-- Mobile Application Container -->
36 <xe:singlePageApp id="singlePageApp1" selectedPageName="home"
37   loaded="true">
38   <!-- Home Page -->
39+ <xe:appPage id="appPage1" pageName="home">[]
73 </xe:appPage>[]
74   <!-- By Name View Page -->
75+ <xe:appPage id="appPage2" pageName="byName" keepScrollPos="true">[]
104 </xe:appPage>[]
105   <!-- By IATA View Page -->
106+ <xe:appPage id="appPage3" pageName="byIATA">[]
137 </xe:appPage>[]
138   <!-- By State View Page -->
139+ <xe:appPage id="appPage4" pageName="byState">[]
171 </xe:appPage>[]
172   <!-- Airport Document Page -->
173+ <xe:appPage id="appPage5" pageName="airportDocument">[]
430 </xe:appPage>[]
431   <!-- Search Page -->
432+ <xe:appPage id="appPage6" pageName="search">[]
435 </xe:appPage>[]
436   <!-- Google Map Page -->
437+ <xe:appPage id="appPage7" pageName="map" resetContent="true">[]
492 </xe:appPage>[]
493 </xe:singlePageApp>
494 </xp:view>
```

The Basic Structure of a Mobile Page (cont.)

- The US Airports app has the following 7 mobile pages. Each one was created with a Mobile Page control:
 - The Home page (pageName = “home”)
 - The By Name View page (pageName = “byName”)
 - The By IATA View page (pageName = “byIATA”)
 - The By State View page (pageName = “byState”)
 - The Airport document page (pageName = “airportDocument”)
 - The Search page (pageName = “search”)
 - And finally, the Google Map Page (pageName = “map”)
- The pageName property of each page is the name of the Mobile Page control name that should be used to display that page

Creating the m_Home XPage

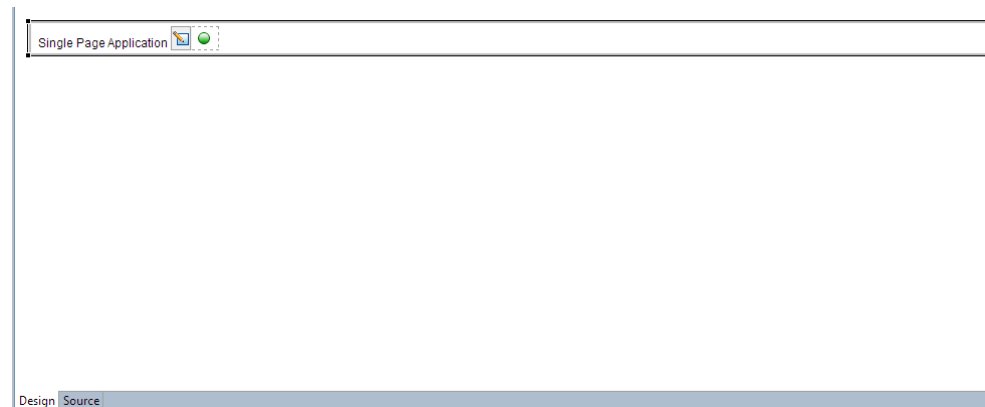
- Create a new XPage and call it m_Home
- On this page, drag from the Mobile Controls palette the Single Page Application
- There are a few properties that you will need to set for this control
 - ♦ This control needs to know which Mobile Page control it should load when the m_Home page is displayed
 - ♦ In the selected pageName property, set this to the Mobile Page name “home”
 - ▶ Note that we have not yet created this Mobile Page, but will in the next steps
- That’s all you need to configure for this control
- Let’s now create the Home page

The Single Page Application Control

- After you drag the Single Page Application control onto your XPage, the markup will look like this:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core"
3     xmlns:xe="http://www.ibm.com/xsp/coreex">
4     <xe:singlePageApp id="singlePageApp1" selectedPageName="home">
5     </xe:singlePageApp>
6 </xp:view>
7
```

- The XPage Design pane will look like this:



Adding a Mobile Page Control

- Now, you need to drag a Mobile Page control onto your Single Page Application control. Set the `pageName` property to “home.”
 - You can either work in the source editor or you can drag the page onto the Single Page Application facet
- Here is what the markup now looks like:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core"
3     xmlns:xe="http://www.ibm.com/xsp/coreex">
4     <xe:singlePageApp id="singlePageApp1" selectedPageName="home">
5         <xe:appPage id="appPage1" pageName="home">
6
7         </xe:appPage>
8     </xe:singlePageApp>
9 </xp:view>
```

- And in the Design view:



Creating the Home Mobile App Page

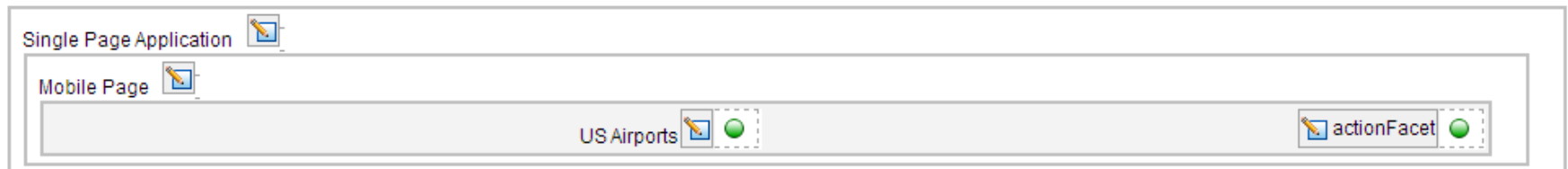
- **As we saw earlier, the Home screen contains 4 things for the user to do:**
 - ♦ **View airports by name**
 - ♦ **View airports by IATA**
 - ♦ **View airports by state**
 - ♦ **Search airports**
- **This screen also contains a “heading,” which tells the user what page they are on. The US Airports home page has its label for the title set to “US Airports.”**
 - ♦ **Start by dragging a Page Heading control into your “home” mobile page and setting the label to “US Airports”**
 - ♦ **In a later slide, we will be adding a button to this page that allows for a new Airport document to be created. We will use a button with a “+” sign to do this.**

Creating the Home Mobile App Page (cont.)

- If we look at the markup, we now see the heading added in

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core" xmlns:xe="http://www.ibm.com/xsp/coreex".
3     <xe:singlePageApp id="singlePageApp1"
4         selectedPageName="home">
5         <xe:appPage id="appPage1" pageName="home">
6             <xe:djxmHeading id="djxmHeading1" label="US Airports">
7             </xe:djxmHeading>
8         </xe:appPage>
9     </xe:singlePageApp>
10 </xp:view>
```

- If we look at the Design Pane, you will now see some extra facets that can be used to add items to the heading (Note the actionFacet). This is where we will place the “+” sign button.



Creating the Home Mobile App Page (cont.)

- Now, we need to add 4 Static Line Item controls for the four things we are going to allow our users to do
 - ♦ Drag four Static Line Item controls below the heading
 - ♦ For each of the controls, you will need to set the following properties:
 - ▶ **The label property**
 - *Airports by Name*
 - *Airports by IATA*
 - *Airports by State*
 - *Search*
 - ▶ **The transition property**
 - *There are three transitions that you can use for page-to-page transitions (slide, flip, fade)*

Creating the Home Mobile App Page (cont.)

- Now, we need to add 4 Static Line Item controls for the four things we are going to allow our users to do
 - For each of the controls, you will need to set the following properties: (cont.)
 - ▶ **The moveTo property**
 - *This property is used to identify which Mobile Page should be navigated when the user selects the line item*
 - *When you specify the page to move to, you will need to preface the name with a # sign*
 - For the US Airports app the values should be set to “#byName,” “#byATA,” “#byState,” and “#search,” respectively
- Note that we still have 6 more mobile pages to create for the US Airports app!
- Let's look at the source and the Design pane to see where we are at on the next slide

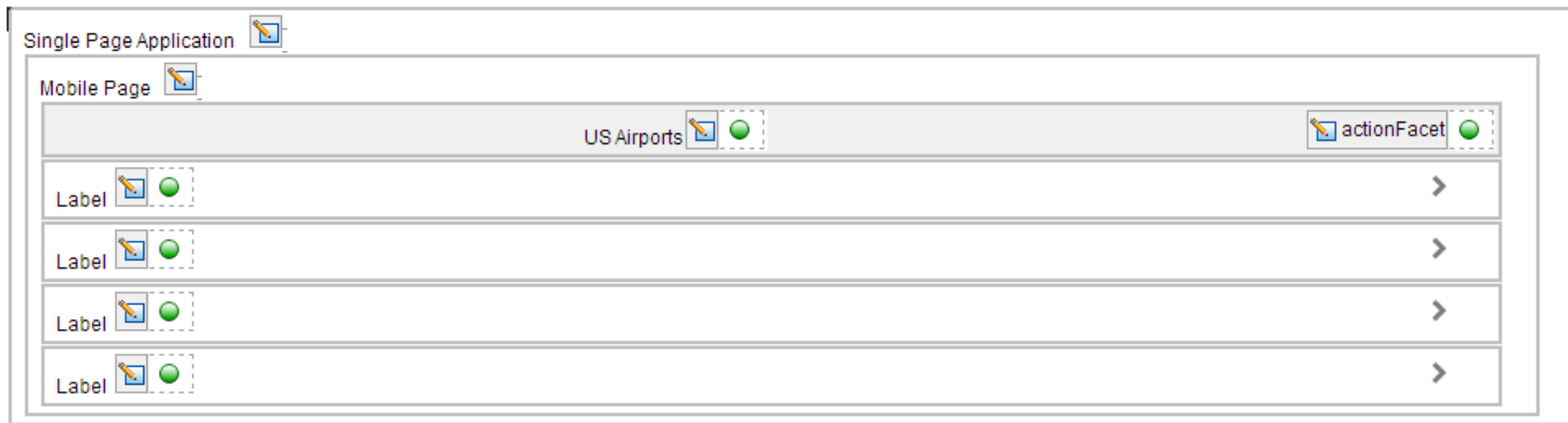
Creating the Home Mobile App Page (cont.)

- The source for the home page (not including the new document button we will be adding later):

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core" xmlns:xe="http://www.ibm.com/xsp/coreex">
3   <xe:singlePageApp id="singlePageApp1"
4     selectedPageName="home">
5     <xe:appPage id="appPage1" pageName="home">
6       <xe:djxmHeading id="djxmHeading1" label="US Airports">
7       </xe:djxmHeading>
8       <xe:djxmListItem id="djxmListItem1" label="Airports by Name"
9         moveTo="#byName"></xe:djxmListItem>
10      <xe:djxmListItem id="djxmListItem2" label="Airports by IATA"
11        moveTo="#byIATA"></xe:djxmListItem>
12      <xe:djxmListItem id="djxmListItem3" label="Airports by State"
13        moveTo="#byState"></xe:djxmListItem>
14      <xe:djxmListItem id="djxmListItem4" label="Search"
15        moveTo="#search">
16      </xe:djxmListItem>
17    </xe:appPage>
18  </xe:singlePageApp>
19 </xp:view>
```

Creating the Home Mobile App Page (cont.)

- Looking at our home mobile page through the Design pane:



- Now that we have this page built, let's move on and see how we can incorporate views
- 3 of the 4 Static Line Items we are providing in the US Airports app are constructed in much the same way. All three are showing data from existing Notes views.
- You will soon see that the “Airports by State” view is a categorized view

What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

Working with Views

- The Mobile Controls make it very easy to add views to your mobile applications
- In this section, we are going to use the “Data View” component from the Extension Library on our Mobile Pages
- Remember, we are still working from our m_Home XPage, and will be adding 3 Mobile Page controls to support the Airports by Name, Airports by IATA, and Airports by State views
- First, we will create the “Airports by Name” mobile page
 - The “Airports by IATA” page construction will be skipped in this slide deck
 - The process for creating this page is identical to the “Airports by Name” page, with the exception of the view being used

Working with Views (cont.)

- In Designer, go to the source pane of your m_Home XPage and drag in 3 Mobile Page controls
- Ensure that these controls are created within the Single Page Application
- Assign a name for each mobile page using the `pageName` property in the All Properties view
 - ♦ `byName`
 - ♦ `byATA`
 - ♦ `byState`
- We also need to set the property “resetContent” to true in the “basics” section of All Properties
 - ♦ Setting this property will ensure that any documents in the view that are added or deleted will be reflected in the contents of the view

Working with Views (cont.)

- Here is the XPage markup after adding the 3 new mobile pages:

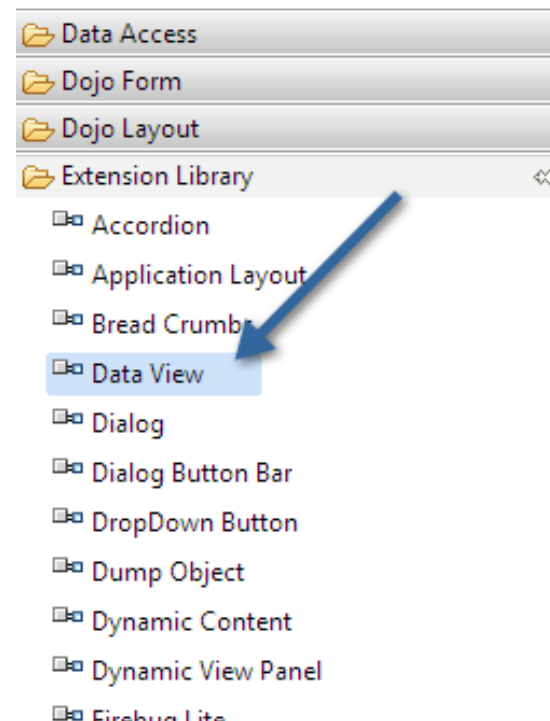
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core" xmlns:xe="http://www.ibm.com/xsp/coreex">
3   <xe:singlePageApp id="singlePageApp1"
4     selectedPageName="home">
5     <xe:appPage id="appPage1" pageName="home">
17   </xe:appPage>
18   <xe:appPage id="appPage2" pageName="byName" resetContent="true"></xe:appPage>
19   <xe:appPage id="appPage3" pageName="byIATA" resetContent="true"></xe:appPage>
20   <xe:appPage id="appPage4" pageName="byState" resetContent="true"></xe:appPage>
21   </xe:singlePageApp>
22 </xp:view>
```

Building a Mobile Page

- Now that we have created the 3 mobile pages for our views, let's include the other controls we need to provide for the user of our app
 - ♦ Drag a Page Heading control onto the mobile page
 - ♦ Set the label to the name of the view (by Name, by State, etc.)
 - ♦ Set the “back” property to “Home.” This will give the user a back button in the heading.
 - ♦ Set the “moveTo” property to “#home.” This is the name we gave to the first mobile page we added.
 - ▶ We will soon show you how to programmatically set the moveTo property for the back button
- Now, we need to get our view data onto the mobile page

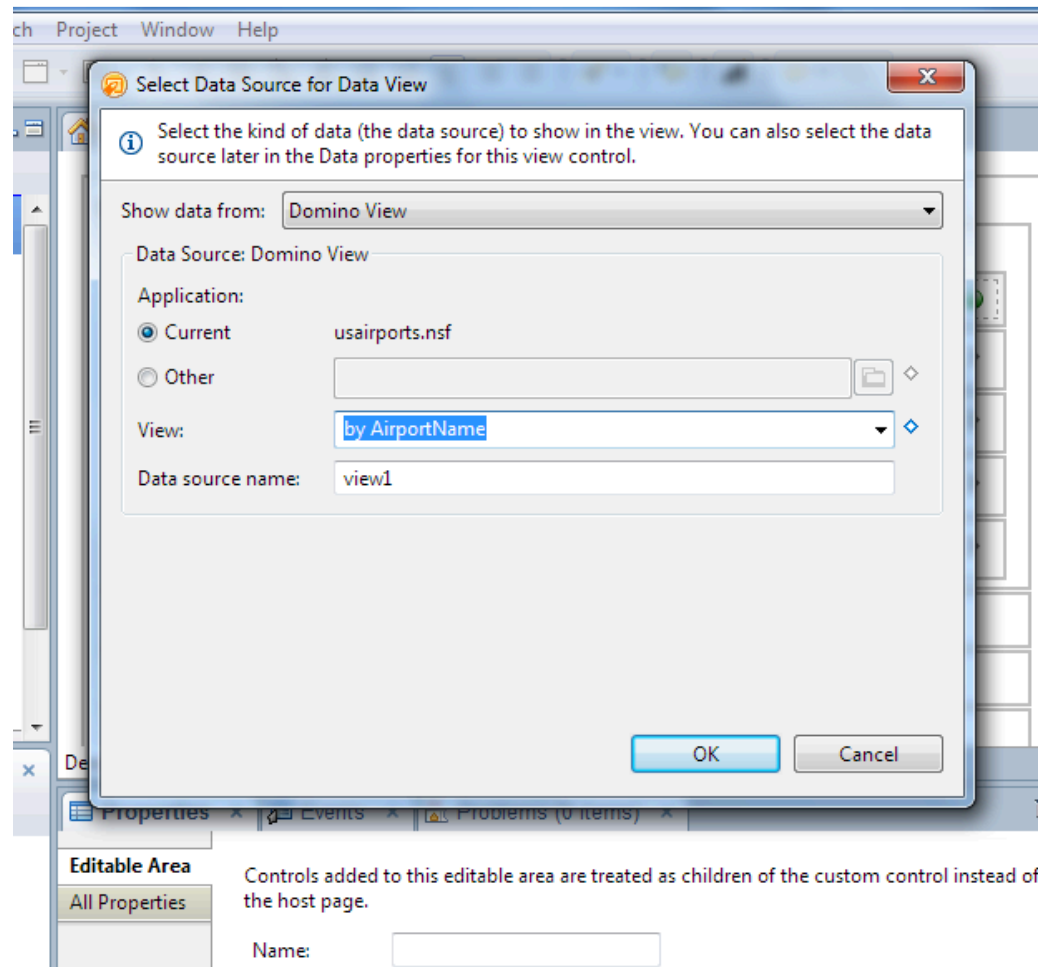
Adding a Data View Control

- Directly below the Page Header control on your mobile page you just added, drag a Data View control onto the page
 - ♦ Remember, you can also drag onto the Mobile Page facet from the Design Pane view
- This control is located in the Extension Library section of the Controls Palette



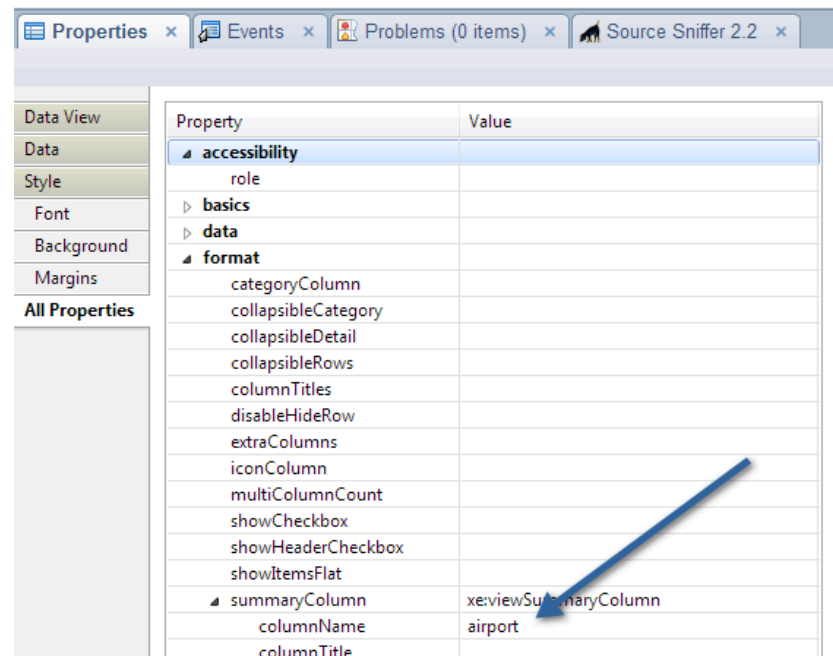
Adding a Data View Control (cont.)

- After dragging the control onto the mobile page facet, we are presented with a dialog box asking us to select the view to be used



Adding a Data View Control (cont.)

- What we haven't done yet is specified the data we want to show on the view
- This is done by setting the “summaryColumn columnName” property in the data property of the Data View control
 - ♦ Since we want to show the name of the airport, set the value of this property to “Name”

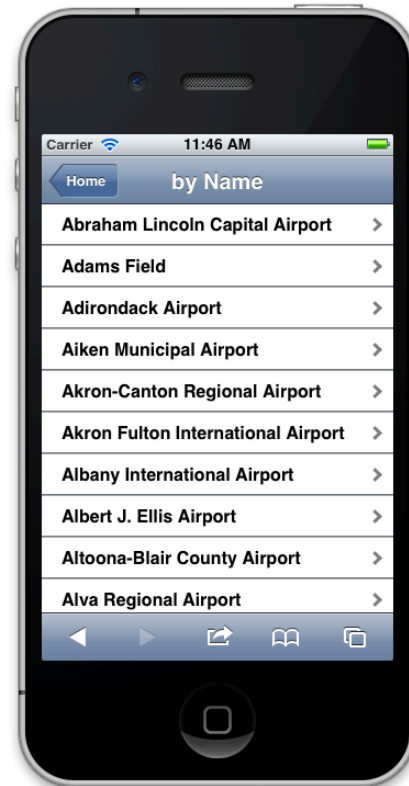


Setting the Data Views Properties

- Now that the view has been added, there are several properties that need to be set
 - ♦ The `pageName` property will need to contain the name of the mobile page that should be used to display documents for this view
 - ▶ Set this name to “`#airportDocument`.” We have not yet created this mobile page.
 - ▶ The next property to set is “`openDocAsReadonly`,” which should be set to true
 - *This property will force the document to opened up in read mode, rather than edit mode*
 - *Set this to false to open the document immediately for editing*

The Completed byName Mobile Page

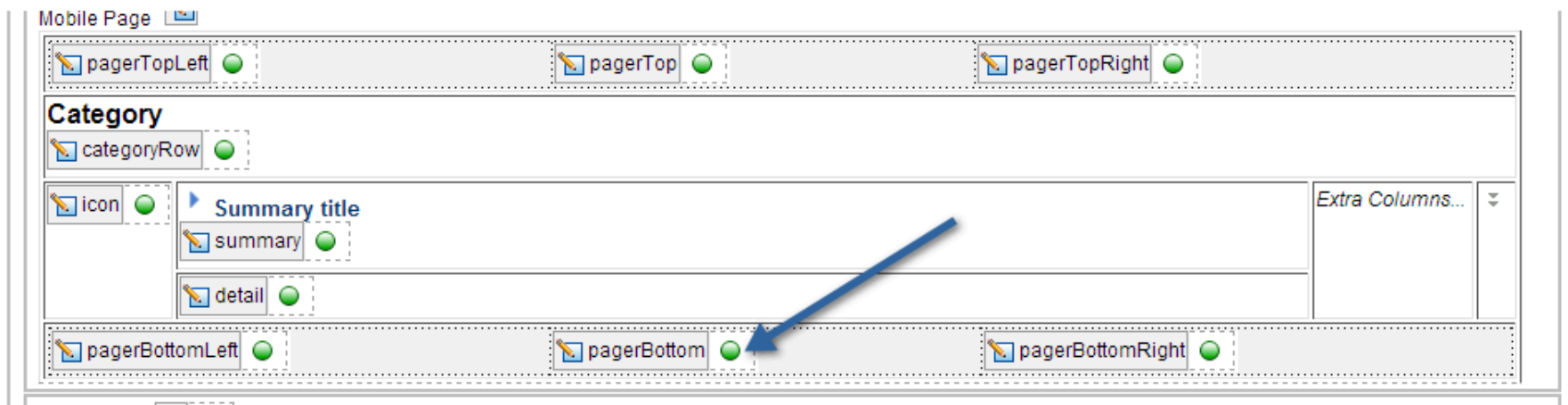
- If we now preview our XPage in the iOS simulator, we will see the Airports by Name view



- The mobile controls will not display the contents of the entire view at one time
- We will need to add a “More Airports ...” button to the view

Adding a Link to Display More View Items

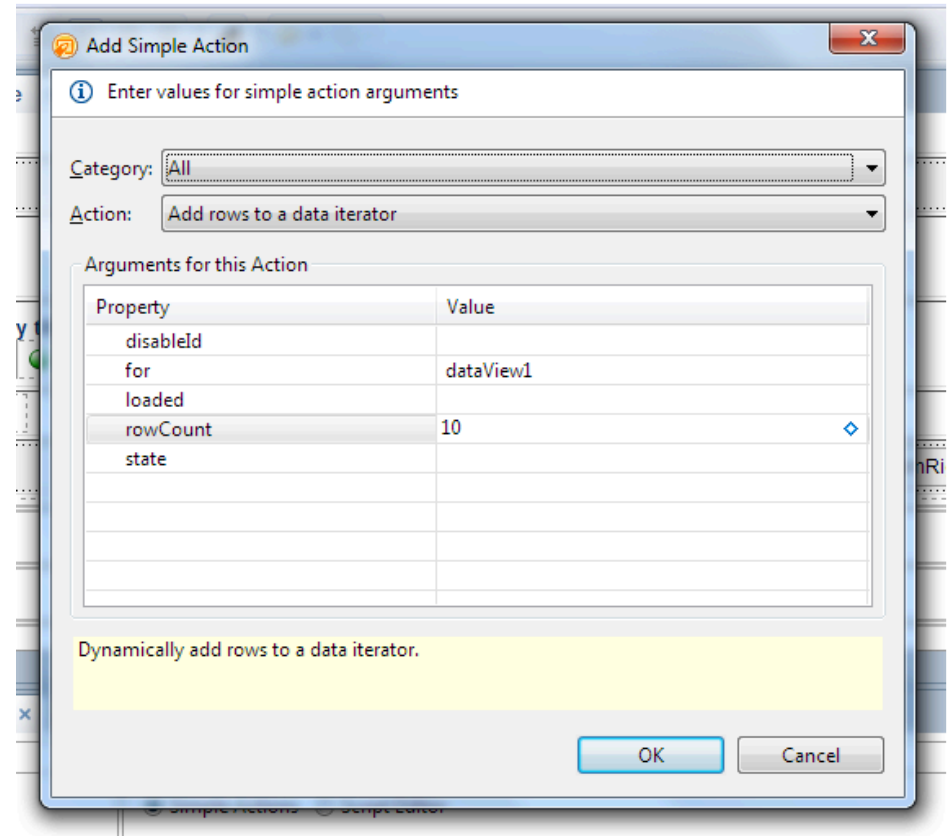
- To add “More Airports ...” to the bottom of the view, drag a “link” control from the Core Controls palette onto the pagerBottom facet (the link will appear as a button on the page)



- Set the label for the link control to “More Airports ...”
- We will now need to configure what happens when a person taps on the “More Airports ...” link

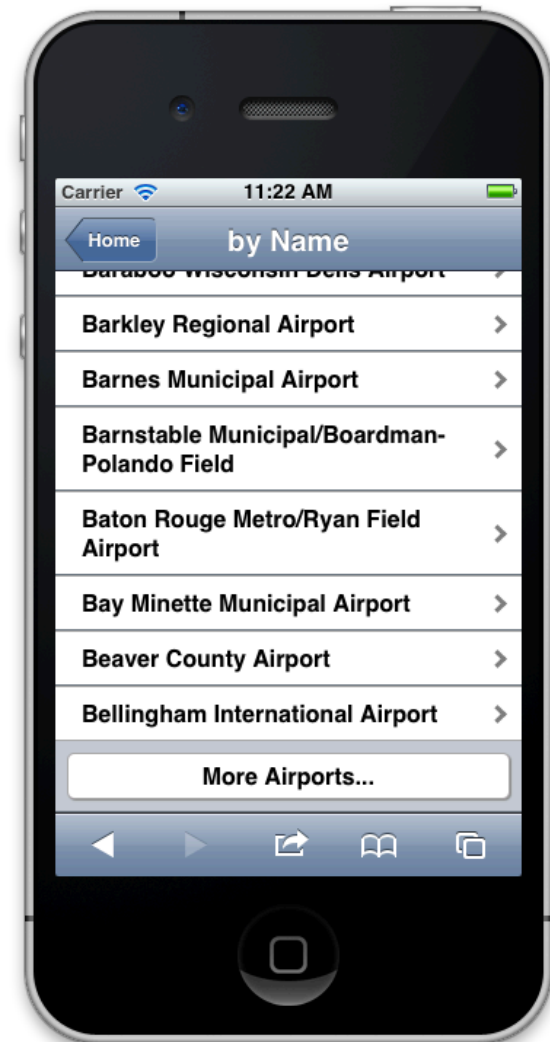
Adding a Link to Display More View Items (cont.)

- In the “Events” tab of the Link, select the “Client” tab and then click on the “Add Action” button
- This will bring up a dialog box where you can specify what happens when “More Airports ...” is tapped
 - ♦ Set the “for” property to `dataView1`
 - ♦ Set the `rowCount` to the number of additional rows you want to display



Previous and Next Buttons

- As you can now see, we have a way for the user to display more data as they need it
 - ♦ You can also choose to display Previous and Next buttons
 - ♦ Use the `pagerBottomLeft` and `pagerBottomRight` facets in the View Control to add these buttons
 - ▶ Remember you can drag and drop Link controls onto the View Control or you can drag them into the XML source

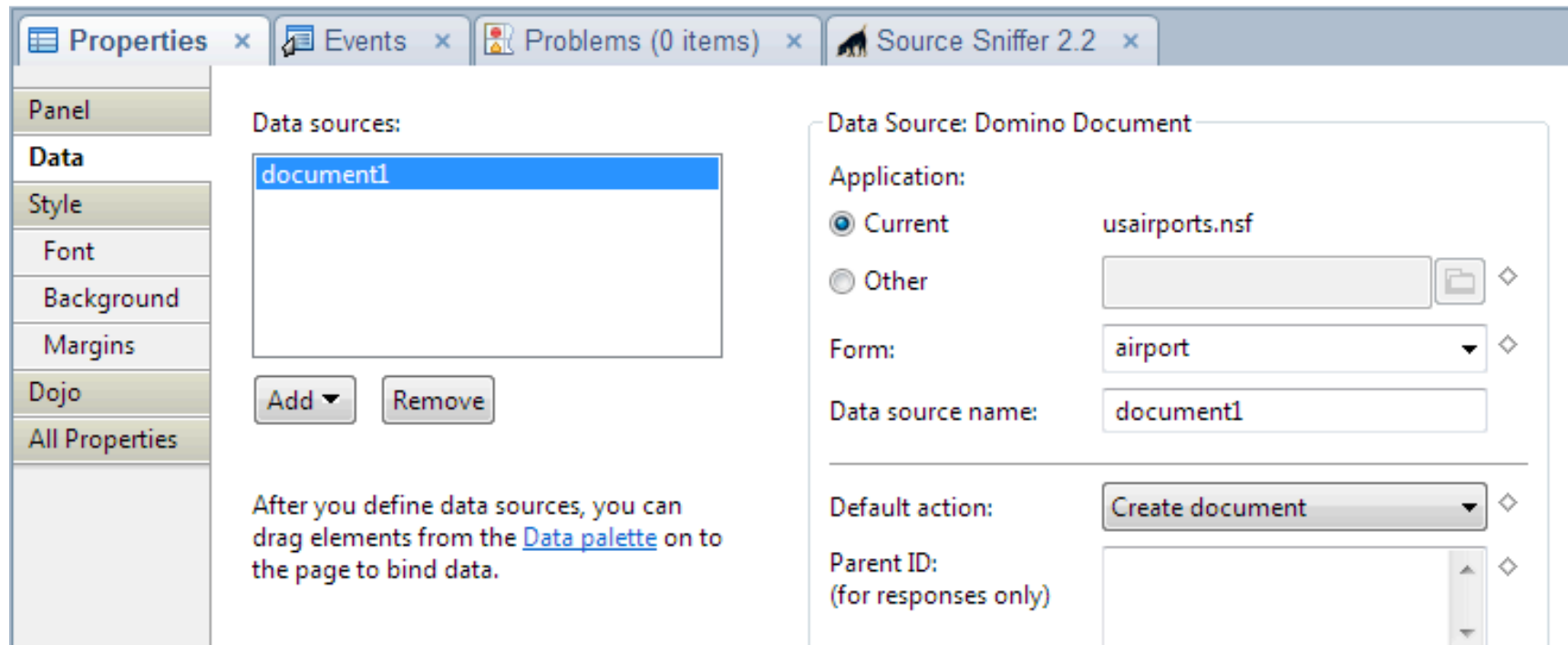


Displaying Document Data

- Now that we have created a mobile page with a view on it, we need a way to display the contents of a document
- Remember earlier that we defined the `pageName` to display for the “by Name” view to be “`#airportDocument`”
- Now, we need to create a new Mobile Page and assign its `pageName` to “`airportDocument`”
- The `airportDocument` page is going to display:
 - ♦ IATA, Name, State, Municipality, Latitude, Longitude, Runway Length, Runway Elevation, and Comments
- Let's start by adding a panel control to our mobile page. We will bind this panel to our airport form using a Data Source.
 - ♦ Place the panel control at the top of your mobile page

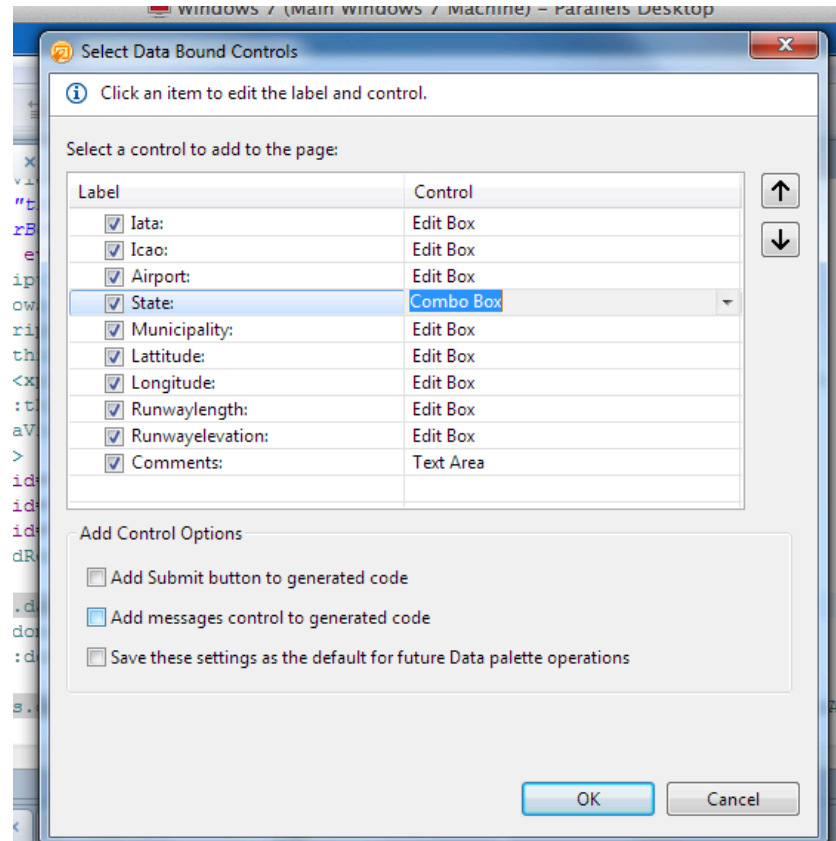
Displaying Document Data (cont.)

- Under the Data section of the panel, you can bind a document data source



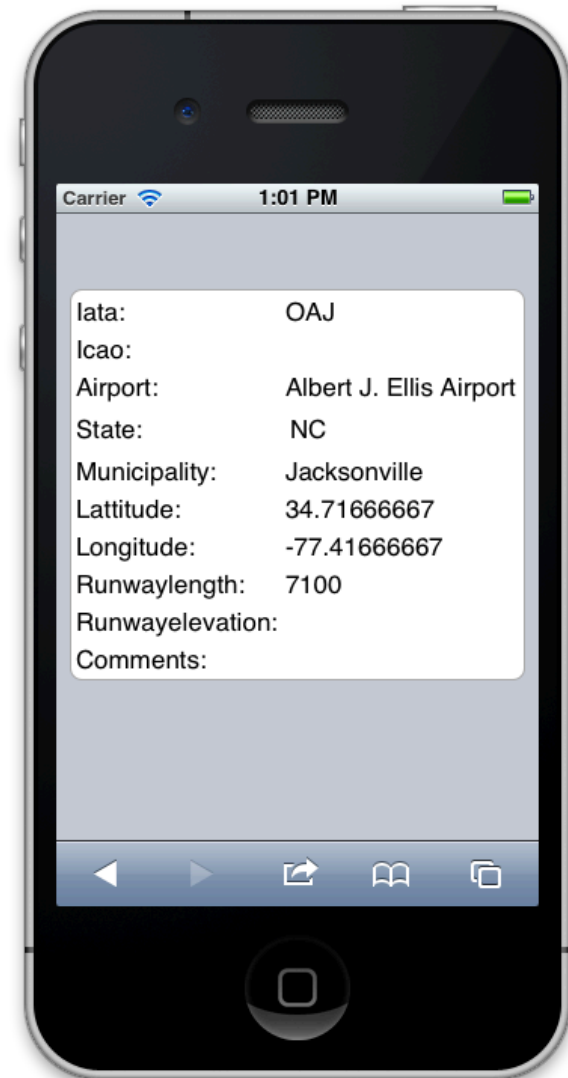
Displaying Document Data (cont.)

- To make the presentation of the document data nicer, drag a Rounded List control below the Page Heading
- Now, within the Rounded List, drag from the Data panel all of the fields from the airport form that you want to be displayed



Displaying Document Data (cont.)

- When we display a tap on an airport name from the view, we now see a document!
- But what's missing?
 - ♦ A Page Header with a back button
 - ♦ A Title for the page
 - ♦ How about the ability to Edit and Save the document?
- You already know how to add the header, and you can add a computed value for the title such as *document1.getItemValueString('iata')*



What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

Editing Documents

- Now that we can display documents, how can we give users the ability to edit them?
- To do this, we are going to use a Tab Bar Control
- Within the Tab Bar Control that we add to the airportDocument page (below the Page Heading), we will use 3 Tab Bar Button controls to give users the ability to:
 - Cancel the operation
 - Edit the document
 - Save the document if it's in edit mode
- Drag a Tab Bar Control out on your airportDocument mobile page, and then drag 3 Tab Bar Button controls within the tab bar
 - Set the property barType to have a value of *"segmentedControl"*

Adding a Cancel/Close Button

- First, let's look at the tab bar button for a button that allows the user to close a document that they are viewing or to cancel the editing of a document

```
<!--Start of Cancel/Close Button-->
<xe:tabBarButton id="tabBarButton1">
  <xe:this.label><![CDATA[#{javascript:@If(document1.isEditable(),"Cancel","Close");}]]>
</xe:this.label>
  <xp:eventHandler event="onClick" submit="true"
    refreshMode="complete" immediate="true">
    <xp:this.action>
      <xe:moveTo targetPage="home"
        transitionType="flip">
      </xe:moveTo>
    </xp:this.action>
  </xp:eventHandler>
</xe:tabBarButton>
```

- The value of the button label is calculated based on whether or not the document “is editable”
 - The **onClick** event will transition the user back to the “home” mobile page

Adding an Edit Button

- The Edit tab bar button should only be rendered if the user has the rights to edit the document
- Use the rendered property of the button to determine if the user can, in fact, edit the document. If they have the rights, display the button and, in its onClick event, set the mode to “edit.”

```
<!--Start of Edit Button-->  
<xe:tabBarButton id="tabBarButton3"  
    rendered="#{javascript:!document1.isEditable();}" label="Edit">  
    <xp:eventHandler event="onClick" submit="true"  
        refreshMode="complete">  
        <xp:this.action>  
            <xp:changeDocumentMode mode="edit"  
                var="document1">  
            </xp:changeDocumentMode>  
        </xp:this.action>  
    </xp:eventHandler>  
</xe:tabBarButton>
```

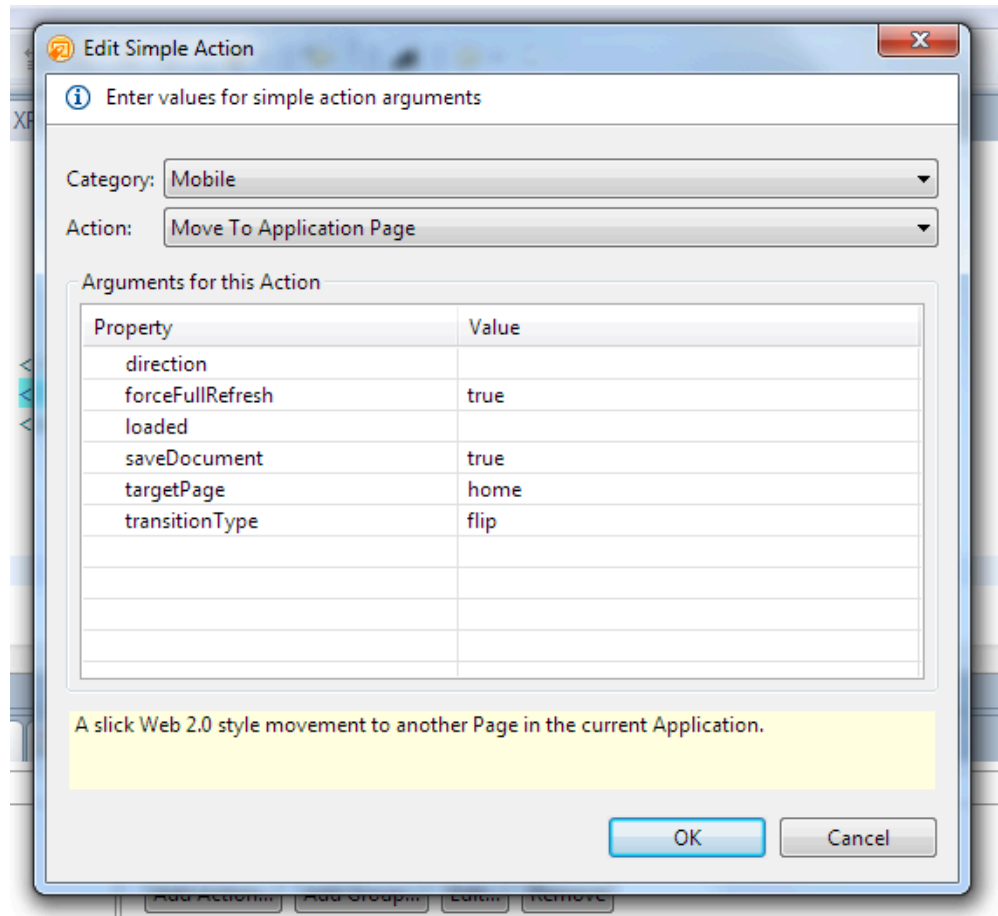

Adding a Save Button

- The Save button on the tab bar should only be rendered when the document is in edit mode
- When the Save button is clicked, we want to send the user back to the home screen
- We also send them back home with a “flip” transition

```
<!--Start of Save Button-->  
<xe:tabBarButton id="tabBarButton2" label="Save"  
    rendered="#{javascript:document1.isEditable();}">  
    <xp:eventHandler event="onClick" submit="true"  
        refreshMode="complete">  
        <xp:this.action>  
            <xe:moveTo transitionType="flip"  
                targetPage="home" saveDocument="true"  
                forceFullRefresh="true">  
            </xe:moveTo>  
        </xp:this.action>  
    </xp:eventHandler>  
</xe:tabBarButton>
```

The Move to Application Page (moveTo) Simple Action

- You may have noticed on the Cancel/Close and Save buttons that there is a `<xe:moveTo>` tag. You specify this action from the `onClick` event.
 - ♦ From the Server tab, click on “Add Action”
 - ♦ Here is where you specify what should happen when the button is clicked



The Back Button

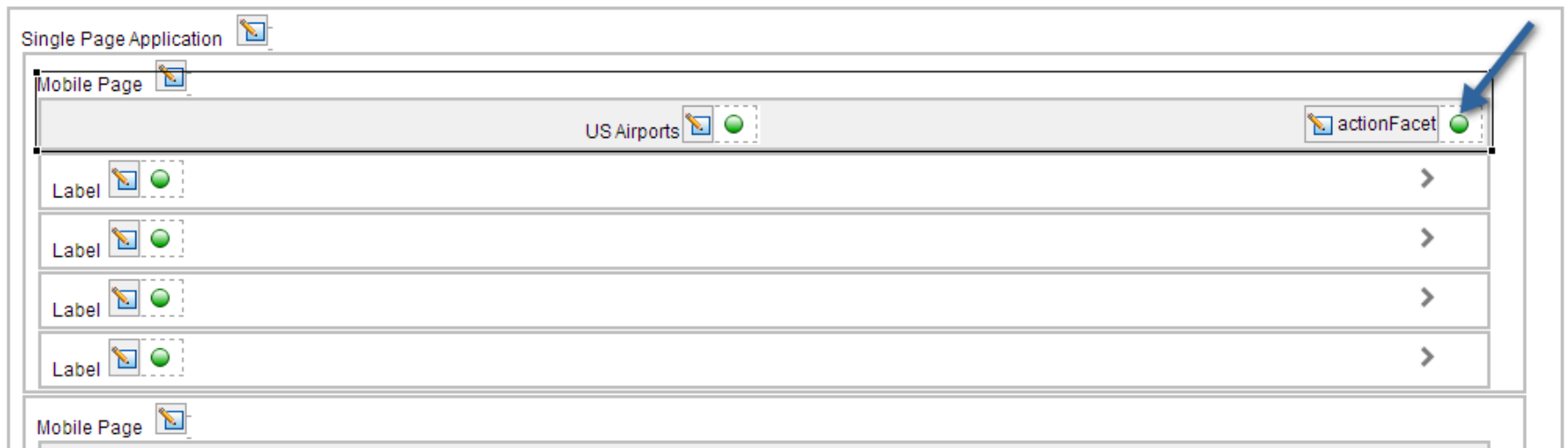
- The back button provides an easy way for users to navigate back to the page that they came from
- The `moveTo` property is used to set the mobile page a user should be directed to
- Now think about the `airportDocument` page for a moment
 - ♦ There are several views which get a user to this page
 - ▶ The Airports by Name view
 - ▶ The Airports by IATA view
 - ▶ The Airports by State view
 - ♦ Which page should the user be directed to if they are on the `airportDocument`?
 - ♦ Let's look at a solution for you to use

The Back Button (cont.)

- Essentially, what needs to be done is to track the page that a user is coming from, and store the value in a scope variable
- This value can then be computed by reading the scope variable
- The best way to set the scope variable is from the Page Heading
 - ♦ This control has a rendered property that we can use to set the scope variable
 - ♦ Use the scope variable “from” to save the pageName value you are navigating away from
 - ▶ `sessionScope.from = “byName”`
 - ▶ You can also use `viewScope`
- On the Back button, set the computed value for `moveTo` to:
 - ♦ `var from = sessionScope.get("from"); return from;`

Creating a New Document from the Home Screen

- Earlier in this presentation, we mentioned that you should provide a nice way for your users to create new documents
- Let's take a quick look to see how easy this is to do
- Go back and open up the m_Home XPage in Designer
- Look at the first mobile page (home) and you will see something called an “actionFacet”

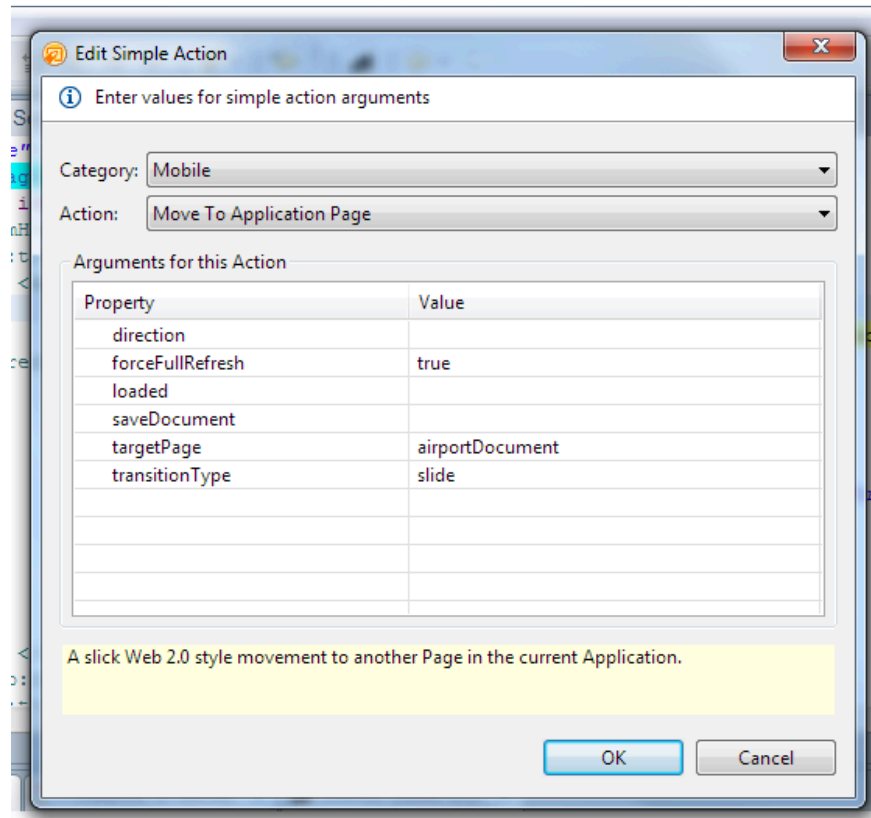


Creating a New Document from the Home Screen (cont.)

- Drag a Div control onto the actionFacet. Within this div, drag in a Button control.
- For the button, change its label to “+”
- Now, we want to only show the button to users who have logged in and have the ability to create new documents
- To accomplish this, we are going to use the UserBean, which is part of the Extension Library
 - ♦ In the rendered property, set the value to computed and use this formula:
 - ▶ **userBean.canCreateDocs**
- Now, in the onClick event for the button, we are going to click on the “Add Action” button
 - ♦ We are going to use the “Move to Application Page” that we used in the Cancel/Close and Save buttons

Creating a New Document from the Home Screen (cont.)

- When you click on the “Add Action” button, select the Mobile Category and then the “Move to Application Page” action



- That's it! Your users will now be able to create new documents.

What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

Customizing the User Interface with CSS

- The XPages Mobile Controls out-of-the-box styles match those of the native device that they are rendered on
- For iOS, you see the white and grey UI, and for Android, the darker black UI
- You can override the styles being generated by the Mobile Controls by creating your own style sheet and adding it to your mobile page
 - You will need to create a style sheet for each platform your app will be running on

Customizing the User Interface with CSS (cont.)

- Here is the US Airports app with the app.css file added to the page resources:



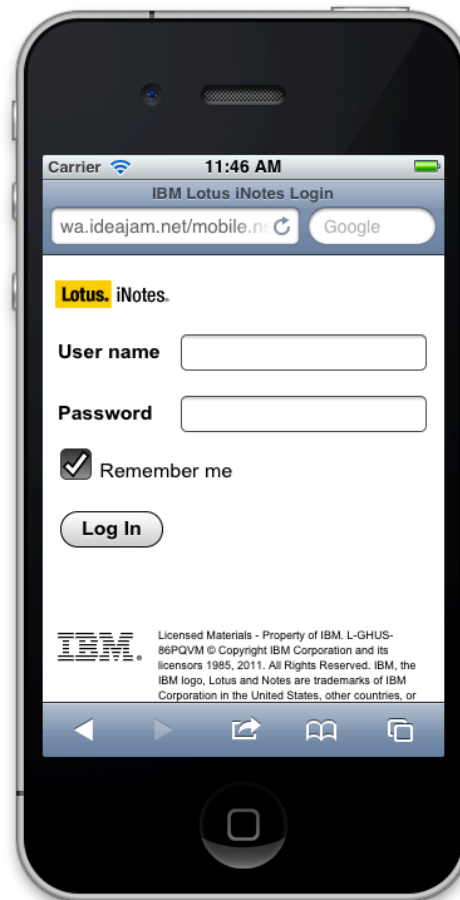
Customizing the User Interface with CSS (cont.)

- If you open up the app.css file in the US Airports database, you can see how the styles in the previous slide were overridden

```
1  /* Page background image */
2  .mobile body {
3      background-image: URL(bg.jpg);
4  }
5
6  /* Background Image for Window Title Bars */
7  .mblHeading {
8      background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#4c4
9  }
10
11 /* Tab Panel */
12 .mblTabPanelHeader {
13     background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#ffa
14 }
15
16 /* Tab Panel Button */
17 .mblTabButton {
18     background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#4c4
19 }
20
21 /* Back Buttons and other Buttons */
22 .mblArrowButtonBody, .mblHeadingActionFacet button {
23     background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#4c4
24     border-color: #cccccc;
25 }
26
27 .mblArrowButtonHead, .mblHeadingActionFacet button {
28     background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#4c4
29     border-color: #cccccc;
30 }
31
32 .mblArrowButtonNeck, .mblHeadingActionFacet button {
33     background: -webkit-gradient(linear, left top, left bottom, color-stop(0%,#4c4
```

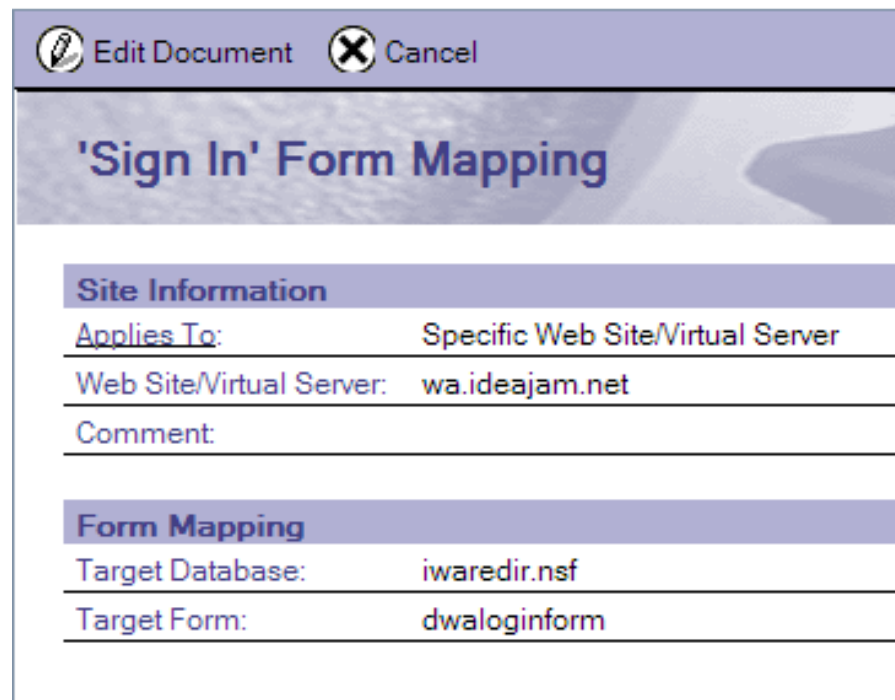
Give Your App the Right Login Screen

- If you have used iNotes Ultralite from your mobile phone, you will have seen this IBM supplied mobile login page:



Give Your App the Right Login Screen (cont.)

- You can (and should) use this login page for your app
- To have this page displayed for your app (and other apps), modify or add a new “Sign in Mappings” for your domain in the “Domino Configuration” database on your Domino server
- Set the Target Database to iwaredir.nsf and the Target Form to dwaloginform



The screenshot shows a dialog box titled "'Sign In' Form Mapping" with a purple header bar containing "Edit Document" and "Cancel" buttons. The dialog is divided into two sections: "Site Information" and "Form Mapping".

Site Information	
Applies To:	Specific Web Site/Virtual Server
Web Site/Virtual Server:	wa.ideajam.net
Comment:	

Form Mapping	
Target Database:	iwaredir.nsf
Target Form:	dwaloginform

What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

Redirecting Users to the Right Place

- In an application's "Application Properties" in the Launch tab, developers can set the launch options for Notes client access and for Web access
- There is, however, no option available to set which page to launch for mobile users

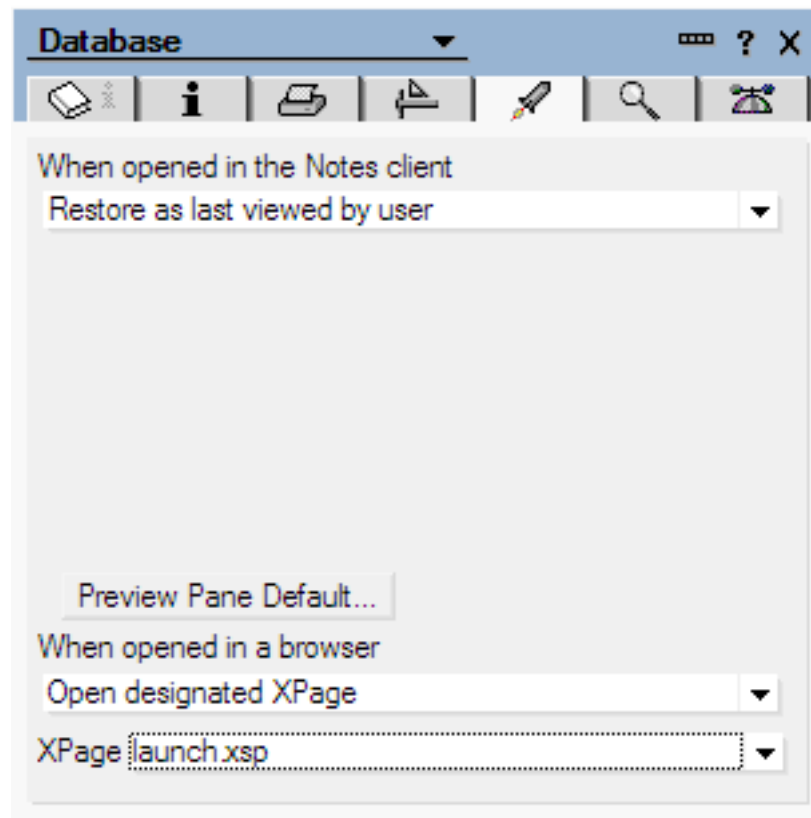
Redirecting Users to the Right Place (cont.)

- You can, however, set up a designated XPage to launch when accessed from a Web browser that will detect the User Agent
- Create an XPage with the following code and set the launch option in “Application Properties” in the launch tab. Call this page launch.xsp.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core" viewState="nostate"
3   rendered="false">
4   <xp:this.afterPageLoad>
5     <![CDATA[#{javascript:
6       var uAgent = context.getUserAgent().getUserAgent();
7       if((uAgent.match("iPhone") !== null || param.platform=="iphone") ||
8         (uAgent.match("Android") !== null || param.platform=="android") ||
9         uAgent.match("iPad") !== null){
10        context.redirectToPage("/m_Home.xsp", true);
11      }else{
12        context.redirectToPage("/home.xsp", true);
13      }
14    }]]>
15   </xp:this.afterPageLoad>
16 </xp:view>
```

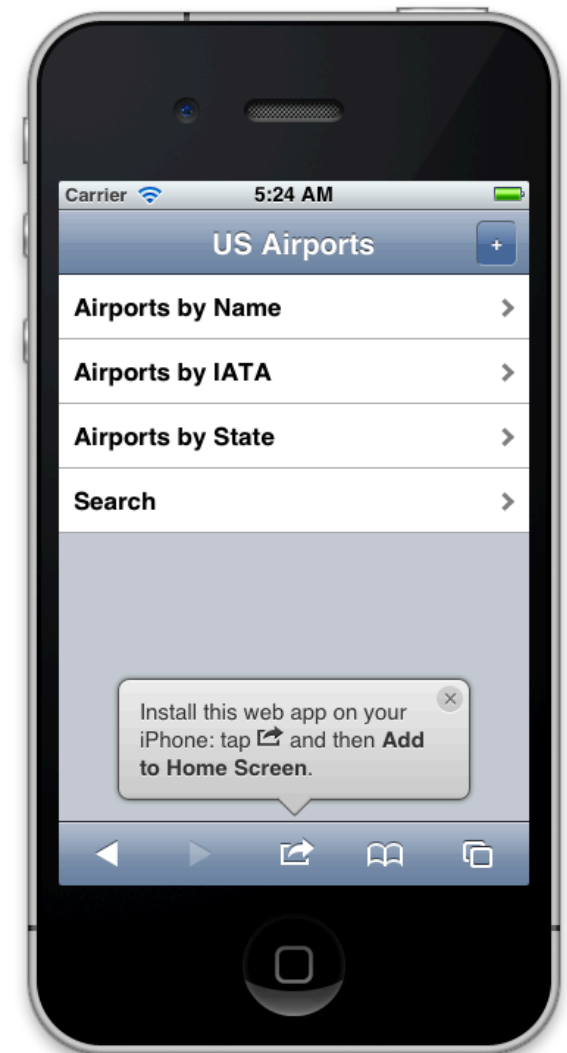

Redirecting Users to the Right Place (cont.)

- Now set the Application Properties to launch your new launch.xsp XPage. Mobile devices will get redirected to m_Home.xsp, and Web browser users to home.xsp.



Adding a Bookmark Page Helper

- You can improve the usability of your mobile app by allowing users to bookmark the app and have it appear on their iOS Home screen
 - This tip is iOS only
- Notice in this screenshot that there is a bubble popup that instructs the user how to add a bookmark for your app
- The code for this feature is available in the demo database and from <http://cubiq.org>
- Let's look at how you can add this feature to your mobile app

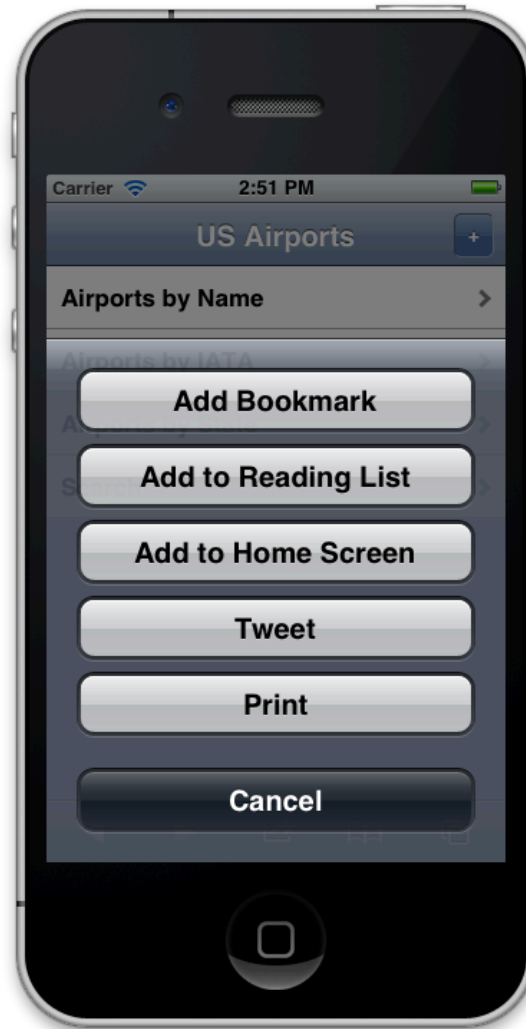


Adding a Bookmark Page Helper (cont.)

- Once you have downloaded the code from cubiq.org, you will need to add the following files to your app:
 - ♦ **add2home.js**
 - ▶ **Add this file as a JavaScript Script Library**
 - ♦ **add2home.css**
 - ▶ **Add this file as Style Sheet in the Resources → Style Sheets**
- On your XPage, on the Resources tab, add the add2home.js and add2home.css resources
- The popup will now be displayed and, when tapped, will present the user with the menu to bookmark the app

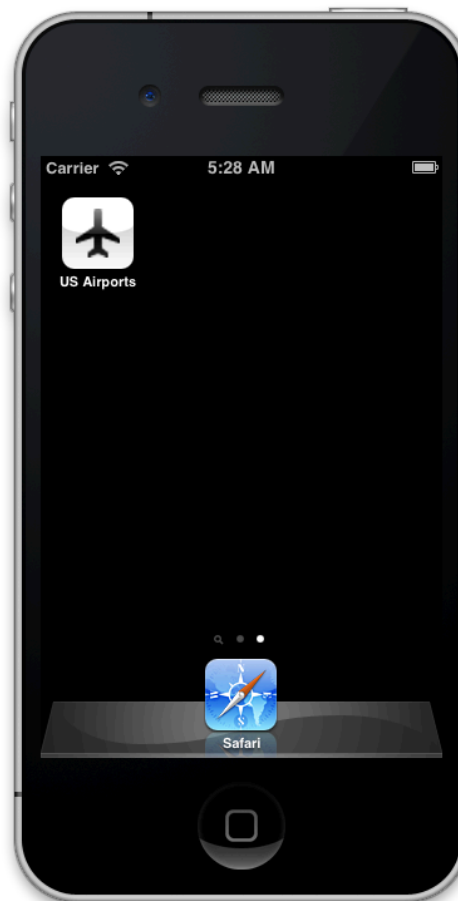
Adding a Bookmark Page Helper (cont.)

- Here is what the user will see when they tap the popup:



Adding an App Icon

- You can give your app a cool Home screen icon that users can tap to access your mobile app
- Here is what you will see if you bookmark the US Airports App:



Adding an App Icon (cont.)

- To add an icon for your app:
 - Add a 57 x 57 pixel .png icon to Image resources
 - Name the file icon.png
 - In the `<xp.this.resources>` section of your XPage, add the following markup:

```
<xp:headTag tagName="link" rendered="true" loaded="true">  
  <xp:this.attributes>  
    <xp:parameter name="rel" value="apple-touch-icon"></xp:parameter>  
    <xp:parameter name="href" value="icon.png"></xp:parameter>  
  </xp:this.attributes>  
</xp:headTag>
```

Adding an App Splash Screen

- Native applications typically have a splash screen that appears when the application is loading
- You can also add a splash screen to your mobile apps
- Here is the splash screen for the US Airports app:



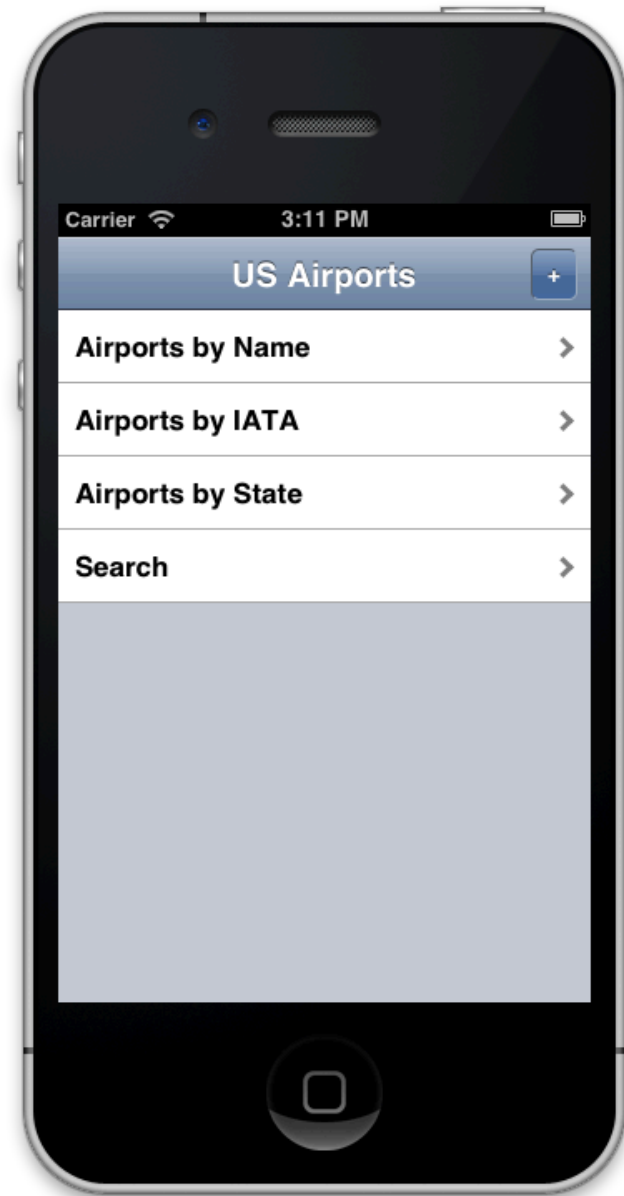
Adding an App Splash Screen (cont.)

- To add a splash screen to your app:
 - Add a 320 x 460 pixel .png graphic to Image resources
 - Name the file splash.png
 - In the <xp.this.resources> section of your XPage, add the following markup:

```
<xp:headTag tagName="link" rendered="true" loaded="true">  
  <xp:this.attributes>  
    <xp:parameter name="rel"  
      value="apple-touch-startup-image">  
    </xp:parameter>  
    <xp:parameter name="href" value="splash.png"></xp:parameter>  
  </xp:this.attributes>  
</xp:headTag>
```

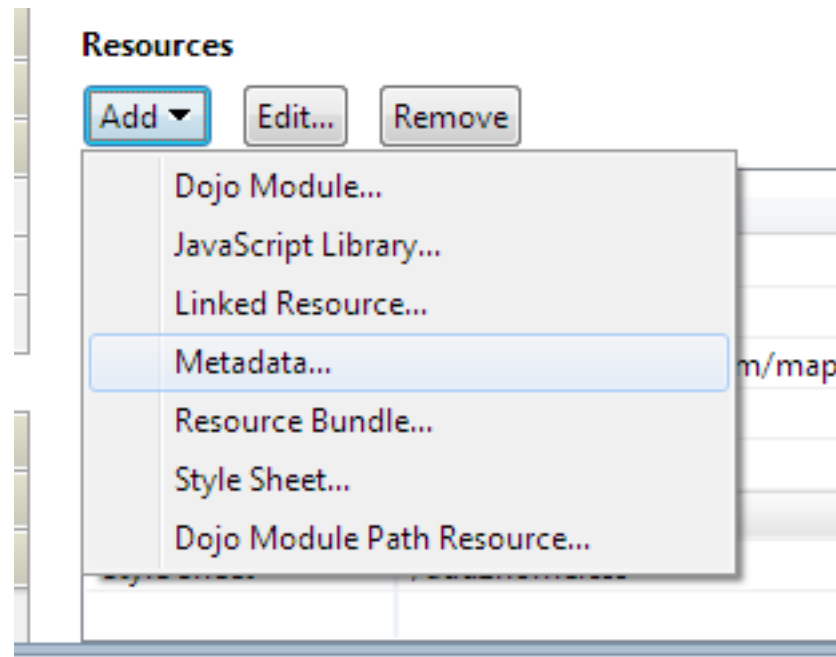

Make Your App Run Full Screen

- When you run a bookmarked app, you can have it run as a “full-screen” app
- Users won’t have the ability to navigate out of the app other than pressing the Home button
- Apps that run full screen appear to the user like a native app
- Notice in this screen shot that there are now controls at the bottom of the screen:



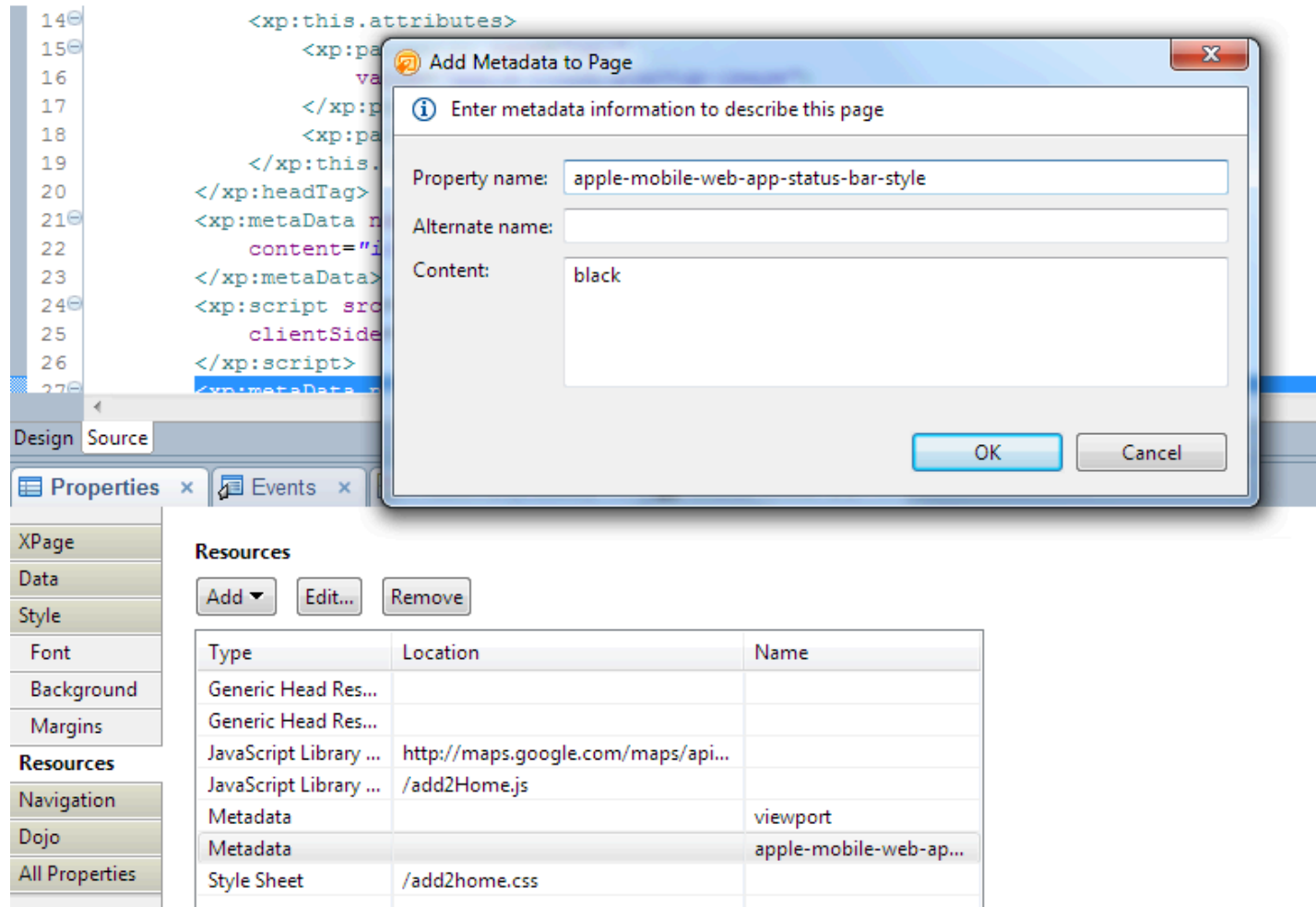
Make Your App Run Full Screen (cont.)

- On your XPage, go to the Resource tab and click on the Add button



Make Your App Run Full Screen (cont.)

- Now, enter the following in the dialog box and then click OK:



What We'll Cover ...

- Introducing the XPages Mobile Controls and our demo app
- Development environment requirements
- A review of each of the controls and what they do
- Configuring your application to be mobile
- The basic structure of a mobile page
- Working with views
- Editing data in documents
- Customizing the user interface using CSS
- More tips and tricks to make users love your mobile apps
- Meet TSAzr – an XPages-powered native iPhone app
- Using Appcelerator to build the native front-end user experience
- How we used XPages to build the back-end services
- Wrap-up

This Is XPages



XPages, You Say?

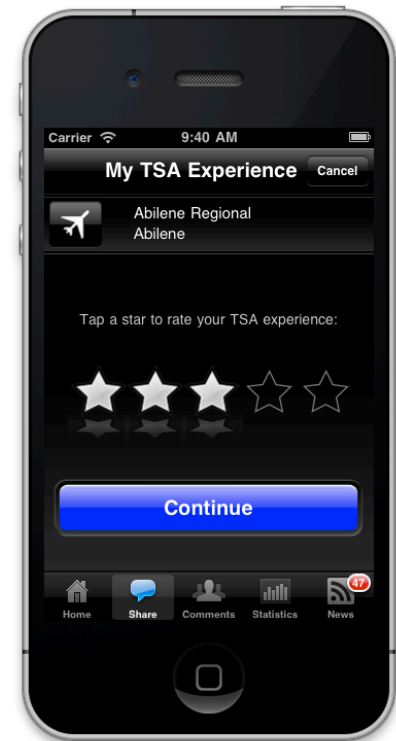
- Earlier, we looked at building mobile Web browser interfaces for our existing Notes and Domino applications
- The XPages Mobile Controls give you the power to rapidly mobilize your applications
- What if ...
 - ♦ You wanted to build a native app for an Android, iOS, or BlackBerry that needed to retrieve data from your Notes and Domino databases
 - ♦ Better yet ...
 - ▶ What if you could use the power of XPages to build NEW mobile applications that use NEW Notes databases?
 - *What are you talking about?*

Think Outside of What You Know

- **What if I told you could use XPages to:**
 - ♦ **Power a native iPhone application**
 - ♦ **Service thousands of iPhone users, using a clustered Domino server on the Amazon Elastic Cloud**
 - ♦ **Do all of this with very little code**
 - ♦ **Oh, and users never, ever see an “XPage”**
- **Let's tear TSAzr apart and see what lurks below**

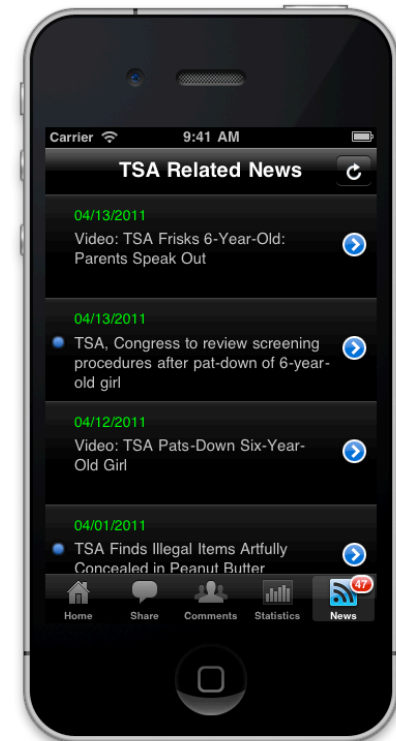
What Does TSAzr Do?

- TSAzr allows the flying public to share their experiences about going through TSA screening points at airports around the United States
- It allows users to enter data for each airport, including:
 - ♦ Comments about their experience
 - ♦ Assigning a star rating
 - ♦ Selecting which type of screening process they went through
 - ▶ Body scan
 - ▶ Pat down
 - ♦ Sharing to Facebook



What Does TSAzr Do? (cont.)

- **Users of the app can see:**
 - ♦ **Comments left by others by airport**
 - ♦ **Graphical statistics**
 - ▶ **Number of pat downs**
 - ▶ **Number of body scans**
 - ▶ **Rating by airport**
 - ▶ **And others**
 - ♦ **A news feed containing stories about the TSA**



What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

Using Appcelerator to Build the Native Front-End User Experience

- To build the native iPhone app, a product called Appcelerator Titanium was used

- www.appcelerator.com
- Titanium allows you to easily build native iPhone apps
- JavaScript is the language you use for Appcelerator
- Domino and Domino Designer V8.5.3
- Google Charts
- Photoshop



The TSAzr API

- The TSAzr API has 19 methods
 - ♦ 18 Get
 - ♦ 1 Post
 - ▶ These are not REST services
- All of these methods are actually XPages
 - ♦ They are XPages that don't render a UI response
- The GET methods return JSON
- The POST method accepts JSON

TSAzr Server API

JavaScript Code Generator:

- ▶ GET JS airport array for app (DONE)

Basics:

- ▶ GET airport data (DONE)
- ▶ POST user rating (DONE)
- ▶ GET comments (DONE)
- ▶ GET news links (DONE)
- ▶ GET news badge (DONE)

Charts:

- ▶ GET home page meter (DONE)
- ▶ GET airports w/ highest rates (DONE)
- ▶ GET airports w/ lowest rates (DONE)
- ▶ GET airports w/ most scans (DONE)
- ▶ GET airports w/ fewest scans (DONE)
- ▶ GET airports w/ most rates (DONE)
- ▶ GET airports w/ fewest rates (DONE)
- ▶ GET airports w/ most scan refusals (DONE)
- ▶ GET airports w/ fewest scan refusals (DONE)
- ▶ GET airports w/ most patdowns (DONE)
- ▶ GET airports w/ fewest patdowns (DONE)
- ▶ GET airports w/ most junk touchings (DONE)
- ▶ GET airports w/ fewest junk touchings (DONE)

The TSAzr API (cont.)

- **GET JS airport array for app**
 - ♦ This is a service that returns the list of airports and their latitude and longitude and other information
 - ▶ This data IS NOT stored in an NSF
- **GET airport data**
 - ♦ Returns all of the information for a specific airport, given the latitude and longitude
 - ▶ This uses the iOS location services

TSAzr Server API

JavaScript Code Generator:

- ▶ GET JS airport array for app (DONE)

Basics:

- ▶ GET airport data (DONE)
- ▶ POST user rating (DONE)
- ▶ GET comments (DONE)
- ▶ GET news links (DONE)
- ▶ GET news badge (DONE)

Charts:

- ▶ GET home page meter (DONE)
- ▶ GET airports w/ highest rates (DONE)
- ▶ GET airports w/ lowest rates (DONE)
- ▶ GET airports w/ most scans (DONE)
- ▶ GET airports w/ fewest scans (DONE)
- ▶ GET airports w/ most rates (DONE)
- ▶ GET airports w/ fewest rates (DONE)
- ▶ GET airports w/ most scan refusals (DONE)
- ▶ GET airports w/ fewest scan refusals (DONE)
- ▶ GET airports w/ most patdowns (DONE)
- ▶ GET airports w/ fewest patdowns (DONE)
- ▶ GET airports w/ most junk touchings (DONE)
- ▶ GET airports w/ fewest junk touchings (DONE)

The TSAzr API (cont.)

- **POST user rating**
 - ♦ This service is called when a user submits their rating for an airport
 - ♦ Parameters include:
 - ▶ Mac address of device
 - ▶ Airport code (IATA)
 - ▶ Users name
 - ▶ Gender
 - ▶ Comment
 - ▶ Star rating
 - ▶ Scanner
 - ▶ Patdown

TSAzr Server API

JavaScript Code Generator:

- ▶ GET JS airport array for app (DONE)

Basics:

- ▶ GET airport data (DONE)
- ▶ POST user rating (DONE)
- ▶ GET comments (DONE)
- ▶ GET news links (DONE)
- ▶ GET news badge (DONE)

Charts:

- ▶ GET home page meter (DONE)
- ▶ GET airports w/ highest rates (DONE)
- ▶ GET airports w/ lowest rates (DONE)
- ▶ GET airports w/ most scans (DONE)
- ▶ GET airports w/ fewest scans (DONE)
- ▶ GET airports w/ most rates (DONE)
- ▶ GET airports w/ fewest rates (DONE)
- ▶ GET airports w/ most scan refusals (DONE)
- ▶ GET airports w/ fewest scan refusals (DONE)
- ▶ GET airports w/ most patdowns (DONE)
- ▶ GET airports w/ fewest patdowns (DONE)
- ▶ GET airports w/ most junk touchings (DONE)
- ▶ GET airports w/ fewest junk touchings (DONE)

The TSAzr API (cont.)

- **GET comments**
 - ♦ Used to retrieve recent comments
 - ▶ Can return all comments
 - ▶ Can return comments for a specific airport
- **GET news**
 - ♦ A service that returns a list of news items and links that are rendered in the app by the News tab
- **GET news badge**
 - ♦ A service that updates the article count when the app loads

TSAzr Server API

JavaScript Code Generator:

- ▶ GET JS airport array for app (DONE)

Basics:

- ▶ GET airport data (DONE)
- ▶ POST user rating (DONE)
- ▶ GET comments (DONE)
- ▶ GET news links (DONE)
- ▶ GET news badge (DONE)

Charts:

- ▶ GET home page meter (DONE)
- ▶ GET airports w/ highest rates (DONE)
- ▶ GET airports w/ lowest rates (DONE)
- ▶ GET airports w/ most scans (DONE)
- ▶ GET airports w/ fewest scans (DONE)
- ▶ GET airports w/ most rates (DONE)
- ▶ GET airports w/ fewest rates (DONE)
- ▶ GET airports w/ most scan refusals (DONE)
- ▶ GET airports w/ fewest scan refusals (DONE)
- ▶ GET airports w/ most patdowns (DONE)
- ▶ GET airports w/ fewest patdowns (DONE)
- ▶ GET airports w/ most junk touchings (DONE)
- ▶ GET airports w/ fewest junk touchings (DONE)

The TSAzr API (cont.)

- **The Chart API calls**
 - ♦ All of these API calls return data that is used to generate the charts that are shown throughout the app
 - ♦ The chart API generates Google Charts' URLs that are passed down to the iOS device, which are, in turn, rendered inside TSAzr
 - ♦ The chart API also returns data points that supplement the chart displayed on the device
 - ▶ **Star rating**
 - ▶ **Airport name**

TSAzr Server API

JavaScript Code Generator:

- ▶ GET JS airport array for app (DONE)

Basics:

- ▶ GET airport data (DONE)
- ▶ POST user rating (DONE)
- ▶ GET comments (DONE)
- ▶ GET news links (DONE)
- ▶ GET news badge (DONE)

Charts:

- ▶ GET home page meter (DONE)
- ▶ GET airports w/ highest rates (DONE)
- ▶ GET airports w/ lowest rates (DONE)
- ▶ GET airports w/ most scans (DONE)
- ▶ GET airports w/ fewest scans (DONE)
- ▶ GET airports w/ most rates (DONE)
- ▶ GET airports w/ fewest rates (DONE)
- ▶ GET airports w/ most scan refusals (DONE)
- ▶ GET airports w/ fewest scan refusals (DONE)
- ▶ GET airports w/ most patdowns (DONE)
- ▶ GET airports w/ fewest patdowns (DONE)
- ▶ GET airports w/ most junk touchings (DONE)
- ▶ GET airports w/ fewest junk touchings (DONE)

The Titanium App Structure

- **App.js**
 - ♦ The main controller for the app
 - ▶ The GET news badge API is called
- **Home.js**
 - ♦ The code that renders when the app is loaded
 - ♦ This includes:
 - ▶ The National Statistics page
 - ▶ The Recent Comments page(s)
 - *National*
 - *Local airport*
 - ♦ The GET airport data API is called
 - ♦ The GET home page API is called
 - ♦ The GET comments API is called

The Titanium App Structure (cont.)

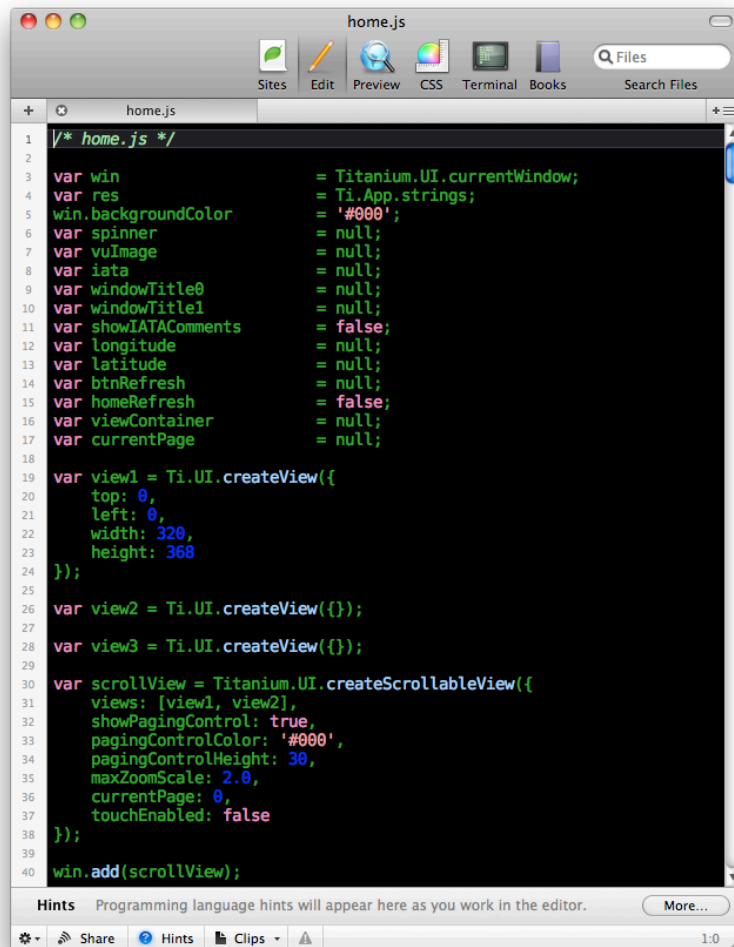
- **Rate.js**
 - ♦ This is code that is used to walk the user through submitting their feedback
 - ▶ **Select an airport**
 - ▶ **Walk them through scan type**
 - ▶ **Ask for star rating**
 - ▶ **Option to enter a comment**
 - ▶ **Save rating to the Domino server**
 - ***Share to Facebook***
 - ♦ The POST user rating API is called

The Titanium App Structure (cont.)

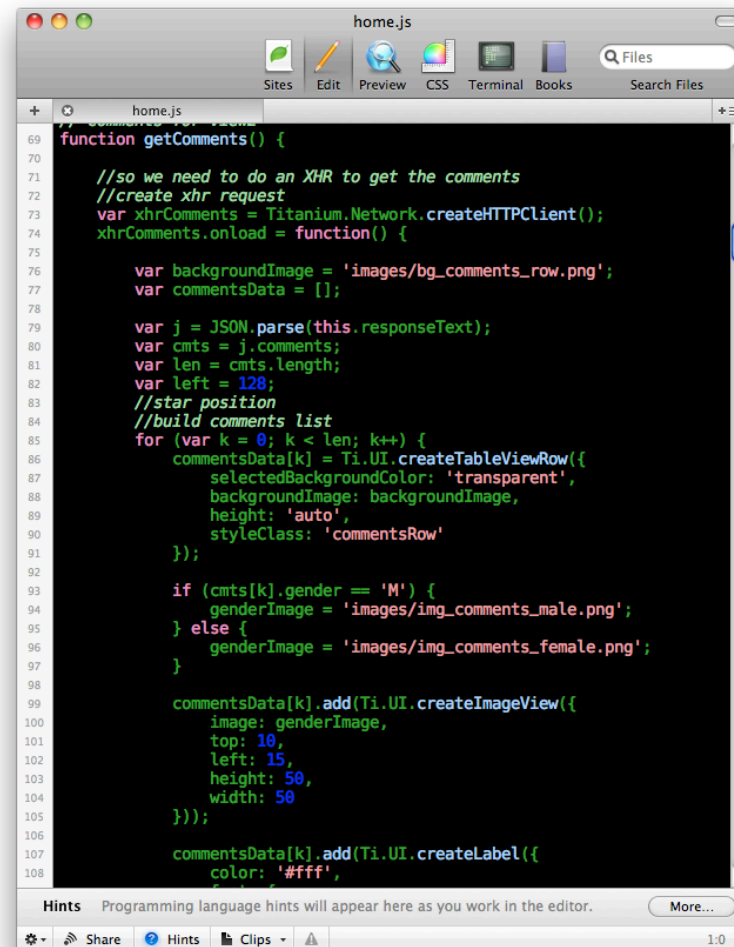
- **Comments.js**
 - ♦ This is code that is used to display:
 - ▶ **Recent comments**
 - ▶ **Recent comments by airport**
 - ♦ The GET comments API is called
- **Statistics.js**
 - ♦ This code renders the choice of charts to display on the device
 - ♦ Once a chart is selected, the appropriate chart is then displayed
 - ♦ The respective GET chart API is called, depending on which chart the user selected to view
- **News.js**
 - ♦ This code displays the list of news articles on the news tab
 - ♦ The GET news links API is called

Sample Titanium Code

- Here are some screenshots of the home.js code:



```
1  /* home.js */
2
3  var win = Titanium.UI.currentWindow;
4  var res = Ti.App.strings;
5  win.backgroundColor = '#000';
6  var spinner = null;
7  var vuImage = null;
8  var iata = null;
9  var windowTitle0 = null;
10 var windowTitle1 = null;
11 var showIATAComments = false;
12 var longitude = null;
13 var latitude = null;
14 var btnRefresh = null;
15 var homeRefresh = false;
16 var viewContainer = null;
17 var currentPage = null;
18
19 var view1 = Ti.UI.createView({
20     top: 0,
21     left: 0,
22     width: 320,
23     height: 368
24 });
25
26 var view2 = Ti.UI.createView({});
27
28 var view3 = Ti.UI.createView({});
29
30 var scrollView = Titanium.UI.createScrollView({
31     views: [view1, view2],
32     showPagingControl: true,
33     pagingControlColor: '#000',
34     pagingControlHeight: 30,
35     maxZoomScale: 2.0,
36     currentPage: 0,
37     touchEnabled: false
38 });
39
40 win.add(scrollView);
```



```
69 function getComments() {
70
71     //so we need to do an XHR to get the comments
72     //create xhr request
73     var xhrComments = Titanium.Network.createHTTPClient();
74     xhrComments.onload = function() {
75
76         var backgroundImage = 'images/bg_comments_row.png';
77         var commentsData = [];
78
79         var j = JSON.parse(this.responseText);
80         var cmts = j.comments;
81         var len = cmts.length;
82         var left = 128;
83         //star position
84         //build comments list
85         for (var k = 0; k < len; k++) {
86             commentsData[k] = Ti.UI.createTableViewRow({
87                 selectedBackgroundColor: 'transparent',
88                 backgroundImage: backgroundImage,
89                 height: 'auto',
90                 styleClass: 'commentsRow'
91             });
92
93             if (cmts[k].gender == 'M') {
94                 genderImage = 'images/img_comments_male.png';
95             } else {
96                 genderImage = 'images/img_comments_female.png';
97             }
98
99             commentsData[k].add(Ti.UI.createImageView({
100                 image: genderImage,
101                 top: 10,
102                 left: 15,
103                 height: 50,
104                 width: 50
105             }));
106
107             commentsData[k].add(Ti.UI.createLabel({
108                 color: '#fff',
```

What We'll Cover ...

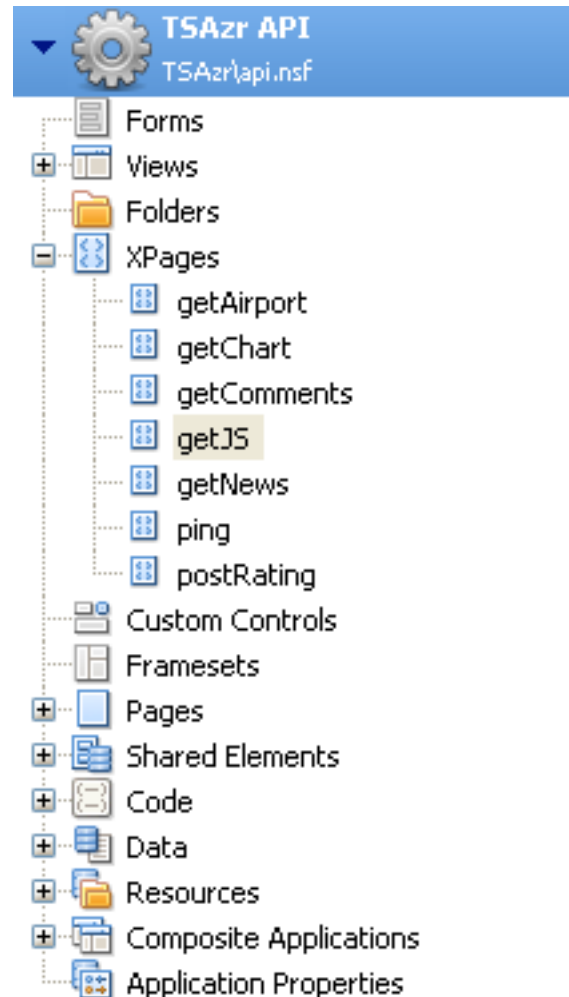
- **Introducing the XPages Mobile Controls and our demo app**
- **Development environment requirements**
- **A review of each of the controls and what they do**
- **Configuring your application to be mobile**
- **The basic structure of a mobile page**
- **Working with views**
- **Editing data in documents**
- **Customizing the user interface using CSS**
- **More tips and tricks to make users love your mobile apps**
- **Meet TSAzr – an XPages-powered native iPhone app**
- **Using Appcelerator to build the native front-end user experience**
- **How we used XPages to build the back-end services**
- **Wrap-up**

The Notes Databases

- There are 3 databases that support TSAzr
 - ♦ TSAzr API
 - ▶ This is the NSF that contains all of the code to deliver API services
 - *Contains 7 XPages*
 - *Contains 0 Custom Controls*
 - ♦ TSAzr News
 - ▶ This is the NSF that contains all of the news articles that the GET news links and GET news badge APIs query
 - ♦ TSAzr Ratings
 - ▶ This NSF stores all of the ratings data submitted by users
 - ▶ It also is used to determine ratings, number of scan types, and more

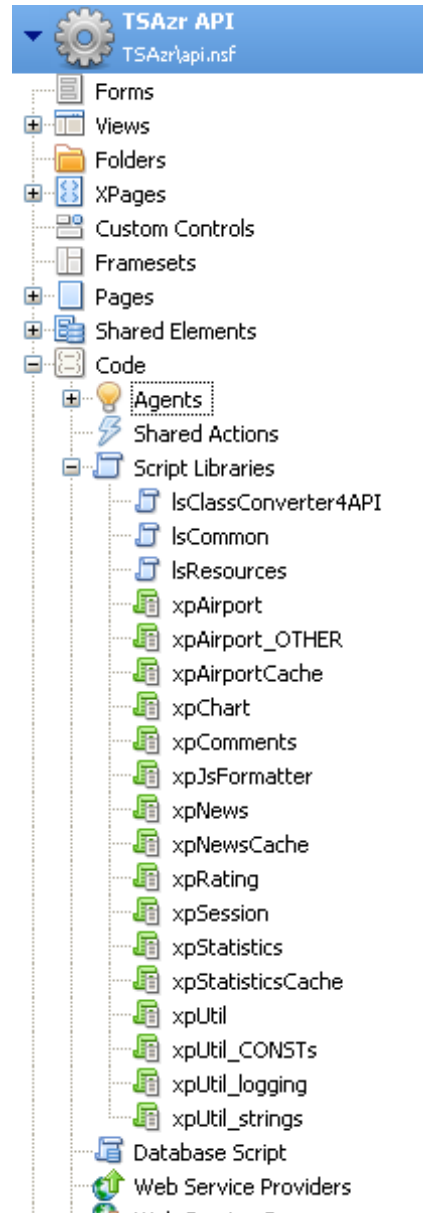
The XPages Code

- 7 XPages support the API



Server-Side JavaScript Libraries

- 16 Script Libraries



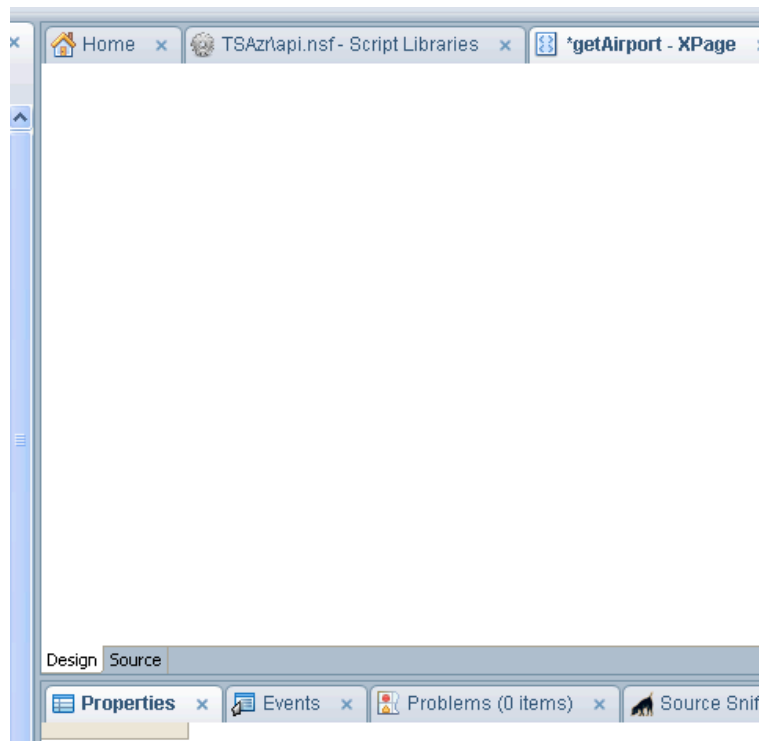
So What's Inside Those XPages?

- **All of the API's logic is contained in the Script Libraries**
 - We won't be tearing the script libraries apart in this session
- **This is SSJS code for getAirport API**
 - Note the inclusion of the xpSession.jss and xpAirport.jss script libraries as resources for the XPage
 - Note that the fSession().init() function is called on the beforePageLoad event of the XPage
 - Note that the fAirport().init() function is called on the afterRenderResponse event of the XPage

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xp:view xmlns:xp="http://www.ibm.com/xsp/core" createForm="false" rendered="false"
3     ViewState="nostate" F>
4
5     <xp:this.resources>
6         <xp:script src="/xpSession.jss" clientSide="false"></xp:script>
7         <xp:script src="/xpAirport.jss" clientSide="false"></xp:script>
8     </xp:this.resources>
9
10    <xp:this.beforePageLoad><![CDATA[#{javascript:fSession().init();}]]></xp:this.beforePageLoad>
11    <xp:this.afterRenderResponse><![CDATA[#{javascript:fAirport().init();}]]></xp:this.afterRenderResponse>
12
13 </xp:view>
```

So What Does the XPage Look Like in Designer?

- **Yeap – absolutely blank**
 - ♦ It doesn't render anything!
 - ♦ All of the magic is done in the SSJS



What We'll Cover ...

- **Introducing the XPages Mobile Controls and our demo app**
 - **Development environment requirements**
 - **A review of each of the controls and what they do**
 - **Configuring your application to be mobile**
 - **The basic structure of a mobile page**
 - **Working with views**
 - **Editing data in documents**
 - **Customizing the user interface using CSS**
 - **More tips and tricks to make users love your mobile apps**
 - **Meet TSAzr – an XPages-powered native iPhone app**
 - **Using Appcelerator to build the native front-end user experience**
 - **How we used XPages to build the back-end services**
- **Wrap-up**

Additional Resources

- www-10.lotus.com/ldd/ddwiki.nsf/dx/XPages_Mobile_Controls_Tutorial_
 - ♦ IBM Notes and Domino Developer Wiki Mobile Controls Tutorial
- www.OpenNTF.org
 - ♦ Open source community for Lotus Notes and Domino
- <http://cubiq.org>
 - ♦ Cubiq.org – Mobile app utilities
- www.XPages.info
 - ♦ A great starting point for learning XPages
- www.appcelerator.com
 - ♦ Appcelerator Titanium
- www.tsazr.com
 - ♦ TSAzr

7 Key Points to Take Home

- The XPages Mobile Controls allow you to quickly give your Notes and Domino applications a nice mobile interface
- All of your mobile app code gets placed on a single XPage
- The XPage uses appPage custom controls for each page of your mobile application
- When you customize the style of your mobile app using CSS, be sure to create a style sheet for iPhone and one for Android
- You can use categorized views in your applications
- XPages don't always have to be things people interact with
- XPages can render responses that aren't seen by a browser

Your Turn!



Questions?

**How to contact me:
Bruce Elgort
bruce.elgort@elguji.com**