DBx GEOMATICS

Developing Interactive Cross-Platform Mobile Applications for Apple iOS (iPhone/iPad/iPod) and Google Android (Phone and Tablets) with Adobe Flash Builder and CartoVista Mobile

Dany Bouchard, DBx GEOMATICS inc.

Abstract. Developing cross-platform mobile mapping applications is a challenge that requires good design and a sound understanding of target devices and platform capabilities. Adobe Flash Builder and the Adobe Integrated Runtime (AIR) framework can be used to author applications that will render properly on any device, regardless of its screen resolution and pixel density. Based on Adobe AIR, the CartoVista Mobile Software Development Kit (SDK) enables advanced mapping for Apple iOS (iPhone, iPad, and iPod), Google Android (Phone and tables) as well as Blackberry (Playbook, BB10) devices.

Keywords: Mobile, Apple iOS, iPhone, iPad, iPod, Google Android, Blackberry, Adobe Flash Builder, Adobe AIR, CartoVista

Table of Contents

Contents

Table of Figures

1. Introduction

You might be surprised when you attempt to access your browser-based web mapping site on a smart phone and discover that your content is not laid out or perform as you had envisioned; and as you try to alter your content so that it's displayed equally well on the desktop as it is on a smart phone or tablet, you will discover that there are some important challenges to overcome.

With the proliferation of smartphones and tablets, mobile applications provide an opportunity to deliver the richness of an immersive mapping experience, similar to the desktop, with the addition of GPS-enabled functionality.

1.1. The Mobile Application Design Challenge

Making your map application device-friendly is quite challenging when you take into consideration the following factors.

1.2. Performance

Smartphones and tablets have fewer hardware resources than desktop computers today—less RAM, CPU power, GPU resources, etc. In particular, this is an important factor to consider when working with complex vector map features and content (e.g. a large layer of polygons).

Figure 1: Complex Polygon Layer (Demographic Data)

1.3. Screen size

As the mobile applications continue to proliferate and reach more devices, developers need to adopt techniques for authoring with multiple screen sizes and resolutions in mind.

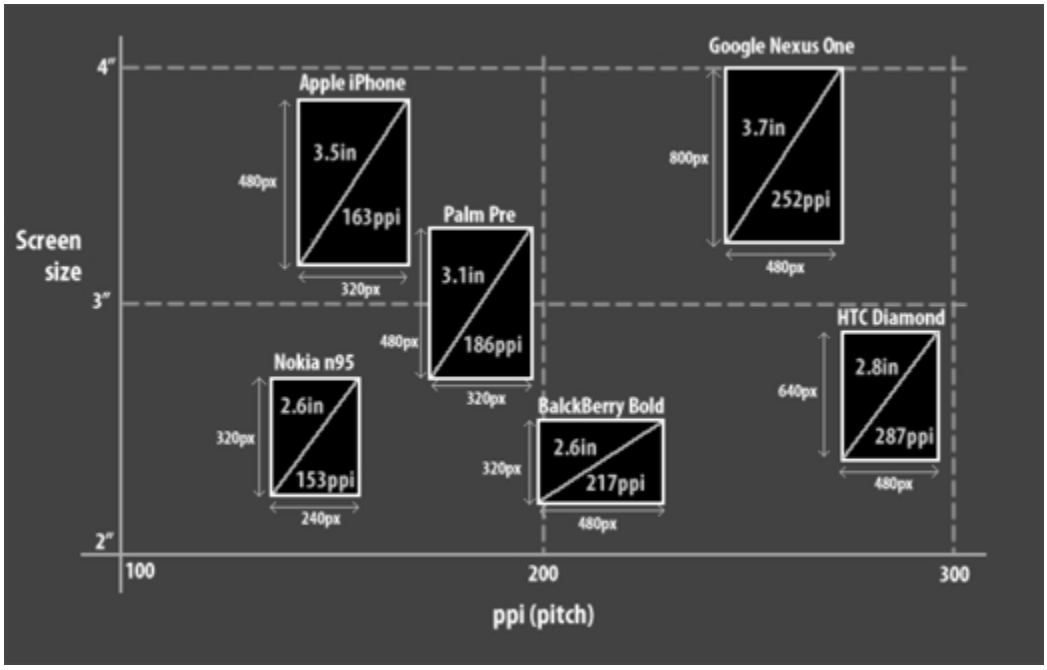


Figure 2: Different screen sizes for common mobile devices (From Adobe Systems)

1.4. Interaction

The application user interface should anticipate touch-based input, accelerometers, screen orientation, etc. and shouldn't assume the presence of a physical keyboard or mouse.

Figure 3: Mobile device interaction: touch-based input

2. Adobe Flash Builder / Adobe AIR

Several techniques within Adobe Flash Builder can help developers author content that will render properly on any device, regardless of its screen resolution and pixel density. As an open source framework, Adobe Flex offers a very rich environment to help developer design their user interface with ease, with a high quality application debugger and profiler. With Adobe Flash Builder, mobile applications are developed with the same authoring tools as on the desktop (targeting the Adobe Flash Player). This single code base and the ability to reuse assets makes optimization easy and simple.

2.1. Developing with Adobe AIR

Adobe AIR is a unique stand-alone application development across mobile, desktop, and digital home devices. Developing mobile apps with Adobe AIR makes it possible to create a single application that can be deployed across multiple mobile smartphones and tablets running Android, iOS, or BlackBerry 10/Playbook. It is an attractive development platform in part because of its broad reach.

At the same time, each of these environments places a real challenge to create a seamless, predictable experience between the end users and your cartographic content.

AIR makes this possible by providing cross-platform abstractions where they are helpful (such as for accessing the camera roll), discovering device properties dynamically (such as screen size), etc.

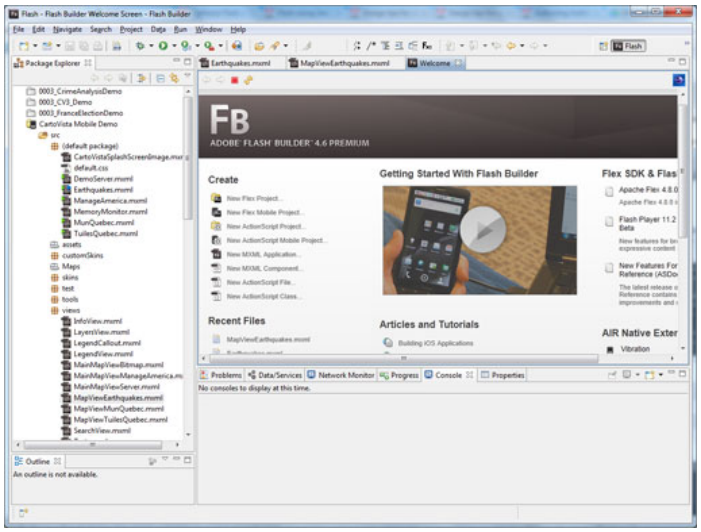


Figure 4: Adobe Flash Builder Interface – Development Tools

2.1.1. Mobile App Design – View-Based Model

In addition to skinning and extending the core components to work well on mobile devices, Flex contains a set of new application components specifically designed to make it easy to build applications that follow standard design patterns for touch screen smartphones and tablets.

Because of the limited screen real estate, applications on these devices are typically structured as a series of **views**, each of which is focused on displaying a single list of data or details about a single data item. The user navigates between views by tapping on data items or other controls, and goes back by either using on-screen UI or a hardware "Back" button. Additional actions can be provided through on-screen UI or a menu overlay.

The following figure illustrates this typical mobile application design pattern.

Figure 5: Flex View-based Model for Mobile Applications

The View component represents a single screen of UI. Typically, you'll create custom MXML or ActionScript components based on View, and add whatever components you want to appear within the content. For example, a shopping cart application might have a home view that displays a list of featured items and categories. Tapping on a category navigates to a product list view displaying items in a given category, and tapping on a product navigates to a product detail view showing information about the product.

When the user rotates the screen between portrait and landscape orientations, the View is automatically resized by default to the appropriate aspect ratio. As a result, if you use the standard Flex layout managers, your application can handle orientation changes with little extra work on your part. For finer-grained control, you can use the Flex states mechanism to define portrait and landscape states that specify exactly how the View should look in each state.

When starting a new Flex Mobile project, template options are available to use the view-based model. In addition to the model you have to pick the target platform for your application, as well as several other options.

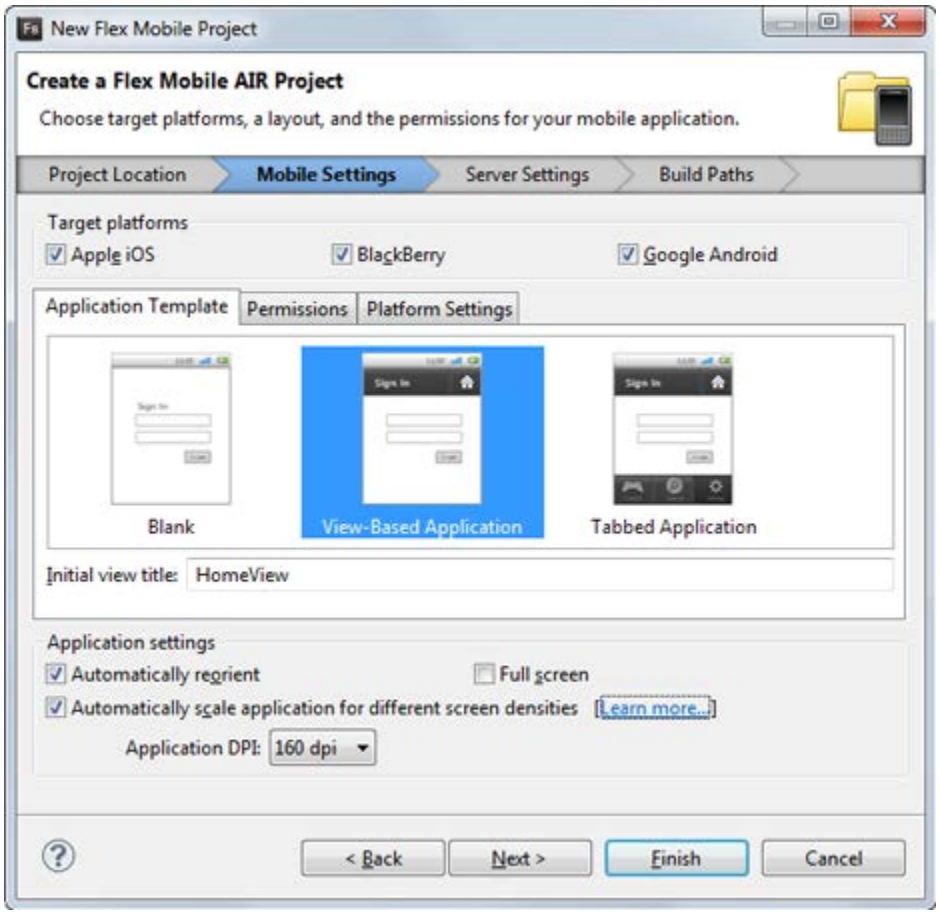


Figure 6: Adobe Flash Builder Mobile Project Settings

Once you have chosen the view-based model, the main view for your application is available in MXML. Flex provides both a source and design view which is very useful when working on mobile applications.

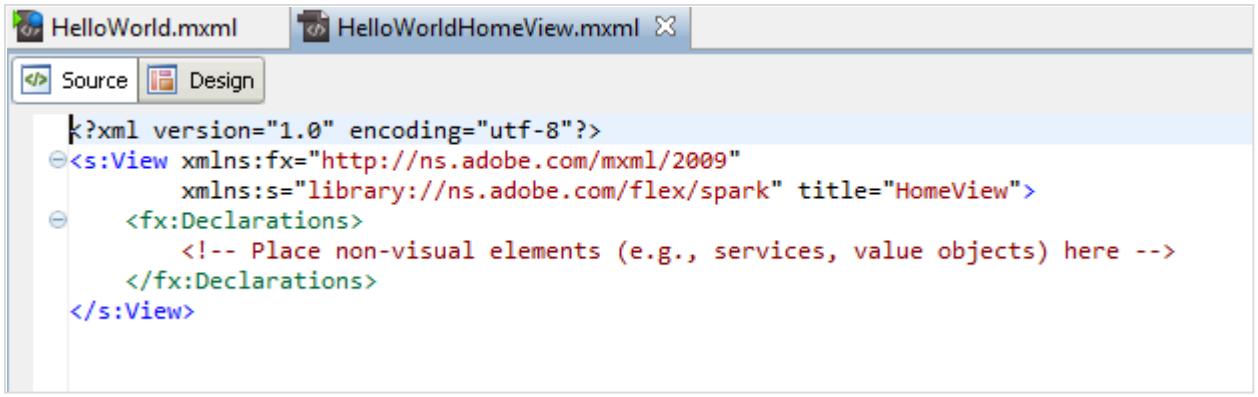


Figure 7: Basis of a Flex Mobile App in MXML (Source View)

When using the design view, you can select the target device and see how your map controls will look in your application.

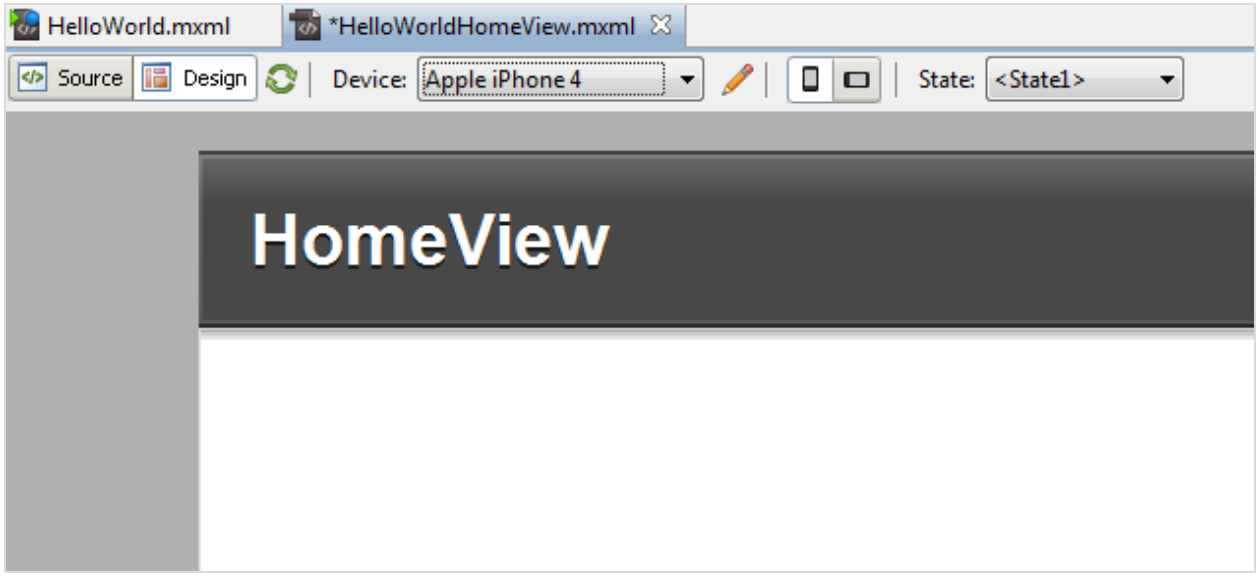


Figure 8: Flex Mobile App in MXML (Design View)

Flex includes a series of rich, common mobile controls that are ready to use to define the functionality of your view.

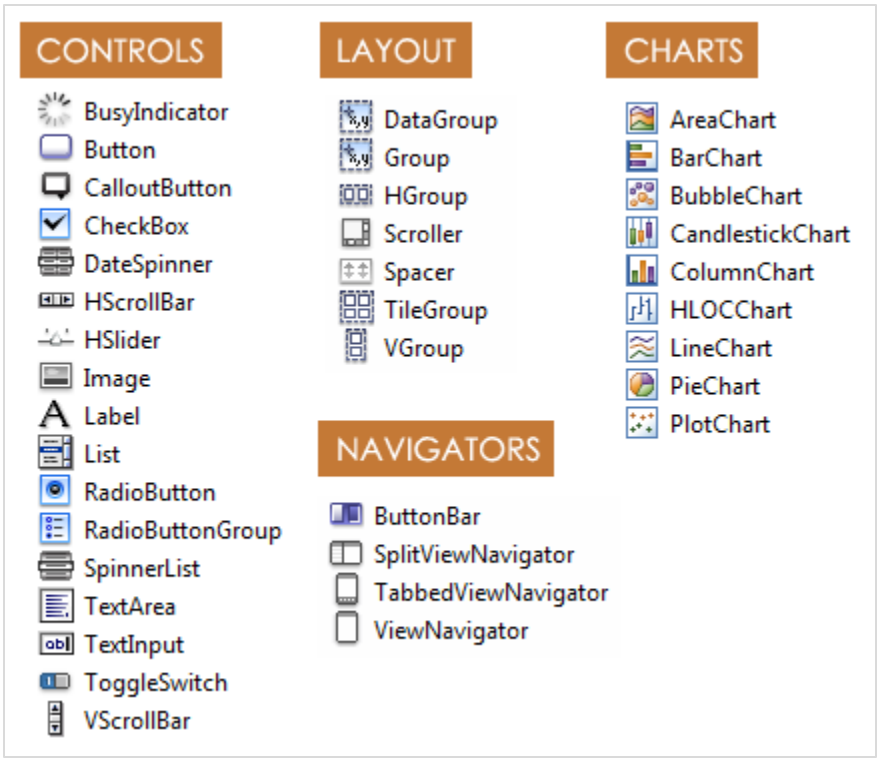


Figure 9: Flex Mobile Components

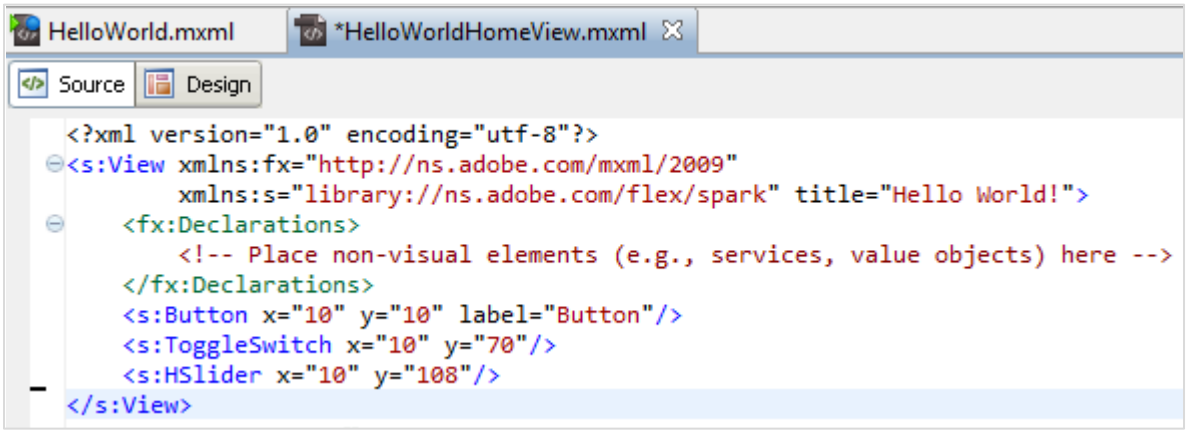


Figure 10: Source View - with Components

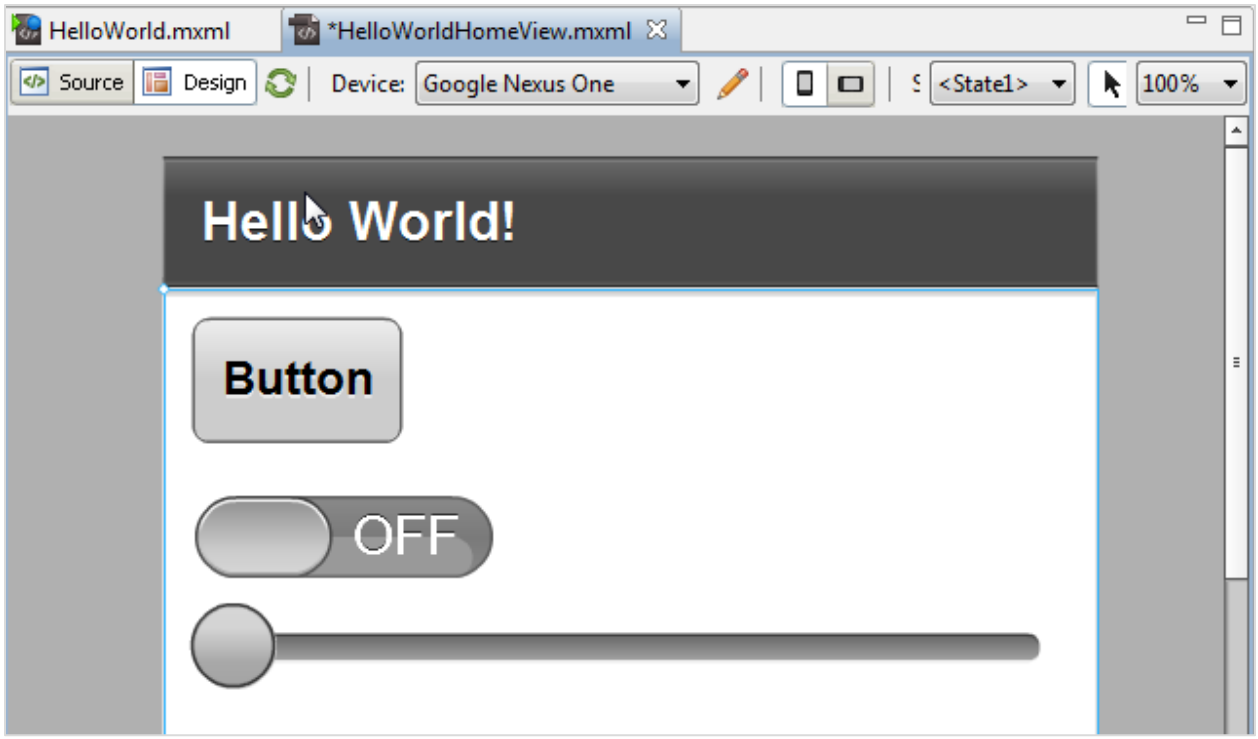


Figure 11: Design view with Components

When you are ready to test your application, Adobe Flash Builder includes a built-in emulator to properly test and debug your application.

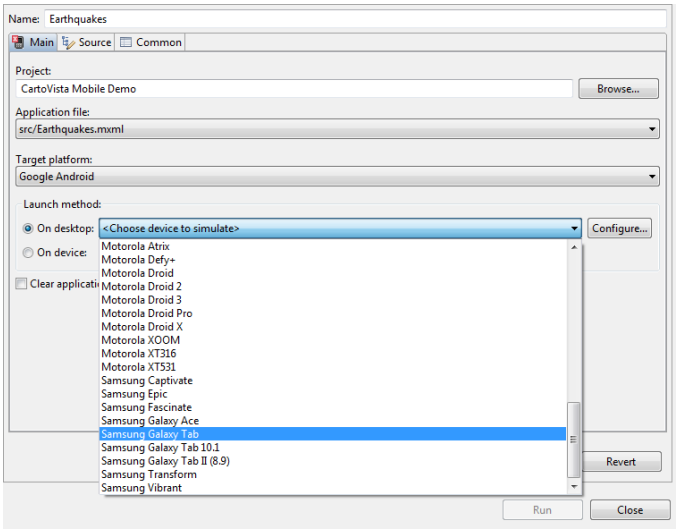


Figure 12: Adobe Flash Builder Interface – Launch configuration for application – Desktop Device emulation

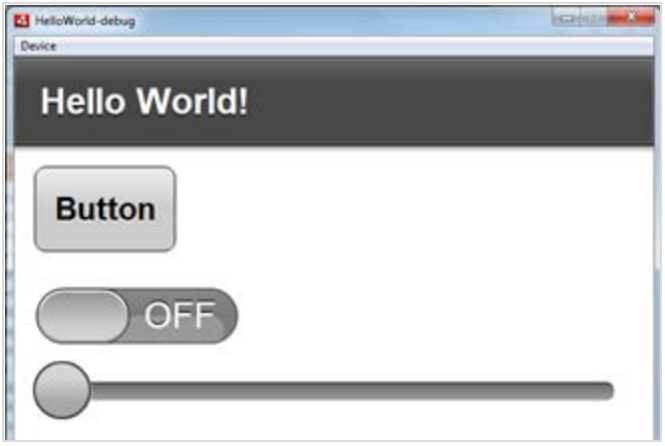


Figure 13: Adobe AIR Emulator (iPhone)

2.1.2. Screens – Getting information on the device display capabilities and status

Perhaps the first and most obvious consideration when targeting a mobile device is the screen. It is relatively small, both physically and in the number of pixels it can display. It also has a high density (pixels per inch), and various devices feature different combinations of densities and dimensions. Mobile devices can also be held in either landscape or portrait orientation.

To operate across this wide variability in size and density, Adobe AIR supports the following key APIs:

Stage.stageWidth and Stage.stageHeight	These two properties provide the actual screen dimensions at runtime.
Capabilities.screenDPI	This provides the number of pixels per inch on the screen.

By combining the information provided by these properties, an application can adapt its display to a wide range of screens—even to sizes and densities not anticipated when it was written.

Mobile applications need not support screen rotation, but should at least consider that not all mobile devices default to portrait (height greater than width) displays. Applications that do not want to be aware of screen rotations can opt out entirely by setting <autoOrients> to false in their application descriptor.

2.1.3. Map Size and Device Rotation

On mobile applications, maps are usually designed to take the entire screen real estate in a view.

Often a top bar is in available to navigate within the application and access certain functionality.

Figure 14: CartoVista Map with Top Bar and Map Taking Most of the Screen

When the device is rotated, the map and its layer content needs to resized to fit the new orientation. This behavior is handled automatically by AIR if the map component is set to a

width and height of 100% (width="100%" height="100%")

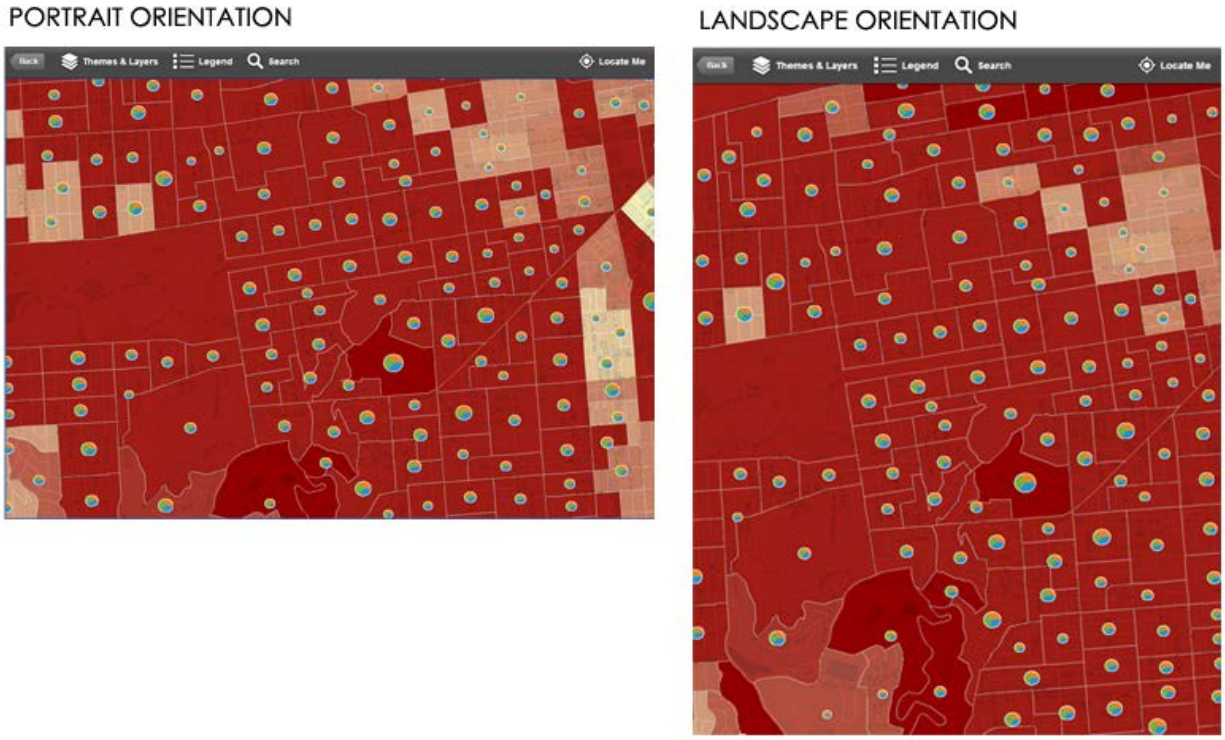


Figure 15: Map Mobile Application - Portrait and Landscape Orientation (Width and Height set at 100%)

2.1.4. Detecting the size of the screen at run time and applying styles with ActionScript

The following ActionScript functions are used to determine if the application is run on a phone or a tablet and apply the proper style accordingly.

```
import mx.events.FlexEvent;
protected function screenIsGreaterThan5Inches():Boolean
{
// Find the physical size in inches, using the devices real DPI
 var width:Number = this.width / this.runtimeDPI;
 var height:Number = this.height / this.runtimeDPI;
//this will resolve to the physical size in inches...
 //if greater than 5 inches, we assume it's a tablet
  return (width >= 5 || height >= 5);
 }
```

The styles are applied at run time when the application initializes.

```
protected function
viewnavigatorapplication_initializeHandler(event:FlexEvent):void
		{
			if (this.screenIsGreaterThan5Inches() == true)
			{
				this.styleName = "tablet";
			}
		}
```

The styles are defined in the application css file (default.css)

```
s|ViewNavigatorApplication s|Button {skinClass:
ClassReference("customSkins.CustomButtonSkin");colorNormal:#ffffff;colorDown:#7ab4ec}
s|ViewNavigatorApplication s|ToggleButton {skinClass:
ClassReference("customSkins.CustomButtonSkin");colorNormal:#ffffff;colorDown:#7ab4ec}
s|ViewNavigatorApplication.tablet s|Button {skinClass:
ClassReference("customSkins.CustomTabletButtonSkin");colorNormal:#ffffff;colorDown:#7
ab4ec}
s|ViewNavigatorApplication.tablet s|ToggleButton {skinClass:
ClassReference("customSkins.CustomTabletButtonSkin");colorNormal:#ffffff;colorDown:#7
ab4ec}
```

The user interface components (such buttons) have their skin in actionscript code. The skin assets used depends on the application DPI.

```
switch (applicationDPI)
		{
			case DPIClassification.DPI_320:
			{
				upBorderSkin =
spark.skins.mobile320.assets.Button_up;
				downBorderSkin =
spark.skins.mobile320.assets.Button_down;
				layoutGap = 10;
				layoutCornerEllipseSize = 20;
				layoutPaddingLeft = 20;
				layoutPaddingRight = 20;
				layoutPaddingTop = 20;
				layoutPaddingBottom = 20;
				layoutBorderSize = 2;
				measuredDefaultWidth = 64;
				measuredDefaultHeight = 86;

				break;
			}
			case DPIClassification.DPI_240:
			{
				upBorderSkin =
spark.skins.mobile240.assets.Button_up;
				downBorderSkin =
spark.skins.mobile240.assets.Button_down;
```

```
                            layoutGap = 7;
                            layoutCornerEllipseSize = 15;
                             layoutPaddingLeft = 15;
                             layoutPaddingRight = 15;
                             layoutPaddingTop = 15;
                             layoutPaddingBottom = 15;
                             layoutBorderSize = 1;
                             measuredDefaultWidth = 48;
                             measuredDefaultHeight = 65;

                            break;
                    }
                    default:
                    {
                           // default DPI_160
                           upBorderSkin =
spark.skins.mobile160.assets.Button_up;
                           downBorderSkin =
spark.skins.mobile160.assets.Button_down;

                           layoutGap = 5;
                           layoutCornerEllipseSize = 10;
                           layoutPaddingLeft = 10;
                           layoutPaddingRight = 10;
                           layoutPaddingTop = 10;
                           layoutPaddingBottom = 10;
                           layoutBorderSize = 1;
                           measuredDefaultWidth = 32;
                           measuredDefaultHeight = 43;

                           break;
                    }
             }
```

2.2. Mobile Map Content and Performance

Achieving good performance in mobile map applications is best reached by selecting a solid fundamental approach to each aspect of how the map data is stored and most importantly rendered.

2.2.1. The separation of subject and base map data

As map data can be very complex by nature, a clear separation of the subject data from the base map is the first step to plan for getting decent performance on a mobile device.

Tiled-map deployments have proven to be a very popular method to develop mapping functionality on the web, GoogleMaps being one the most popular example of this approach. This method of organizing the base map data is also ideal for mobile applications because tiling offers many tangible benefits:

- Seamless map navigation
- Screen resolution independence
- Low resources consumption
- Unlimited amount of details

2.2.2. Overlay of layers of interest

Setting up the base map with tiles allows focusing properly on the layers of interest for the mobile application. This is accomplished by overlaying one or more vector layers of interest on top of the map tiles base map. The layers of interest are defined with the required interactivity and potential thematic analysis options.

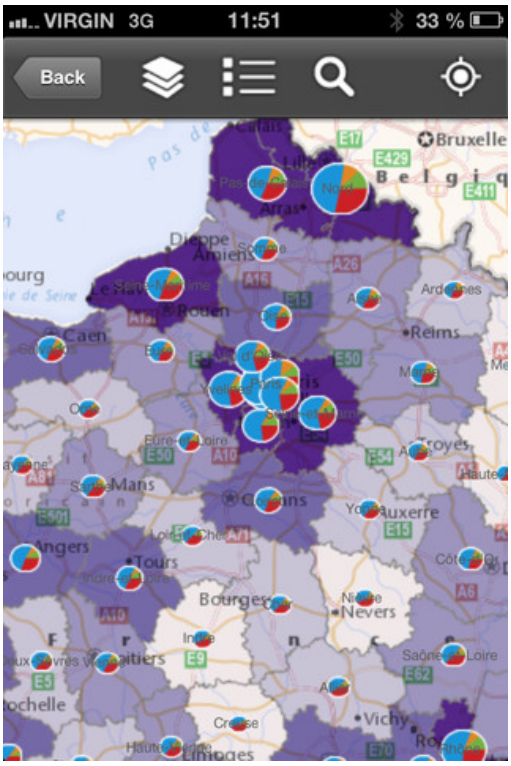


Figure 16: Tiled Base Map with Thematic Overlays (Choropleth and Pie Charts)

2.2.3. Map Features (ESRI Shapefiles)

CartoVista mobile applications can work with both vector and raster (tiles, etc.) data sources. Map outputs for mobile applications are produced with the CartoVista Publisher.

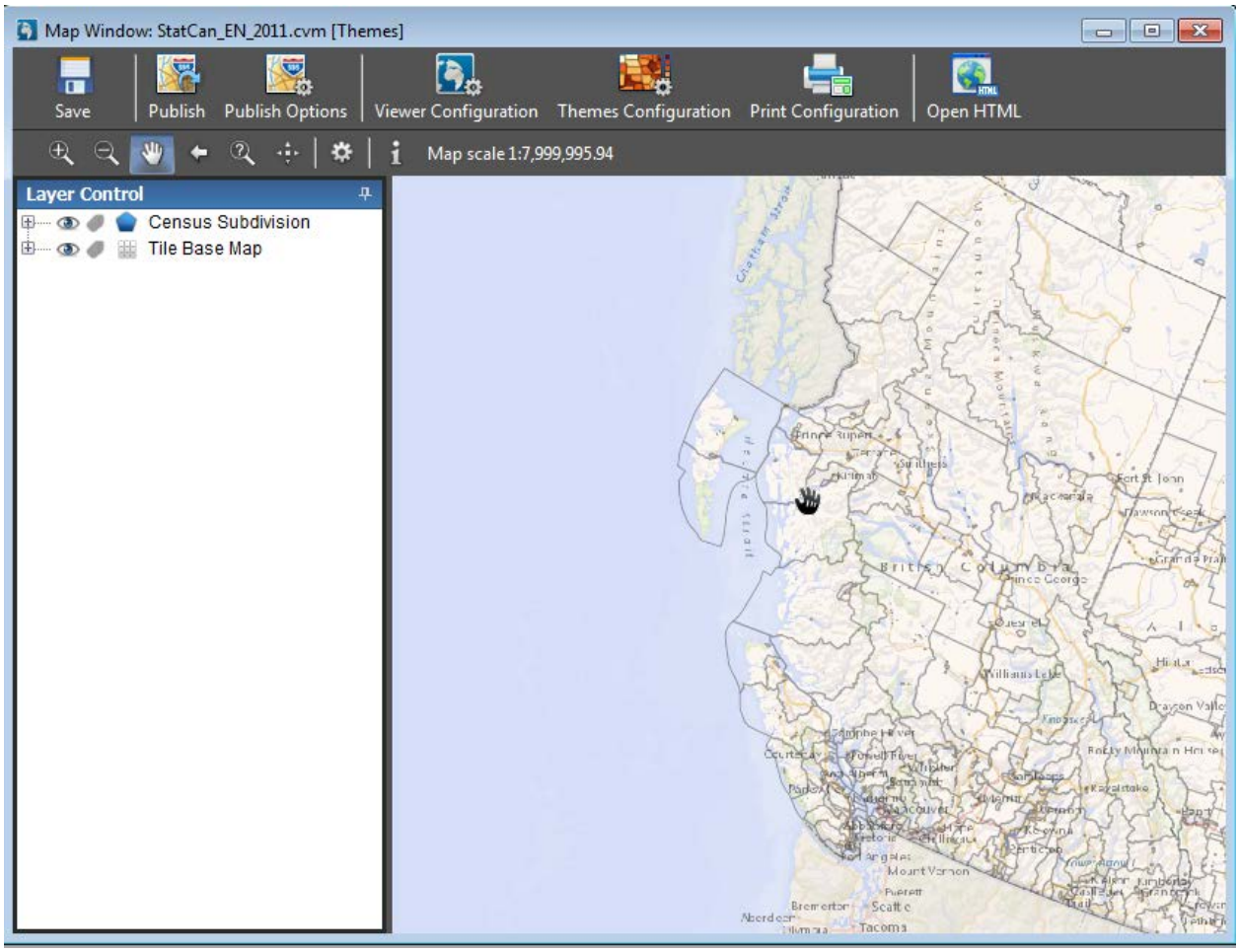


Figure 17: CartoVista Publisher Map with tiles and vector layer (Census Subdivisions)

The publisher includes advanced tools to configure the representation of the map layers through style classes.

Figure 18: CartoVista Publisher - Styles of Map Features – Polygon Style

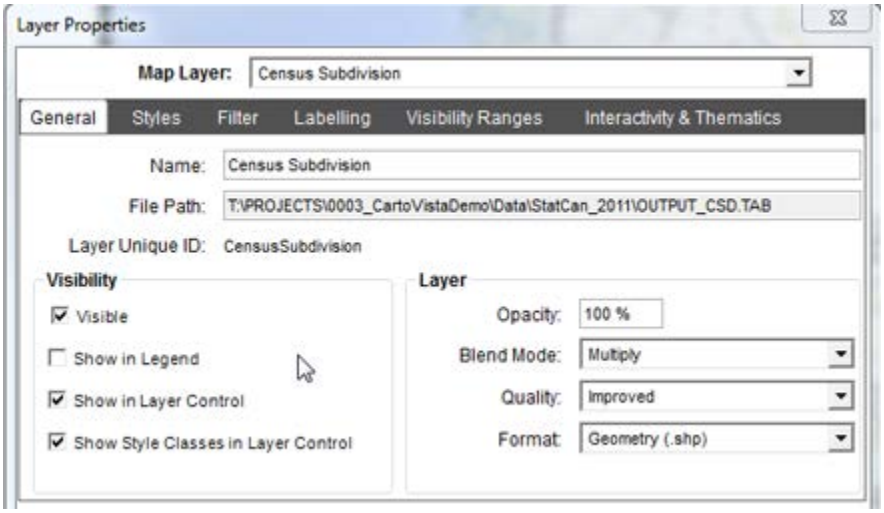


Figure 19: CartoVista Publisher - Layer Properties

The publisher outputs layers in ESRI Shape File format (.shp). Several options are available to control the quality of the output (level of generalization) and other layer settings.

2.2.4. Quad-Tree Index

The publisher map output for mobile applications includes a Quad-tree index for better performance. The quad-tree is stored in a file with a .qix extension.

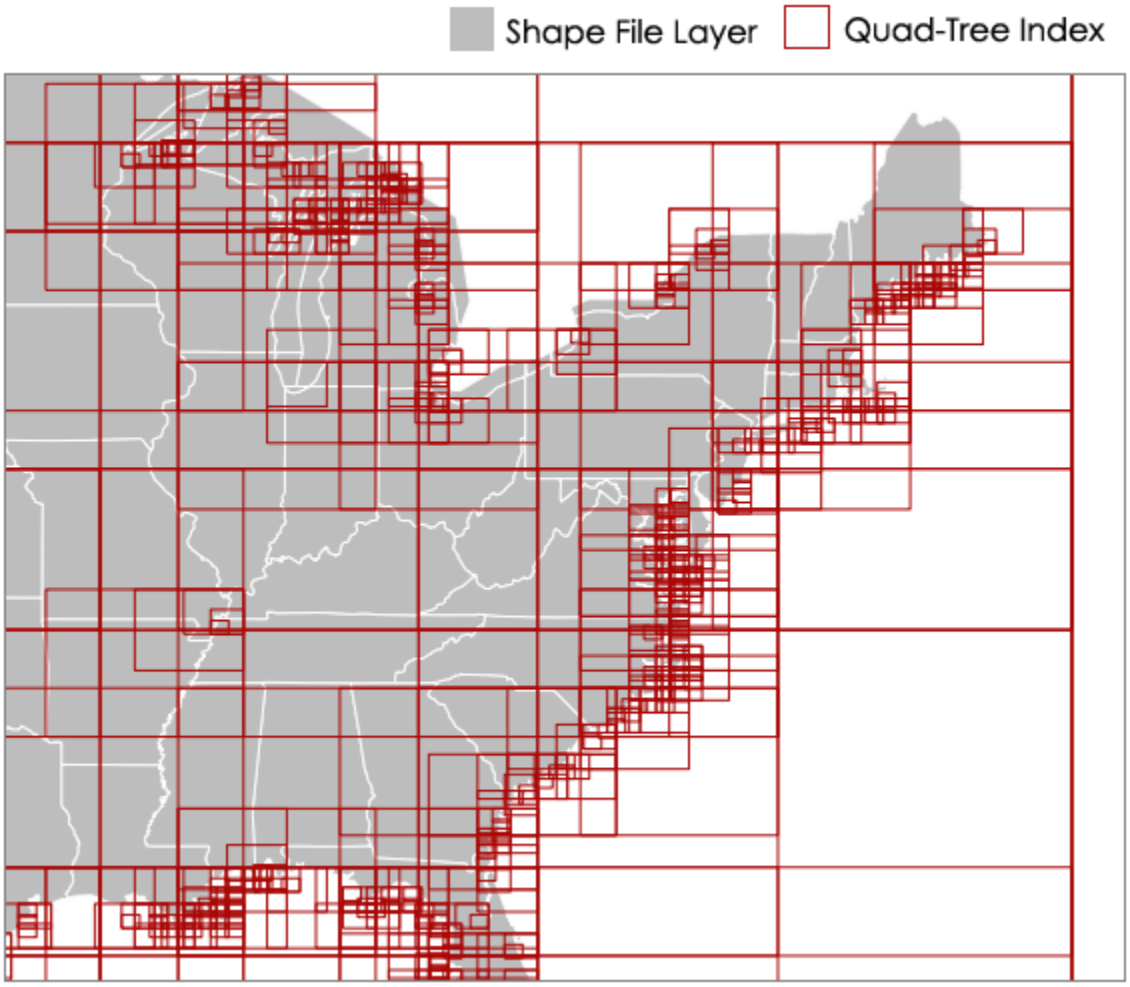


Figure 20: Quad-tree index for a shape file of the USA

2.2.5. Rendering – Leveraging the Graphics Processing Unit (GPU)

The rise of GPUs in recent information technology has essentially inverted the performance characteristics of a typical rendering pipeline.

When rendered on a CPU, each pixel is expensive. It's therefore advantageous to render from a description of shapes, performing pre-processing such that each pixel on the screen is drawn only once. This is the basic approach taken by Adobe AIR when rendering normal vector-based content such as map data. However, when working with maps that contain hundreds or thousands of features, this approach is simply not a solution because of the rendering overhead.

A GPU, on the other hand, renders shapes poorly but can easily move enormous numbers of pixels around—often, several times more pixels than can actually fit on the screen. The best way to use a GPU is to think about composing your graphic content out of a set of bitmaps and then restrict yourself to transformations of those bitmaps.

In AIR, it is possible to get the best of both worlds. Map developers can use the full capabilities of the AIR rendering model to draw and then cache the result as a bitmap that can be efficiently rendered to the screen.

In CartoVista for example, the BitmapData.draw() is one of the useful methods to get fast map feature drawing while maintaining decent performance and footprint.

2.2.6. Memory on mobile device is different

Although today's mobile devices contain plenty of RAM, it is key to remember that they do not use the same memory management model as traditional desktop operating systems. On the desktop, if memory requirements are exceeded, the contents of memory can be spilled to disk and then brought back in to memory later. This enables the operating system to keep an effectively unlimited number of programs running at once.

On mobile devices, this spill-to-disk approach is simply not available. Instead, if the demand for memory exceeds what is physically available, background applications are forced to exit, thus freeing up the memory they were using. If a request for memory cannot be satisfied at all, then the requesting application itself is exited.

Mobile app developers should strive to use as little memory as possible when it's in the background. Explicit memory management is also critical when dealing with multiples of a given object. For example, if your map application loads a set of complex vector objects, if written naively, your application will always run out of memory when that set is too large. On the other hand, if the implementation caps the number of objects that can be in memory at any one time, it will not run out of memory no matter how large the set of feature objects is. This can be achieved by freeing up an old feature before loading a new one or, even more efficiently, by keeping a fixed number of features in memory and cycling features through them.

2.3. Map Navigation and Basic User Controls

Today's smartphones and tablets are becoming predominantly touch-screen devices for any native applications. This is a key concept for mapping applications to enable simple and straightforward map navigation. The Adobe AIR gesture support allows zooming and panning the map while taking advantage of the direct tactile experience that touch screens enables.

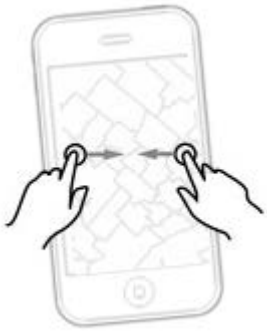
In Adobe AIR, the *GestureEvent* and *TouchEvent* class lets developer handle complex movement input events (such as moving fingers across a touch screen) that the device or operating system interprets as a *gesture*. Developers can use the properties and methods of these classes to construct event handlers that respond to the user touching the device.

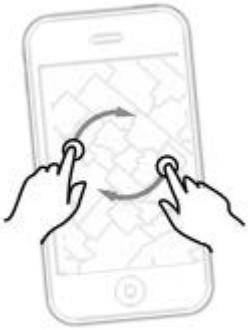
It is also possible to use the Multitouch class to determine the current environment's support for touch interaction, and to manage the support of touch interaction if the current environment supports it.

2.3.1. Navigating the map with gestures on smartphones

The following map navigation methods are common and fully supported on almost all smart phone devices on the market. The following methods are supported by Adobe AIR and the CartoVista Mobile SDK (except rotation).

One-Finger Drag		MAP ACTION: Recenter This common gesture used also for media content (images, etc.) allows to recenter the content view by a simple drag. To drag, users place a finger on the screen and move it in the desired direction without lifting it from the screen.
Two-Finger Zoom-In		MAP ACTION: Zoom in This common gesture used also for media content (images, etc.) allows zooming in the content. This gesture is executed by moving two fingers farther apart on the device (pinch open).

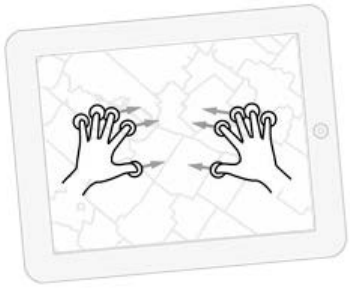
Two-Finger Zoom-Out		MAP ACTION: Zoom out This common gesture used also for media content (images, etc.) allows zooming out of the content. This gesture is executed by moving two fingers closer together on the device (pinch close).
One-Finger Double Tap		MAP ACTION: Zoom in This user gesture is commonly used to zoom in of content or an image. A double tap consists of two quick taps. On the map, one finger double tap allows zooming-in by factor of 2.
Two-Finger Tap		MAP ACTION: Zoom out This user gesture is commonly used to zoom out of content or an image. On the map, two-finger tap allows zooming-out by factor of 2.
Two-Finger Double Tap		MAP ACTION: Zoom out This user gesture is commonly used to zoom out of content or an image quickly. A double tap consists of two quick taps. On the map, two-finger double tap allows zooming-out by factor of 4.

One-Finger Tap		MAP ACTION: Get information on map feature (Attribute Info) A user gesture used to display an information bubble, magnify content under the finger, or to perform specific actions in built-in applications and features. To touch and hold, users touch the screen, leaving their finger motionless until the information is displayed or the action occurs.
Two-Finger Rotate		MAP ACTION: Not available This gesture is commonly used to rotate an object by moving two fingers in a circular motion. Since CartoVista does not support rotation this is not implemented but is available in Adobe AIR.

2.3.2. Navigating the map with gestures on tablets

Tablets have more real screen real estate so that they are well suited for supporting additional interaction, on top of the smartphone-type interactions seen above. On a tablet, the interaction is not limited to one hand. Both hands and fingers can be used simultaneously to navigate the map.

Single-Hand Drag		MAP ACTION: Recenter This hand gesture used also for media content (images, etc.) allows to recenter the content view by a simple drag. To drag, users place the hand fingers on the screen and move them in the desired direction without lifting from the screen.

Two-Hands Zoom-In		MAP ACTION: Zoom in This common gesture used also for media content (images, etc.) allows zooming in the content. This gesture is executed by moving two hands fingers farther apart on the device.
Two-Hands Zoom-Out		MAP ACTION: Zoom out This hand gesture used also for media content (images, etc.) allows zooming out of the content. This gesture is executed by moving the two hands fingers closer together on the device.
Two-Hands Rotate		MAP ACTION: Not available This gesture is commonly used to rotate an object by moving two hands in a circular motion. Since CartoVista does not support rotation this is not implemented but is available in Adobe AIR.

2.3.3. Sample ActionScript Code

The following code shows how the two finger tap gesture is captured in ActionScript and mapped to the CartoVista zoom out event.

```
protected function onGestureTwoFingerTap(event:GestureEvent):void
            {
      var point:Point = new Point(this.map.width / 2, this.map.height / 2);
      MapEventManager.getInstance(this.map).dispatchEvent(new ZoomByAboutEvent(point,
this.TWO_FINGER_TAP_ZOOM_FACTOR, false, false));

                  event.updateAfterEvent();
            }
```

2.3.4. Other Common Mobile Gestures

Mobile application users also expect support for other common mobile gestures such as swiping, flipping and panning when working with map user controls (legends, information windows, etc).

One-Finger Swipe (Flick)		This user gesture is used to scroll or pan quickly. To flick, users place a finger on the screen and quickly swipe it in the desired direction. This is also used for other actions (menus, etc.).
Two-Finger Swipe (Scroll)	 	This user gesture used to scroll content in an element with overflow capability or a scrollable inline frame element. A two-finger scroll is a drag performed with two fingers moving together in the same direction.
Three-Finger Pan (Aggressive)		Move three fingers in a horizontal line to pan an object in the X plane.
Three-Finger Tilt (Aggressive)		Move three fingers in a vertical line to tilt an object in the Y plane.

2.3.5. Mobile User Controls

The CartoVista mobile SDK includes a top bar with simple tools:

- A **Theme and Layer Control** to turn map layers on or off or choose a theme set
- A **Legend** menu to display the map legend in the interface
- A **Search** menu to search for map features
- A **Locate Me** bottom to use the GPS to show the current location on the map.

Figure 21: CartoVista Mobile Sample Map: Range Thematic Analysis – iPad Device – Gesture-based map navigation and use of icons for menus and actions

Figure 22: CartoVista Mobile Sample Map: Range Thematic Analysis – Legend / Theme Selection and Layer Control - iPhone Device.

2.4. Searching and Querying

When working with the map, mobile end users are expecting to be able to tap on map features to retrieve attribute information. Callout components are available in the Adobe Mobile framework to provide rich data tips while views can be used to layout detailed information about map features.

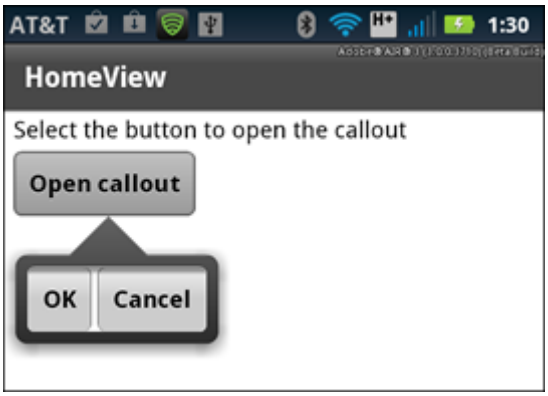


Figure 23: Callout Component in Adobe Flex 4.6

```
<?xml version="1.0" encoding="utf-8"?>
<!-- components\mobile\views\CalloutButtonSimpleHomeView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    title="HomeView">
    <s:layout>
        <s:VerticalLayout
            paddingLeft="10" paddingTop="10"/>
    </s:layout>

    <s:Label text="Select the button to open the callout"/>

    <s:CalloutButton id="myCB"
        horizontalPosition="end"
        verticalPosition="after"
        label="Open callout">
        <s:calloutLayout>
            <s:HorizontalLayout/>
        </s:calloutLayout>

        <!-- Define buttons that appear in the callout. -->
        <s:Button label="OK"
            click="myCB.closeDropDown();"/>
        <s:Button label="Cancel"
            click="myCB.closeDropDown();"/>
    </s:CalloutButton>
</s:View>
```

Figure 24: CartoVista Mobile Sample Map: Earthquake Map – DataTips – Detailed Info Window - iPhone Device.

When it comes to provide searching capabilities, the end user experience can be greatly enhanced if the interface provides suggestions as the user is typing. The Adobe AIR includes full access to the native keyboard of the device.

Figure 25: CartoVista Mobile Sample Map: Search by Municipalities – iPhone Device

When working with vector map features, spatial indexing (e.g. Quad-tree, R-Tree) is a recommended approach to help ensure that the map querying and drawing is optimized for speed. For example, only drawing features that are visible in the current view can help save on memory/graphics footprint of your mapping application.

2.5. Using Local Data & Accessing the GPS/Cameras on the Mobile Device

2.5.1. Local Device Storage

Mobile devices provide local file systems that applications can use to store preferences, documents, and the like. In general, applications should assume that this storage is accessible only to the app itself and cannot be shared with other apps. This storage can be accessed on all platforms via the `File.applicationStorageDirectory` property in Adobe AIR.

Being able to work with a mapping application without relying on a network connection is an important advantage, especially for data entry related operations in the field or disconnected data querying.

Packaging data locally is possible in Adobe AIR using files or with a SQLite database. Map tiles can also be installed in a SQLite database that can be packaged with an application to run locally without a network connection.

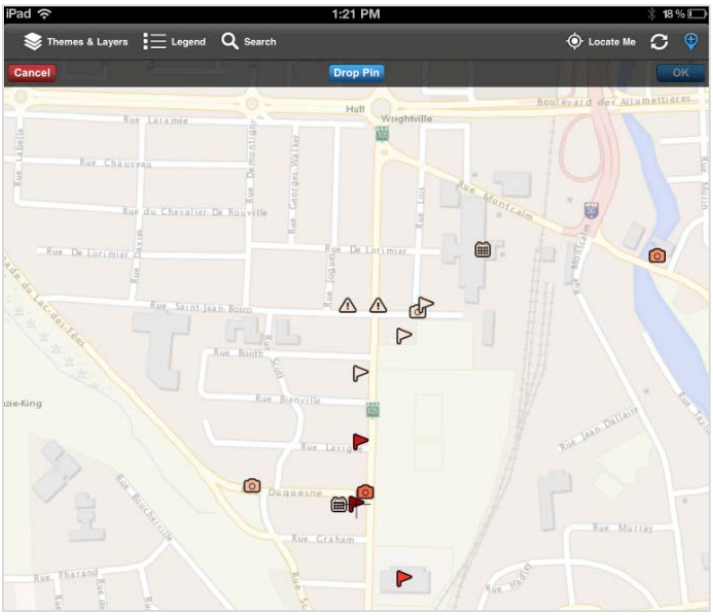


Figure 26: Data Entry Application: Enter Point on the map for local storage

2.5.2. Accessing the Mobile Device Camera (CameraUI and CameraRoll)

When it comes to taking pictures, the question for mobile devices these days is not whether they have a camera, but how many cameras they have. AIR for mobile includes simple APIs providing integration both with the cameras and with any photos already stored on the device.

Built-in camera functionality is accessed via the new CameraUI class. As the name suggests, this differs from the familiar Camera class in that it is an API to the camera's user interface, not to the camera directly. Depending on the device, this means the user may have the ability to select between still and video recording, select different resolutions, turn the flash on or off, select between front and rear cameras, and so on.

Mobile devices not only take pictures but also store them. The user's library of recorded images can be accessed via the CameraRoll class. The browseForImage() method can be used to open the device's standard UI for selecting an image from the library. The camera roll is also writable: images can be stored to the library via the addBitmapData() method.

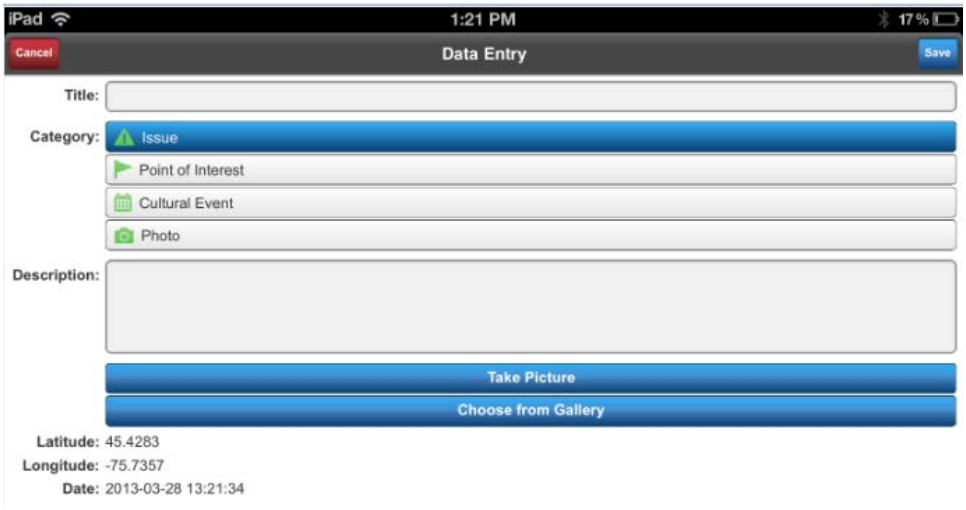


Figure 27: Field Data Entry Application with access to Camera (Take Picture) or existing photos (Choose from Gallery)

2.5.3. Using the device GPS with the GeoLocation API

The Geolocation API in Adobe AIR provides a high-level interface to access the geographical location information of the mobile device. The geographical location can be displayed on the device in the form of latitude and longitude coordinates. The API is designed to enable both one-shot location request and repeated location updates (useful while building a Geo tracking application).

In your device, when its location changes, your application can receive updates about the changes, including information on altitude, accuracy, heading, speed, and timestamp using one of the following location sources:

GPS Satellites: Enables accurate positioning if a GPS sensor is available.

Wireless networks: Enables approximate positioning if a data connection is available.

2.5.4. Code sample in ActionScript to read the device GPS data

First a listener is setup for the GeoLocationEvent.UPDATE event. The function updateLocation is called to display the current location on the map.

```
private function startTracking():void
                        {
                                this.geo.addEventListener(GeolocationEvent.UPDATE,
updateLocation, false, 0, true);
                                this._centerOnPosition = true;


NativeApplication.nativeApplication.addEventListener(Event.DEACTIVATE,
onAppInterruption);

NativeApplication.nativeApplication.addEventListener(Event.ACTIVATE, onAppActivate);

                                this.btnLocateMe.setStyle("color", 0xFF0000);
                        }
```

Then the updateLocation function received the event, evaluates the accuracy and updates the position of the point location on the map.

```
private function updateLocation(geoEvent:GeolocationEvent):void
                        {
//Deal with the accuracy….

if (geoEvent.verticalAccuracy >= 500 && geoEvent.horizontalAccuracy >= 500)
{
     this.btnLocateMe.setStyle("color", 0xFF8C00);
}
else if (geoEvent.verticalAccuracy >= 100 && geoEvent.horizontalAccuracy >= 100)
{
     this.btnLocateMe.setStyle("color", 0xE1FF00);
}
else if (geoEvent.verticalAccuracy < 100 && geoEvent.horizontalAccuracy < 100)
{
     this.btnLocateMe.setStyle("color", 0xA0FF00);
}
else if (geoEvent.verticalAccuracy < 50 && geoEvent.horizontalAccuracy < 50)
{
     this.btnLocateMe.setStyle("color", 0x00FF00);
}

//Add and update point position
if (!this._currentPositionPoint)
{
  if (!this._featureFactory)
  {
   this._featureFactory = new FeatureFactory(this.map);
  }
  this._currentPositionPoint = this._featureFactory.createImagePointFeature(id,
```

```
geoEvent.longitude, geoEvent.latitude, true, _currentPositionLayer,
_currentPositionStyle);
 }
 else
 {

this._featureFactory.positionPointFeatureWithLatLong(this._currentPositionPoint,
geoEvent.longitude, geoEvent.latitude, _currentPositionLayer.isReprojectable);
 }
 // Put visible features that are within the map extents
 // Update visibility of this layer

this._currentPositionLayer.updateFeaturesAndVisibility();
this._currentAccuracy = (geoEvent.horizontalAccuracy + geoEvent.verticalAccuracy) /
2;
this.updateCurrentPosition();

//Center on the point
if (this._centerOnPosition)
{
//Check if point is not outside the maximumExtent of the map
var mapExtent:Extent = this.map.maximumExtents;
if (mapExtent)
{
if (!(this._currentPositionPoint.cX > mapExtent.maxX ||
this._currentPositionPoint.cX < mapExtent.minX || this._currentPositionPoint.cY >
mapExtent.maxY || this._currentPositionPoint.cY < mapExtent.minY))
  {
   this.map.centerOnFeature(this._currentPositionPoint);
  }
 }
}

if (geoEvent.verticalAccuracy < 100 && geoEvent.horizontalAccuracy < 100)
{
this._centerOnPosition = false;
//Check if point is still outside the maximumExtent of the map
var mapExtent2:Extent = this.map.maximumExtents;
if (mapExtent2)
  {
   if (this._currentPositionPoint.cX > mapExtent2.maxX ||
this._currentPositionPoint.cX < mapExtent2.minX || this._currentPositionPoint.cY >
mapExtent2.maxY || this._currentPositionPoint.cY < mapExtent2.minY)
     {
       this.stopTracking();
       this.btnLocateMe.setStyle("color", 0xCECECE);
     }
   }
 }
}
```

Figure 28: CartoVista Mobile Sample Map: Detailed street map tile and GPS (Locate Me) Button – iPhone Device

2.6. Deploying your mobile application

In the mobile space, deployment happens primarily via app marketplaces. All marketplaces include on-device functionality for discovering, installing, and updating them.

AIR applications are prepared for deployment to a particular marketplace by packaging them in the appropriate platform-specific format.

For example, developers should package their application as an .ipa file for uploading to the Apple App Store and as an .apk file for uploading to an Android marketplace. These options are available from within Flash Builder, or can be scripted via the ADT command-line tool.

All mobile application marketplaces require that applications be signed that are published to them. For iOS, signing must occur with a certificate issued by Apple. For Android devices, developers should create a self-sign certificate and must use the same certificate to sign all updates to their application.

The CartoVista Mobile application is available on the following app marketplaces:

- Apple App Store (iOS): (https://itunes.apple.com/ca/app/cartovista/id618390264?mt=8)
- Google Play store (Android): (https://play.google.com/store/apps/details?id=air.com.dbxgeomatics.cartovista).
- Blackberry World (Blackberry 10 / Playbook): (http://appworld.blackberry.com/webstore/content/25893905/).

3. Conclusion

This paper has walked through the steps necessary to eliminate the various potential points of variation so that developers can think about creating a mobile mapping experience that is similar what users can find on the desktop, while leveraging the innovative capabilities of mobile devices.

The combination of Adobe Integrated Runtime (AIR) and CartoVista offers a rich development environment that enables reaching all of the major mobile platforms while building expressive, compelling mapping applications.

Developers and GIS specialists can think about building their own engaging mobile apps with a single code-base that works with most emerging hardware and new form-factors.

The methods and code samples in this presentation will help developers and cartographers understand how to build map applications that are well-suited for mobile and tablet devices while leveraging the benefits of today's mobility.

References

Goldman O, Adobe Systems inc., Adobe Developer Connection, Considerations for developing Adobe AIR applications for mobile. URL: http://www.adobe.com/devnet/air/articles/considerations-air-apps-mobile.html

Zuverink D, Adobe Systems inc., Adobe Developer Connection, Design tips for creating mobile websites, URL: http://www.adobe.com/devnet/devices/articles/design_tips_mobile_ria.html

Jennigs, F, Adobe Systems inc., Adobe Developer Connection, Using the Adobe AIR Geolocation APIs on Android, URL: http://www.adobe.com/devnet/air/quick_start_as/quickstarts/qs_as_geolocation_api.html

Adobe Systems inc., Adobe Developer Connection, Develop mobile applications for iOS (iPhone and iPad), Android, and BlackBerry Tablet OS with Flash Builder and Flex, or with HTML and JavaScript. URL: http://www.adobe.com/devnet/devices/mobile-apps.html

DBx GEOMATICS inc., CartoVista for iOS (Apple iPhone/iPad/iPod) on Apple App Store. URL: https://itunes.apple.com/ca/app/cartovista/id618390264?mt=8

DBx GEOMATICS inc., CartoVista for Android on Google Play Store. URL: https://play.google.com/store/apps/details?id=air.com.dbxgeomatics.cartovista

DBx GEOMATICS inc., CartoVista for Blackberry on BlackberryWorld. URL: http://appworld.blackberry.com/webstore/content/25893905/

Bouchard D, DBx GEOMATICS inc, CartoVista Architecture & Design, URL: http://www.cartovista.com/solutions.aspx