Welcome to the Kony Platform Developer Bootcamp

Copyright Information

Agenda

Agenda for This Week

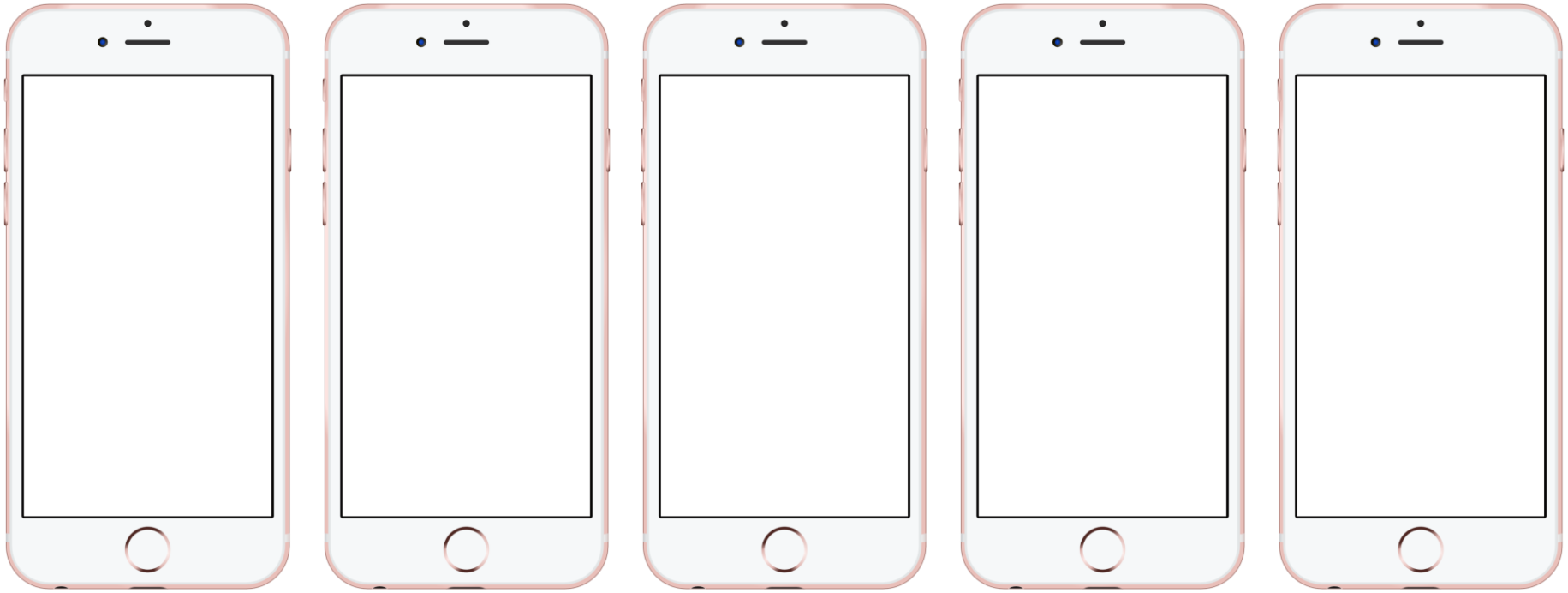
Day1 – Monday	Day2 – Tuesday	Day3 – Wednesday	Day4 – Thursday	Day5 - Friday
• Introduction to Mobility • Visualizer Basics • Actions and Animation • Using Widgets • Advanced Widgets • Design Techniques • Outside the Design	• Using JavaScript • Kony API • Animation API	• Selection Widgets • Services • Kony Widget API	• Other Application APIs • Kony Widget API	• Other Channels • More Services • Pre-Post Processor

Introduction to Mobility

What Are Your Goals?

- By the end of this week, I will be able to:
- _____.
- _____.
- _____.
- _____.

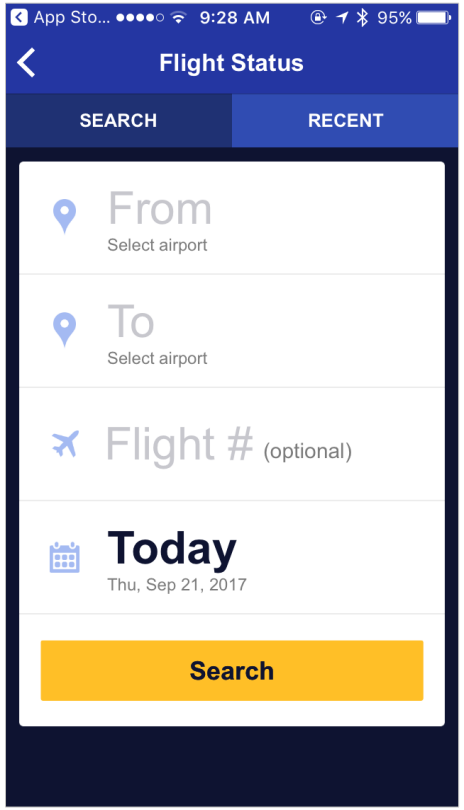
What Does Your App Look Like?

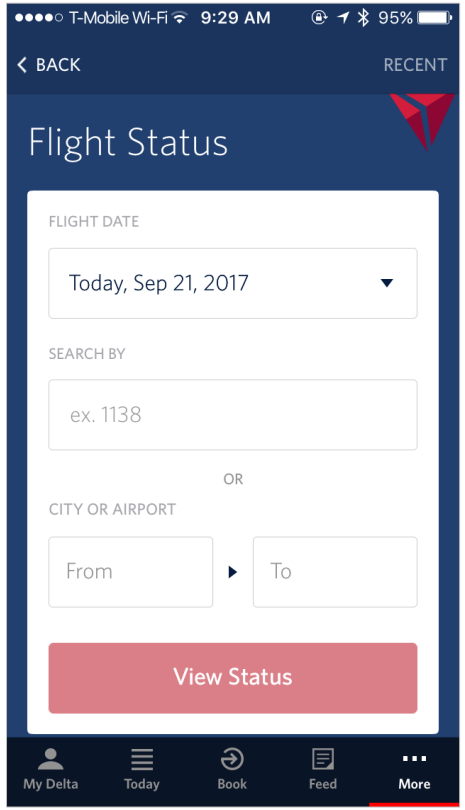


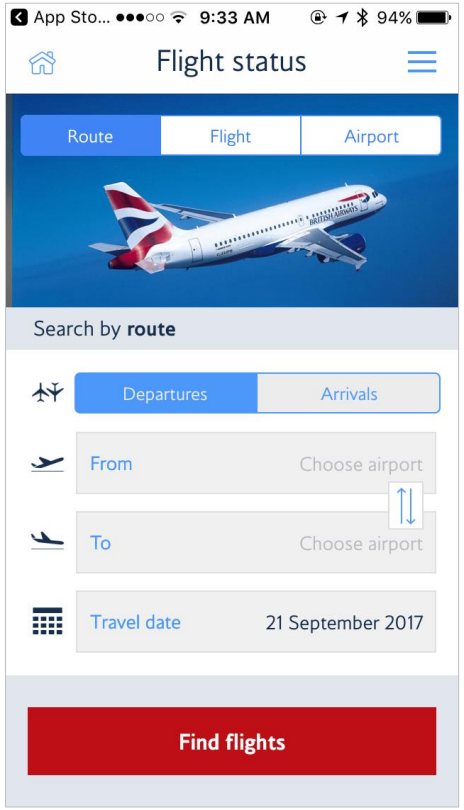
Which Elements Do You See In Your Screens?

Label	Picture	Button	Container	Menu
Line	Box	Animation	Switch	Segment
Table	Data	Map	Chart	Other?

HOW And WHY Are These Screens Different?







The Psychology Of Colors

RED: Red is the color of fire and blood, so its associated with energy, war, danger, strength, power, determination as well as passion, desire, and love

ORANGE: Orange combines the energy of red and the happiness of yellow. It is associated with joy, sunshine, and tropics.

BLACK: Black is associated with power, luxury, elegance, formality, death, evil, and mystery.

Grey: disabled;

YELLOW: Yellow is the color of sunshine. It's associated with joy, happiness, intellect, and energy

WHITE: White is associated with light, goodness, innocence, and purity. It symbolizes safety, purity, and cleanliness.

PURPLE: Purple combines the stability of blue and the energy of red. Purple is associated with royalty. It symbolizes power, nobility, luxury, and ambition.

BLUE: Blue is the color of the sky and the sea. It is often associated with depth and stability. It symbolizes trust, loyalty, wisdom, confidence, intelligence, faith, truth, and heaven

GREEN: Green is the color of nature. It symbolizes growth, harmony, freshness, and fertility.

How Does The Device Type Influence Your App?

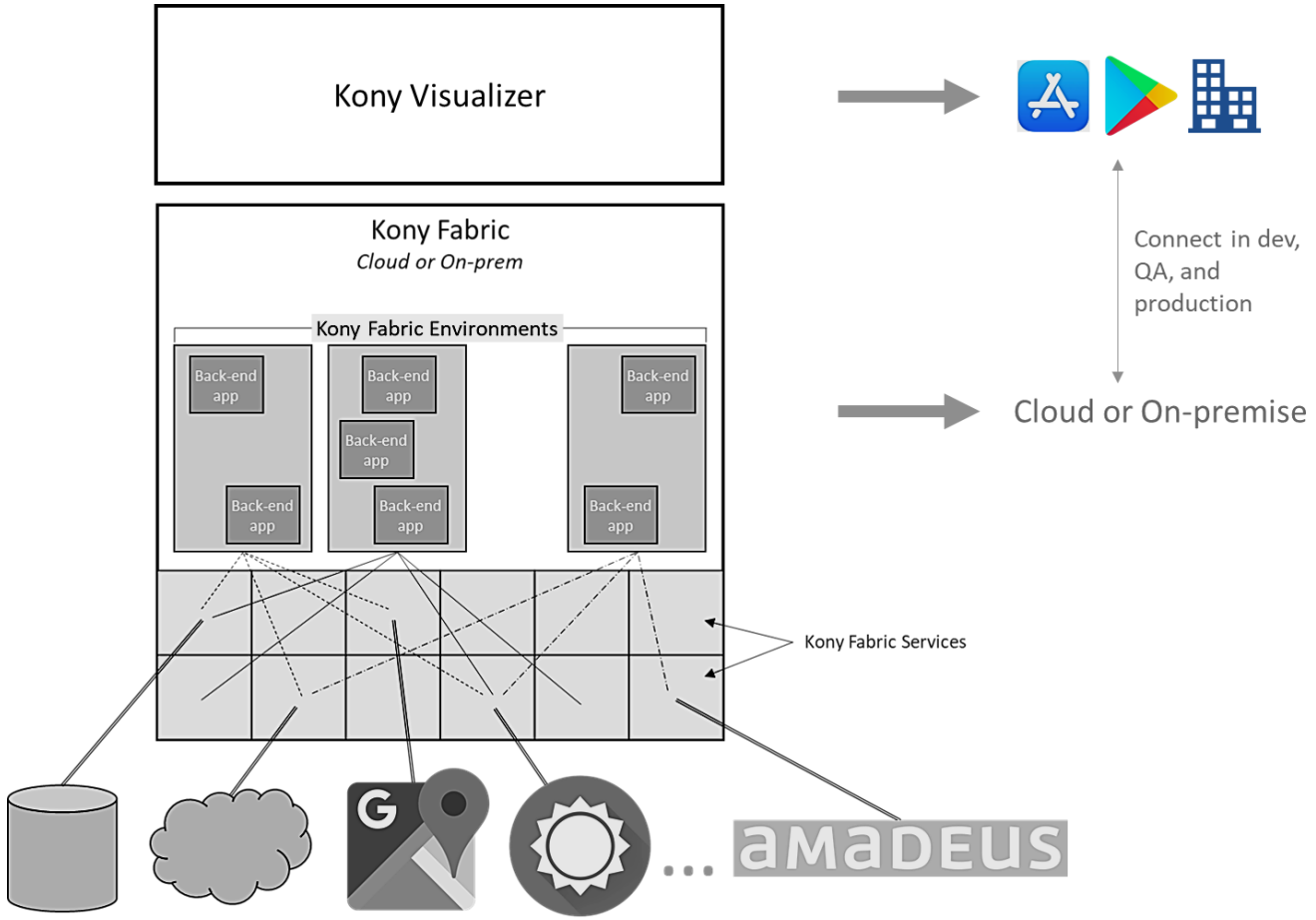
What Are Kony's Tools To Build Mobile Apps?

Front-end apps

Back-end apps

Integration services

Data providers



What Are The Main Programming Mechanisms?

Kony Visualizer

- Forms, Widgets, Actions, etc.
- JavaScript
  - Kony Visualizer uses JavaScript as development language, but NOT in the final app
  - Kony Visualizer generates NATIVE code (no JavaScript will run on the device)
- Kony APIs
- Native APIs

Kony Market Place

- Pre-configured Components
- Mostly Front-end apps; some also contain Back-end apps

Kony Fabric

- Fabric Console
- JavaScript pre and post processors
- Kony APIs

Build The Case!

	What are the **pros** of…	What are the **cons** of…
A **native** approach to app development	Team 1	Team 2
A **platform** approach to app development	Team 2	Team 1

Many Know More Than One

Student name	Programming experience	Mobile app development experience	JavaScript experience	Other experience
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	
	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	Ⓑ Ⓐ Ⓔ	

Visualizer Basics

Introduction to Visualizer

- Visualizer allows you to manage multiple projects and Projects are kept in a "workspace"
- In Visualizer, you're only working on one project at a time
- Whatever UI design you create and see on your Visualizer canvas IS exactly what it will look like on that device and You're able to switch device types and tweak the UI for each device if you want
- Visualizer currently supports Apple, Android, and Windows devices (phones and tablets)
- What we'll talk about in this training is:
  - Designing applications using Visualizer
  - Making your design "come to life" by adding actions and animations
  - Allowing collaboration by publishing your app for others to comment on

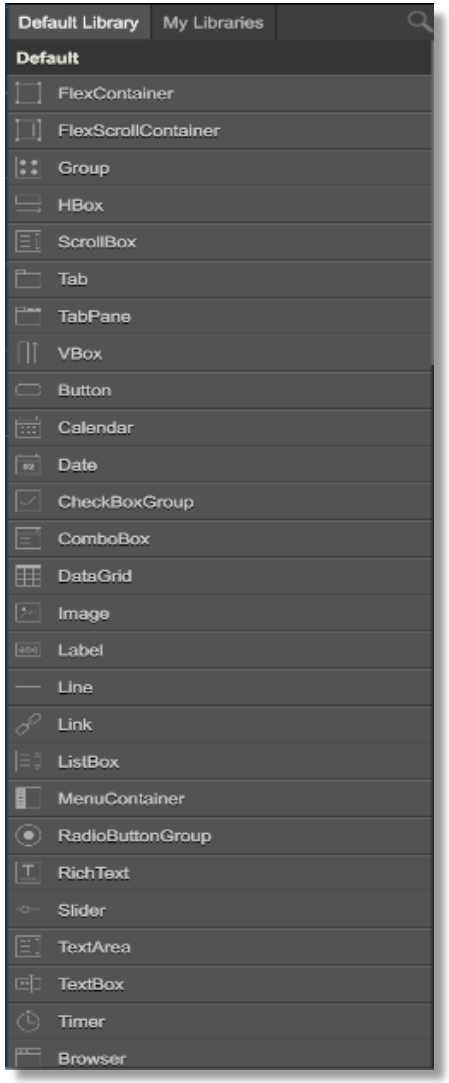
Visualizer Demo

What is a UI really?

- Designing a screen is a matter of placing UI elements on the screen and configuring how they look
- TouchingUsers can interact with the application by touching the screen
- can include:
  - A tap – for a button
  - A gesture – a swipe
  - Entering data in the keyboard
- Your UI should indicate how the user should interact with it by the way the element looks:

Visualizer Widgets Palette

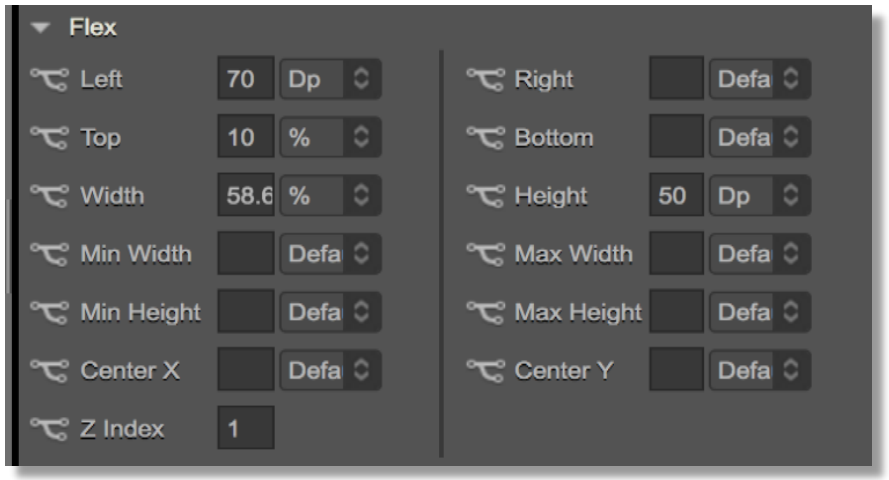
- Visualizer has a collections of UI elements – "widgets“ and each widget has a specific look that is configurable
- Each widget has defined functionality that is configurable
- What's the difference between a Kony widget and a UI element picture?
- Kony widgets allow the DESIGNER to specify how that widget will behave when the designer is done laying out the UI, those aren't pictures of UI elements, they are the real thing!
- Using Visualizer there is no more translation of the design to create the application – the design IS the application.

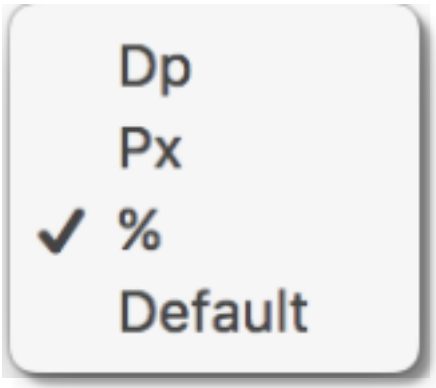


Look Tab / Layout Demo

Layout Basics

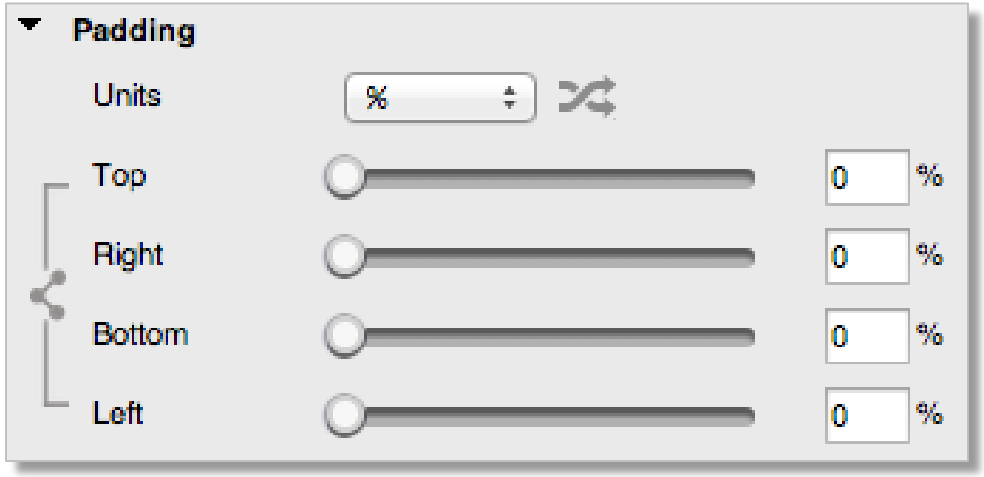
- The Left, Top, Right and Bottom values specify where the widget will be anchored in the parent container
- Width and Height can be used to specify the size of the widget
- Width will ignore Right value if Left value is specified
- Height will ignore Bottom value if Top value is specified
- Use Width % to make widget span a fixed % of the screen
- Useful when switching orientation – widget still spans the screen width by %





Button Widget Features

- For the button widget
  - You can edit the button text by
    - Double-clicking the widget and typing in text (works for all text widgets)
    - Edit the property sheet
    - Right-click it and choose the "Edit Button text" option
- To Change the positioning of the text in the button
  - By changing the content alignment
  - By changing padding settings



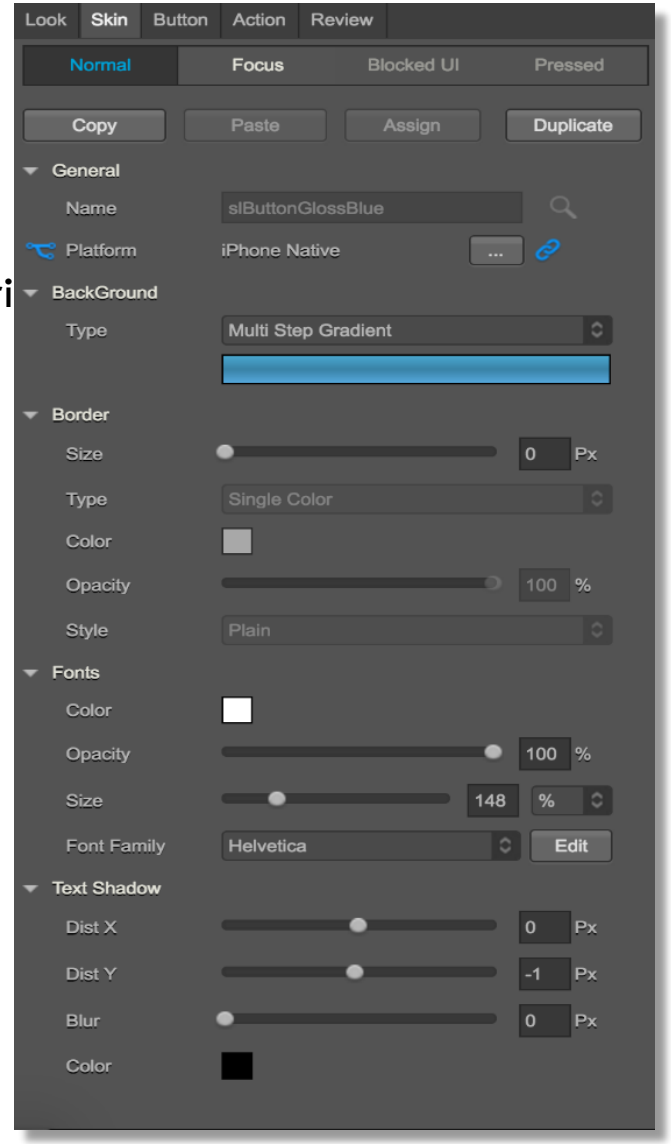
Using Skins

- All widgets can be skinned
- A skin is a definition of a widget's specific look
- A widget can have many different skins defined
- Any widget can have a skin assigned at design time OR runtime in code
- There is a common set of skin components:
  - Widget background
  - Widget border
  - Font type, size and color
  - Widget shadow
  - Text shadow

Skin Demo

Skins - Properties

- We saw that skins determine how a widget looks:
  - Background – specifies the background for the widget
  - Some devices do not support multi-step gradients
  - Border – you must set a border at least of 1 px to get the options for configuri the border, the border is part of the widget
  - Making the border transparent lets you configure the border shape (rounded corners) without seeing a border
  - Fonts – configure fonts including picking fonts for each output type
  - Shadow – for adding a widget shadow
  - Text Shadow – for adding a shadow on the widget text only



Using the Color Picker

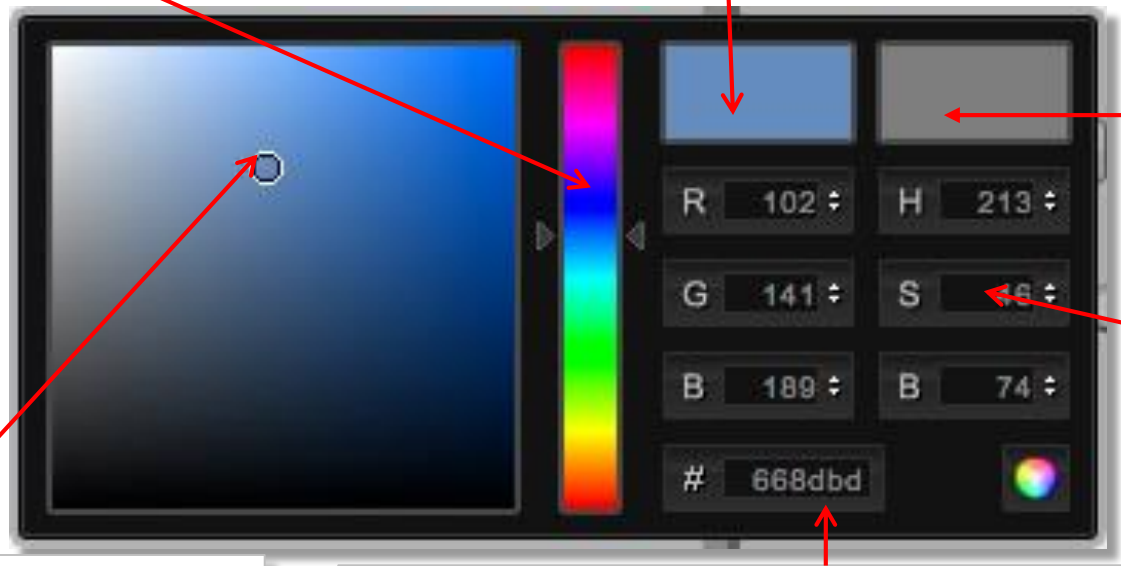
- Let's just go though the color picker features:

Either move the little handles or slide cursor along color bar to change colors

New color...

Old color...

Note: there is no cancel button so clicking this old color puts you back where you started



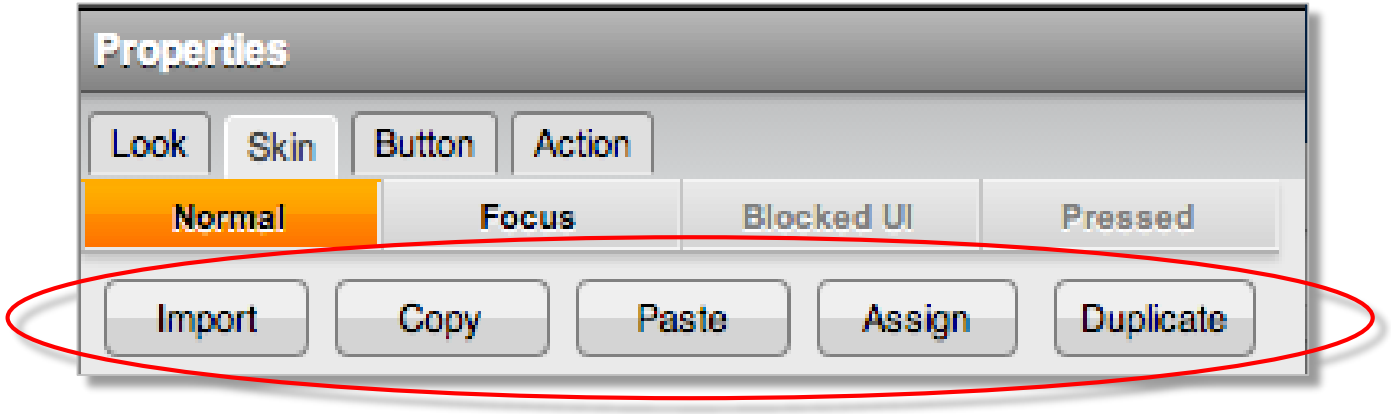
you can change these values by typing in a new number or using the up/down arrows to change values

You have to drag the circle target to change color – clicking on the background won't move it

Very common to simply copy/paste the color Hex code here – the easiest way to get a perfect color match

Applying Skins to Widgets

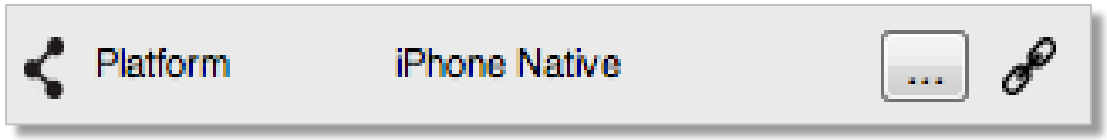
- You can use the control buttons to apply skins from one widget to another:



- Copy - Use to copy a skin from a widget
- Paste - will create a new skin based on the copied skin
  - The new skin will have it's own separate name different from the copied skin
- Assign - will assign the copied skin to the new widget
  - They both now share the same skin definition
  - Change one instance and all instances change
- Duplicate - will take the widget's existing skin and create a new instance basically un-assigns the skin and gives it a new name

Forking Skins

- Skins can be common across devices or forked:



- The  icon is the fork icon
  - When it's black – the skin is forked. Ex: 
  - When it's light – the skin is not forked or it's the common skin that is applied. Ex: 
  - Note: this fork icon means the same thing for any property that is forkable

- The  icon is the lock icon:
  - When it's black – the skin is tied to other device's version of the skin. Ex: 
  - When it's light – the skin is now forked and only the canvas you're currently working on will change – the other device's versions will remain unchanged. Ex: 
  - Clicking on the light icon will re-lock it & possibly apply the original version of the skin

- There is a common version of all skins that will be applied to non-hero platforms not used in Visualizer

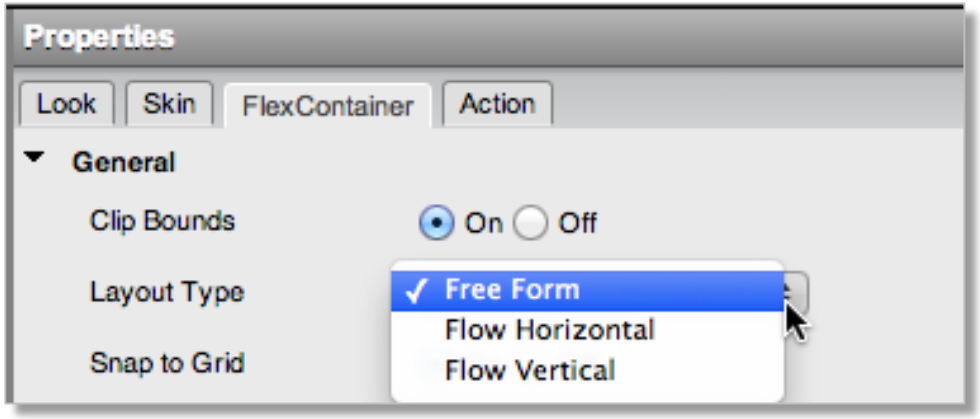
Containers

- So far the only container we've used is the form itself
  - Containers can be used for a lot of different purposes:
    - For grouping several widgets for layout effects
    - For creating interesting visual effects – as an overlay or background
    - For controlling animations
    - For saving work in a library
- The main container we'll use everywhere is the Flex container
- The Flex Scroll Container works the same but allows the configuration of vertical and/or horizontal scrollbars to scroll content
- Let's take a look at how the Flex containers work...

Container Demo

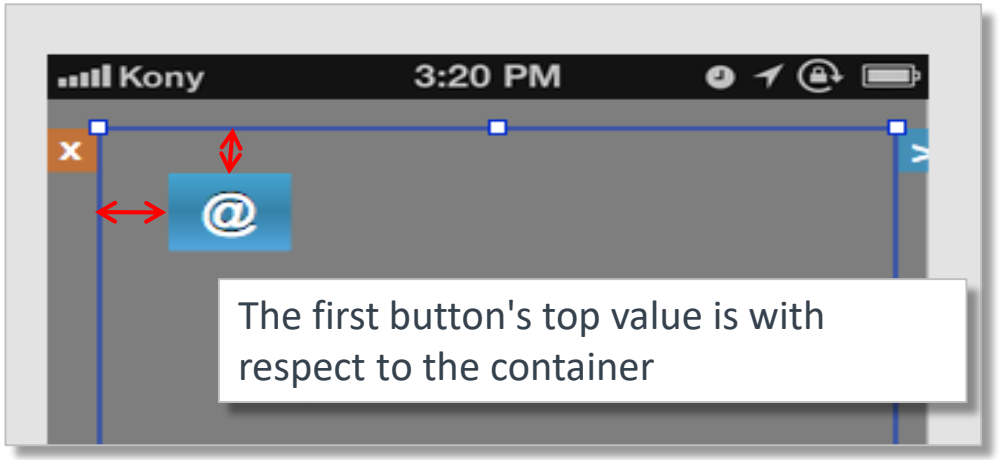
Flex Container Properties

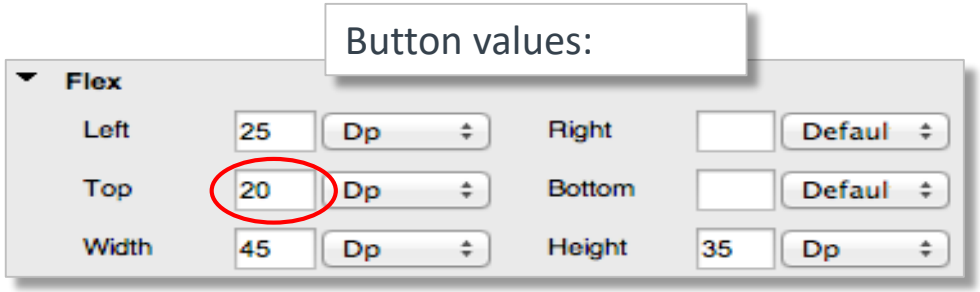
- Flex Containers allow you to group your UI elements and don't "cost" anything so use as many as you need/want in your screen designs
- Flex container properties:
  - Clip Bounds is on by default – typically what you want
    - Clips widgets off at container border
  - Layout type:
    - Free Form – where all child widget layout properties are with respect to container
    - Flow Vertical/Horizontal – where all child widget layout properties are with respect to the previous widget in the container



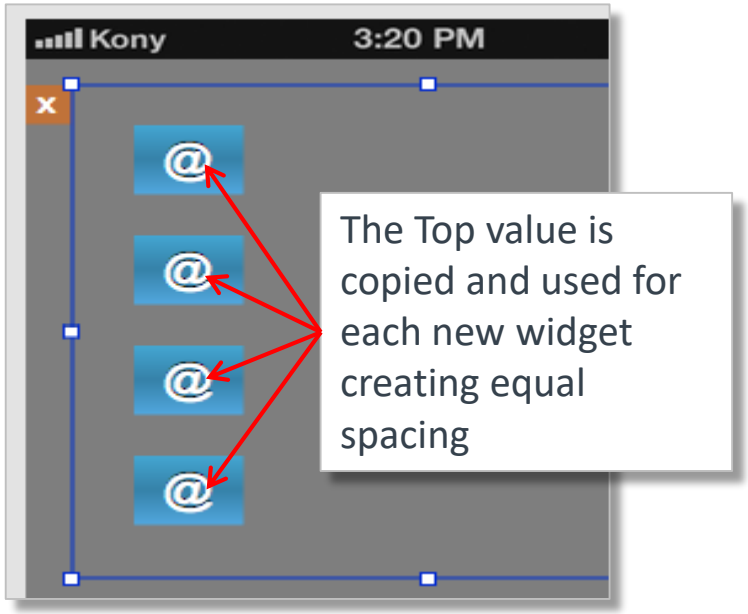
Flex Container - Flow Vertical

- Here is a Flex Container with a button and the Flex Container is set to Flow Vertical





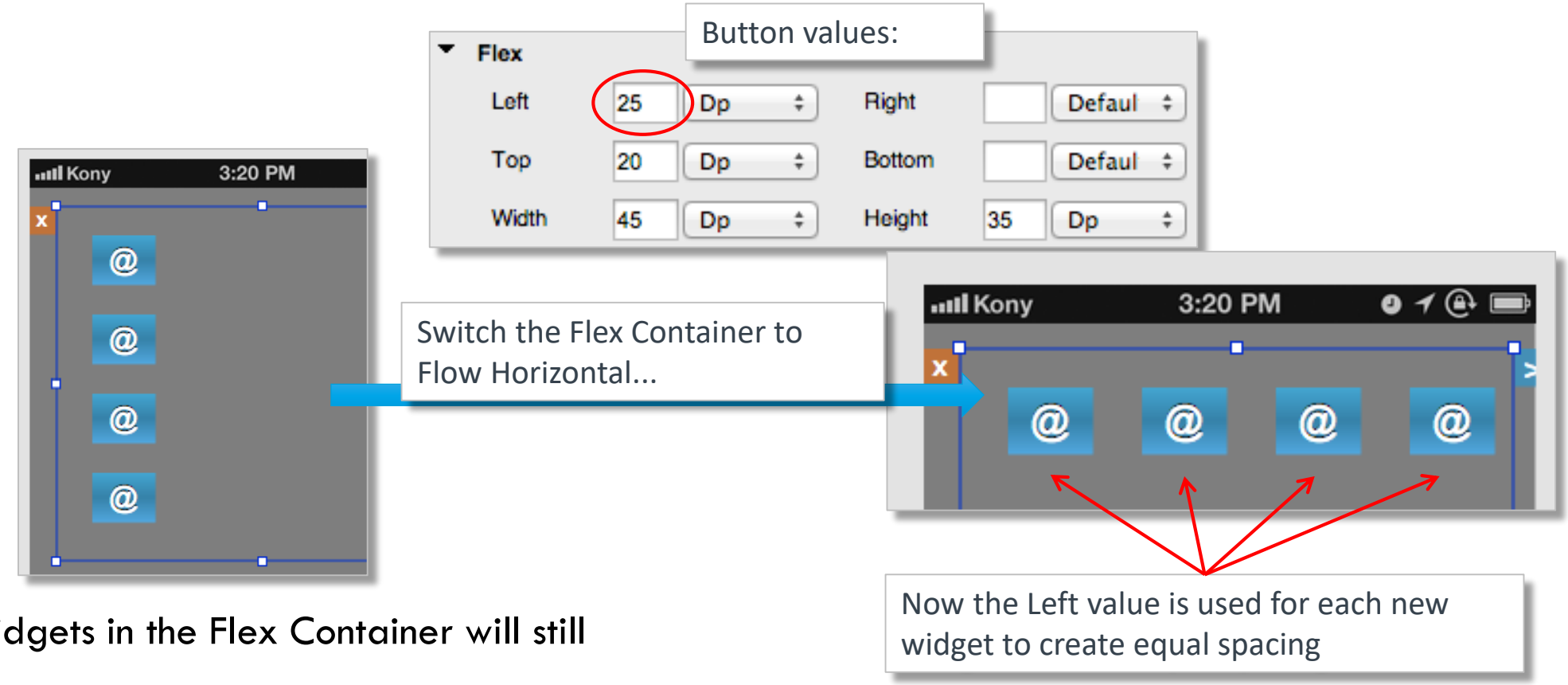
- The copied button will adopt the SAME top value
  - The spacing matches the original top value
  - Each item's top is with respect to the previous widget



- This makes layout very easy as you copy/paste elements

Flex container - Flow Horizontal

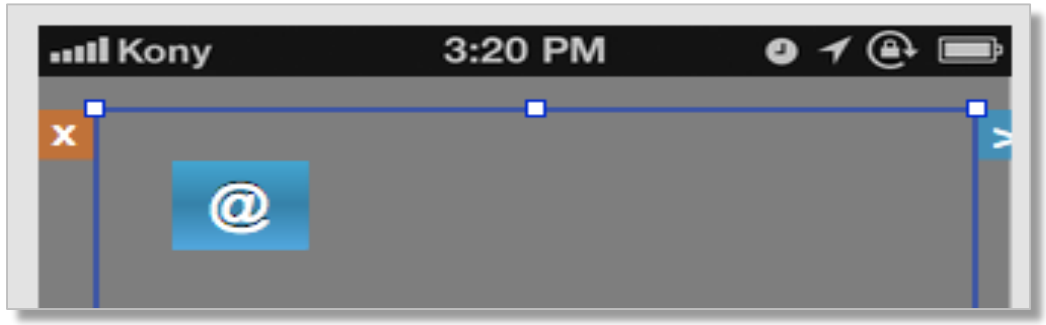
- When the Flex Container is set to Flow Horizontal:
  - The Left value is used and copied with respect to the previous object



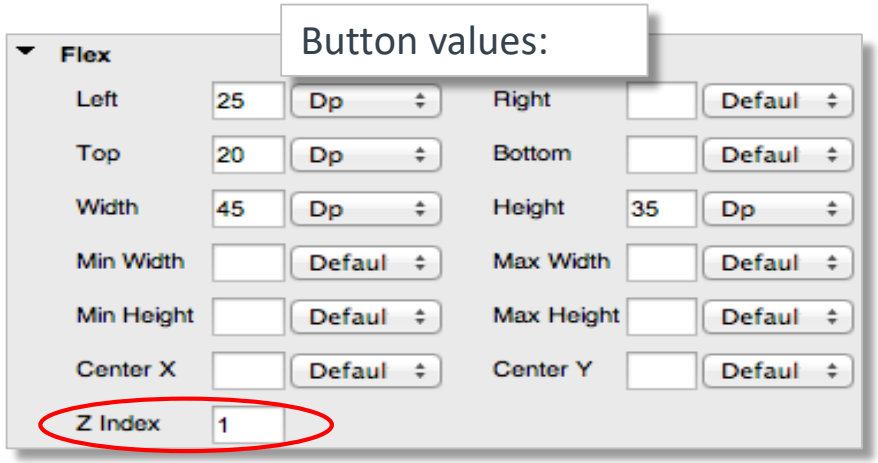
- Any new widgets in the Flex Container will still use relative spacing

Flex Container - Free Form

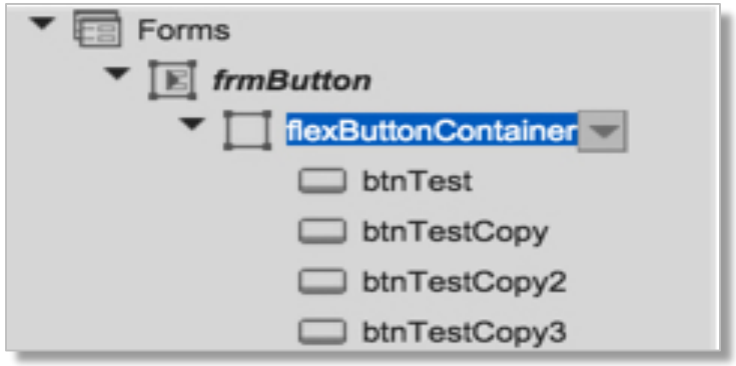
- When the Flex Container is now set back to Free Form:



Button values:

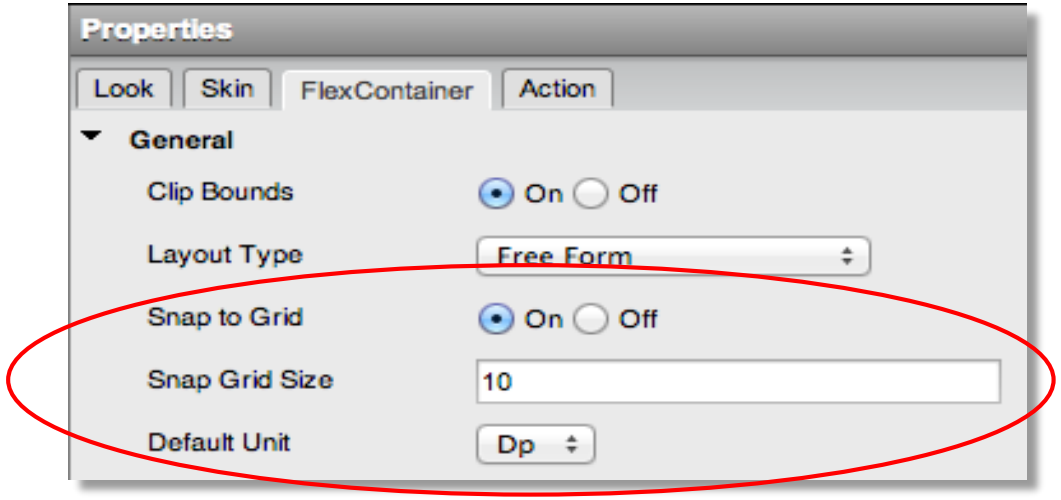


- All copied widgets had the SAME values
  - These Top/Left values are now with respect to the form
  - They are all in exactly the same location stacked on top of each other
- Z Index – is used to stack objects
  - By default, if Z Index is the same, the order will be according to the widget hierarchy
  - Higher Z order moves widget to the top



Flex Container Grid

- Just like the form has a default grid size of 10Dp, containers can have a grid too:

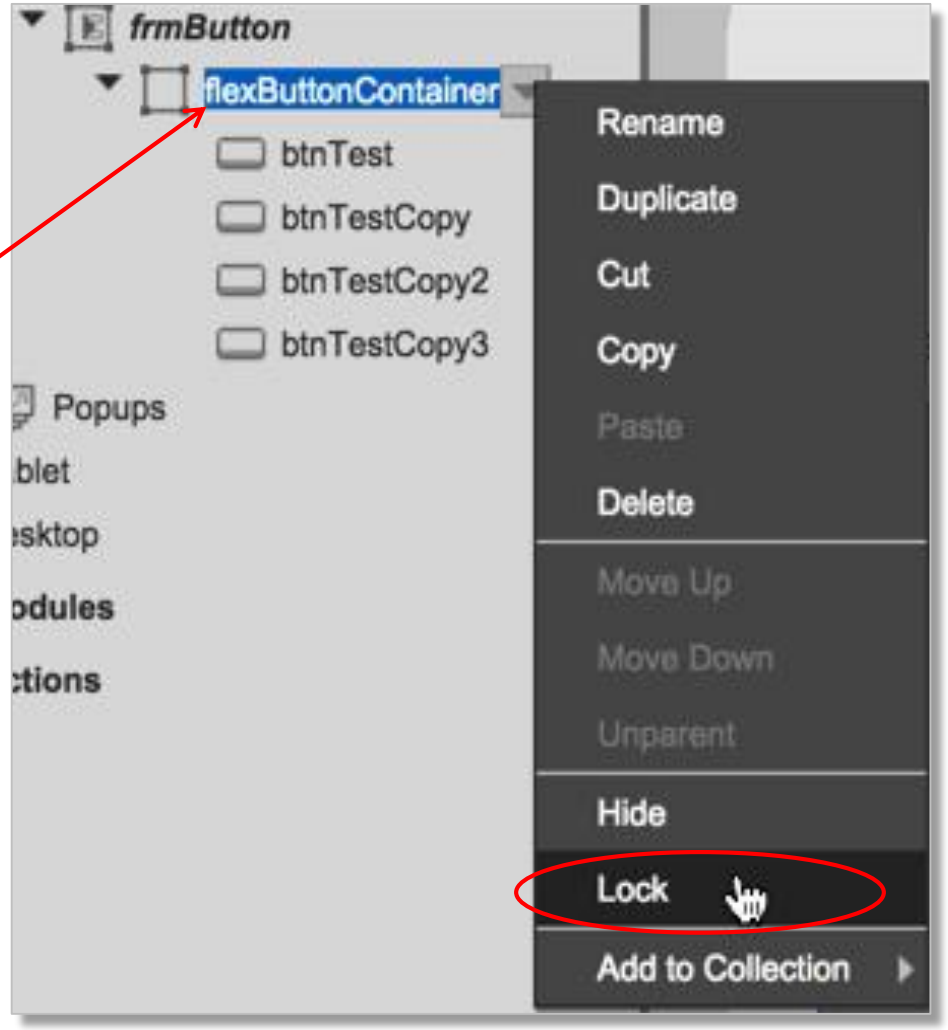


- The default grid configuration for a Flex Container is the same defaults as the form
- Remember – regardless of the grid size:
  - You can enter any layout values you want
  - You can use the arrow keys to move a widget 1 Dp at a time
- Grid just helps you when dragging widgets around

Locking Widgets

- Making mistakes can sometimes take a bit of time to correct

In this example, locking that flex container locks all the widgets inside too



- Best practice is to Lock your widget when you've finished configuring it so that you can't accidentally alter it:

Exercise – Calculator

- This exercise will give you some practice laying out a screen using containers and the layout principles we talked about.

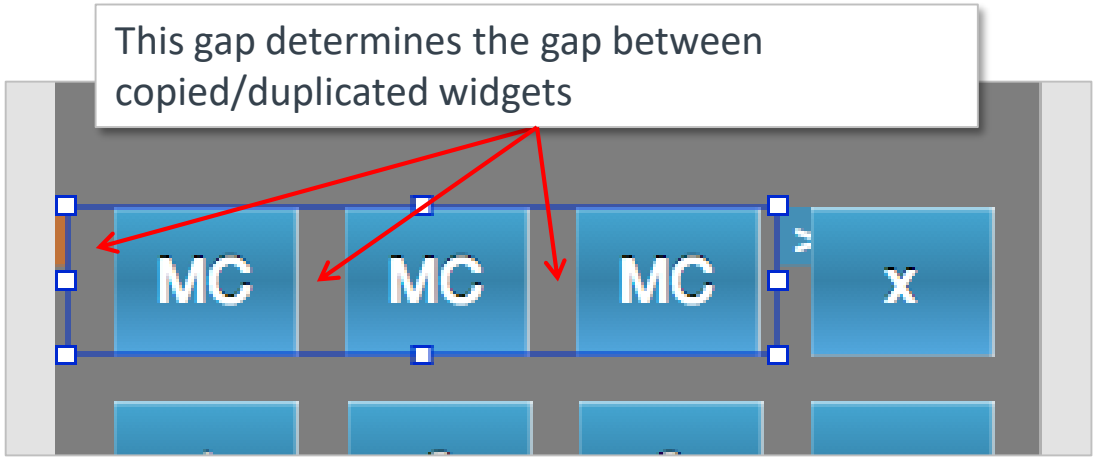
- Create this:
  - Each button is 60Dp wide/50Dp tall
  - Each button has a 15 Dp gap

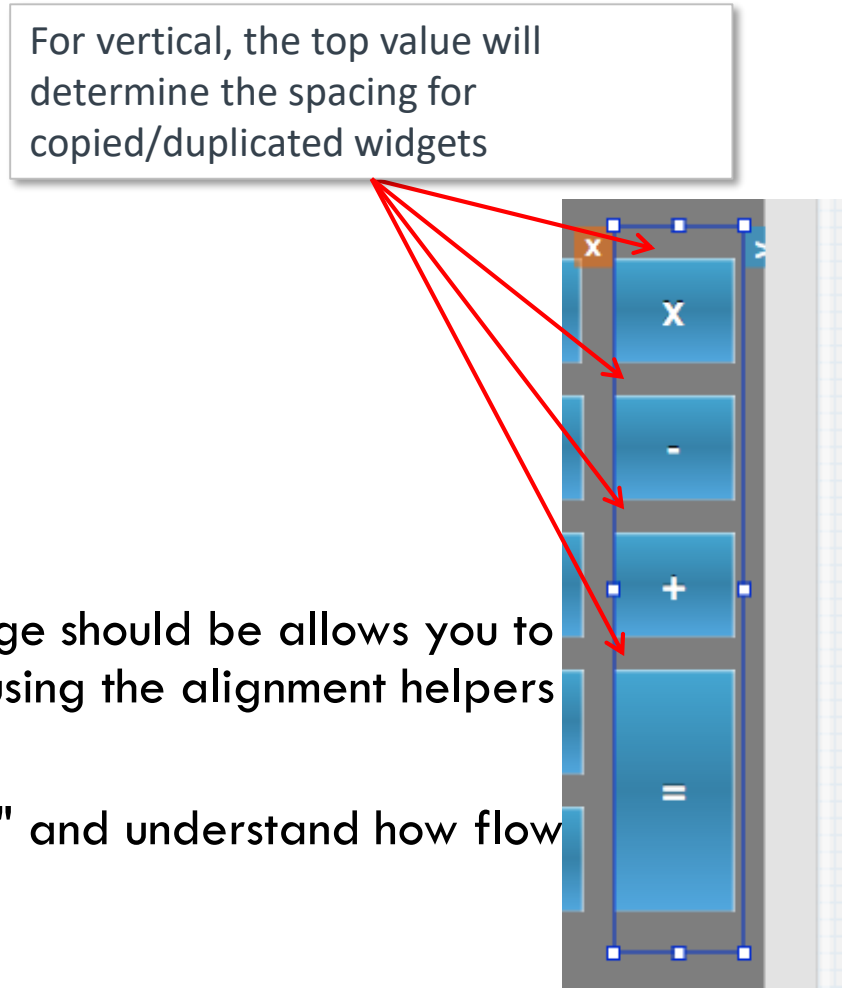
- Build the top 3 M buttons in a flex container set horizontal
- Build the 4 function keys in a flex container set vertical
- Build the rest free form

- Next slide talks about tips/techniques

Exercise – Calculator (cont'd)

- Top buttons in flex – duplicating button replicates settings, so the left edge of the flex should be the gap between buttons you want to reproduce – i.e., 15 Dp





- The same type of calculations apply to the vertical one
- To adjust the main buttons – knowing the math of where each edge should be allows you to edit them by typing in top and left values for one row and then using the alignment helpers move the other ones to match and line up
- NOTE: This exercise is to practice our flex container layout "math" and understand how flow vertical/horizontal work

Calculator Example 2

- To really make our calculator robust, we should use % rather than DP to set this up
- You need to do your layout math to get this figured out
- For the first 3 rows:
  - 4 button widths + 3 between button gaps + 2 outside gutters = 100% screen size
    - Each button can be 20% width
    - Each between button gap can be 4% width
    - Each outside gutter can be 4% width
- You can do similar calculation on the height and double width/height buttons
- Now it'll render filling up the same screen amount on every device and resolution

Testing Your Design

- We're going to talk about actions and animations in our Visualizer application
  - The Visualizer canvas can show you the static design, but most designs "come to life" when actually run
  - The static WYSIWYG canvas only goes so far in finalizing your design
- We'll now need to actually RUN our application on a device to see it working
- To do this, you must first install the Visualizer Functional Preview application on your device
  - The process will then be:
    - From Visualizer you'll be able to "build" your app – make it available to the Functional Preview application
    - From the Functional Preview application, you'll load up your application and it will run inside the Functional Preview application
- Let's see how all that works...

Functional Preview

- Functional Preview is an application that anyone can download from the App store
  - It's listed as "Kony Visualizer"
- The Functional Preview app lets you "run" your Visualizer apps so you can see all the actions and animations working
  - It ALSO allows for collaboration by letting people enter comments/notes and associate them with your app
  - You can then review comments
  - This is a perfect collaboration tool with the rest of the organization

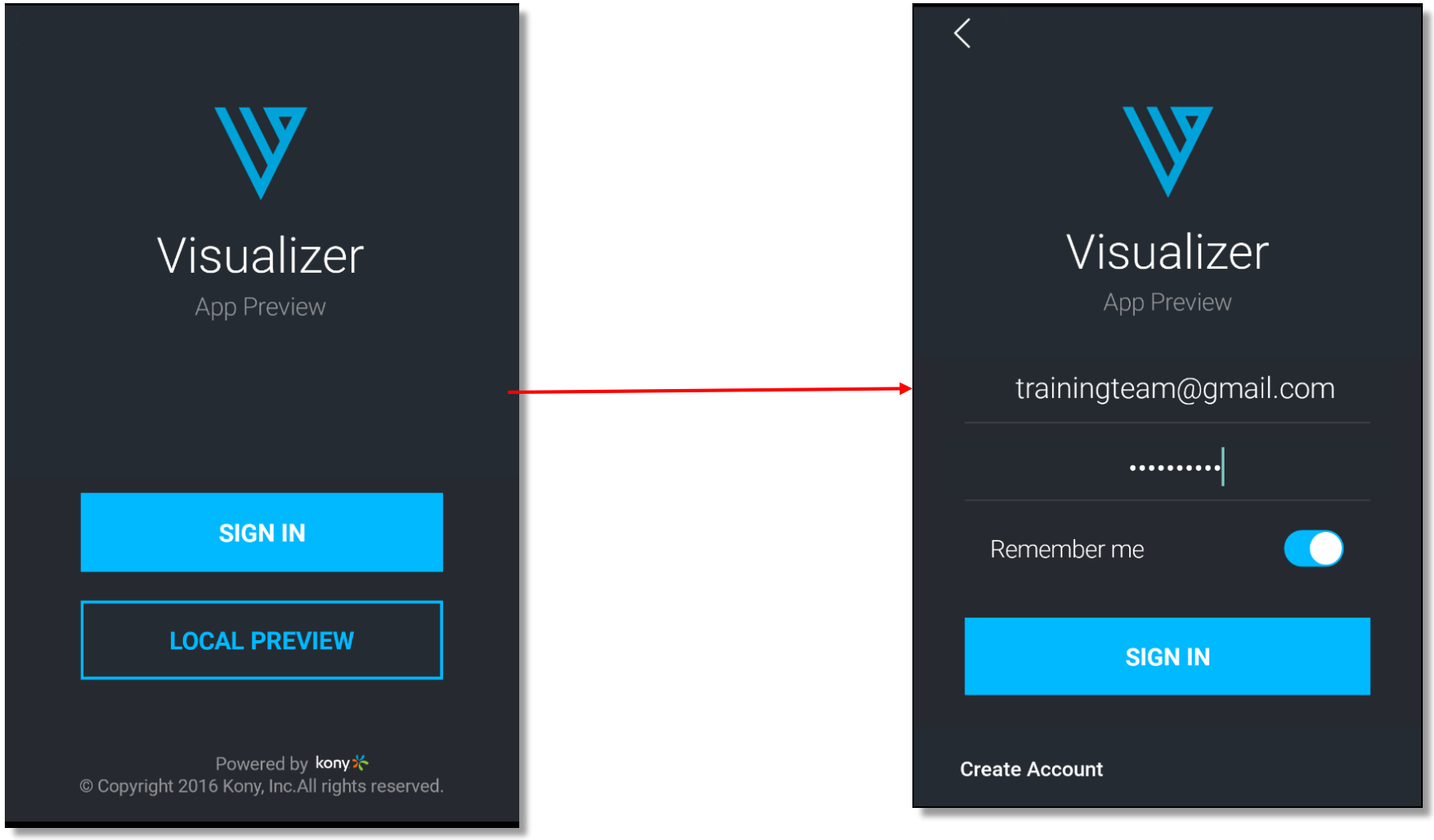
Functional Preview (cont'd)

- Functional Preview can be run in 2 modes:
  - Running the application that's been published on the Kony Cloud
    - For sharing the app with others
  - Running the application locally connecting to Visualizer
    - For designers, testing their design from their desktop/laptop

- Let me show you how this works...

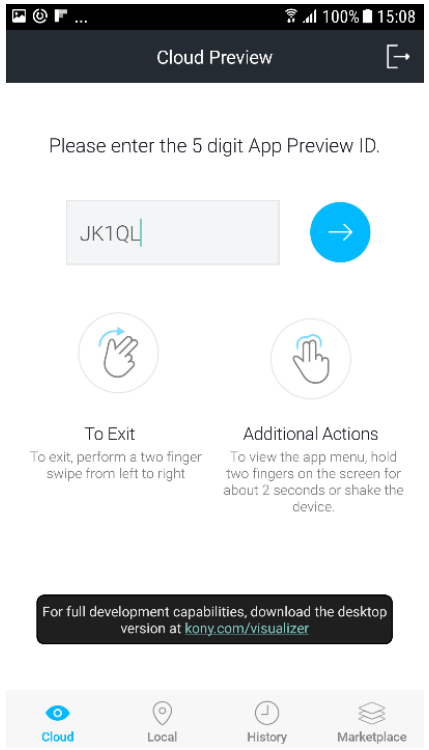
Functional Preview (cont'd)

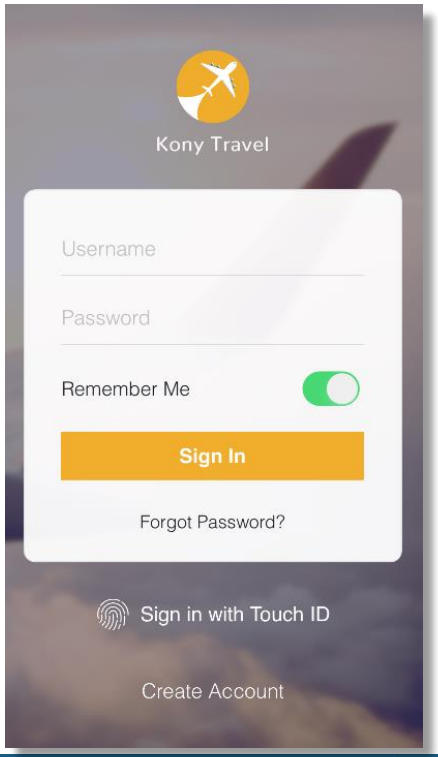
- After launching the application you have to sign in.



Function Preview - Cloud

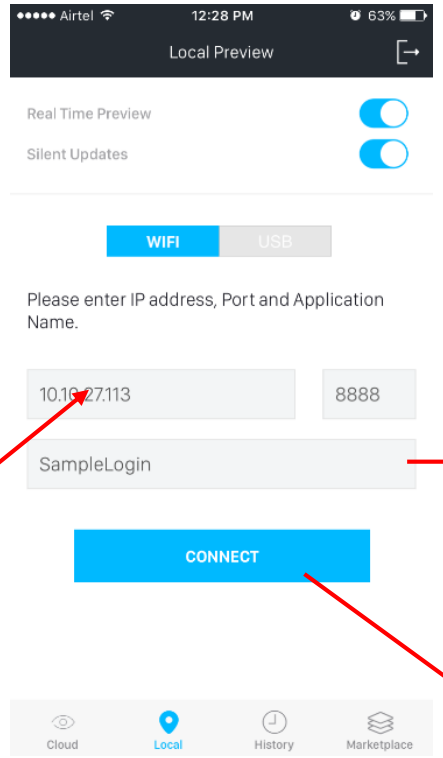
- After logging in to the application, we can see the various modes available as the menu items at the bottom and by default cloud mode is selected
- To use the cloud mode we have to Run the application in Visualizer and publish to the cloud.
- We have to enter 5 digit code and click on arrow to preview the application.



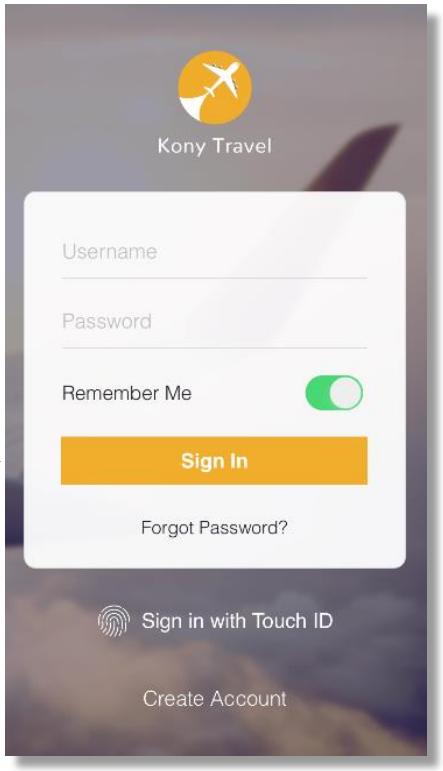


Function Preview - Local

- To use the local mode we have to Run the application in the Visualizer



Enter IP address , Port Number and the Visualizer Application Name

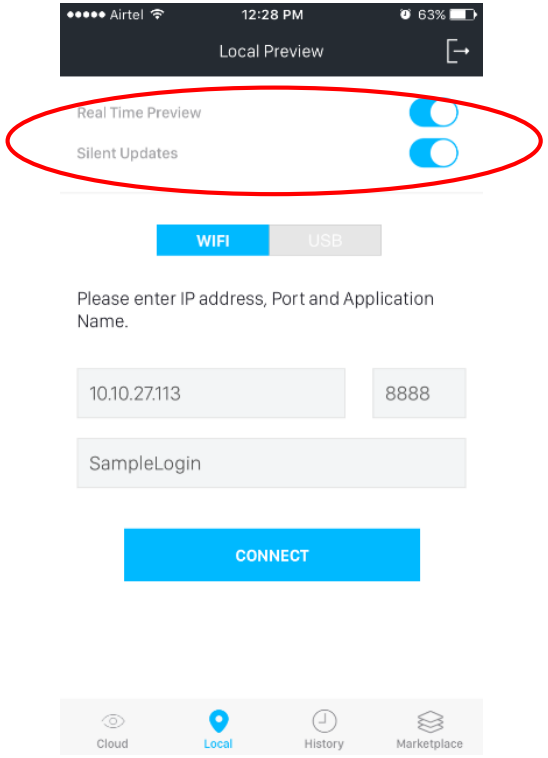


Click on Connect to launch the preview application

Local Mode - Real Time Preview

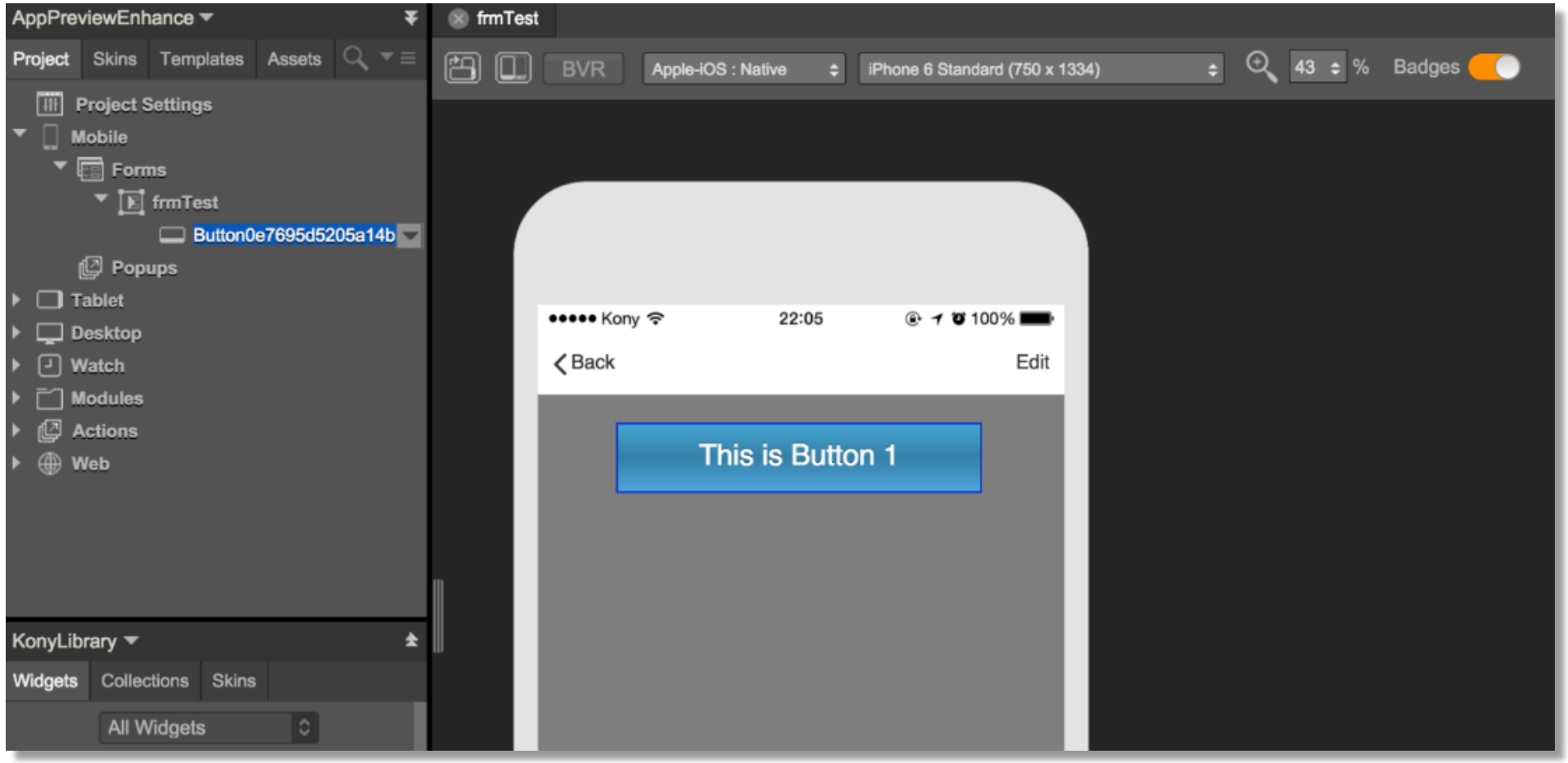
- Kony Visualizer Functional Preview app supports Real time Preview

- If you want the incremental build changes that take place in Kony Visualizer to be automatically reflected in the app preview on the device, enable **Real Time Preview** option in the Functional Preview app



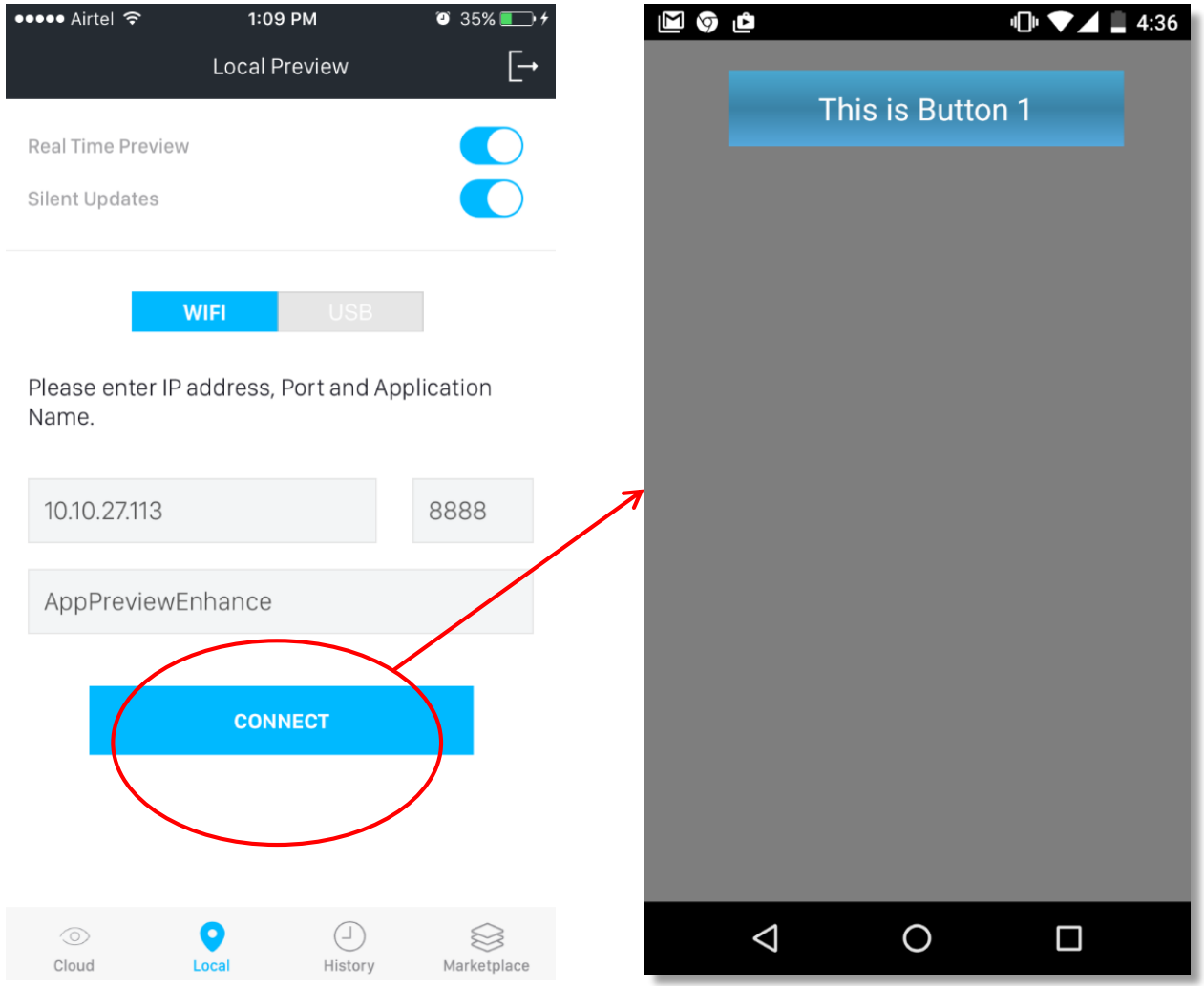
Testing Real - Time Preview (cont'd)

- For example, create a project in Kony Visualizer with sample form like below



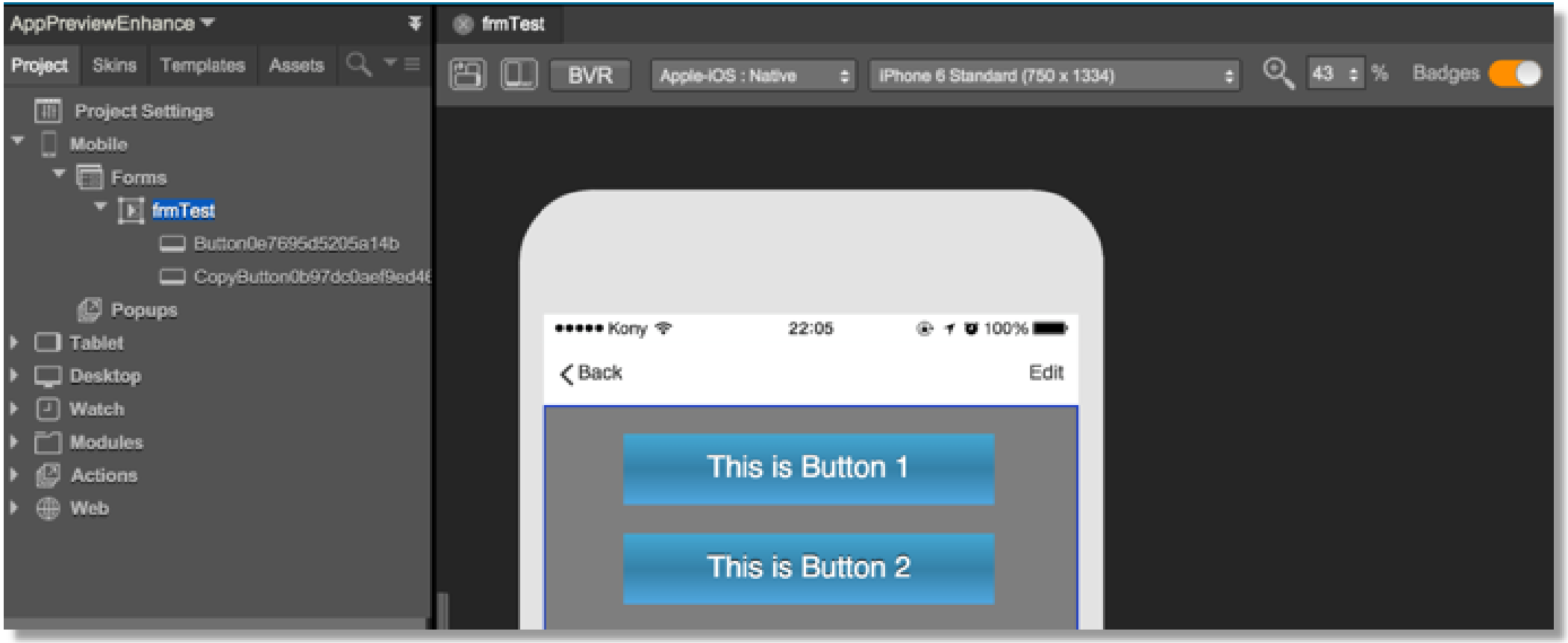
Testing Real-Time Preview (cont'd)

- Now, run the project locally and test it in the Functional Preview app by enabling **Real Time Preview** option



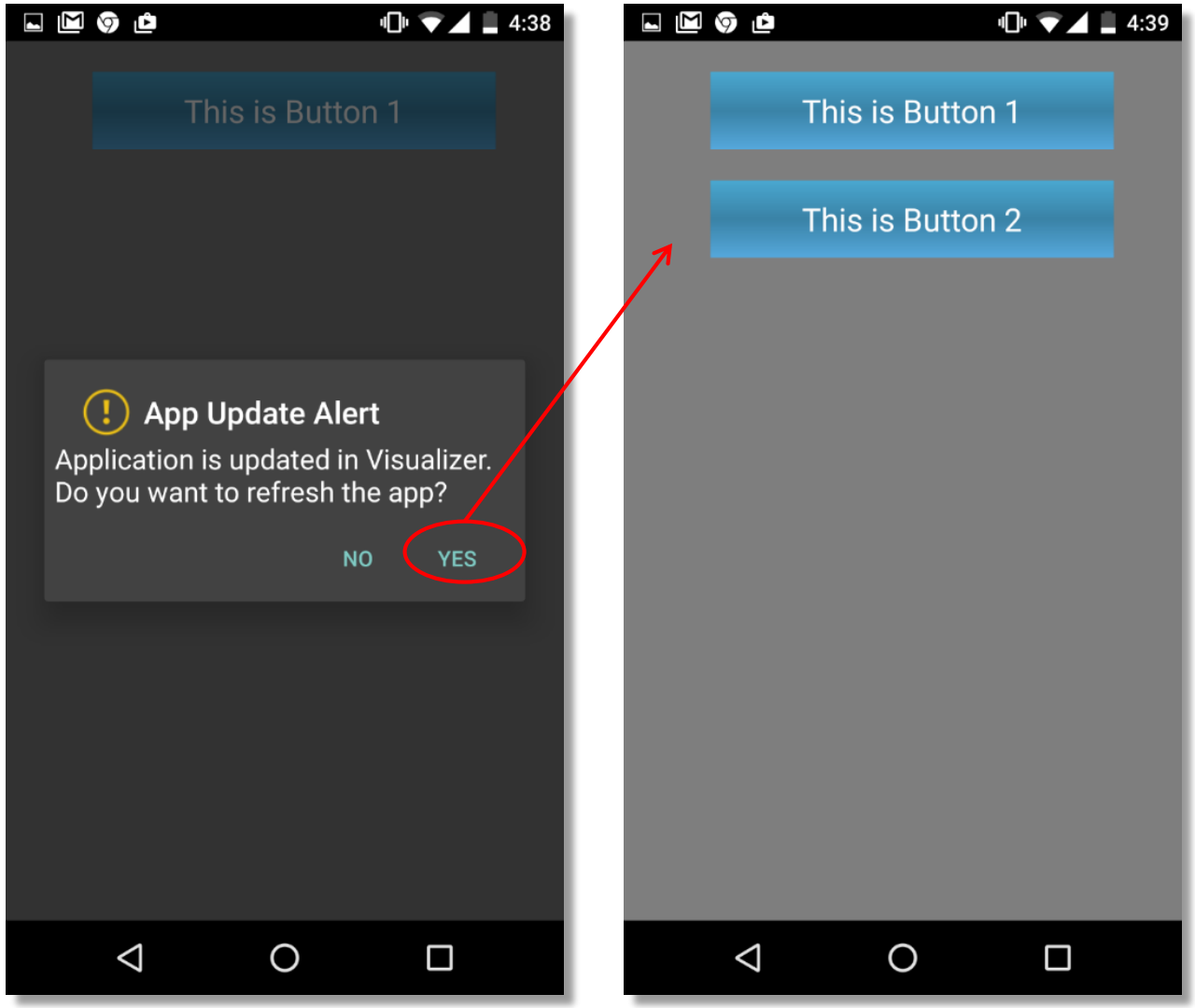
Testing Real-Time Preview (cont'd)

- Now, modify your design (add one more button) in the form and run the project locally.



Testing Real-Time Preview (cont'd)

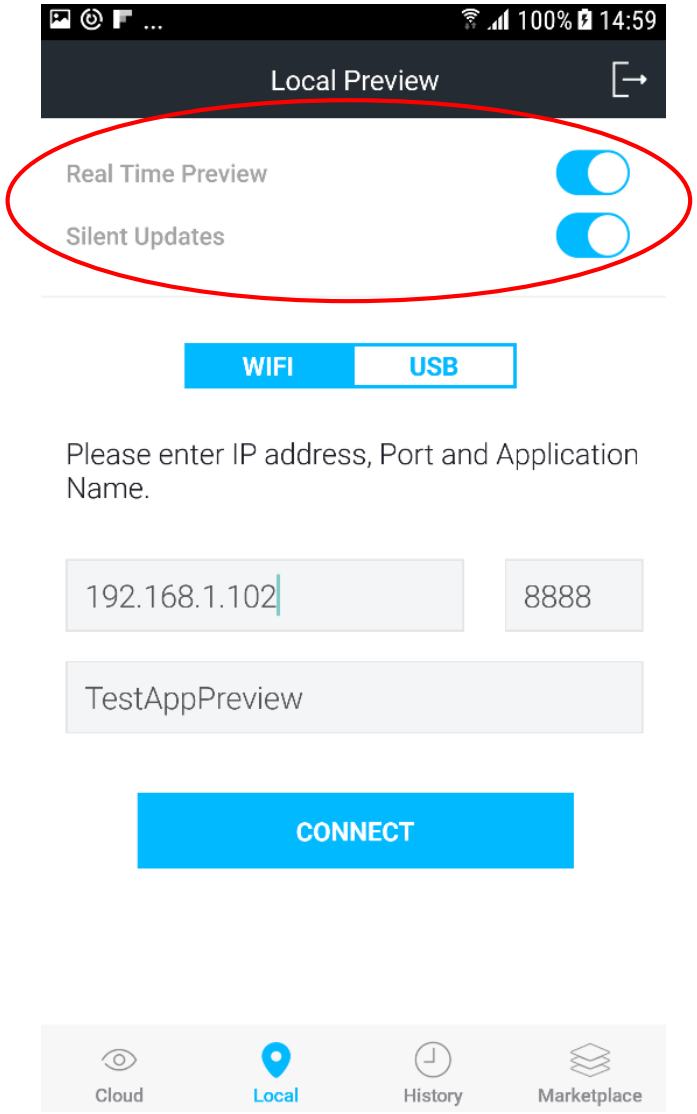
- When trying to test these changes in the Functional preview app, there is an alert that will be displayed about the application upgrade.



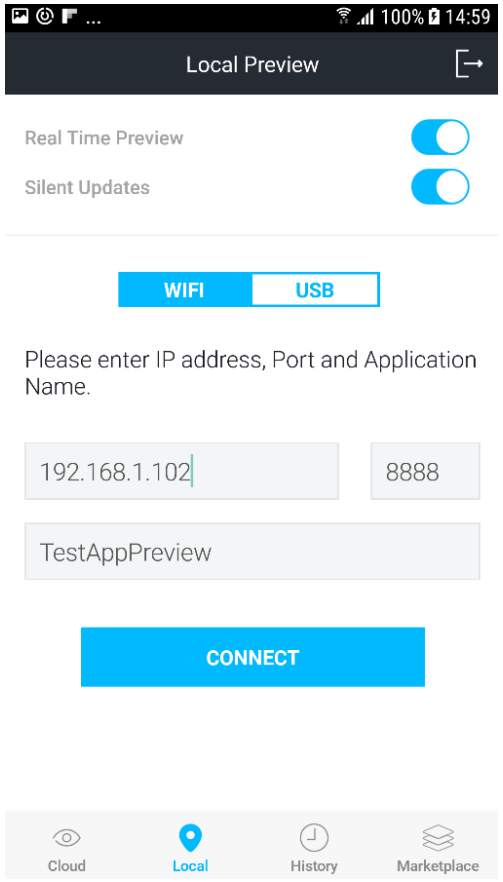
Real Time Preview - Silent Updates

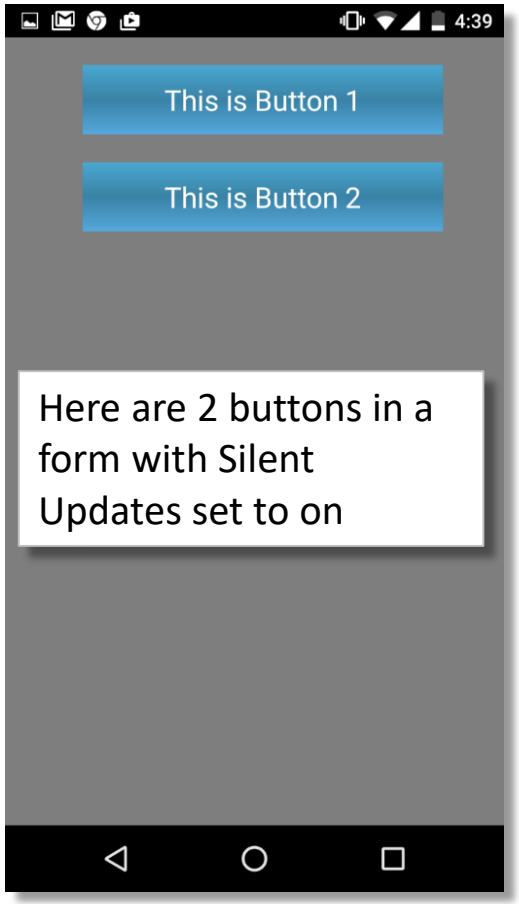
- When the **Real Time Preview** option is enabled, We have the option to enable **Silent Updates**

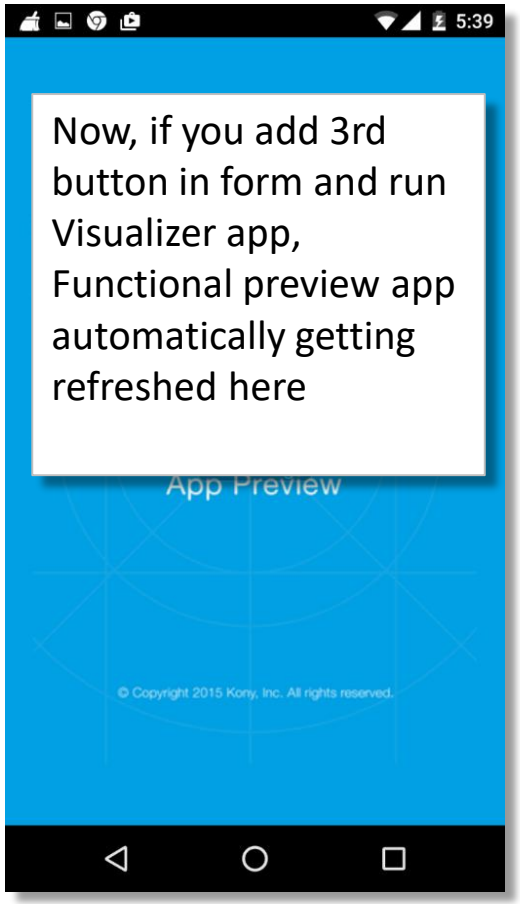
- **Silent Updates** refreshes the app preview automatically without giving any alert (like earlier) whenever we do incremental build anytime for the project in Kony Visualizer

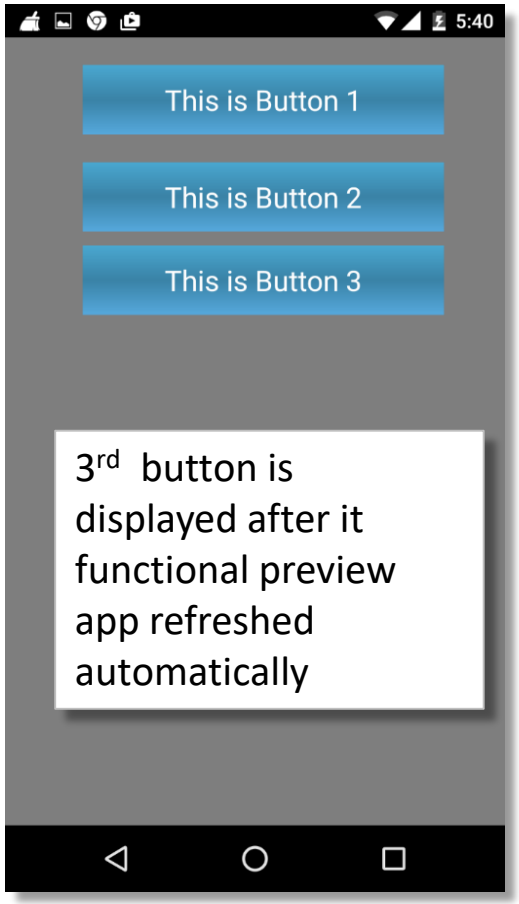


Testing Silent Updates



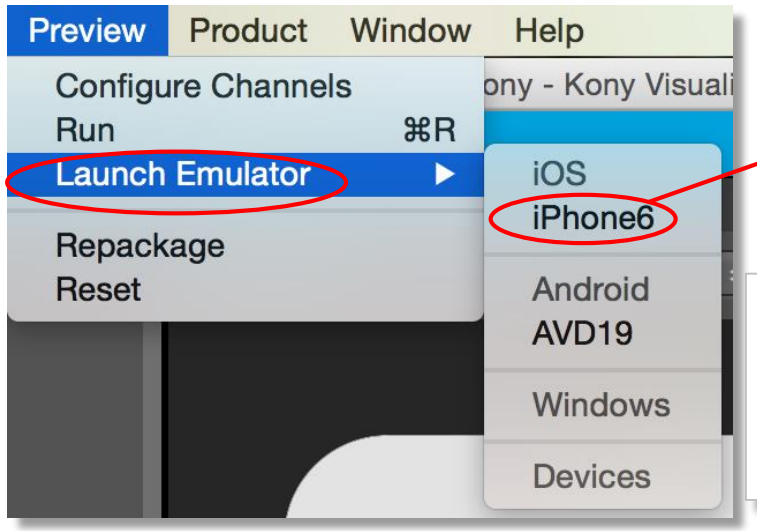




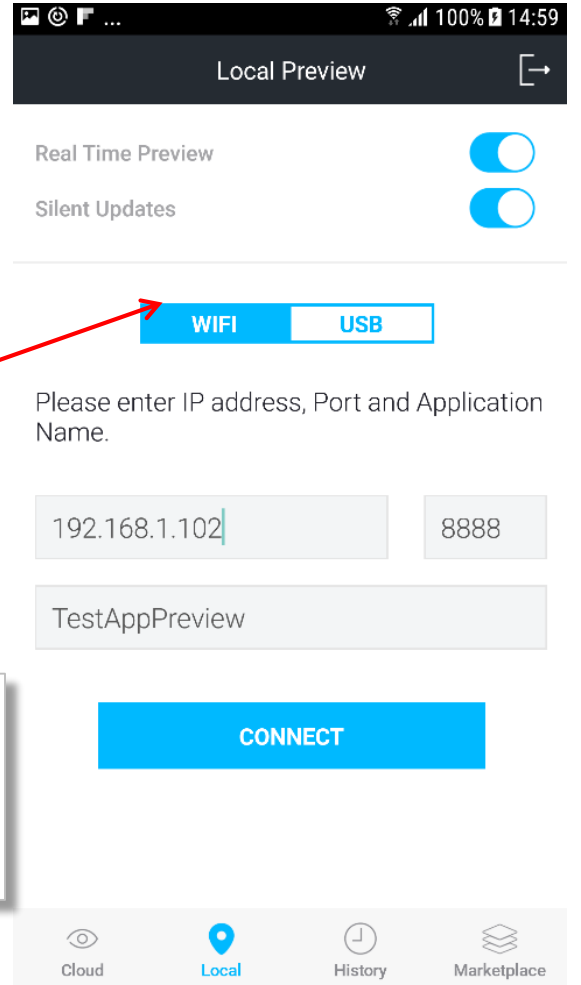


Functional Preview - Emulators

- Running on an actual device is the best way to test.... BUT... who has all the phone types laying around for testing?
- You CAN run the Functional Preview application on the emulators!
- You'll have to launch Visualizer to use this feature
  - Running on the emulators is supported for iOS and Android only



use Visualizer port # to run the Visualizer app – just like we did in the downloaded app



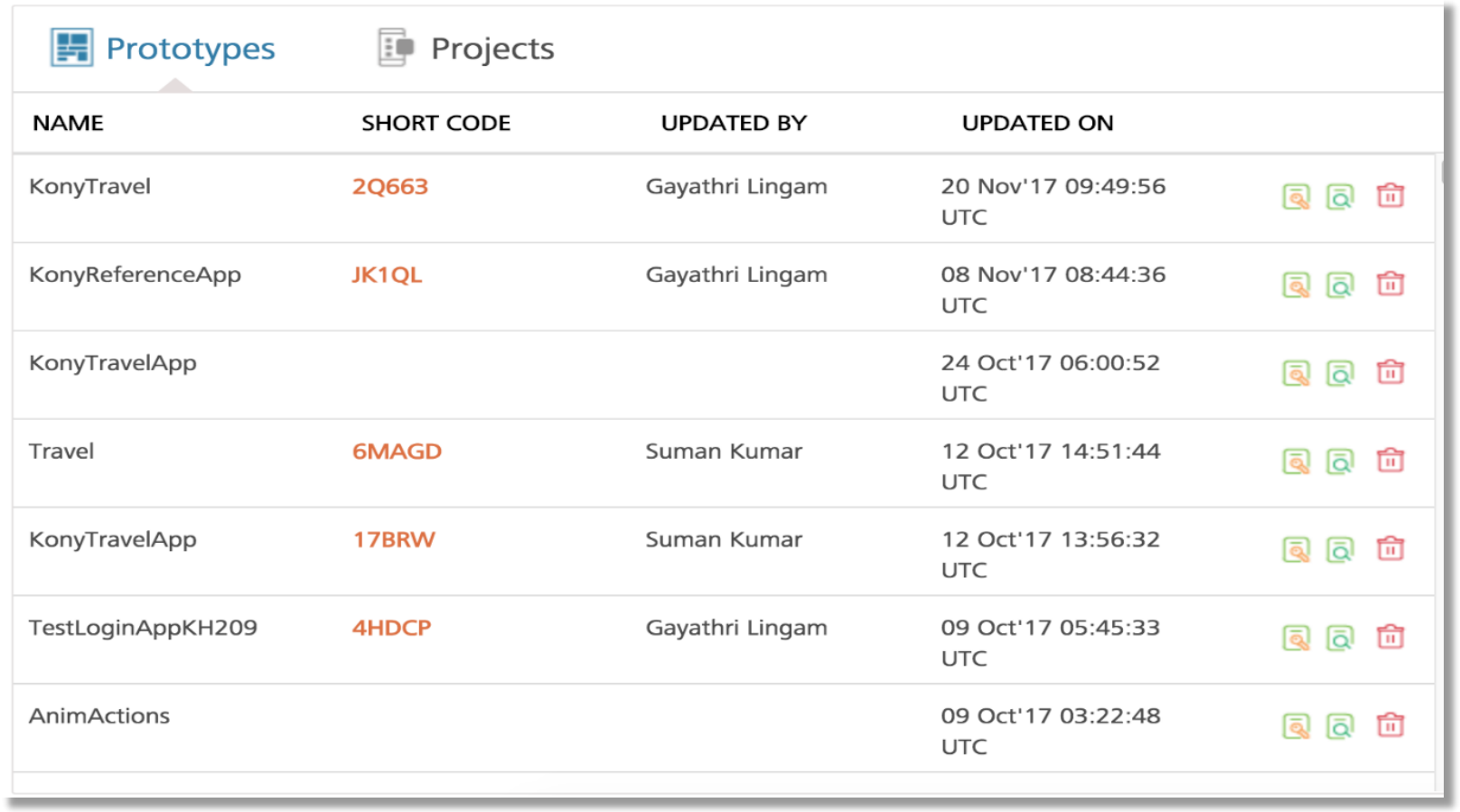
Visualizer Cloud

- The preview code that you generate can be shared with anyone using any cloud instance
  - They'll just need the functional preview app and any valid Visualizer cloud account
  - These preview codes are not private

- What is the Visualizer Cloud?
  - It's a central repository for applications
  - You can share applications with all the developers/designers on your team
- When you log into the cloud – go to Visualization services in your Visualization cloud instance: There will be links to download the installers

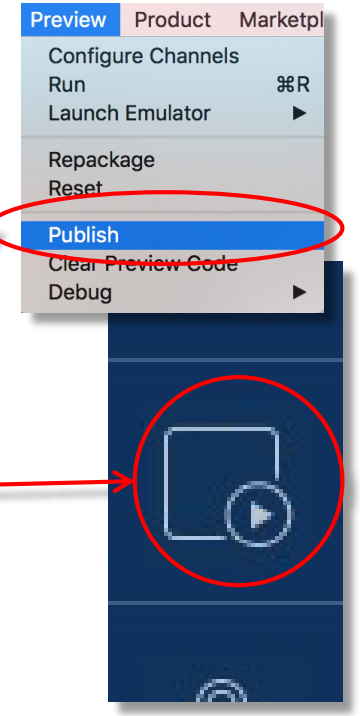
Visualizer Cloud

- You'll see all your applications and preview codes:

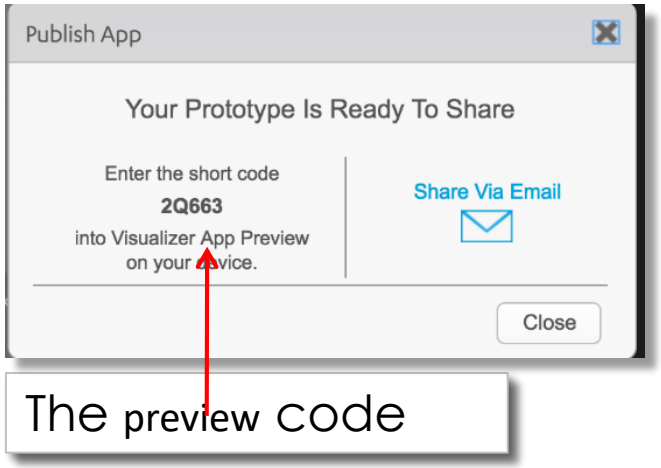


Publishing Project

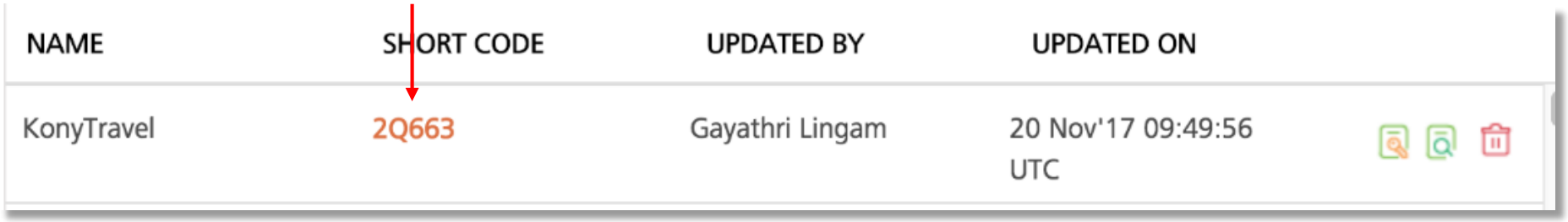
- In the demo I showed you publishing the application to the cloud



- This generates the preview code

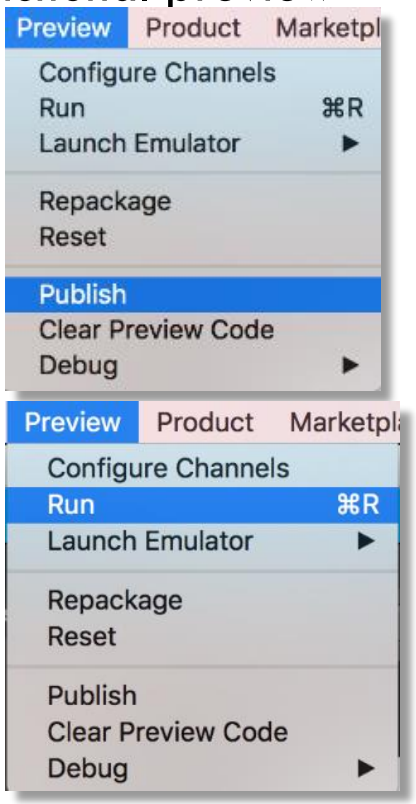


- The project is listed as a "prototype" – shown here in the cloud console



Testing an Application - Exercise

- Let's just make sure that we can publish our apps to the cloud and run them on our functional preview application
- Take any Visualizer project and Publish it to the cloud:
  - Jot down the generated code
- On your device (note only works if you have an iPhone or Android phone):
  - Download and install the Kony Visualizer application
  - Run and log in using your cloud credentials
  - Enter the code and view your design
  - Try entering notes on your form in Visualizer and make sure you can see them after re-publishing
- Now make a change in Visualizer and Run your app:
  - Test in your Functional Preview app in Local mode (port is 9989/8888) using the emulator if necessary – going forward, we'll use Local mode for all our testing



Widgets

Part I

kony Stay Ahead

Using Widgets

- What is a Widget?
  - It's a predefined UI element that represents common design elements found in mobile apps
  - Widgets also have an API for manipulating them in code
  - They can be used to speed up development IF the widget meets your UI AND functionality needs
    - If not, you can create your own UI elements using the basic core widgets like Flex Container, buttons, labels and images
- Let's first talk about these basic core widgets and then we'll talk about the rest of the widgets
  - We'll start by revisiting the Flex Container...

Widgets

Most Frequently Used Widgets

Flex Container

- We already saw that the Flex Container can:
  - Have a skin applied to it - we saw this in a few examples already
    - Use shadow to make container stand out of the screen
    - Use borders and rounding
  - Be animated and all the children are animated along with the container
  - Configure them horizontal/vertical/freeform to assist in our layout needs
  - Use BVR mode to see all child widgets even if they are outside the Flex Container boundaries
- We haven't talked about the FlexScrollContainer yet
  - This lets you have a scrollable area in your UI
  - We'll talk about this at the end of this module

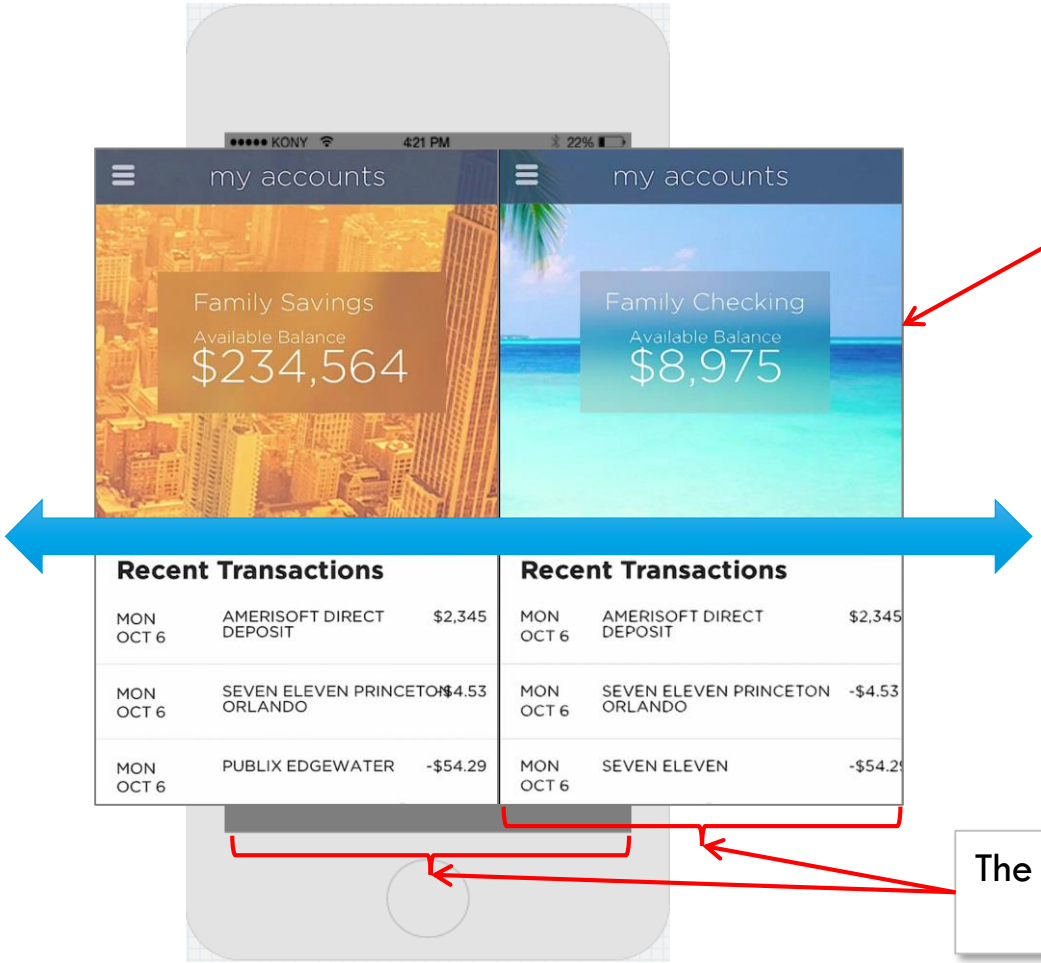
FlexScrollContainer

- The FlexScrollContainer will scroll inside it's boundaries to show UI elements that may initially not be viewable
  - Can add as many widgets as you want (use BVR mode to see them all if outside layout boundaries)
  - Configurable to have scrollbars (horizontal, vertical or both)
  - Useful if you don't want the whole page to scroll but you still need to show a lot of info

- One interesting property is Paging:
  - Rather than smoothly scrolling, the UI will jump in increments of the FlexScrollContainer's height/width (depending which way you've enabled scrolling)
  - This is a very common and nice UI feature to show pages of data
    - Let's look at an example...

FlexScrollContainer

- Here is a screen from a banking app:



With the FlexScrollContainer holding BOTH screens worth of data, you can swipe side to side to move between views

Setting the Page property on will snap to either page since each is sized as the screen width

The FlexScrollContainer is 2x the screen width – perfect for paging

Label Widget

- The Label widget is the main way to put text in the UI
- Label widgets have a new width/height unit: Preferred

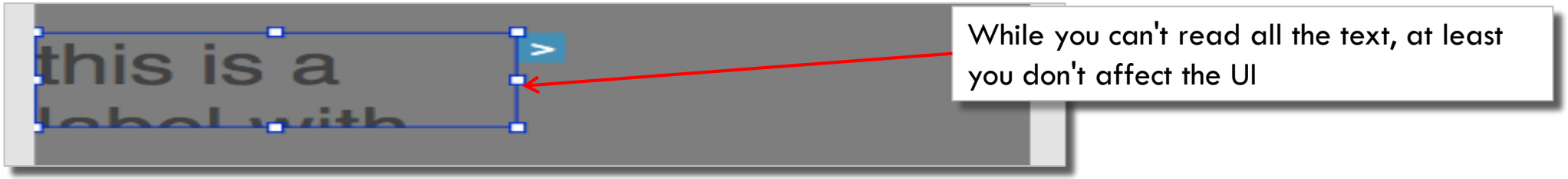
- A Label has a preferred width that is big enough to show the text
  - By default it will wrap when it gets to be as long as the screen
  - Let's look at an example:

  - Now, as we type more text.
  - "Preferred" = drawing the widget as big as it needs to be to show all the text

Label Widget - Preferred Size

- Preferred is fine if you are using static text and you have tested that the widget size renders properly on all devices
- Changing the font affects the preferred size:

- What happens if you are using dynamic text that may be longer than you expect?
- Typically you don't want your labels to overflow into neighbor UI elements so you will limit the space
- the label takes up
  - Turn off Preferred to specify it's size like other widget makes it a fixed size:



- Best practice to test all your dynamic content labels with your real data sizes to validate the UI works in all conditions

Preferred Layout Demo

Images

- Before we get into the Image Widget, let's talk about images in general
- Images are copied into your project using the computer's file explorer
  - Move all your image files into your project's resource folder
  - There are subfolders in the resources folder that relates to the structure you see on the Assets tab in Visualizer

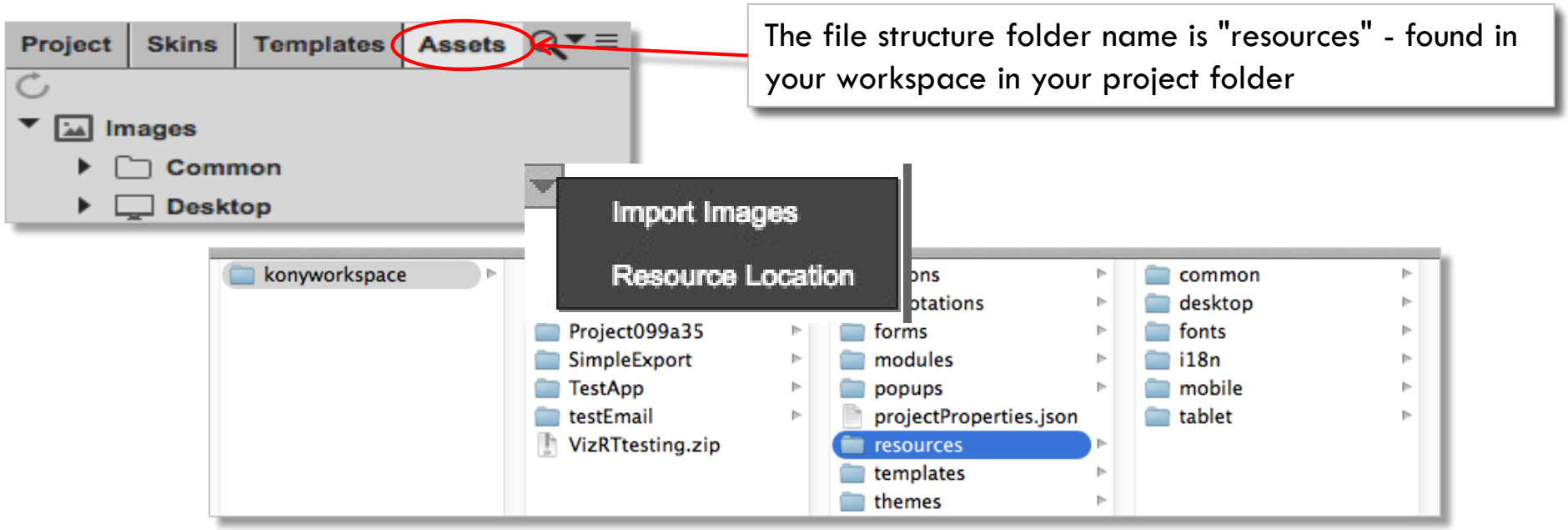
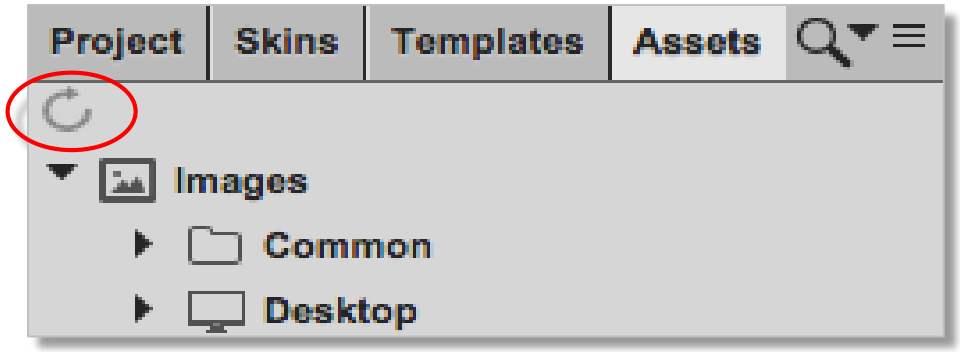


Image Naming Convention

- The only supported image types are PNG, JPEG and GIF
- Images have specific requirements around their naming convention:
  - The file name must contain only lower case characters
  - The file name must start only with a letter
  - The file name may contain numbers
  - Don't use any JavaScript keywords like "return" or "end"
- Once you copy them into the right place, you can refresh Visualizer to know they are there
- If you want EVERY device (mobile and tablet) to use an image, put it in the common folder



Images and Devices

- Since every device and every device type has different resolutions and screen sizes, one image size rarely fits all
  - Images that are stretchable/skewable can be common but that only applies to a very few rare cases (texture image, for example may be okay in certain use cases)
- You will typically need a specific version of the image for each target device or target resolution
  - Expanding an shrinking images (keeping aspect ratio) is very much possible but eventually you may opt for exact size images to be used
- The folder structure supports this:
  - Resource folder has separate sub-folders: mobile (phones) and tablet
  - Each of those folders has a common, native and web sub-folders
  - Web and native folders have sub folders for device types: iPhone & Android phone
  - Each of these sub folders may be further subdivided by screen resolution

Images and Devices (cont'd)

- The documentation specifies how to create the sub-folder structure down to the resolution level
  - iOS supports the @2x and @3x file naming conventions instead of resolution sub-folders
    - For all iOS images:
      - use <filename>.png for low resolution images (non-retina)
      - use <filename>@2x.png for high resolution images (retina)
      - use <filename>@3x.png for the new pixel-tripling images (new iPhones)
- Use the common folders to avoid creating more variances
- The default app menu images that are included in each project give us an example

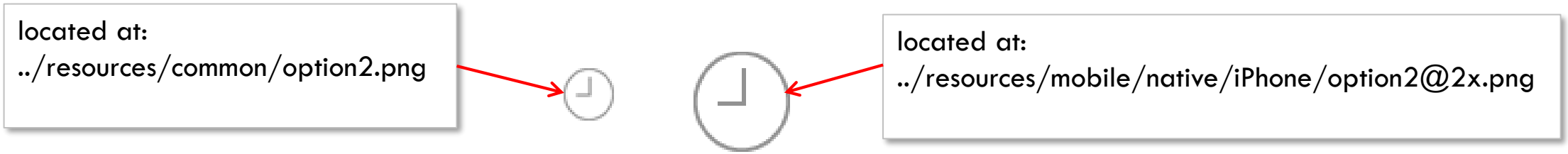
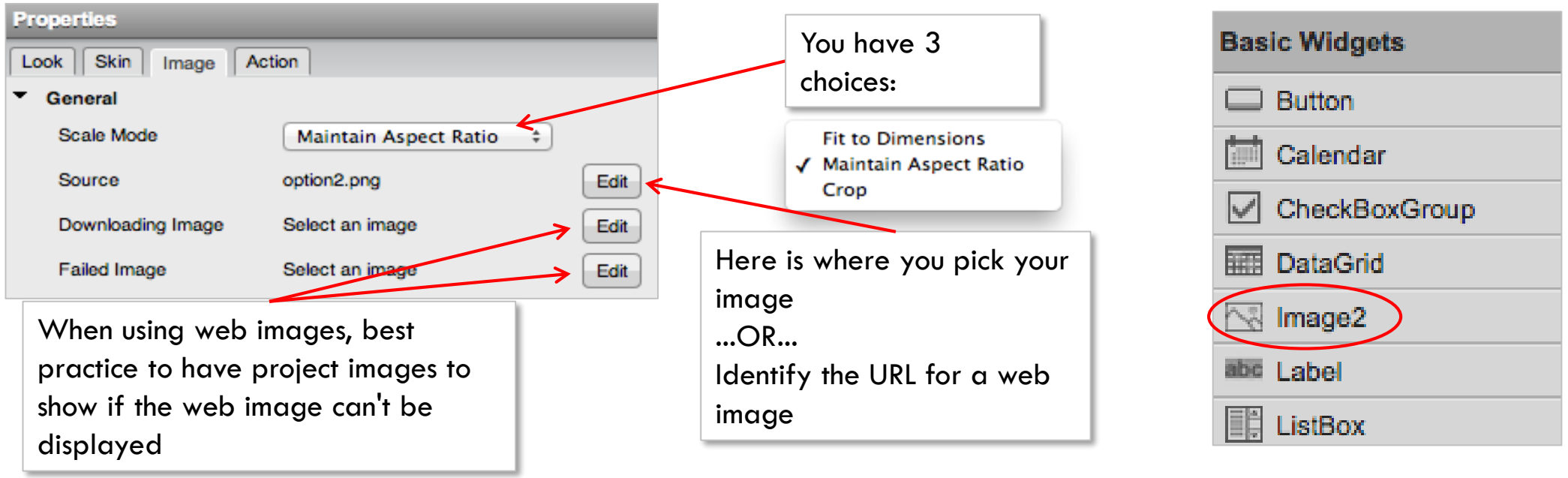


Image Widget

- The Image Widget lets us show images in our app
- Layout is the same as other widgets and skins don't really apply
- Here are the properties available for the image widget:



- Let's look at the Scale Mode and Source properties closer...

Image Widget - Scale Mode

- Scale Mode can be set to one of the following values:
  - **Fit to Dimensions** – the image will be displayed using the size of the image widget
    - the image will stretch, if necessary, to fit the image widget shape/size
  - **Maintain Aspect Ratio** – the image will be displayed, using the size of the image widget, but keep it's aspect ratio
    - the image will stretch, if necessary, to fit the the first edge of the image widget
  - **Crop** – the image will be displayed as it's actual size
    - the image will be cropped, if necessary, to only show the top-left part of the image that is the size of the image widget

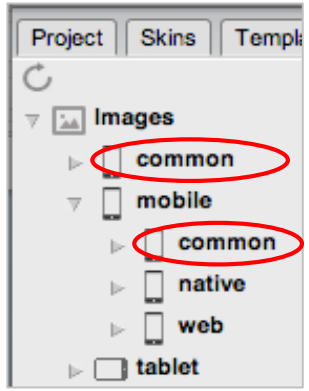
Examples:

Image Widget - Source

- Source - Identify which image you want to display (double-click image for same dialog):

when you have a LOT of images, typing image name filters list

The images in this list are all the images in all relevant common folders
For example: For mobile phone screens, you get the top level common + the mobile/common folders

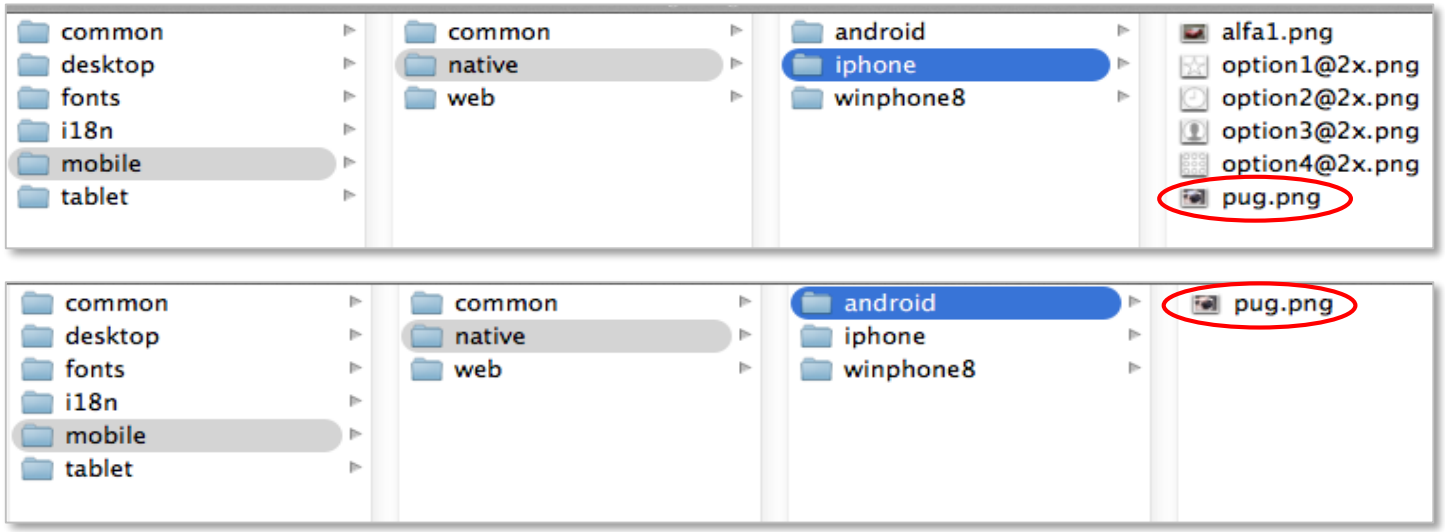


OR you can enter a URL to a web resource – use HTTP:// for all internet URLs

- Device specific images in sub-folders are used for those devices - let's take a look...

Using Device Specific Images

- I created 2 new versions of pug.png as follows:

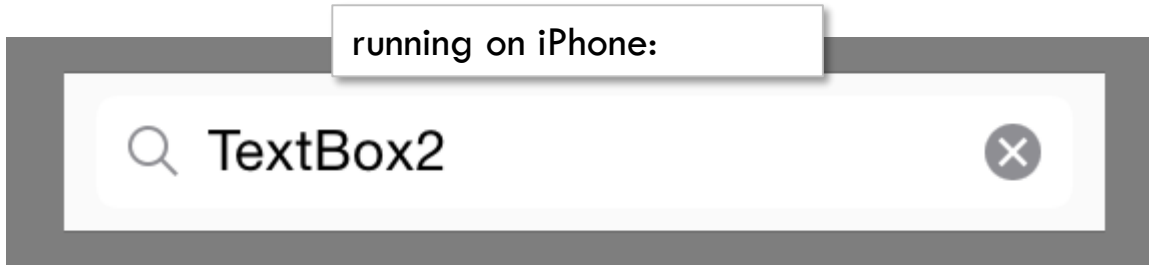


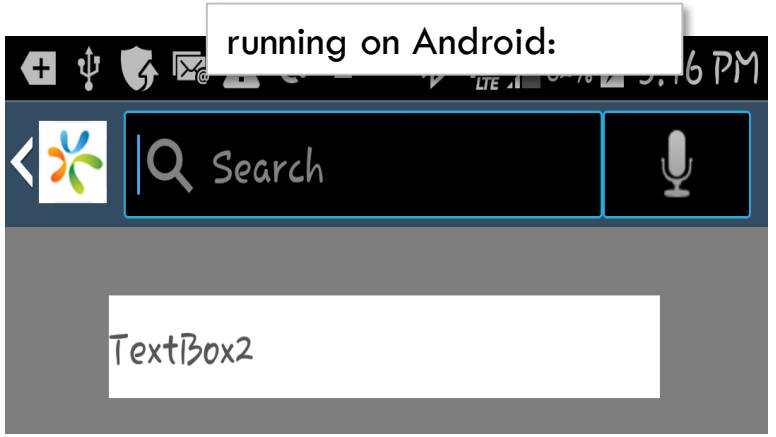
- There is still the original pug.png in the top level common folder
  - Note that the file name is identical in all 3 places
- Now, when you look at iPhone, it'll use the iPhone image, when you look at Android, it'll use the Android image and for all other devices, you get the original default
- Let's see what that looks like…

Image Demo

Getting Text Input

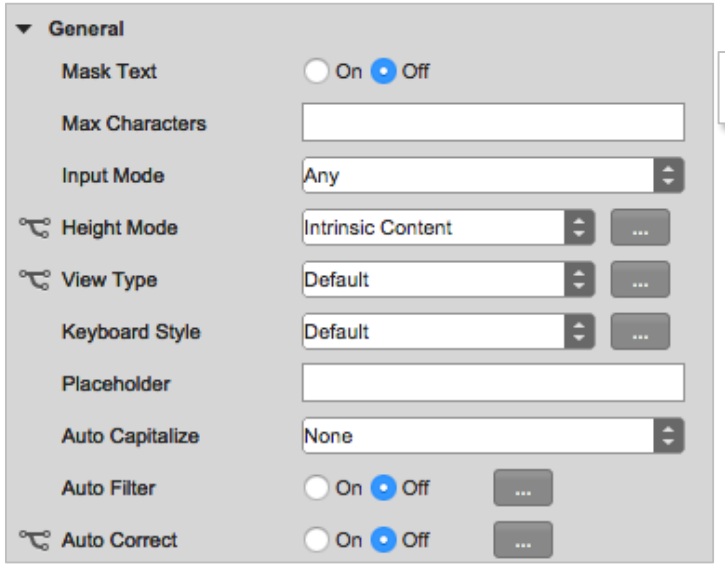
- Getting the user to type in text is a typical UI element that is necessary in any application
- There are 2 widgets that you can use: TextBox and TextArea
  - TextBox – used for a single line or short input
    - Examples: word searches, username/password, address fields
  - TextArea – used for multiple lines of input
    - Examples: user comments, problem description field
- They are both configured almost exactly the same (same options) with a few differences:
  - TextArea has an Auto Correct option but TextBox doesn't
  - TextBox can be configured as a search field:





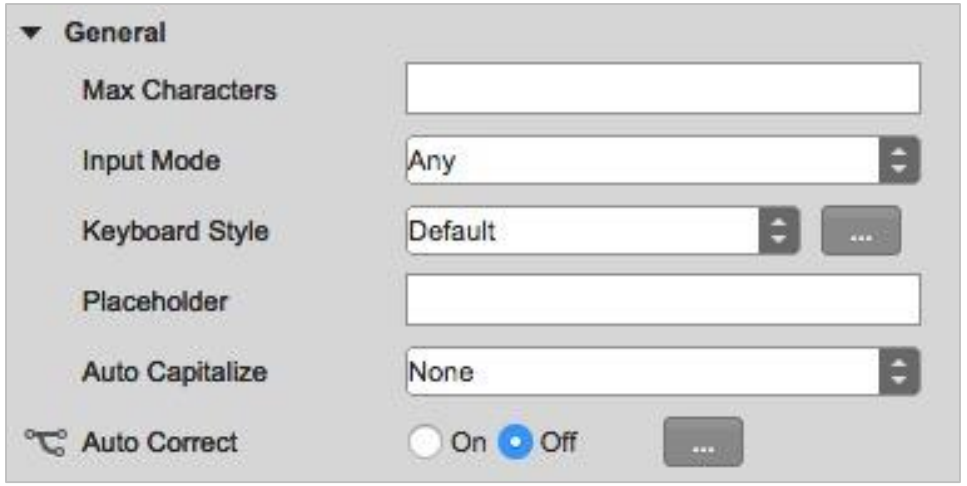
TextArea and TextBox Configurations

- Both widgets have configurations that control what happens inside the textbox:



TextBox:

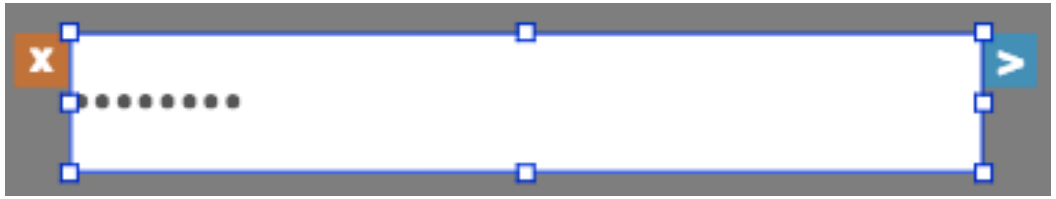
TextArea:



- Note: Most don't change the way the widget looks, only how it acts.
- Exceptions:
  - **Max Characters** - Good to keep the user from typing too much
  - **Placeholder** - The text that disappears as soon as the user taps into the field
  - **Keyboard Style** - Changes the available keys on the keyboard

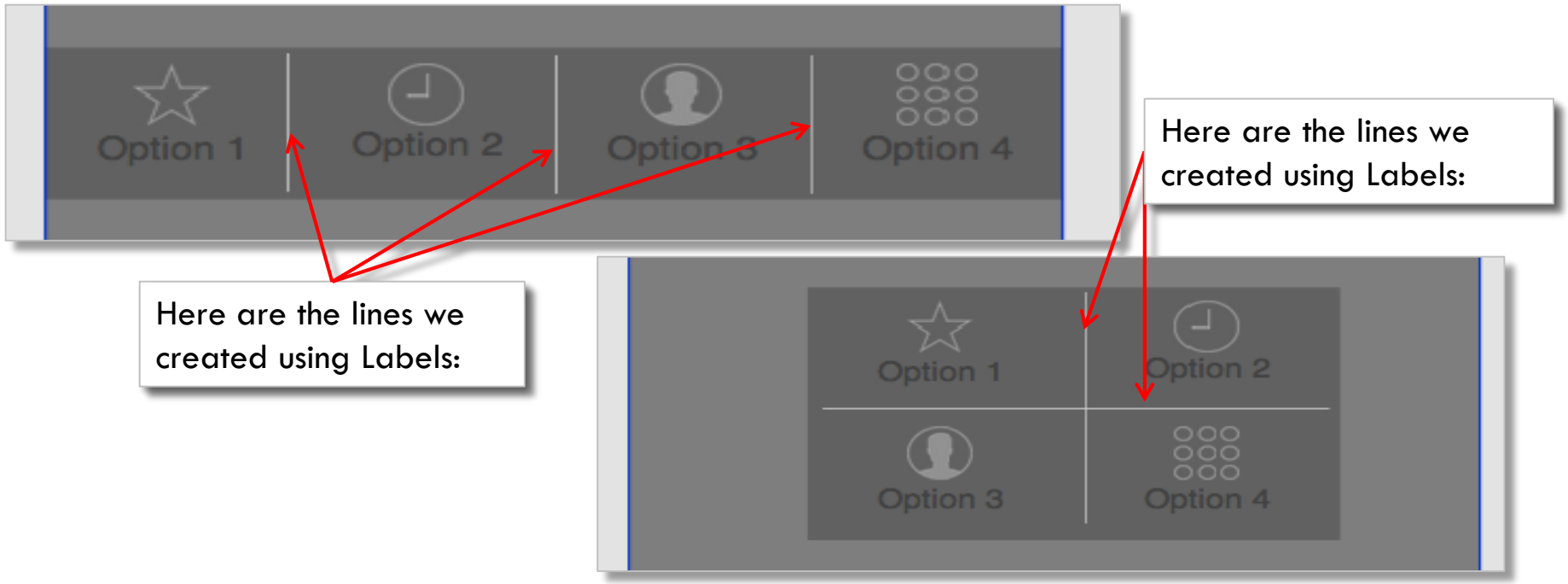
TextArea and TextBox Keyboards

- Both widgets have configurations that control the keyboard
- The keyboard isn't a UI concern, it's more of a usability concern
- However, making text entry awkward reflects poorly on the design
  - Use the right Keyboard Style property for the type of text the user will be entering
- Some keyboard options to note:
  - **Auto Correct /Auto Capitalize** - Don't turn these on if you expect a lot of cryptic notes or abbreviations to be used, for example
  - **Pasterboard Type** (iPhone) - If you have very sensitive data, consider setting this to avoid users copy/pasting text to/from the application
  - **Mask Text** (TextBox) - Necessary for password and other secret fields. Hides characters as they are typed



Label - As UI Lines

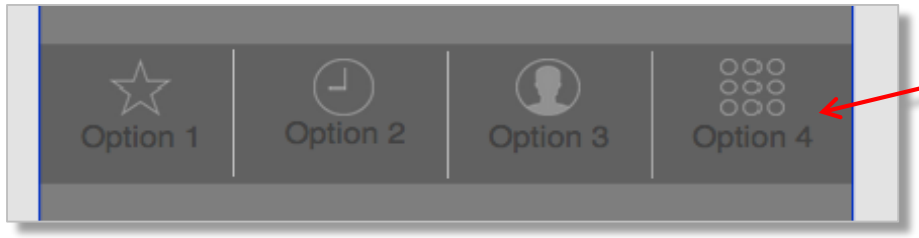
- An additional way to use the Label widget is to create vertical and horizontal lines as shown in these examples:



- Simply remove all the text, change the height/width away from "preferred" and set those values and skin to create your line

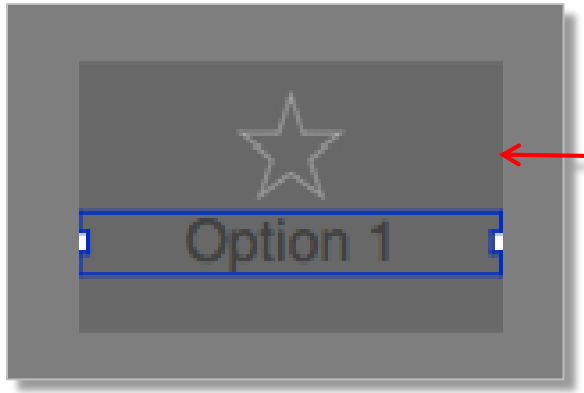
Label Widget - Centering Text in the UI

- It's very typical to center your labels around other widgets in the UI
- Our previous example is a good one:



each label should be centered in the box and with respect to the image

- VERY typical to set this up as follows:
  - Each label has width set to 100%:
  - Content alignment set to Centered

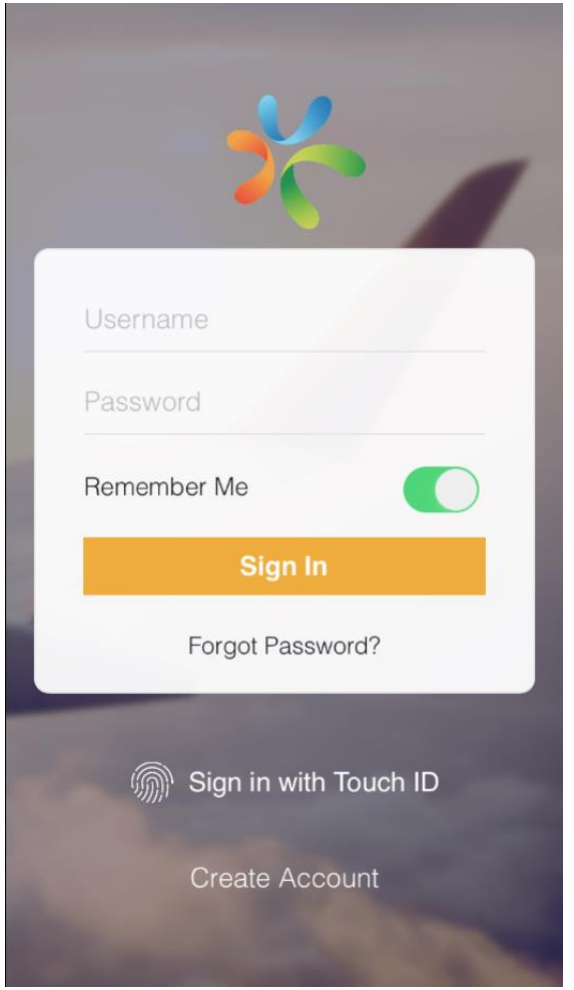


Each menu item is it's own Flex Container

- This is a great example of why we use Flex Containers – it helps us do alignment work for a part of our UI easily

Login Screen - Exercise

- This exercise will give you some practice laying out a screen using containers and the layout principles we talked about
- We will also use the skin properties to change the background
- The intention of this exercise is to just create a basic login screen design
- Next slide talks about tips/techniques



Login Screen - Exercise (cont'd)

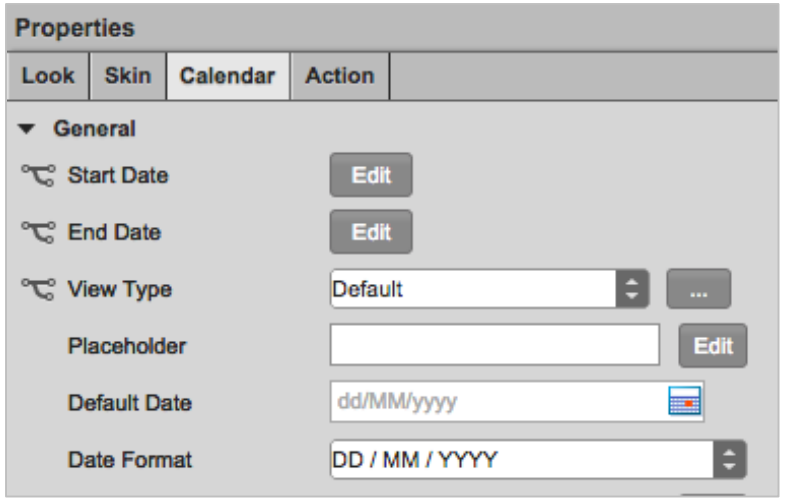
- Here are some hints to create the Screen
  - Add Image widget and make it look square and add image to it.
  - Add Flex Container as shown in the screen and add the following and make the layout type as flow vertical. Use skin property to change the background.
    - 2 text boxes
    - A FlexContainer (add a label and switch widget as shown)
    - Add Button
    - Add Label
  - Add FlexContainer and add the following
    - An Image
    - A Label
  - Add a Button

Widgets

Additional Widgets

Calendar Widget

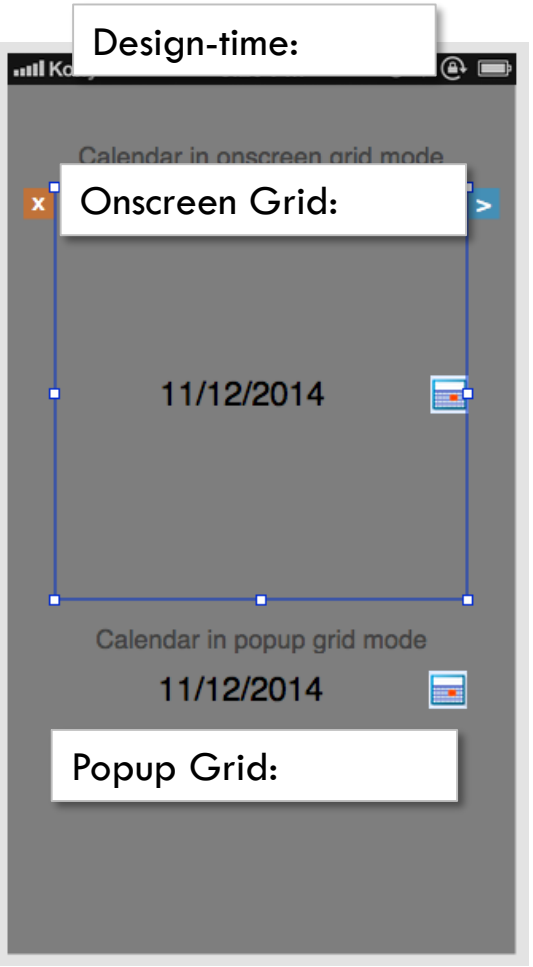
- Calendars are very common features in an application
- Here are the basic configurations:
- **View Type**
  - Onscreen Grid
  - Popup Grid
- **Start Date/End Date -** you can configure these that limit what date range is valid



- **Default Date -** what date do you want to show by default when it first displays?
  - While you can configure this value at design-time – most likely you'll set this dynamically in code to be today or a date that is relative to the current date
- **Date Format -** pick the date format you want
- **Placeholder** - enter text to prompt user
- Let's look at an example...

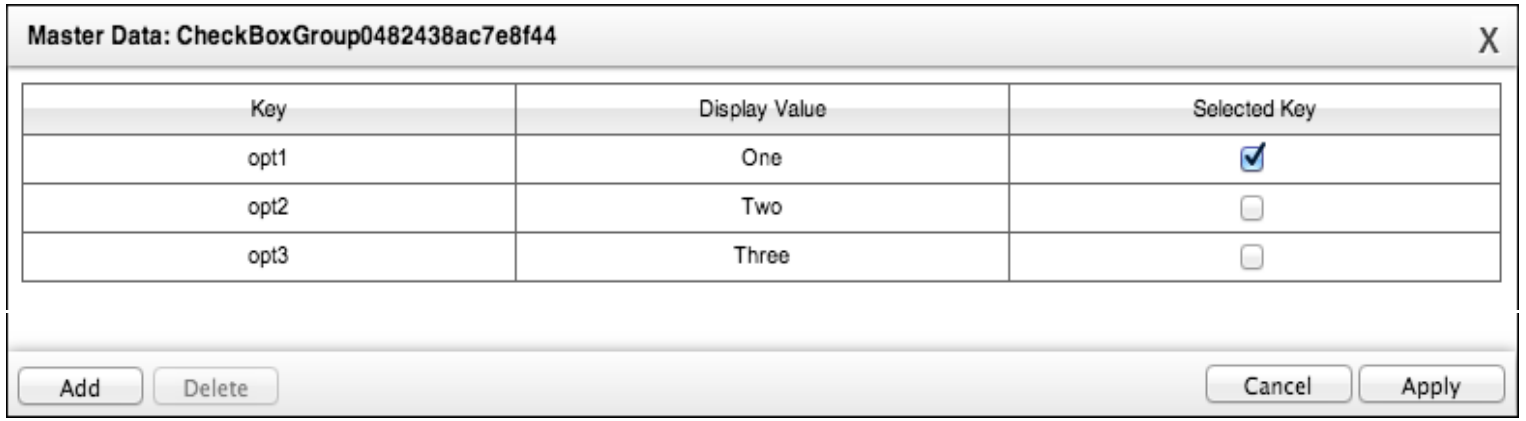
Calendar Widget (Cont'd)

- Here is an example at design-time and at run-time:



CheckBoxGroup Widget

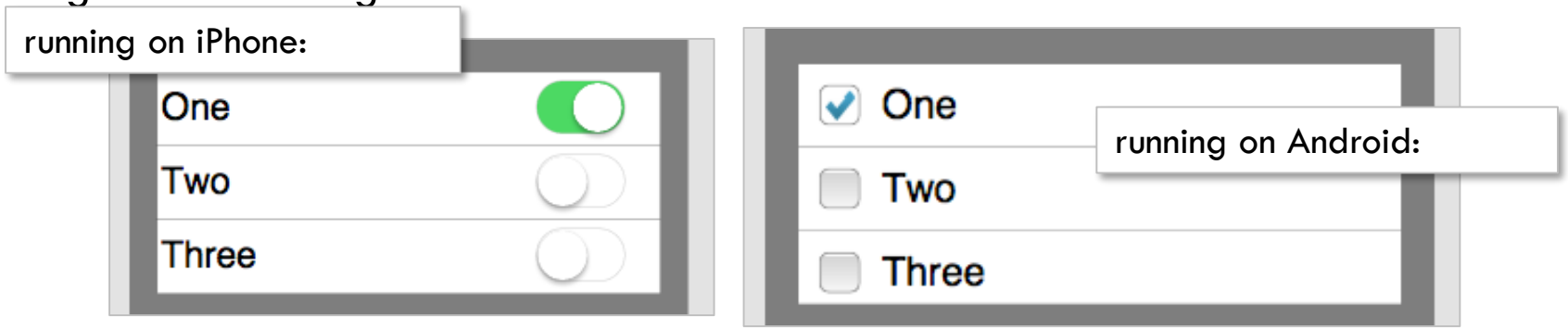
- The CheckBoxGroup widget puts a set of checkboxes on the UI
- The information (text and state of each checkbox) is determined by Master Data:



- Use the Add/Delete keys to add new check boxes
- The key is what is used (typically) in code by the developer to know what was selected
- Display Value is the text shown to the user for each check box
- Selected Key indicates if that check box should initially display as selected or not
- Let's see how this data makes our CheckBoxGroup widget look...

CheckBoxGroup Widget (cont'd)

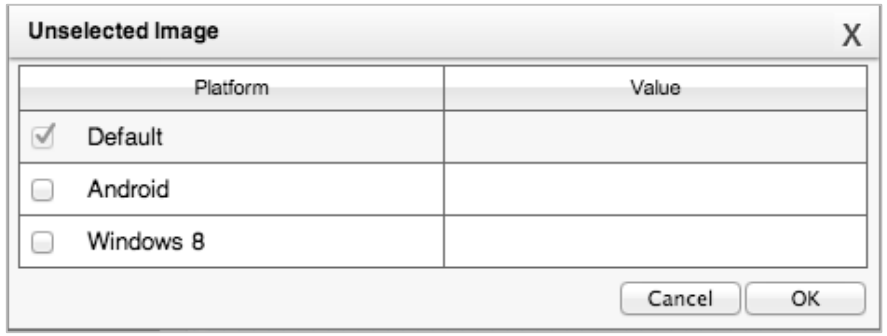
- Here it is using default configurations:

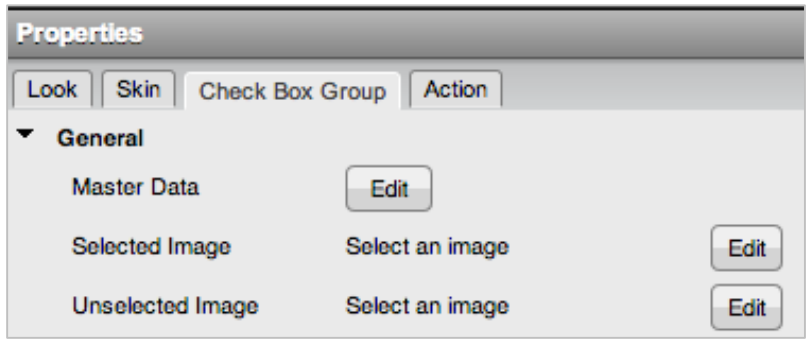


  - Note: The Canvas IS active, so you can:
    - change the selected state by toggling the check box
    - change the item text by double-clicking it and typing a new value
  - The more items you have, the bigger you must make the widget to show all the values
- There IS a skin for this widget both for the normal and focus states
  - Focus is when the user is in the process of checking an option
- Let's look at the configuration properties next...

CheckBoxGroup Widget (cont'd)

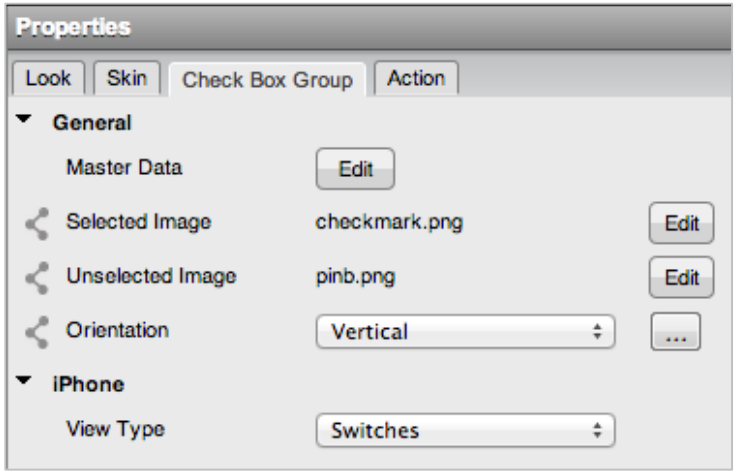
- Here are the first few configurations:
  - **Master Data** – where we edit value
  - **Selected Image/Unselected Image** – you can pick what images you want instead of the defaults:
    - Note: This is a common image dialog whenever you need to specify an image
      - Default applies to all devices
      - Device specific ones let you pick a DIFFERENT image
        - Not the same as creating a device specific version of an image





CheckBoxGroup Widget (cont'd)

- Let's continue with the other properties:
  - **Orientation** – Vertical or Horizontal
    - Horizontal puts them side by side
    - Note – This property is forkable so each device can have it's own orientation setting

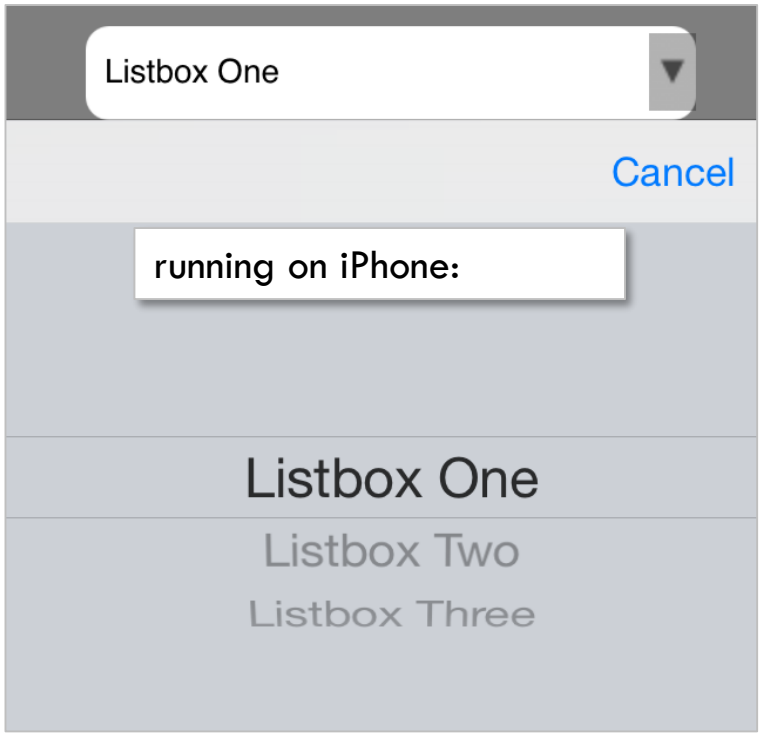


- Important Consideration: This and other widgets have forkable properties that MAY change the layout
  - For example, if we fork the Orientation to one device in Horizontal and the others in Vertical – everything below that widget will now not match
  - Consider very carefully the use of forking properties that change the UI
  - **(iPhone) View Type** - Let's you pick what type of widget to use for the selection
    - Choices: Switches or Table or On-screen wheel

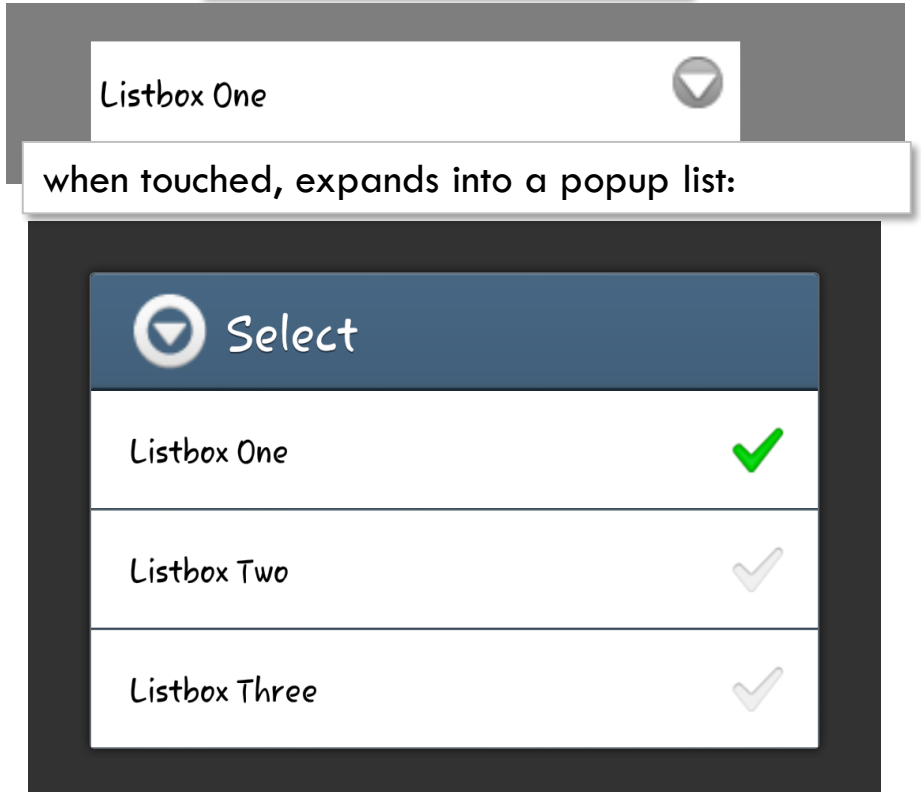
ListBox Widget

- The ListBox widget is a way to let the user pick a single value from a list
  - Slightly different paradigm than the CheckBoxGroup where each check box works independent from the other choices
- Here is an example:

when touched, the list expands below – rotate the wheel to make a selection

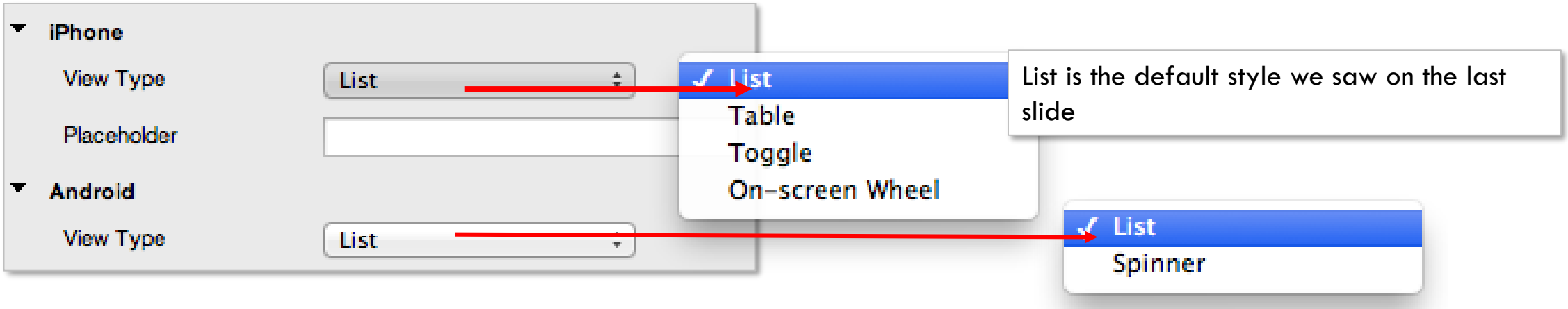


running on Android:



ListBox Widget (cont'd)

- The ListBox has a lot of different combination of views between iPhone and Android

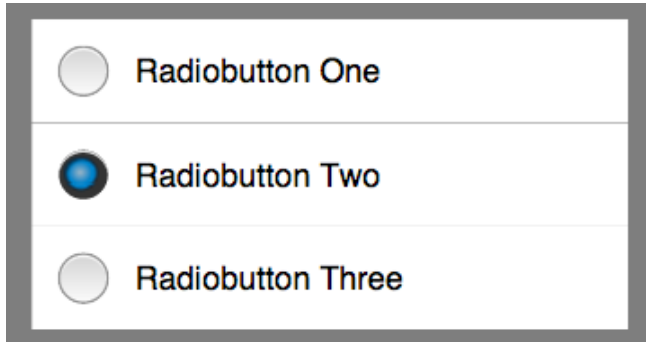


  - Another example of different view types lead to different layouts – careful cross-platform!

RadioButtonGroup Widget

- The RadioButtonGroup widget displays radio buttons
  - Note: This is more of a desktop widget that really doesn't get much mobile use
    - For the iPhone, you get the same display options as the ListBox
    - The issue is that it's very likely your layout WILL differ between devices with different display options you'll want to use
    - This widget is very easy to reproduce using the core widgets and that layout WILL be the same for each device
  - This widget is best for Android and/or Windows Phone designs only and therefore are of limited use
  - Configurations are the same as with the CheckBoxGroup and/or ListBox

running on Android:



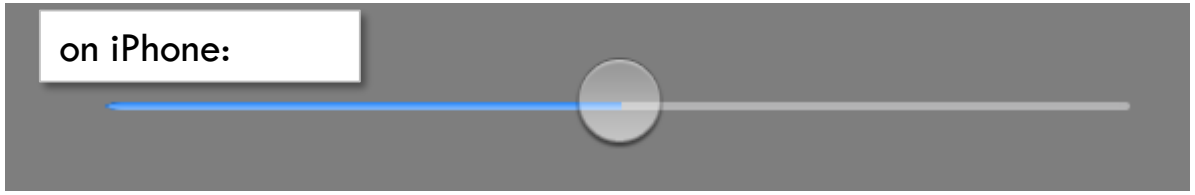
Easy to recreate with an image (toggle between unselected and selected images) and a label

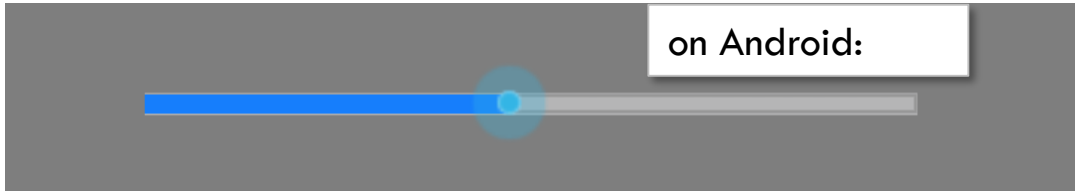
RichText Widget

- The RichText widget lets you display text using HTML formatting for text styles (bold, italics, etc.), links and even images
  - Note: It's NOT full HTML compliant, only basic tags are understood
- Sometimes, it's easier to use HTML formatting than trying to string together a lot of Label widgets to get the same effect
- Sometimes, you can share standard content from a content management system
- Here's a simple example showing the text property AND rendering at design-time
  - Skinning and configurations are the same as a Label widget

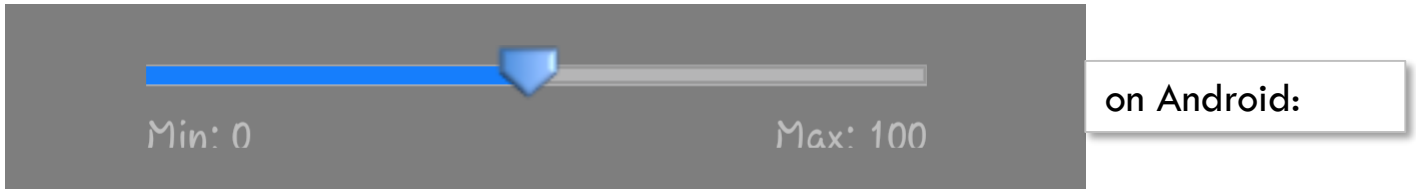
Slider Widget

- The Slider widget is another example of a widget that is easy to create using core building block widgets for a read-only version (to show a value) – animating the drag action is harder
  - The Slider lets you configure the various UI aspects - if those meet your needs then using the slider is a good idea
- Here is an example or a zero-configuration look you get by default:
- Slider has a lot of configurations that allow you to create a more robust look



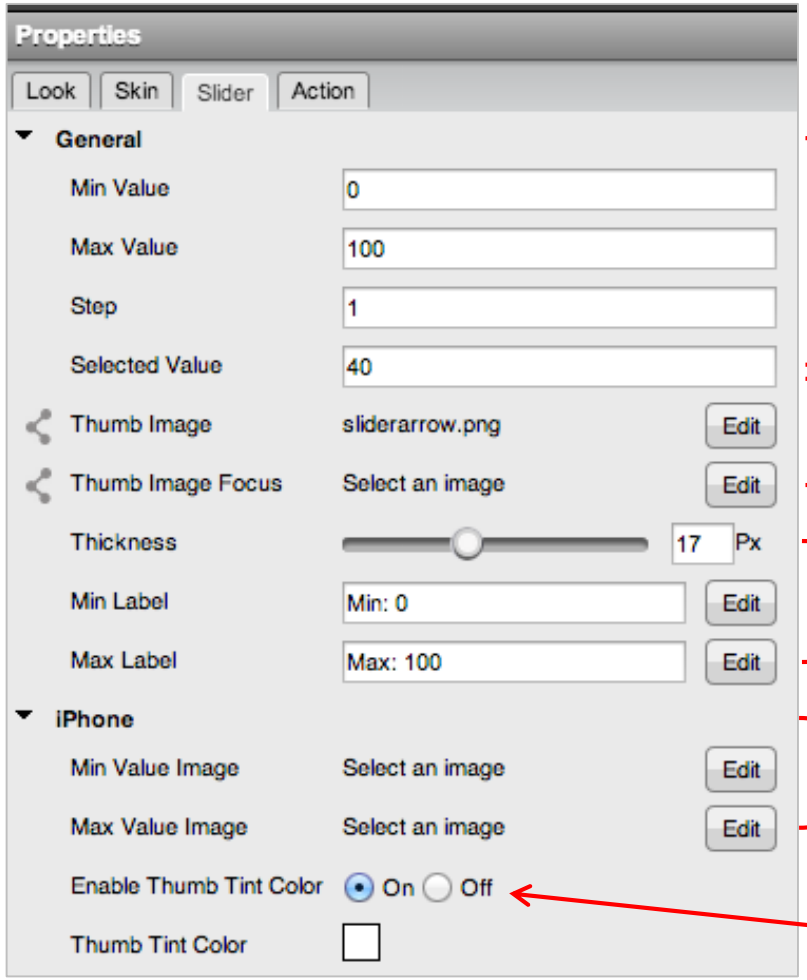


- The simple look is good if you want to put all the labels and values around it yourself...OR...
  - here is an example of the same slider with configurations applied:



Slider Widget Properties

- Let's look at the configurations available for the Slider:



specify the scale and incremental values on the slider – it's typical for the designer to set these but have the developer pick final values

specify custom image(s) for the slider thumb – can optionally show a different image when the user's finger is on the slider

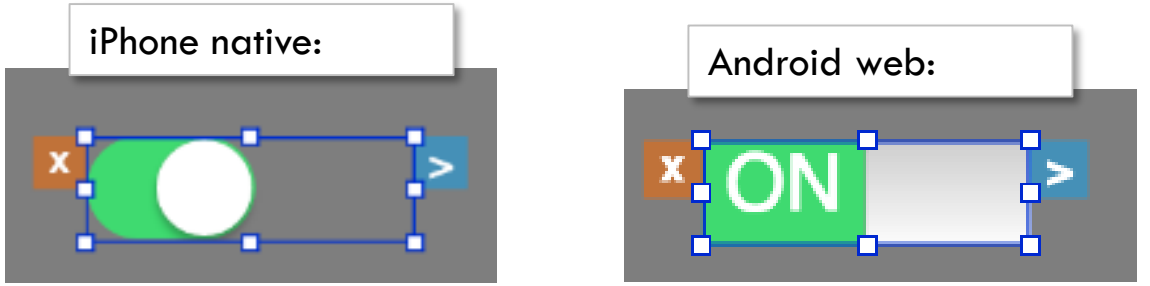
The labels only show on Android, SPA and Windows Phones

Since no labels are available on iPhone, you can optionally provide an image on either side of the slider

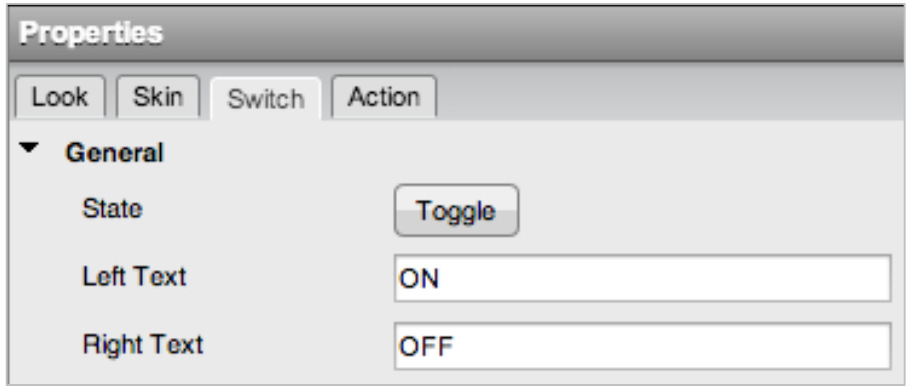
The default On means the iPhone will NOT display custom thumb image – turn off to see it

Switch Widget

- This widget is the typical switch you see on iPhones



- Examples of what it looks like:
- Configurations for this widget are minimal:
  - **State** – sets default switch state
  - **Left/Right Text** – don't apply to iPhone



- Remember the CheckBoxGroup widget lets you configure this the same for iPhone and has options for Android - a better choice since it's really a cross-platform

Most Frequently Used Widgets Summary

- Your UI will be 90% composed of the widgets we've covered so far
- We saw that the Flex Container, Button, Image and Label widgets (and the text input widgets – TextArea and TextBox) are the core widgets that you'll use for sure
- We saw how we can use images and how we can have device specific version of images
- The other basic widgets in the palette are available IF they meet your needs:
  - They come pre-packaged with various configuration options - quick to use
  - They ALSO come with an API that makes the developer's job easy
- This was just a walk-through of these widgets so you can understand what building blocks are at your disposal
  - Compare these widgets with the UI paradigms and UI bits you've used for other projects
  - It'll take a bit of practice and experimentation to know the exact limits of each widget in your UI and to find out what works best for you

Exercise - Basic Widgets

- Let's now test these widgets out
  - We'll create several screens so that we can experiment with these basic widgets
  - Screen #1:
    - Configure a RadioButtonGroup, CheckBoxGroup and ListBox to show the same options but configured to use a different UI
    - Flip canvases to see other devices
    - Experiment with the settings
  - Screen #2:
    - Configure a TextArea and a TextBox widget on the form
    - Run and test the behavior as you enter text into the TextArea and TextBox
    - Try different configurations
  - Screen #3:
    - Put a slider on the screen
    - Play with the configurations

Introduction to Actions and Animations

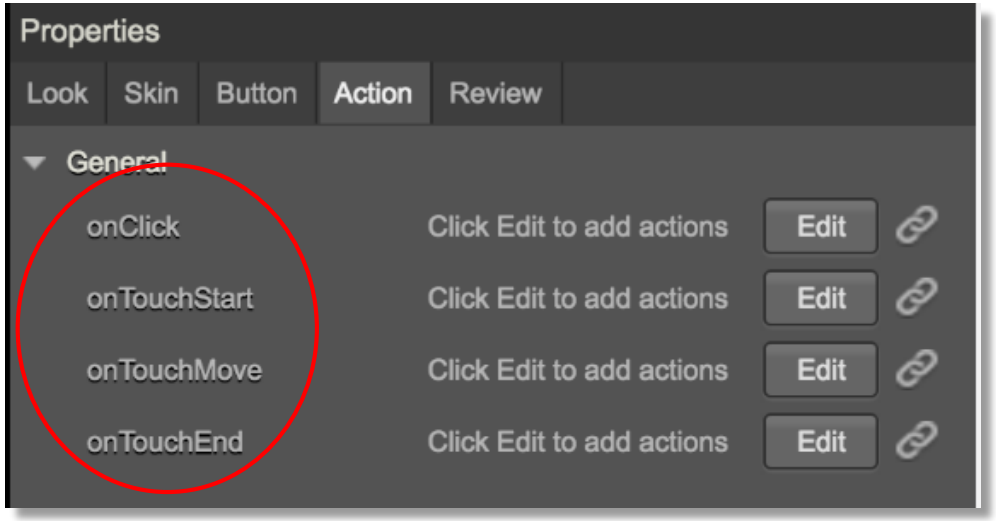
kony Stay Ahead

Actions and Animations

- When we talk about design, we frequently talk about an app's "look and feel"
  - "Look" is what we've talked about so far – making the UI look right at any given point
  - "Feel" is all about the user experience when interacting with the app
- Visualizer let's you do both
  - We can make our design come to life by associating actions with user activity
  - Examples include:
    - Clicking a button to navigate to a new screen
    - Having some animation happen when a screen first shows
    - Showing an alert to a user when checking the "remember me" option on a login form
- Let's take a look at a little demo app to give you some ideas of what I mean

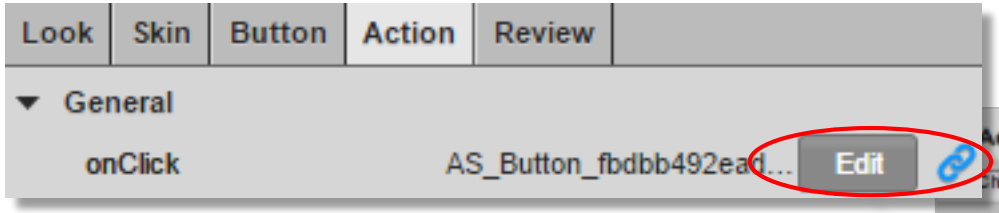
Actions

- Almost EVERY widget has some sort of event that we can use to DO something
- What is an event?
  - An event is where a widget is responding to something that is happening in the app
  - Examples include:
    - A button has an onClick event that triggers when the user taps a button widget
    - A radio button widget has an onSelection event that triggers when the user picks a value
- We can use the Action Editor to do things when any of these events fire
- Here are the events for a Button widget
  - You'll find them on the Action tab



Action Editor

- For any event, you can specify what happens by clicking the Edit button



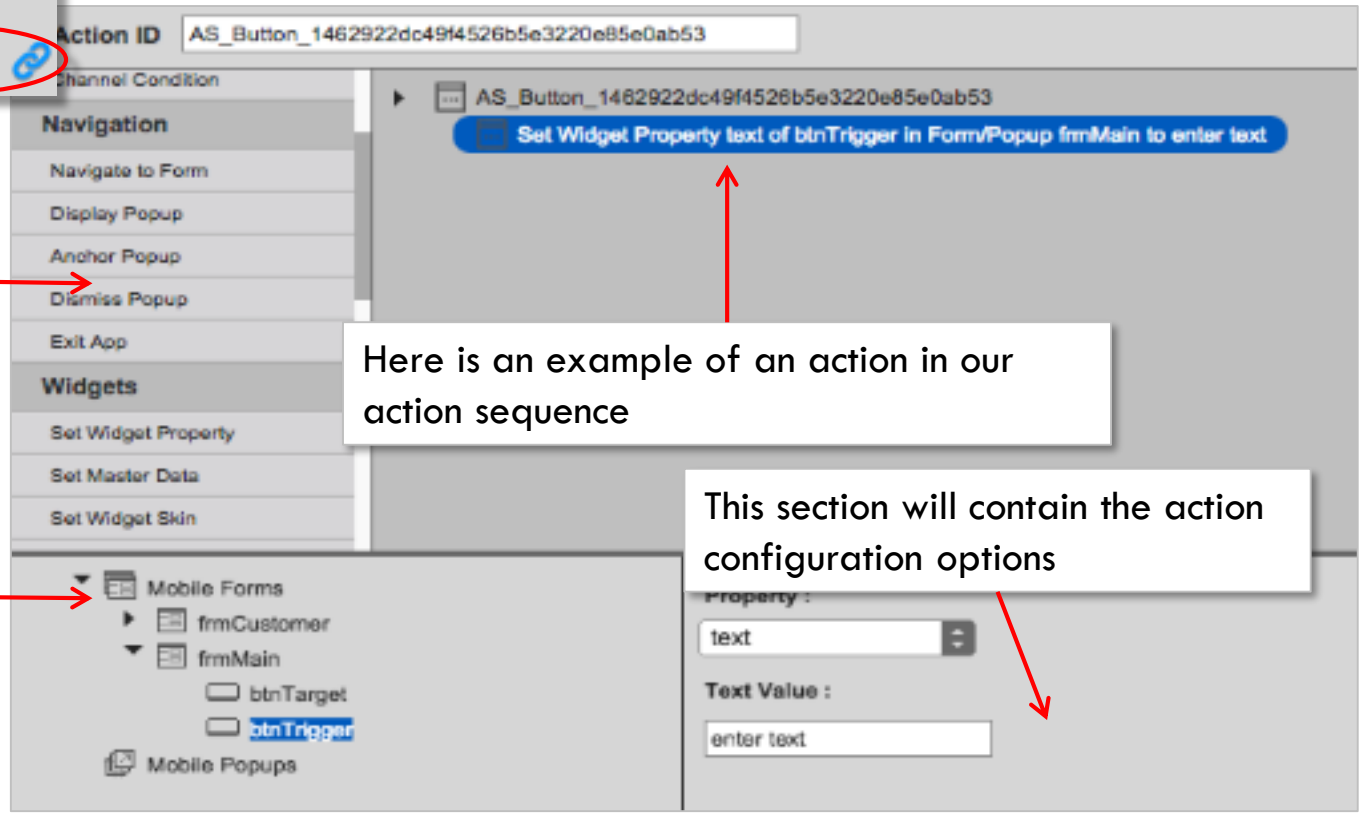
- This brings up the action editor

Here are all the Actions

Most actions will require that you select the widget or form before you can configure the action – this is the "target" of the action

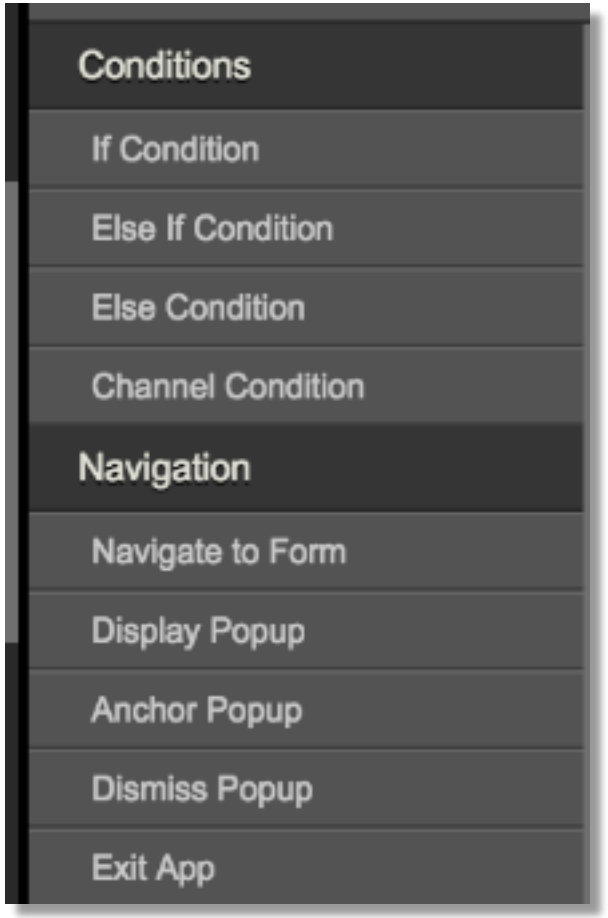
Here is an example of an action in our action sequence

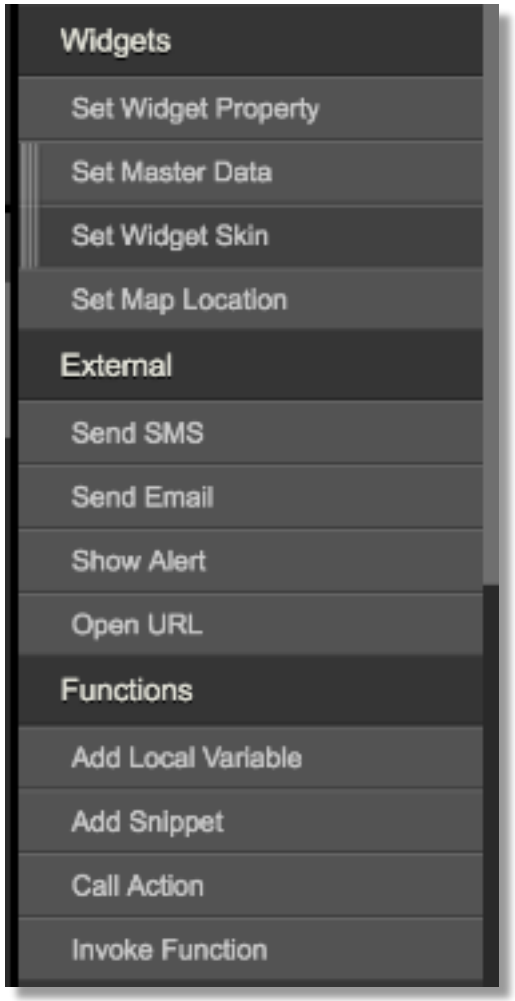
This section will contain the action configuration options

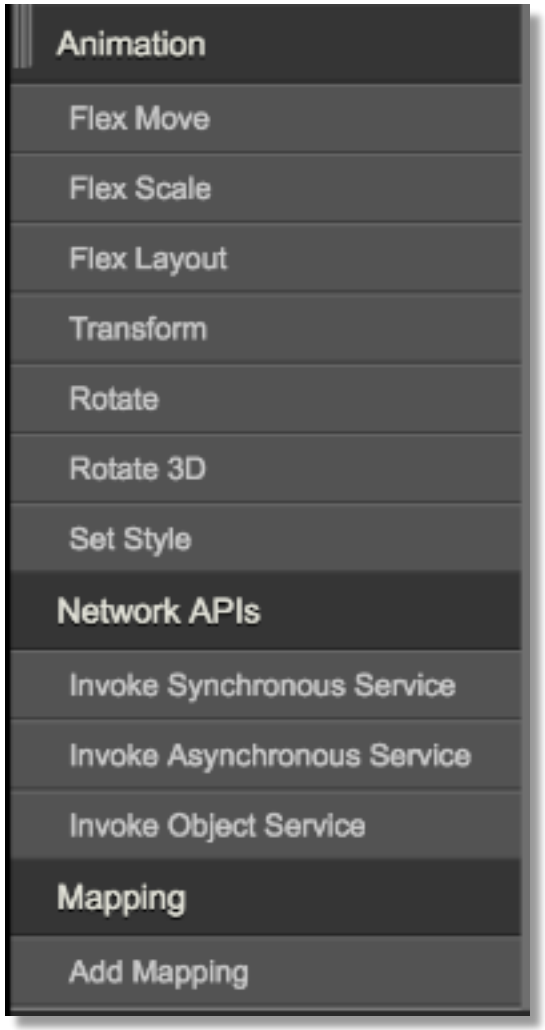


Actions

- Here are all the available actions

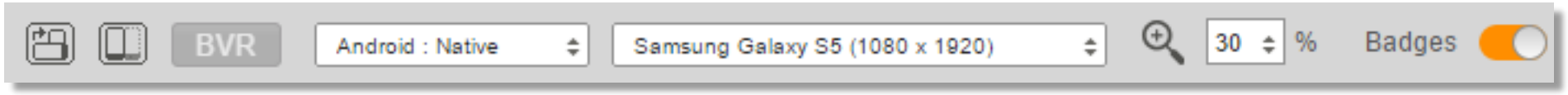






Container BVR Mode

- For any container (form included) you can look at all the child widgets even if they are off the physical boundaries of the container
- Use the BVR button on the canvas:

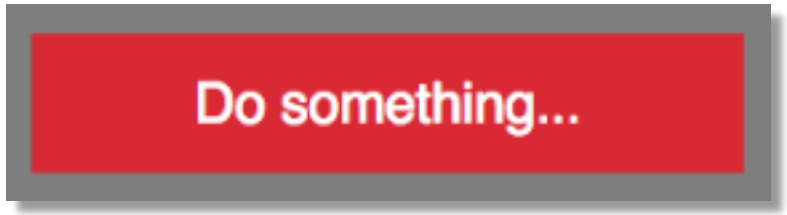


- As you saw, BVR mode is very useful when you need to see all the widgets
- Remember to highlight the container you want to see and then go into BVR mode
- Note that a flex container, by default, will not show up since it's 100% transparent
- The widgets in the flex container will be visible

Other Skins

- Did you notice how the buttons turned Red briefly when they were clicked?
- Most widgets that do something have more than one "state" that the widget can be in
- Let's look at our button widget example:
  - The "normal" state is when the button is just sitting there on the form
    - We saw how to configure the skin

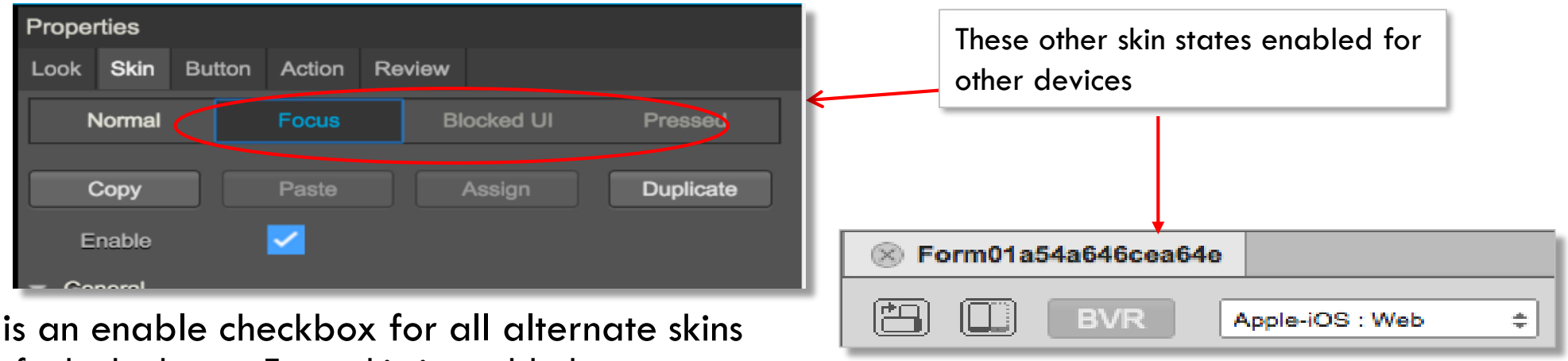
  - The other state is the "focus" state – this is where the user's finger is pressing the button
    - That skin is default showing a Red background



  - When the user lifts his/her finger, the button returns to the normal state

Other Skins (cont'd)

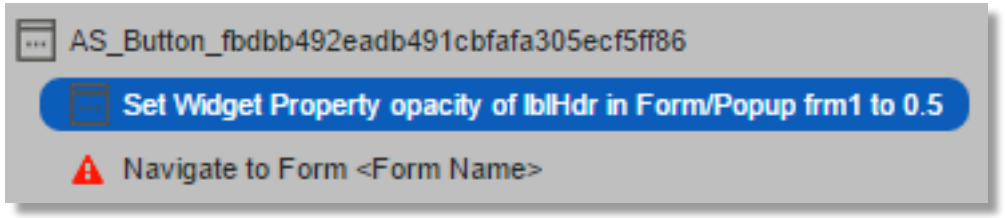
- When we start talking about all the widget types, we'll see what those other skin states are for each widget
  - Note that these skins are managed automatically for us
- The tabs across the top of the skin tab show what alternate skins are available for configuration
  - Note: Blocked UI and Pressed are only available on some devices (SPA/android native)



- There is an enable checkbox for all alternate skins
  - By default, the button Focus skin is enabled
  - For other widgets, you may need to check the enabled checkbox to activate that skin

Action Editor Rules

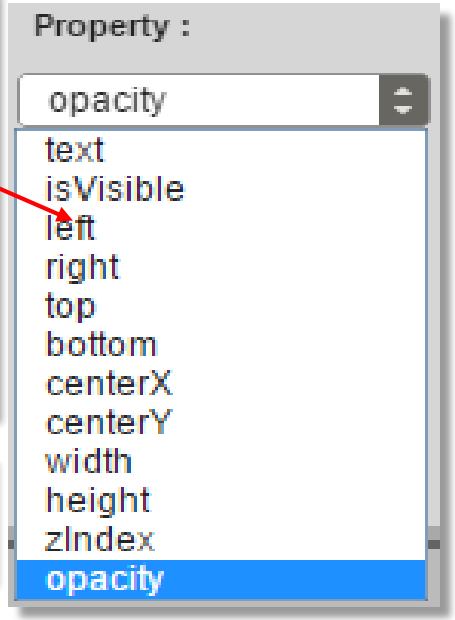
- In every action sequence, to add the first action, you must CLICK it on the left and Visualizer will automatically add to your action sequence
- To add more actions, there are 2 ways
  - Clicking it will add to after whatever action is currently highlighted
  - Dragging it will allow you to put it anywhere in the action sequence EXCEPT for the end
    - To add it to the end, select the last action in the sequence and then click the action and Visualizer will automatically add it as the last action
- You can drag actions to re-order them
  - Yes, you can even drag it to the end



- Actions with exclamation mark are not completely configured

Widget Actions - Set Widget Property

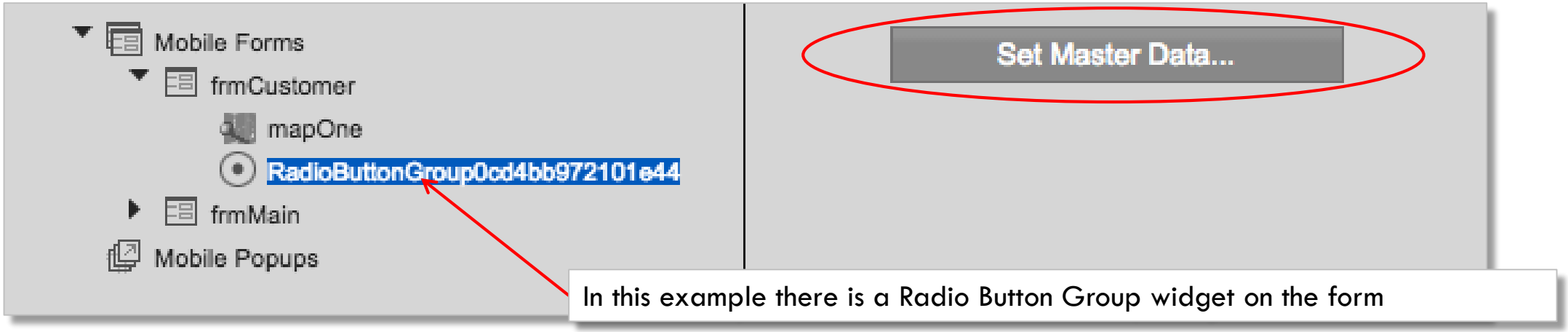
- Let's go through the actions – we'll start with the widget actions:
  - Set Widget Property – change common widget property values
    - For the button widget you can change the text and if the widget is visible or not and All widgets have an isVisible property…some have a text property



First you pick which widget you want to change – this is true for any action that affects a widget

Widget Actions - Set Master Data

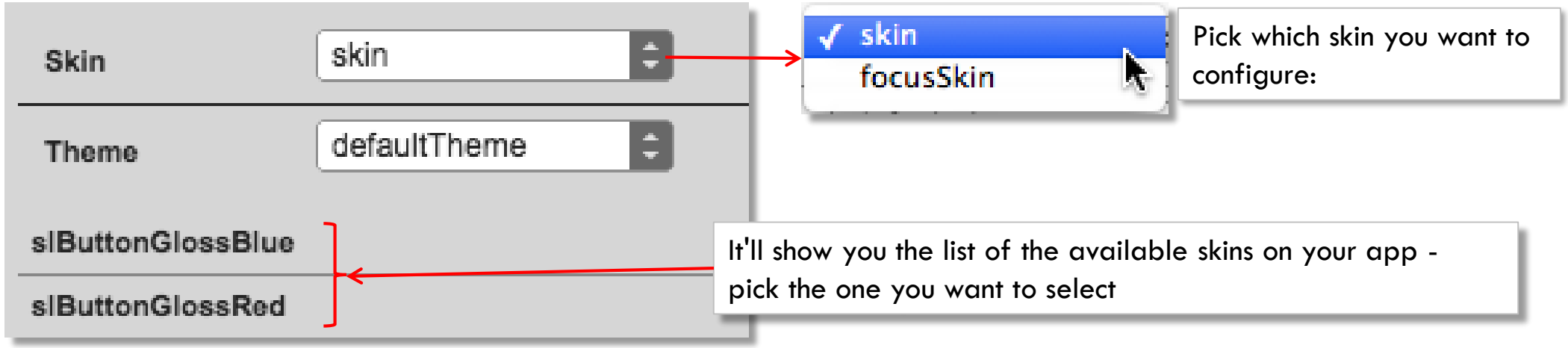
- Set Master Data - Set the data for collection type widgets like Radio Buttons, Check Boxes, etc.
- We've not talked about these widgets yet...
- You'll be able to click the "Set Master Data..." button to configure the data



- Each widget has it's own way of specifying the data
- We'll talk more about Master Data later in the training

Widget Actions - Set Widget Skin

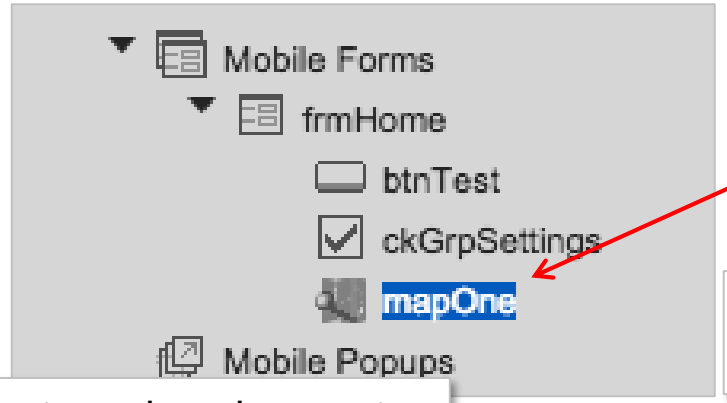
- Set Widget Skin - Assign a skin to a widget
- The example shown here is for a button widget



- There is always a default theme
  - The Theme dropdown will show you all themes you have configured
  - We'll talk about themes later on...

Widget Actions - Set Map Location

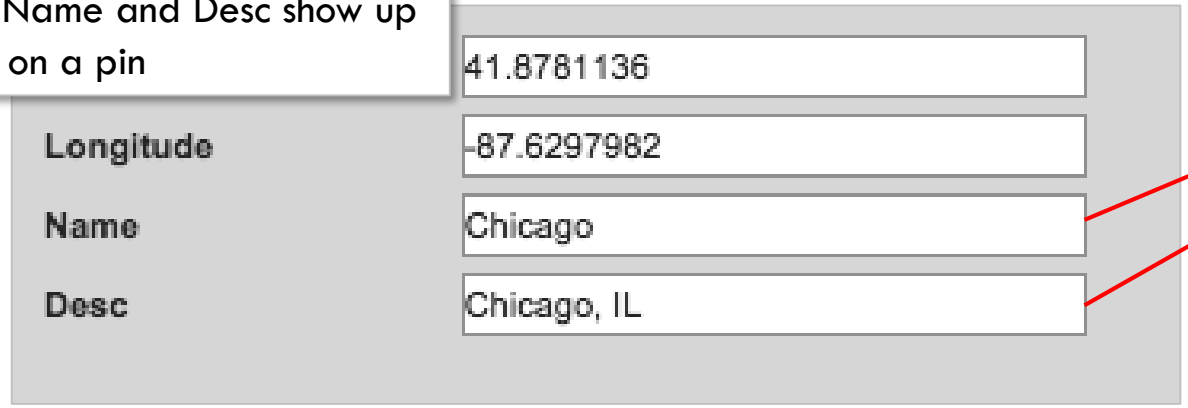
- Set Map Location - Add a location to display on the Map widget

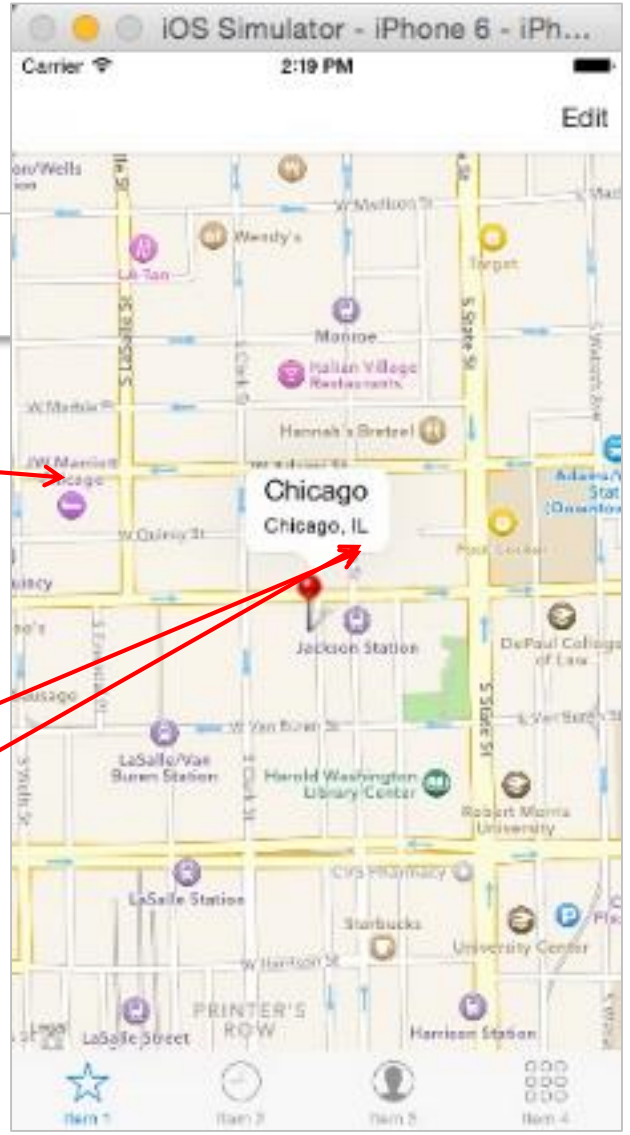


Here is a Map widget on the form

Here's what it looks like running on iPhone

This information is used to place a pin on the map – Name and Desc show up when you tap on a pin

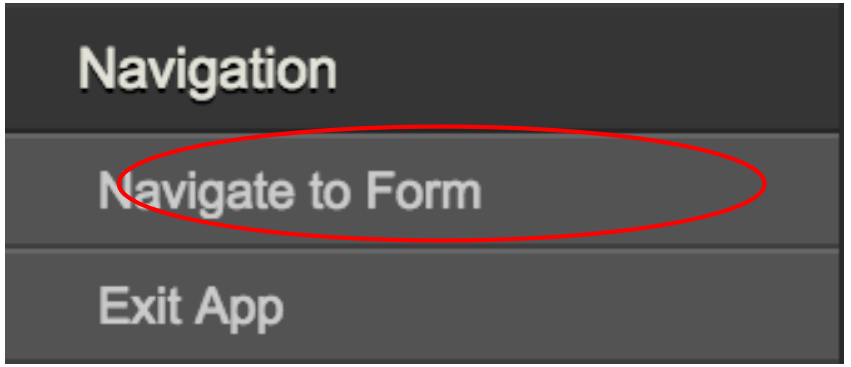


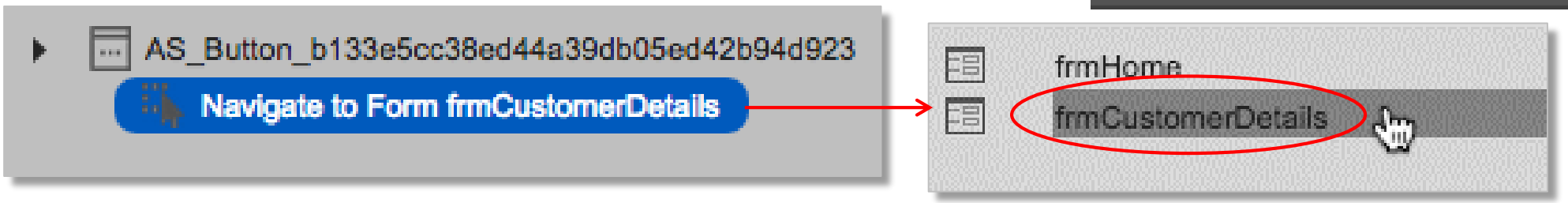


- We'll talk about this and the other widgets later on in this training...

Navigation Actions - Navigate to Form

- Let's now go through the Navigation actions
- Navigation to Form - We already saw this in the demo
- You just need to pick which form you want to navigate to

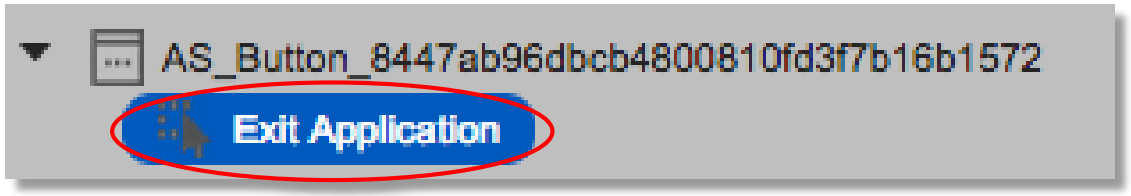


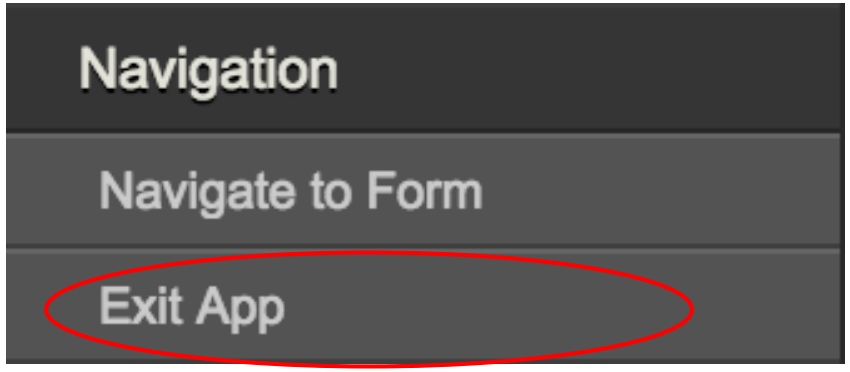


- If you want to set or change anything in the form before showing it, do all those actions BEFORE you show the form
  - The user will be presented the final version on first view

Navigation Actions – Exit App

- Exit App - Shuts down the application running on the device

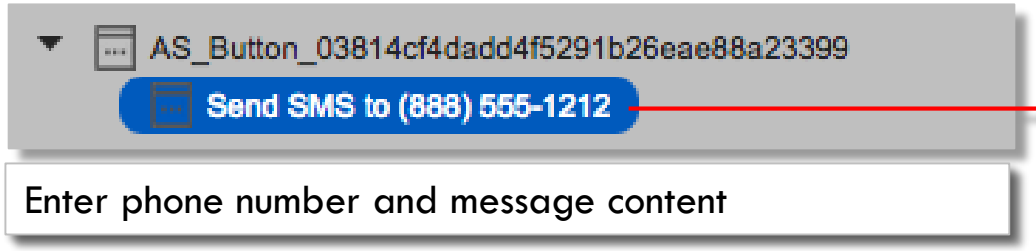


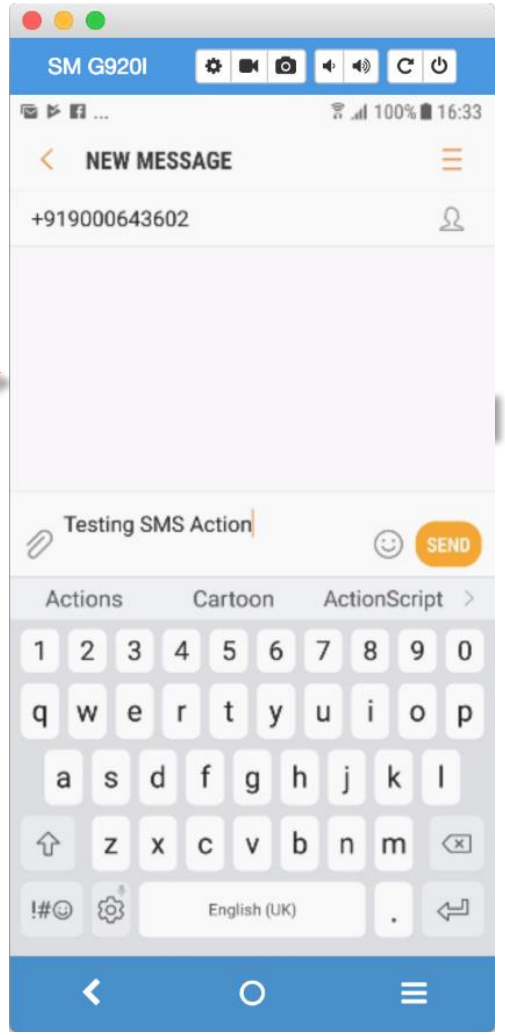


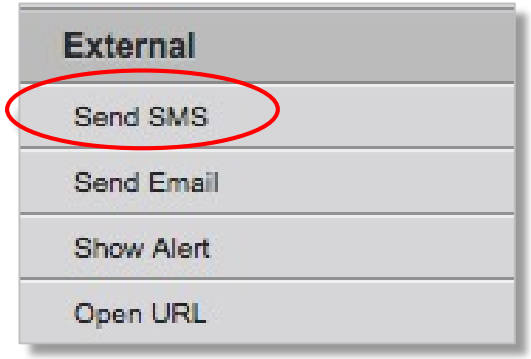
- There isn't anything to configure - just pick that action
- Note that exiting an app is not necessarily something that most applications have
  - Typical paradigm is to leave apps running
- Exiting the app is useful when testing the application
  - For example: you have things going on during application startup and need to test the application starting from scratch

External Actions – Send SMS

- Let's now go through the External actions
  - These all access functions in the Kony API
  - Send SMS - Opens an SMS message to the number provided with the message specified

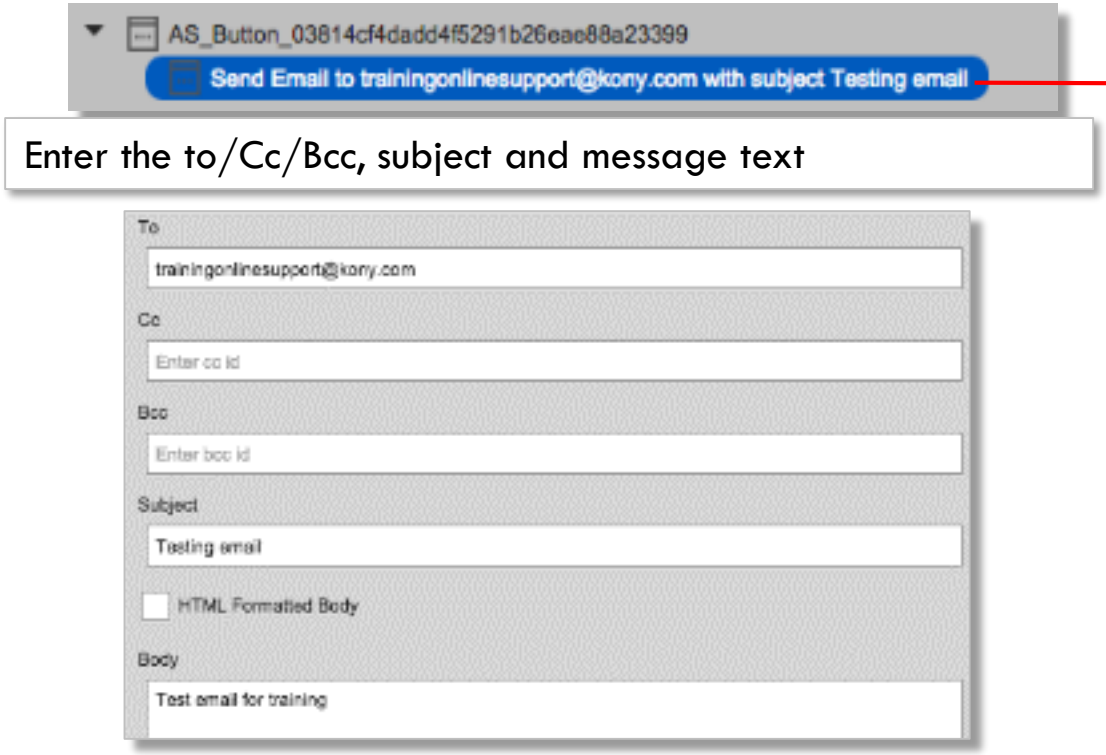


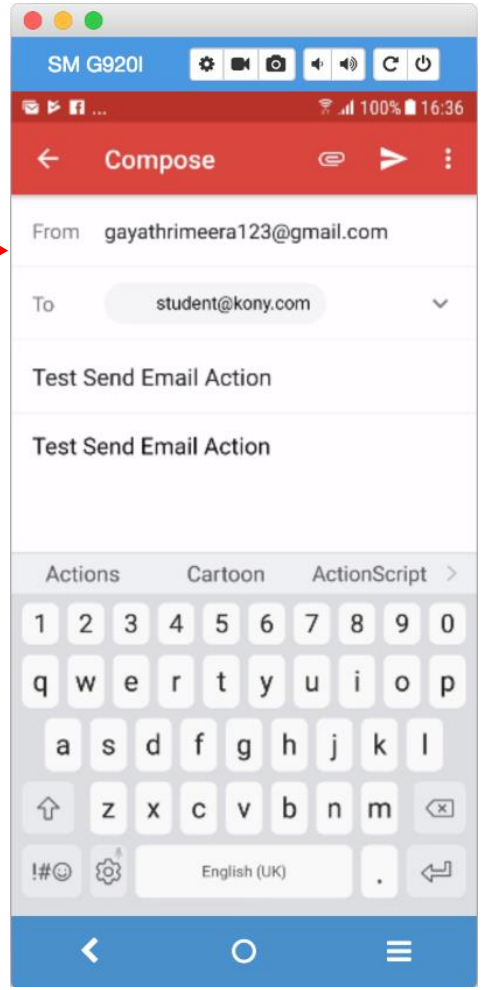


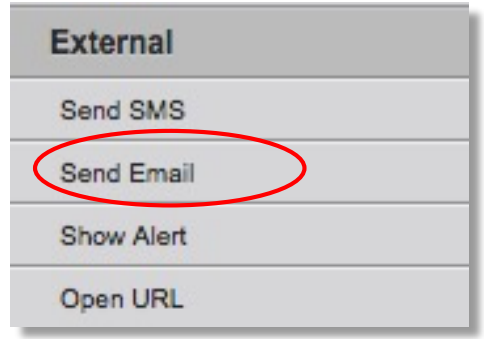


External Actions - Send Email

- Send Email - Opens an email message with the information provided in the action
  - Like SMS, it'll open an email configured as specified but the user must actually send it

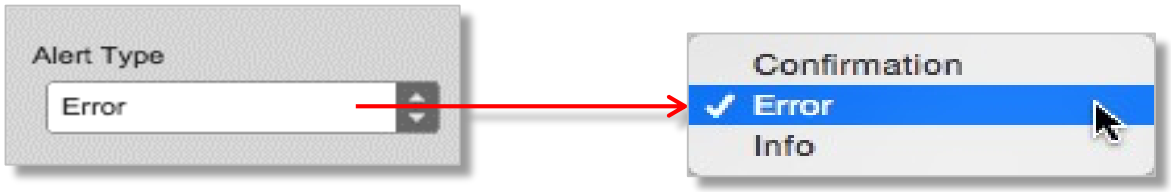


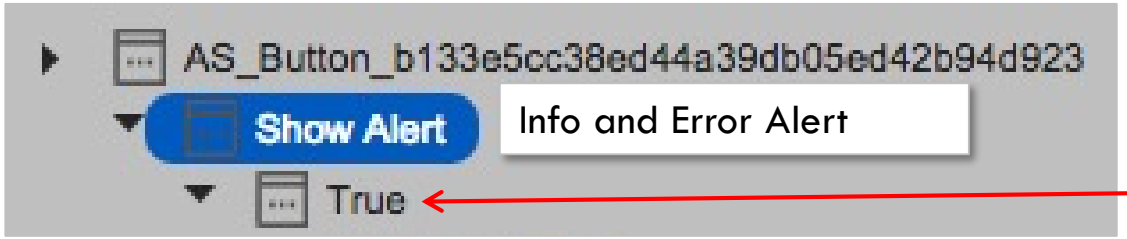




External Actions - Show Alert

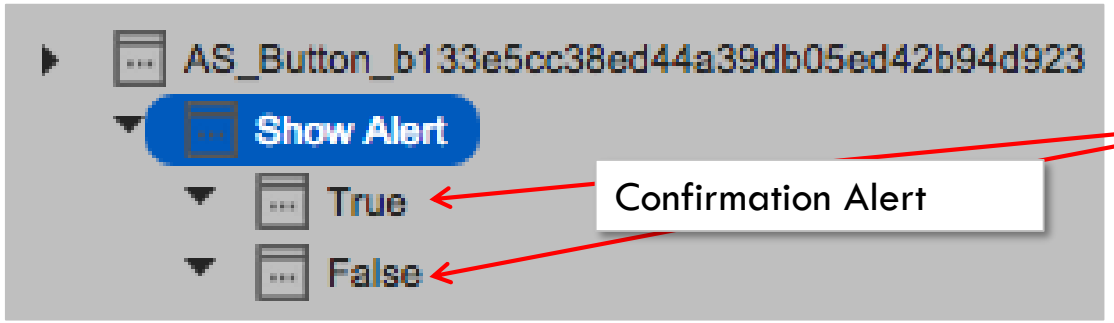
- Show Alert - Displays an on-screen modal dialog for the user
- It comes in three types depending on the Alert type:
  - You'll either have 1 button (ex "Ok") or …
  - 2 buttons (ex "Yes" and "No")





Only one button shown to dismiss dialog

We can specify what to do next by adding actions here

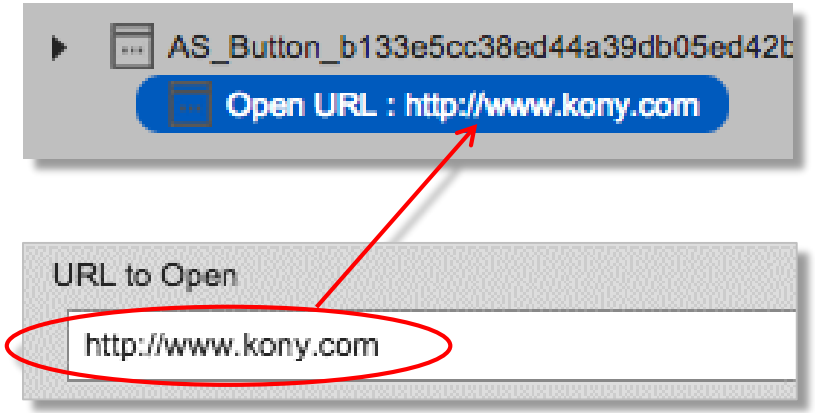


Alerts with 2 buttons allow us to specify what we do next dependent on when the user selected "Yes" or "No"

We can specify actions for each option

External Actions - Open URL

- Open URL - Opens the specified URL in the default device browser (for native apps) or in the current browser (for Single Page Application)
  - We saw an example of this in the demo
  - You only need to configure the URL to open - make sure to use http:// if you are looking to open a web page
- This is a great way to point users to your own website if you have something you want to show the users on the mobile app
- Note that in native apps, this takes the user OUTSIDE of your application
  - We'll see how the browser widget keeps the users in the app...



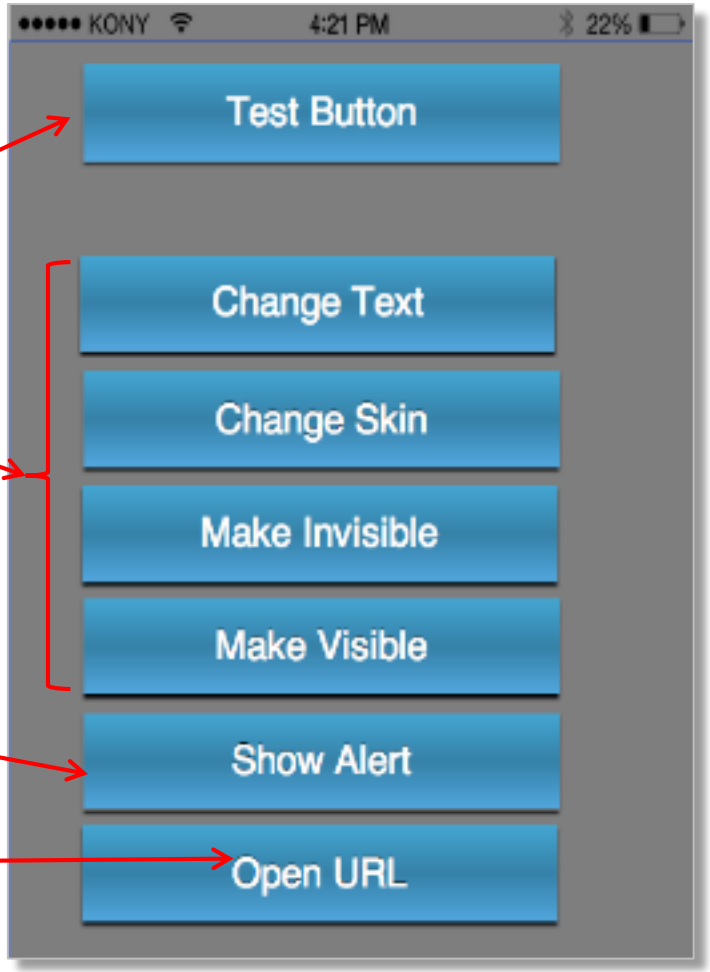
Actions - Exercise

- Let's try some of these to make sure we know how to use actions
- Here is what we'll do:
  - Let's build something similar to this screen:

For all these, use the Test Button as the target for these actions
Each button will change the Test Button according to the button text – i.e., change it's text, change the skin and make it visible or invisible

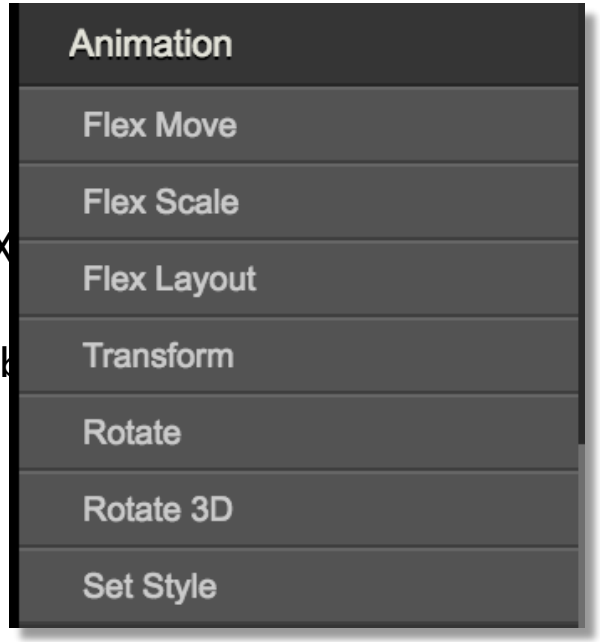
Try showing a Confirmation alert using a custom icon

Try opening http://www.google.com or any other site – don't forget the "http://"



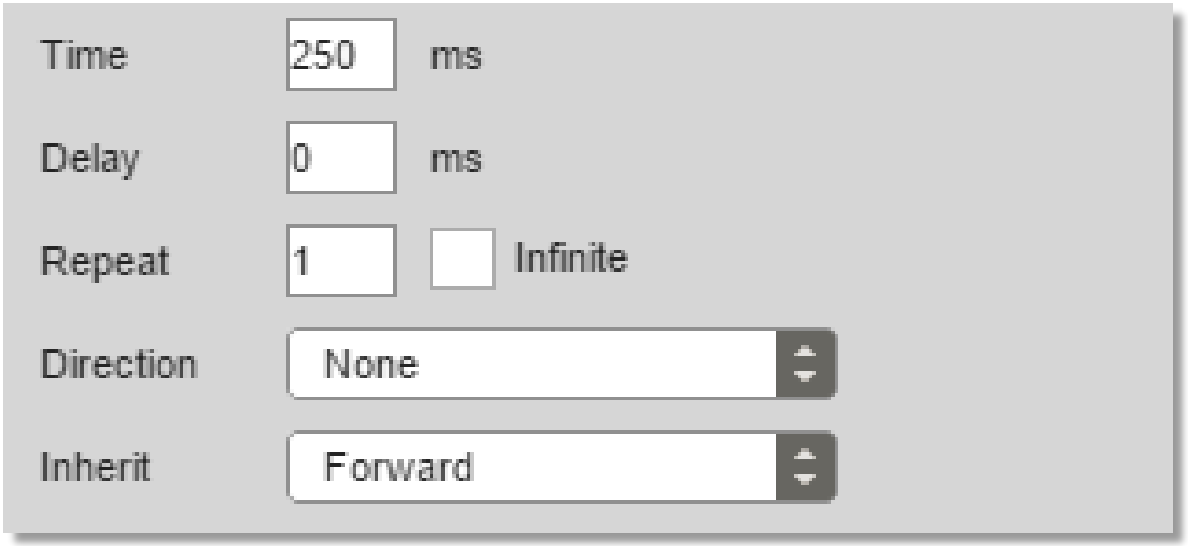
Animation Actions

- Let's now go through the Animation actions:
  - Flex Move – to move a widget in it's parent container
  - Flex Scale – to change the size of a widget
  - Flex Rotate – to rotate the widget
  - Flex Layout - Moves, scales, and rotates a widget with a single action along an X and Y axis (two dimensional).
  - Rotate 3D – to rotate the widget by angle on the unit directional vector formed b rx, ry, and rz
  - Transform – to do a compound animation, combine any or all of the 3 animation types
  - Set Style – to change the widget's background
  - Note: In each case, the animation is inside the parent container – animations may cause clipping (intentionally or unintentionally)
- Animations have 2 components: the widget change and the timing
- Let's look at the timing first – common across all animation actions...



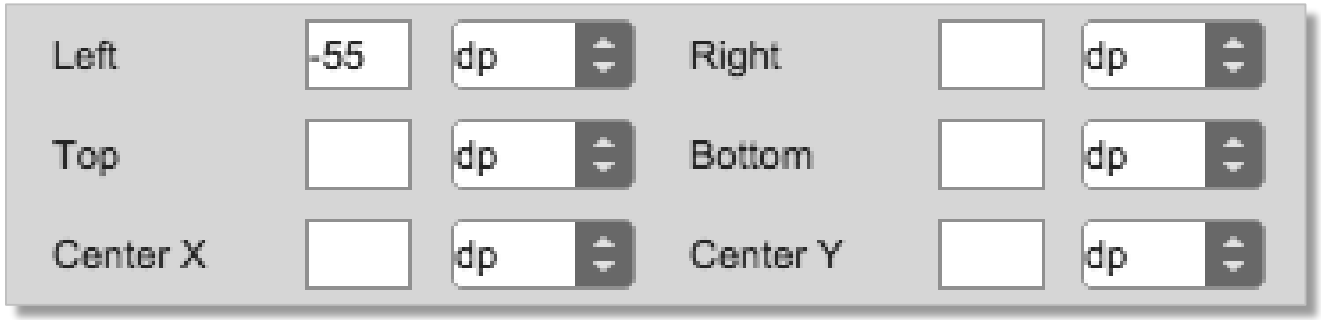
Animation Actions - Timing

- Any animation has a timing configuration component that is same for all the animation types:
  - Time – duration of animation
  - Delay – how long to wait to start
  - Repeat – 1 to whatever
    - 1 means do it once
  - Direction choices (for repeat):
    - None: same direction
    - Alternate: switch directions
  - Inherit choices (for actions, there is duplication, using the animation API, they all have unique features):
    - None: widget returns to original state
    - Forward: end of animation is the final state
    - Backwards: widget returns to original state
    - Both: end of animation is the final state



Animation Actions - Move

- Move - to move a widget in it's parent container
  - We saw an example of this in the demo
  - Here are the configuration options once you pick which widget you want to animate

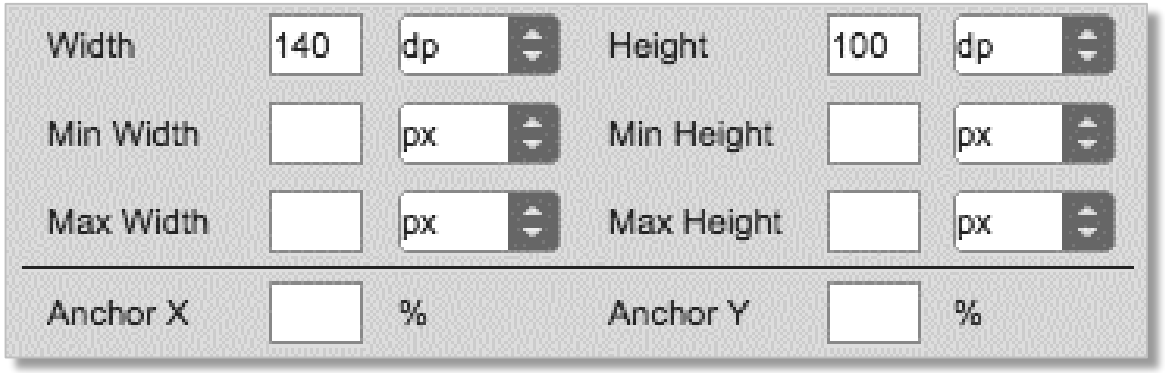


Specify more than one value to move in a diagonal line

  - Each value you specify is the final value you want to have
  - Examples:
    - 55 DP Left value: this moves the widget to the left such that the left 55Dp of the widget is beyond the left side of the container
    - 50% Center X and 50% Center Y values: move the widget to be centered to the middle of the container

Animation Actions - Scale

- Scale - To change a widget's size
  - Here are the configuration options once you pick which widget you want to animate:



  - Each value you specify is the final value for the animation - not the change value
  - Min or Max settings can be set to make sure widgets never get too big or too small
    - For example: On screen rotation, you may not want a button actually still spanning the screen width in landscape - set a Max width
  - The widget will grow or shrink around the layout values initially set for the widget
    - If you specified top/left - that will stay unchanged as the widget size changes
    - If you specified center X/Y, then it will change size leaving the center of the widget at that location

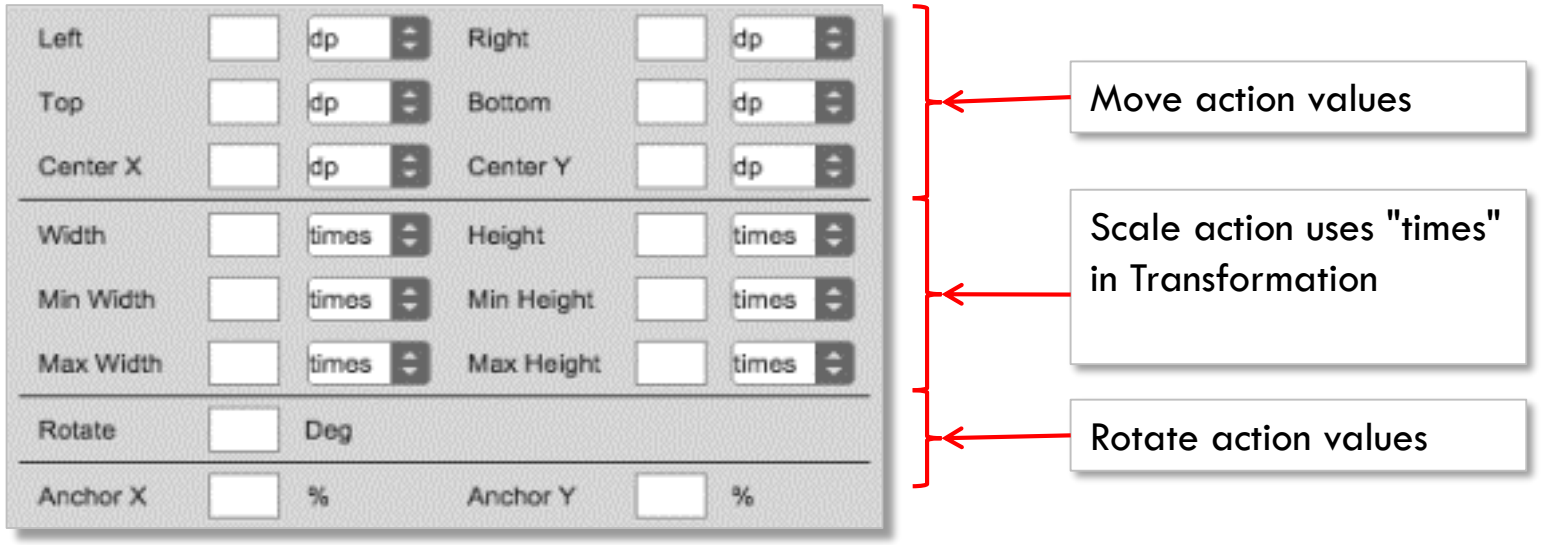
Animation Actions - Rotate

- Rotate - To rotate a widget
  - Here are the configuration options once you pick which widget you want to animate:

  - The rotation amount is the final rotation value – not the amount it will rotate from it's current position.
  - A positive value rotates counter-clockwise
  - A negative value rotates clockwise
  - Avoid a rotation that will rotate more than 179 degrees in either direction
  - For example, to do a complete circle – use 3 rotation animations:
    - rotation #1 set to 120 degrees
    - rotation #2 set to 240 degrees
    - rotation #3 set to 360 degrees
  - Anchor X and Anchor Y specifies the horizontal and vertical anchor points from where widget rotation begins

Animation Actions - Transform

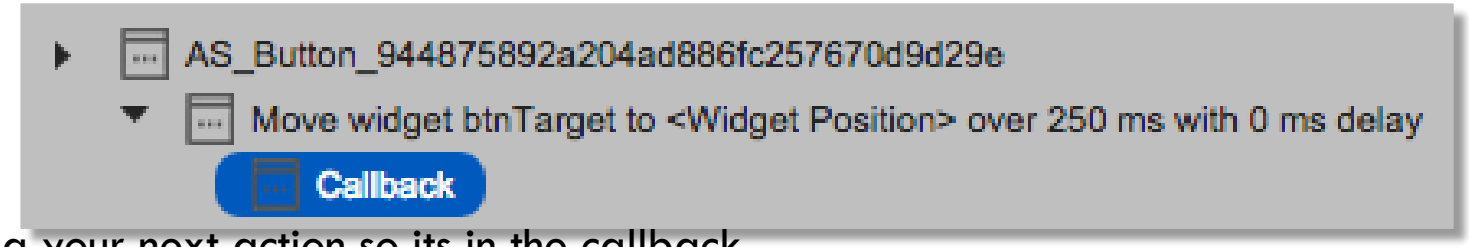
- Transform - To create a composite animation consisting of a move and/or a scale and/or a rotate animation
  - Here you get all the configurations for all the animation types:



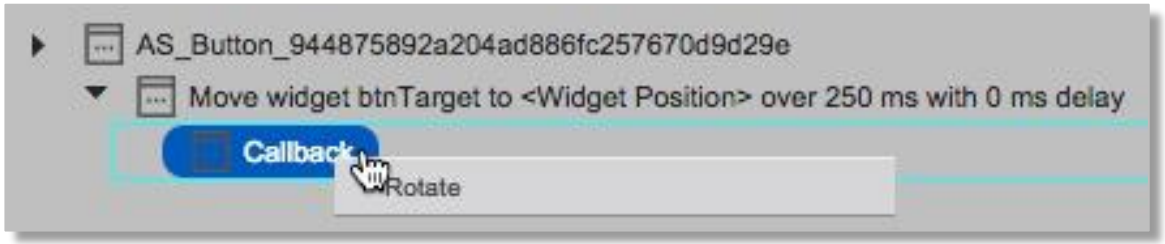
  - For Move and Rotate - unlike the individual actions - your values are the amount of movement/rotation rather than the final value
  - For Scale, we now have a "Times" value - use the value 1.5 for width and height to scale a widget with a 50% increase

Animation Actions - Stacking Actions: New

- Callback - This property allows you to initiate one action to run when the current action completes.
  - In the callback of your current action…



  - Drag your next action so its in the callback



  - You can actually do anything you want here, for example, such as navigate to another form
  - Note: If you drag your next action below the Move action (in our example above) that action will start as soon as the Move action starts

Form Animations

- While we are talking about animations, the form itself has lots of properties that affect how it looks/acts:

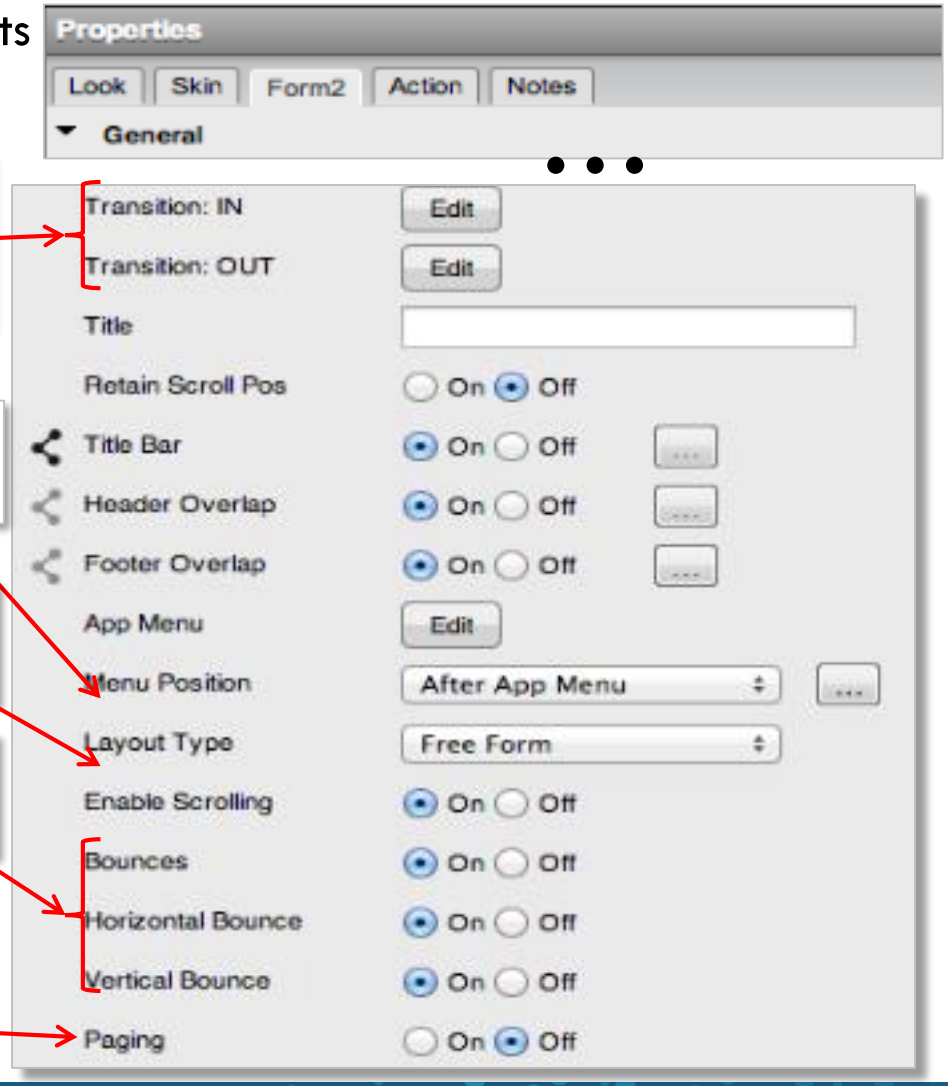
Transition on the form dictates the type of animation used to move THIS form in and out of view – each device has it's own options but there are a LOT of effects you can use

On Android – if the developer adds menu items, this determines if they are put before or after the App Menu items

Like flex containers, the form can be created in Free Form or Vertical or Horizontal layout mode

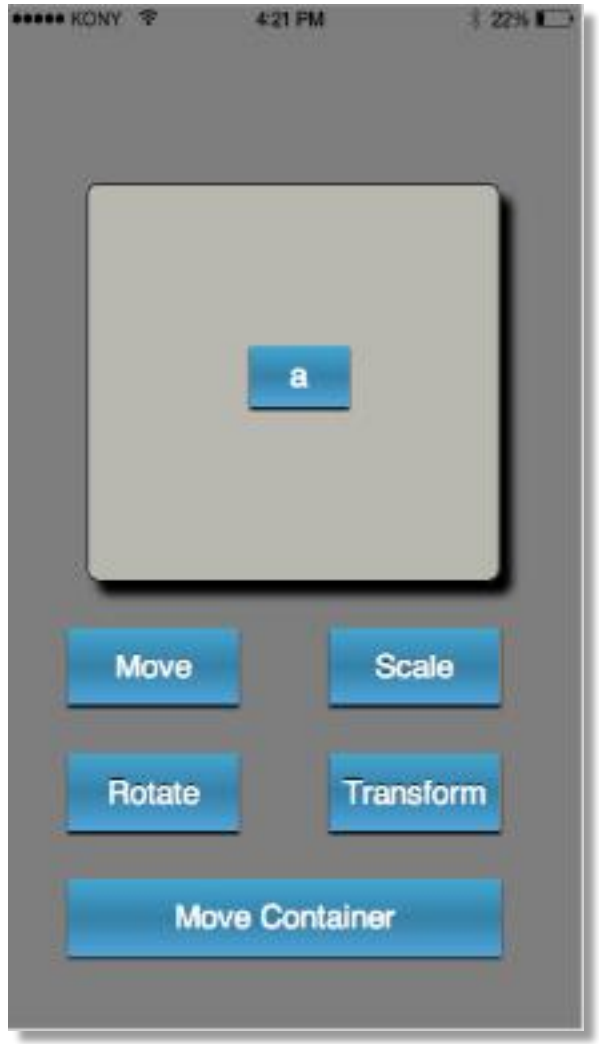
As the user scrolls the screen, you can control the amount of bounce when it hits the top or bottom vertically or horizontally

Can optionally scroll page at a time



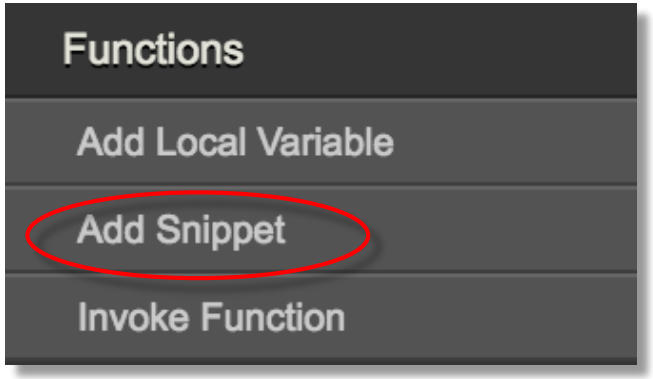
Animation Actions - Exercise

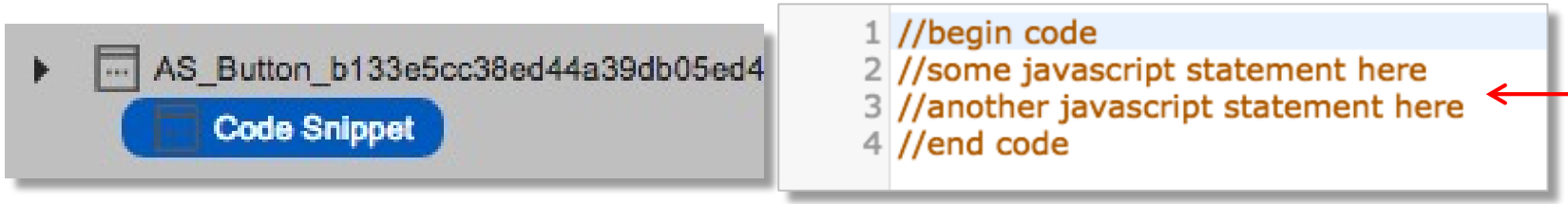
- Ok, let's try it!
  - We'll build the screen on the right that has a button (text is "a") in the middle of a flex container that has rounded corners, a shadow and background color
  - The Move button moves the "a" button to the left and upwards by 50Dp
  - The Scale button makes the "a" button twice as big in width and height
  - The Rotate button rotates the "a" button 120 degrees
  - The Transform button moves the "a" button to the left and upward by 50Dp and rotates by 120 degrees
  - The Move Container button moves the whole flex container down and to the right by 50Dp
- For each action, set repeat to 2, direction to "alternate" and inherit to "forward" – this does the animation and then reverses to go back to the original position...OR...specify inherit as "none" and it'll return to the start position after the animation is complete



Function Actions - Add Snippet

- Let's now go through the Function actions:
  - We'll talk about "Add Local Variable" and "Invoke Function" later
  - Add Snippet – lets you write JavaScript code, including using the Kony API, to do anything you want. For example:
    - Write simple JavaScript code
    - Call commonly used functions



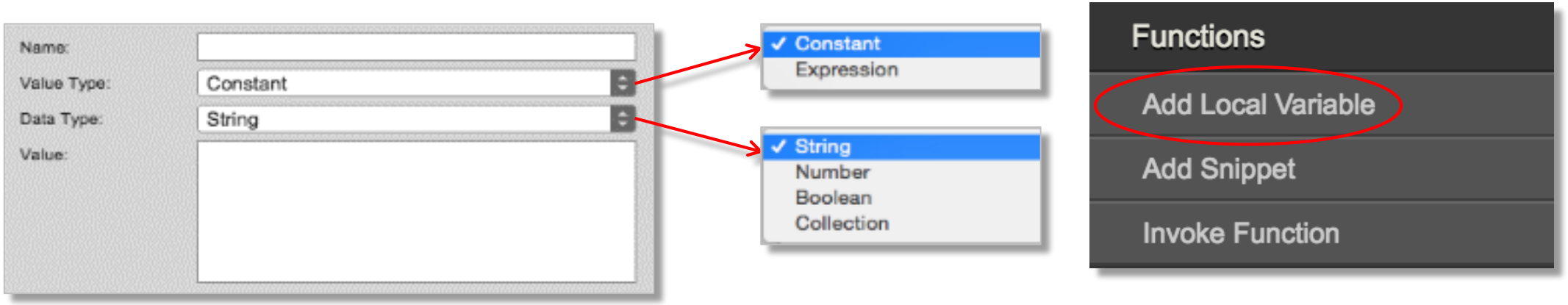


The configuration area for a snippet is a code window where you type in whatever code you want to run

- Snippets are very helpful if you have even basic JavaScript skills. For example:
  - Lets you implement IF...THEN... functionality
  - Access ANY widget property to change values
  - Store code in modules for easy portability

Function Actions – Add Local Variable

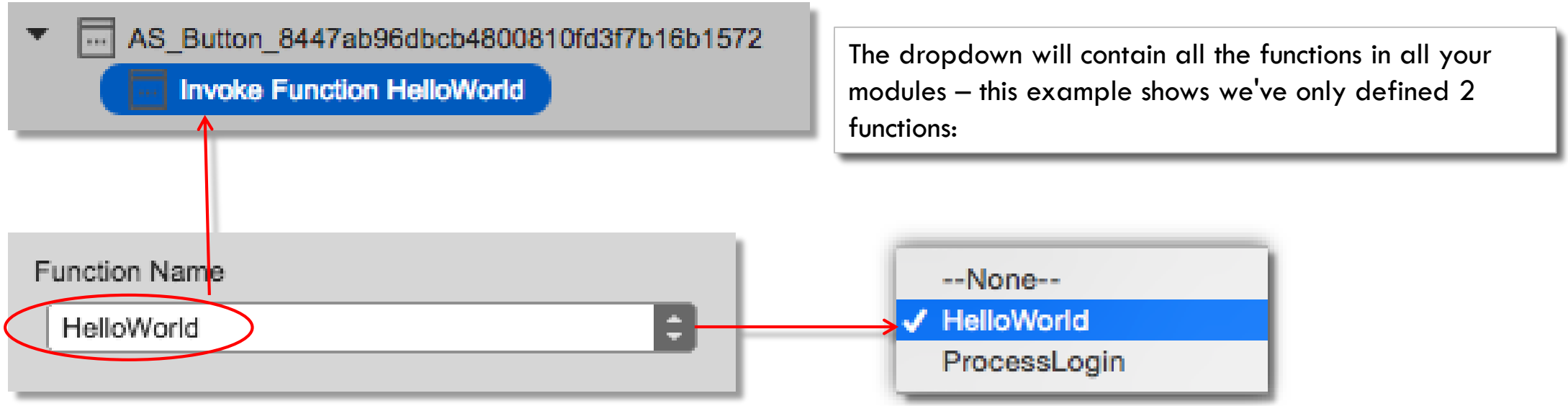
- Used to store data to be used in your action sequences
  - Very limited use – Should only be used in snippets
  - Add Local Variable – Let us create a variable that can be used to pass data from one snippet or action to another
    - We'll talk about code later on, but let's look at a simple example:



- Using a Value Type of Expression will allow you to set the value of the variable in another snippet within the same action sequence

Function Actions - Invoke Function

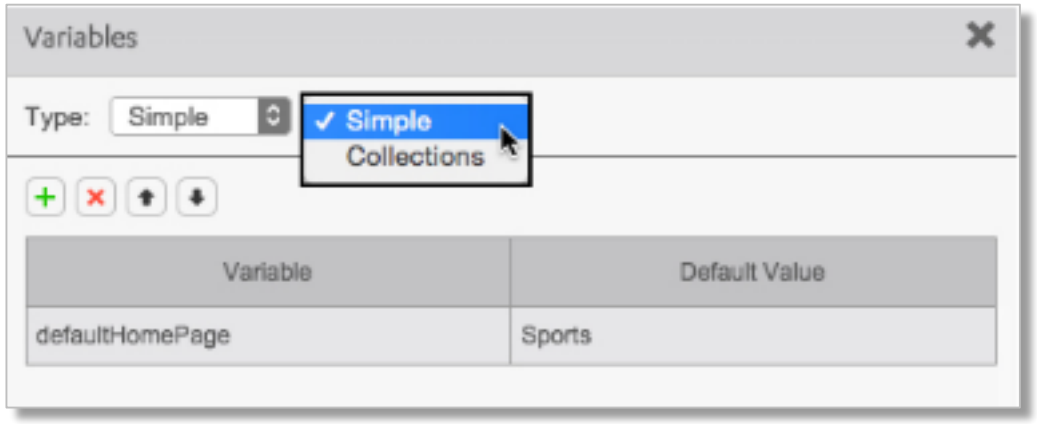
- We'll cover this in more detail when we discuss using JavaScript modules, but any functions you've created are available for use here
- We can call any of these functions with an invoke function

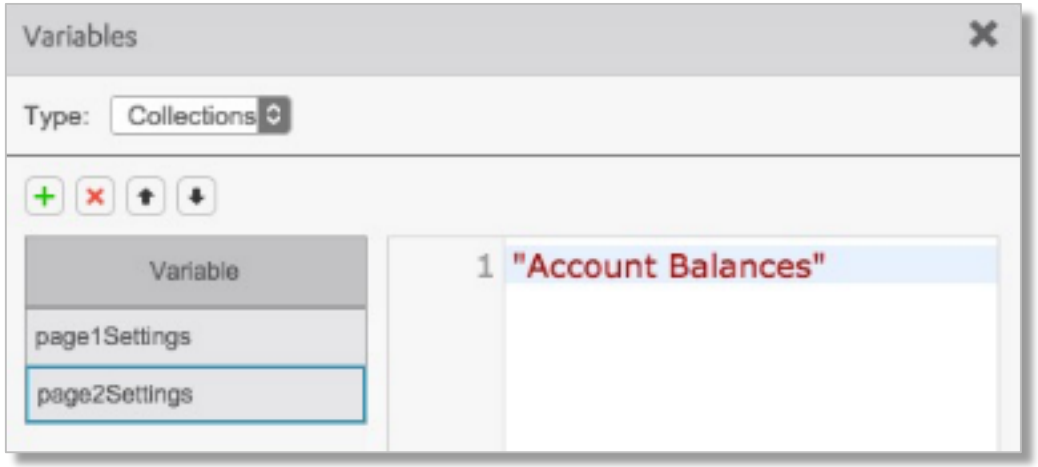


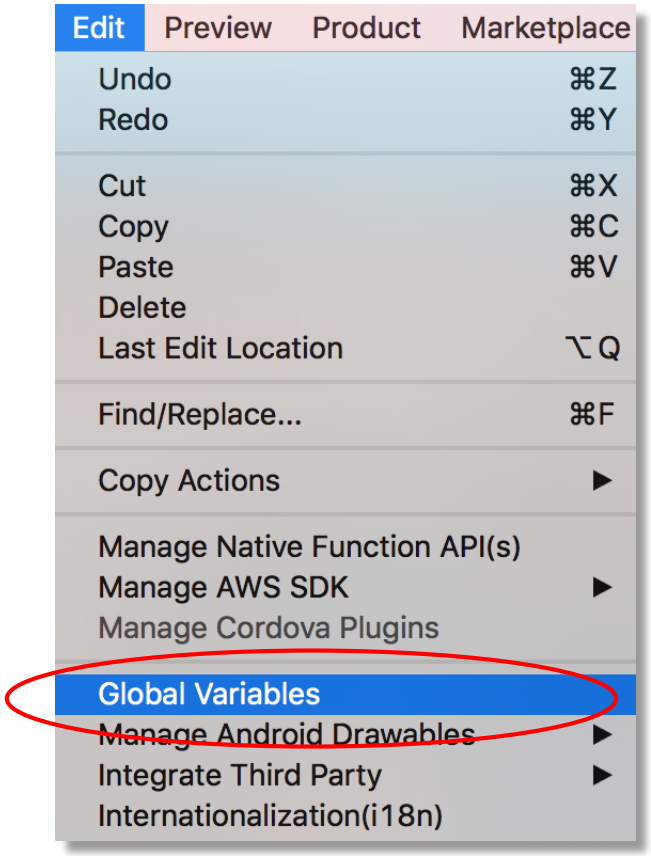
- You'll be able to configure parameters if they are defined

Global Variables

- These can be simple variables or a collection

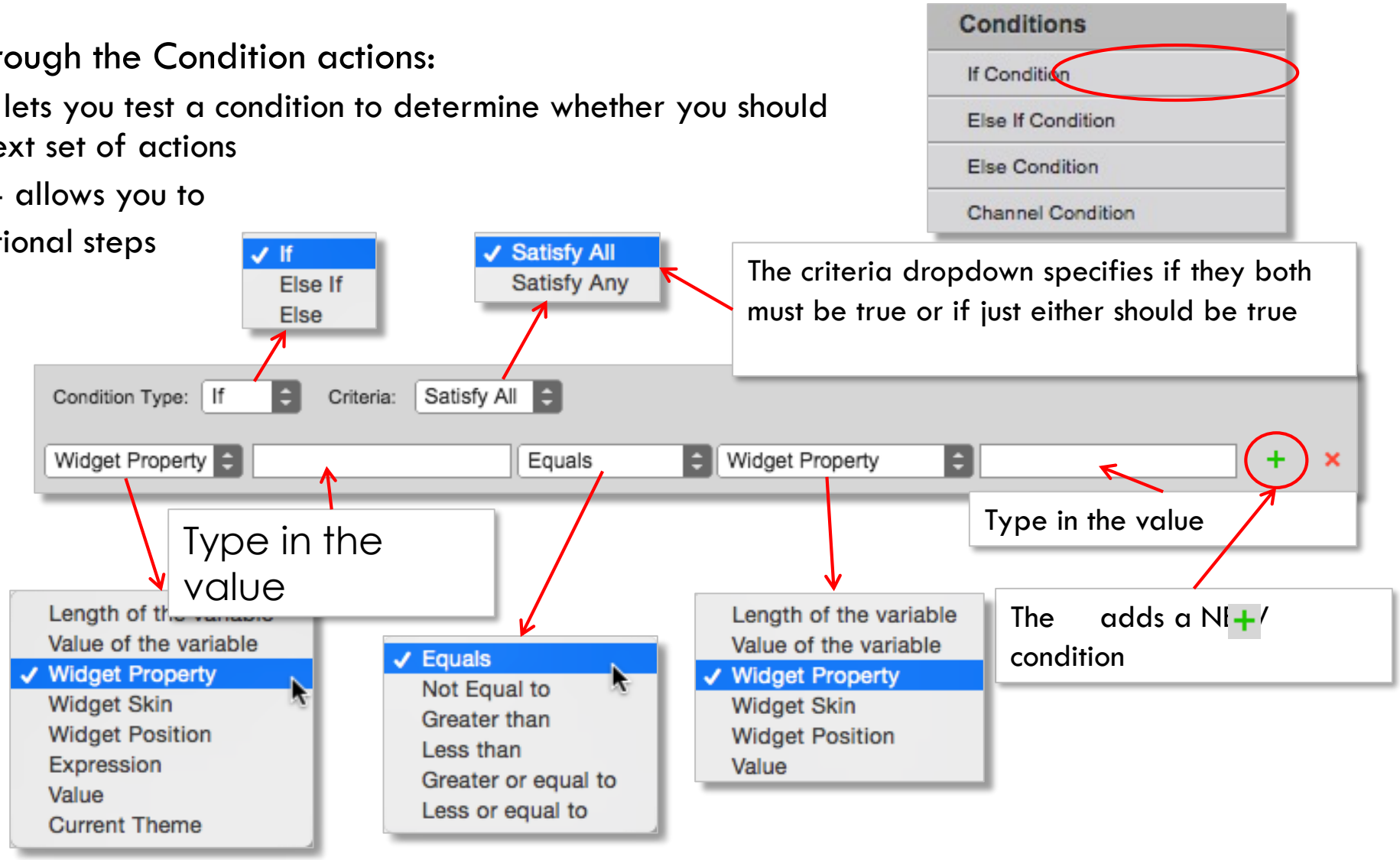






Condition Actions

- Let's now go through the Condition actions:
  - If Condition - lets you test a condition to determine whether you should process the next set of actions
  - Else If / Else - allows you to provide conditional steps



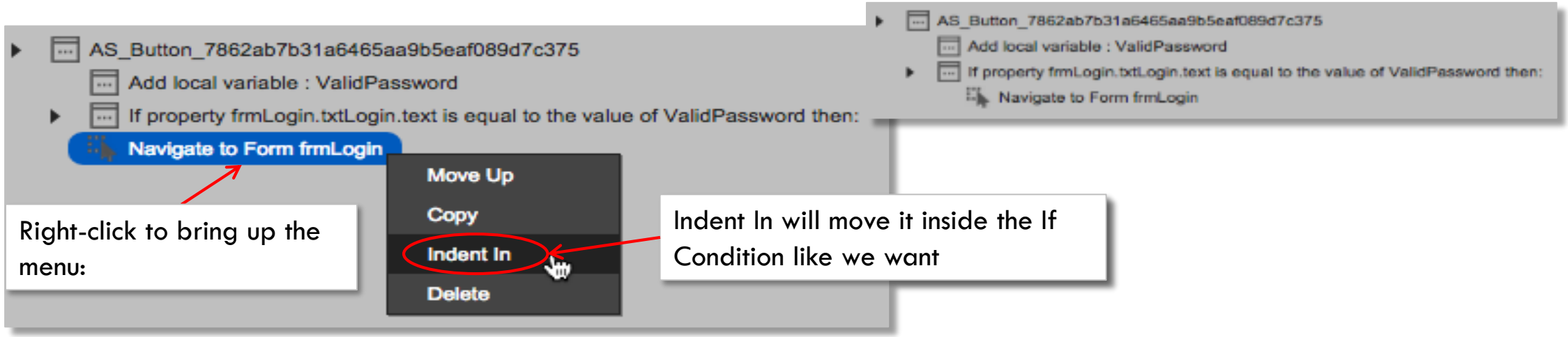
Condition Actions Example

- Let's look at an example to see how this works
- We're building an app with a login screen and we want to be able to test BOTH valid and invalid login situations
  - We want people testing our app (using functional preview) to see how they both work
- Here's what we'll do:
  - We'll create a local variable in our action sequence to be the valid password value
  - We'll compare what the user types in to see if it equals our variable value (our valid password)
    - If it does, we'll navigate to the home screen
    - If it doesn't, we'll show an alert that the password is incorrect

- Let's take a look at our project...

Conditions – Indenting

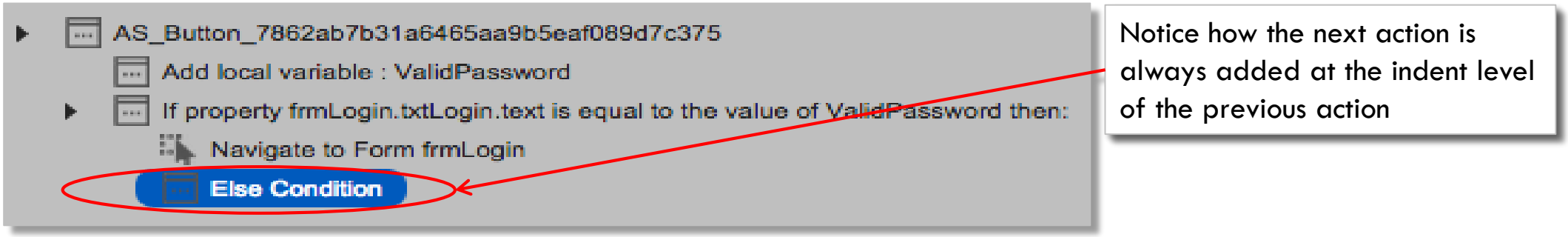
- We want the Navigate action to be INSIDE the If Condition, not the next action
- Right click the Navigate action and choose "Indent In":



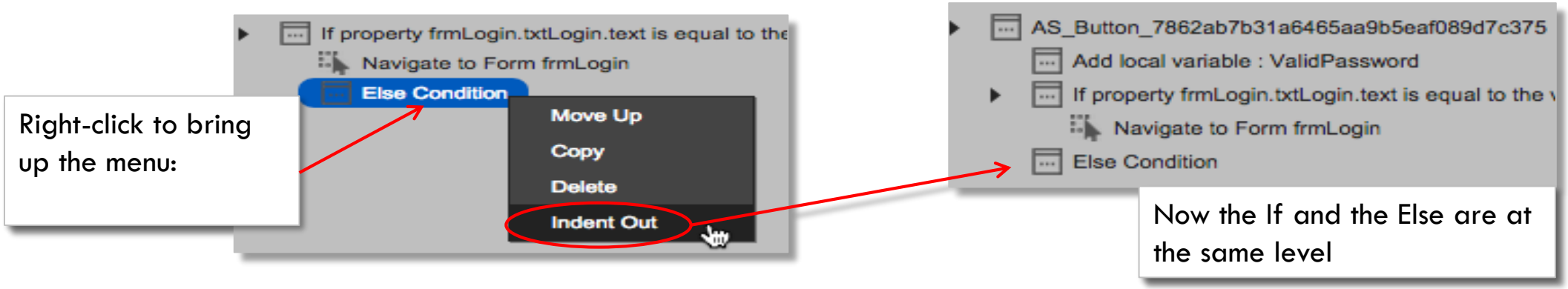
- And now our flow looks correct - the Navigate will ONLY run if the If Condition returns true:
- Now let's configure what happens if the login value is incorrect - we need to display that alert to the user...

Conditions Actions - Else Condition

- The way we specify what happens if the If Condition returns FALSE is to add an Else condition:



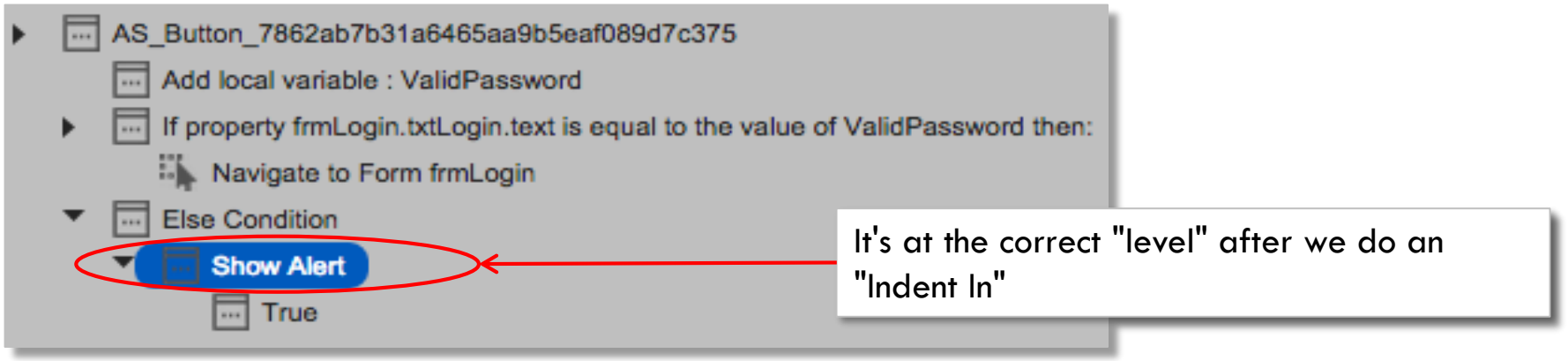
- We want the Else Condition to be at the same "level" as our If Condition
- If needed, right click the Else Condition and this time choose Indent Out:



- The last step is to add our Alert action to our Else Condition...

Conditions - Else Action

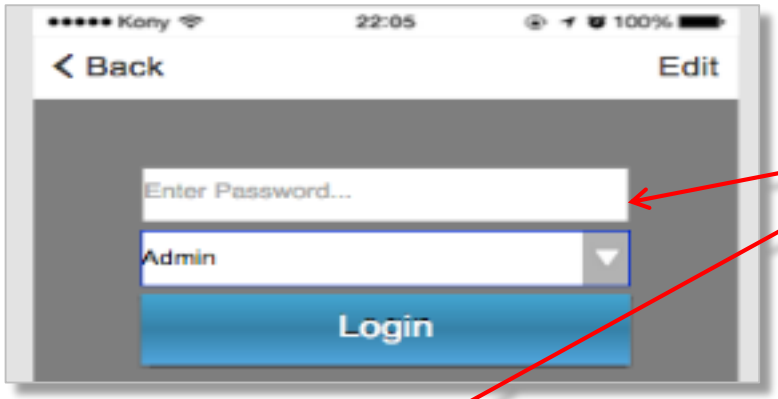
- Now, we can add our Alert for the Else Condition
- Note: we'll have to Indent In to make sure that the action is "under" the Else Condition and not just the next action in the flow:



- Do we need to configure the Alert True condition?
  - Well...what do you want to happen if they typed in a bad password?
    - Typically you just do nothing – leave them on the same screen to try entering something new
    - You may want to blank out their password attempt
      - To do that, for the True condition, do a Set Widget Property action and set the password texbox's text property to a blank

Conditions - Else If Example

- In our example we had a simple true/false decision to make - but sometimes you might want to check for more than one case
  - For example, what if we had the user login AND specify what type of user he/she is – this would determine which page gets displayed to the user
    - Our app might look like:



Here is the new listbox where the user picks the type of user

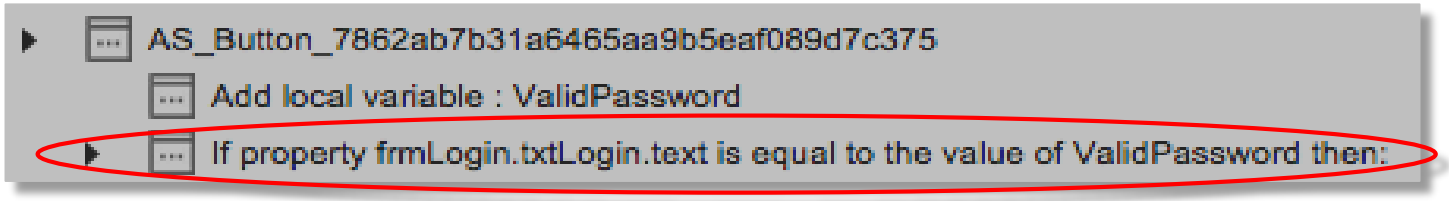
Here is the Master Data – note that we assign key values that are easy to use and understand

Master Data: lstUserType

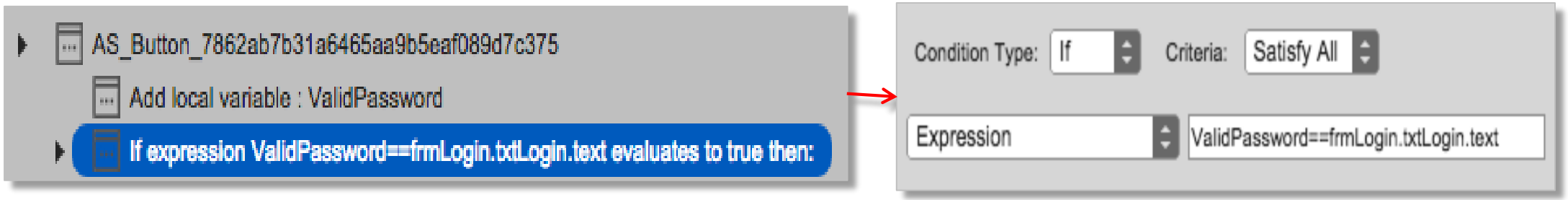
Key	Display Value	Selected Key
admin	Admin	☑
manager	Manager	☐
user	User	☐

Conditions - Using Expressions

- If you are a programmer, you'll maybe find using the Expression type the easiest way so you can enter the condition in code
  - We can replace our original If Condition:



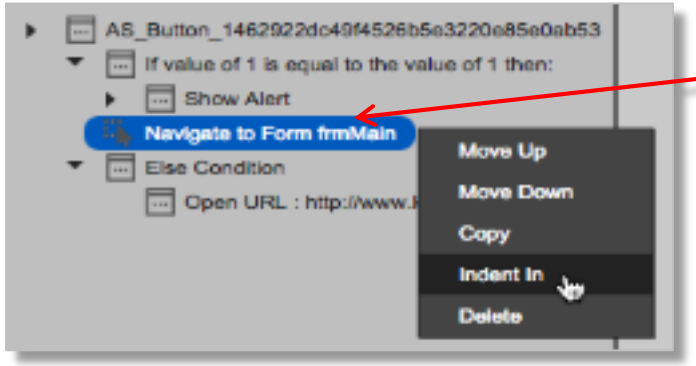
  - With an Expression that does the same thing:



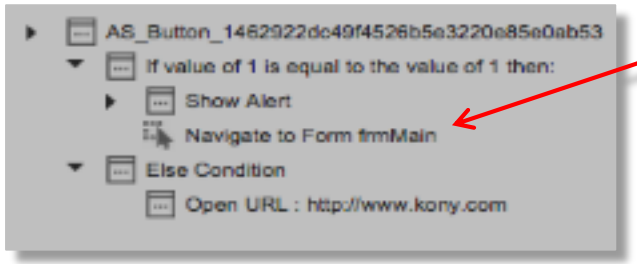
- The Condition builder tool is nice if you don't know the Kony API, but if you do - Expressions are going to be the easiest to use

Indentation Revisited

- To make sure you sequence your actions correctly with a Condition action, indent is critical
- Right click allows you to indent your actions to keep them grouped within your "IF" step, your "Else If" step or your "Else" step



Right-click on the Navigate action…



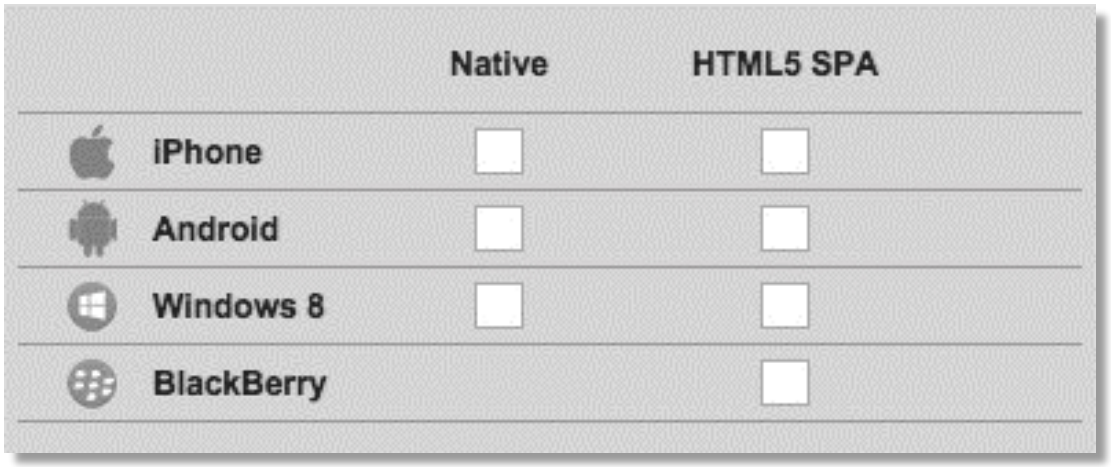
Brings it to the same indentation level as the Show Alert action!

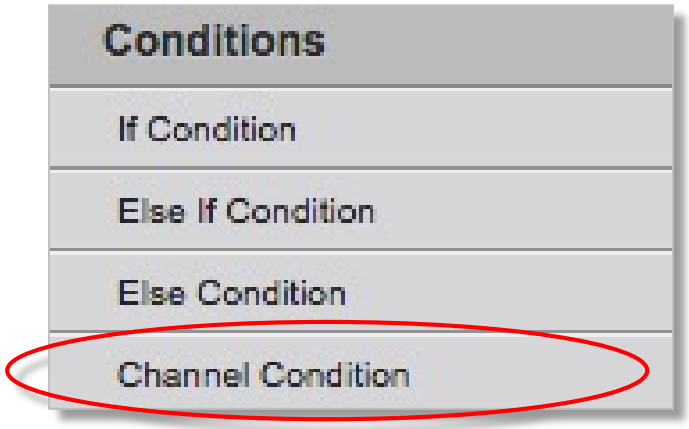
This means the Alert will show and then they'll be taken to frmMain

- Just be sure that you thoroughly test ALL your branches to make sure they are correct

Conditions Actions – Channel Condition

- Finally, Channel Condition:
  - Channel Condition - lets you test for a particular device to determine whether you should process the next set of actions
  - In this condition, you pick which device/output type you want to check - if true, it'll perform whatever actions you put in the condition

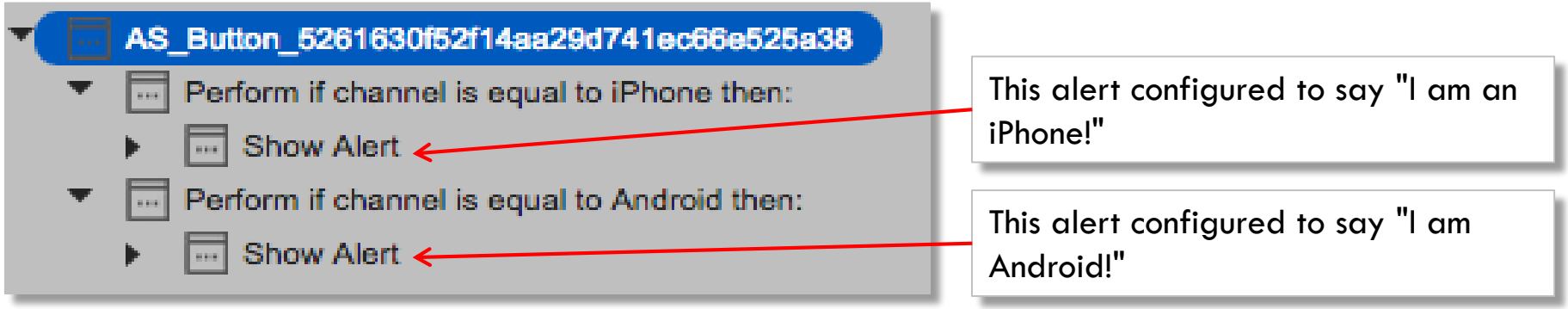




- Let's take a look...

Conditions Actions – Channel Condition (cont'd)

- Here's an example where I will run a different code snippet for Android or iPhone
- Design Time in Visualizer



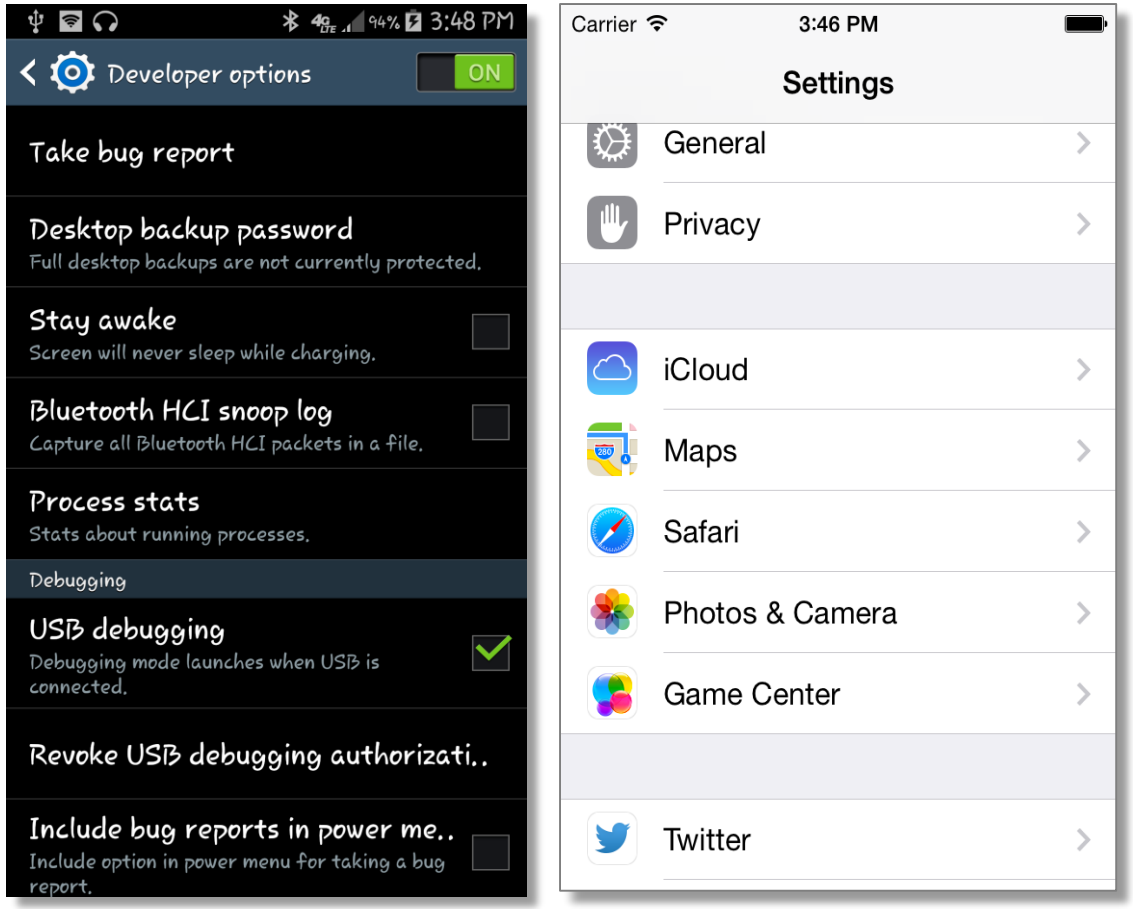
- And at Run Time…

- Use Channel Conditions any time you need to do something unique/different on one or more devices

Advanced Widgets

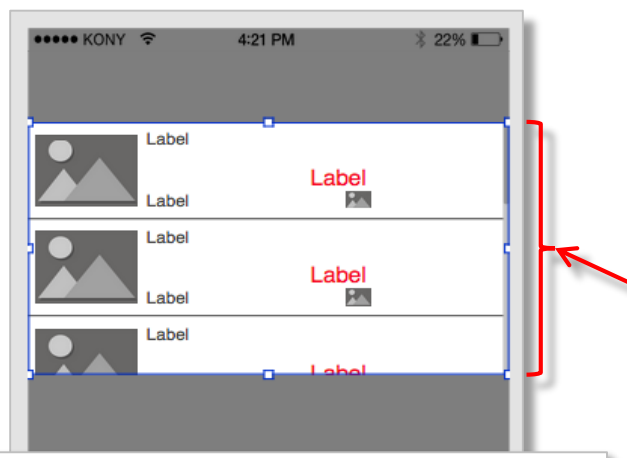
Segment Widget

- The Segment is used for creating repeating rows - the widget will add as many rows as necessary for all the data it's given
  - There are segments in 99.99% of all applications
- Here are some examples from Android and iPhone screen settings:



Segment Widget - Layout

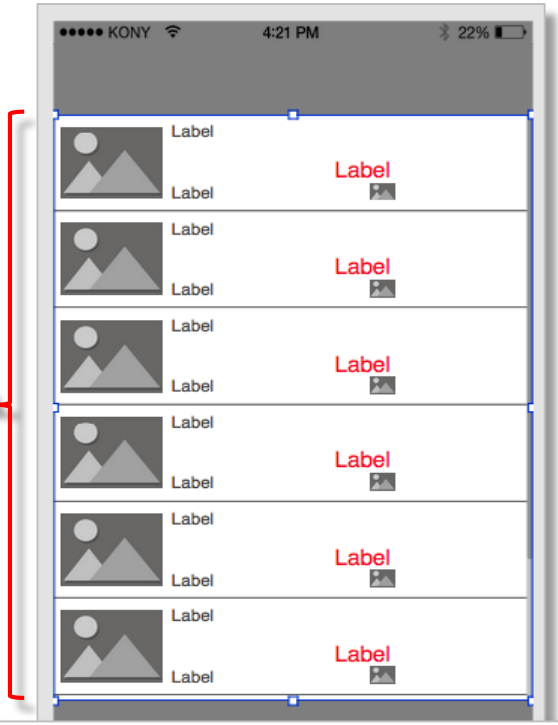
- The Segment widget still has a fixed size - and rows that would extend beyond that size are not shown but the widget is scrollable:



Note: adding widgets to the segment copies the widget to all rows

It shows 3 rows by default but you can add more or remove them if you want

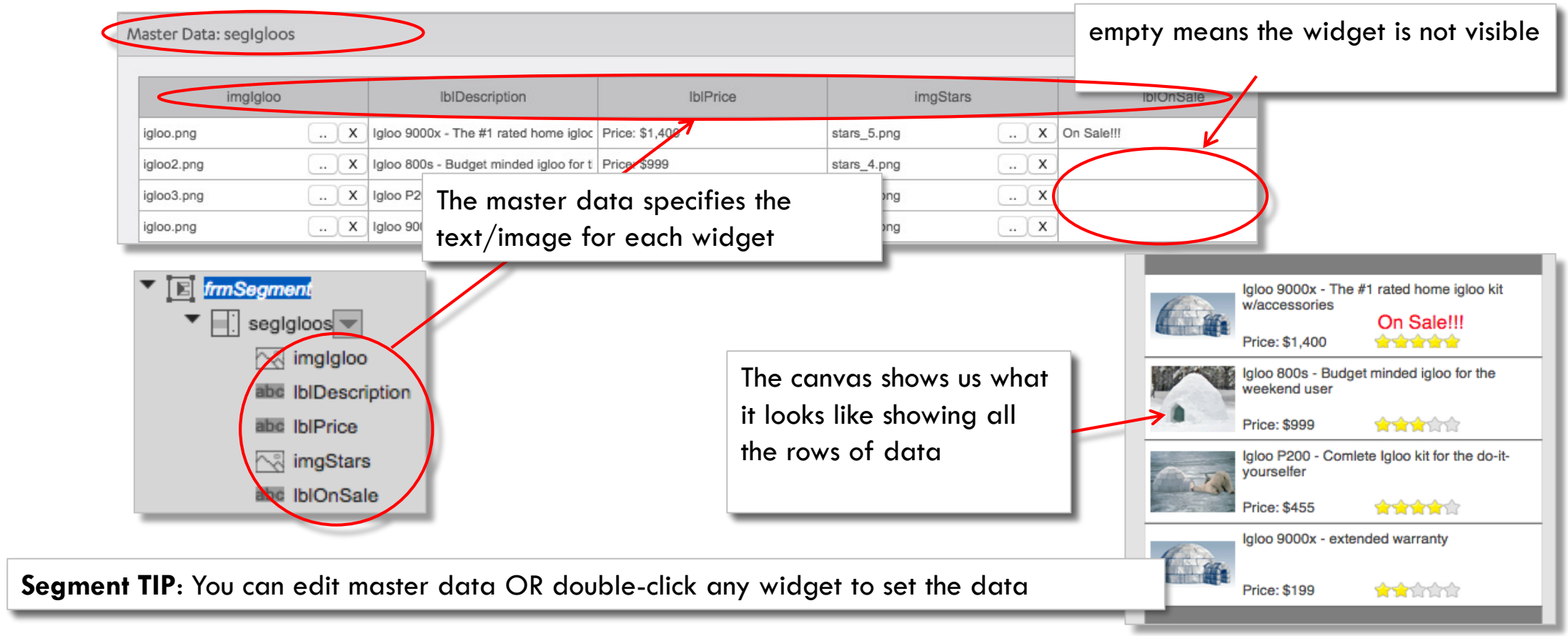
In both configurations, all the rows will scroll within the segment boundaries



You VERY MUCH want to use relative sizing for the segment to make sure it fits right on the screen – i.e., use % and not Dp

Segment Widget - Master Data

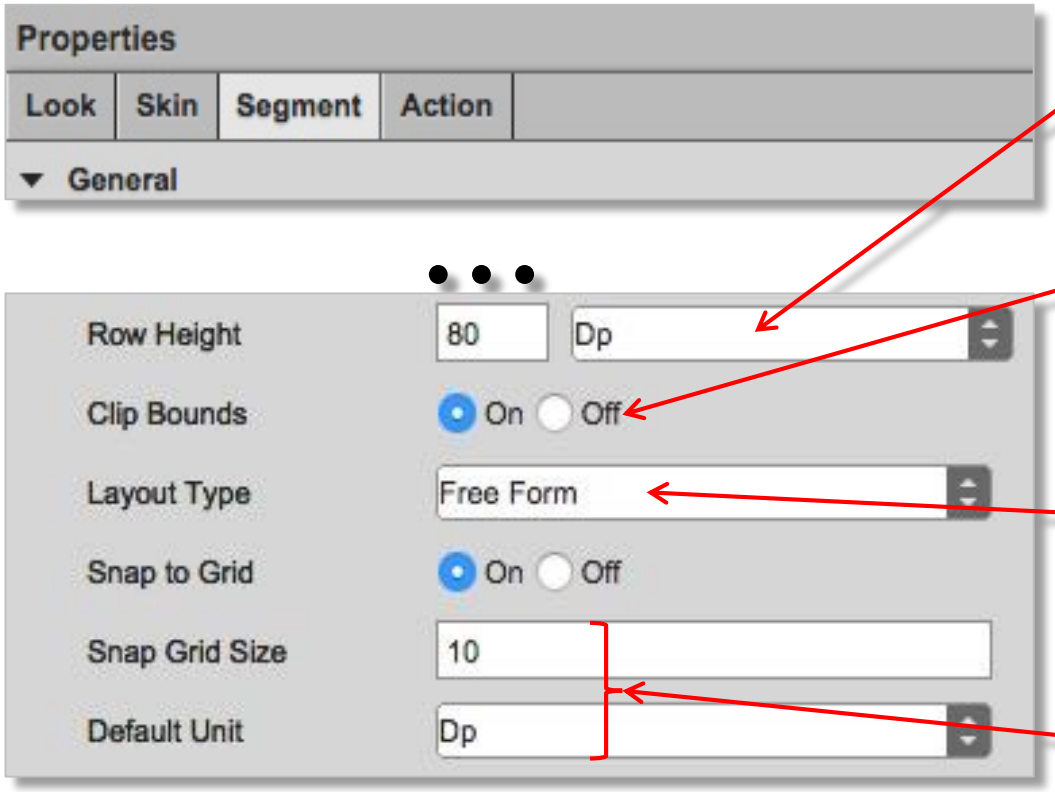
- The Segment is a container - our job is to populate it with whatever widgets we need to show all the information for each "row" of data
  - Here's an example of a segment with 4 "rows" of data defined in Master Data:



Segment TIP: You can edit master data OR double-click any widget to set the data

Segment Widget - Layout (cont'd)

- Let's look at the Segment configuration properties that apply to layout:



here is where you specify the layout size for each row

Like Flex, you typically do want to clip – anything outside of the container boundaries is clipped

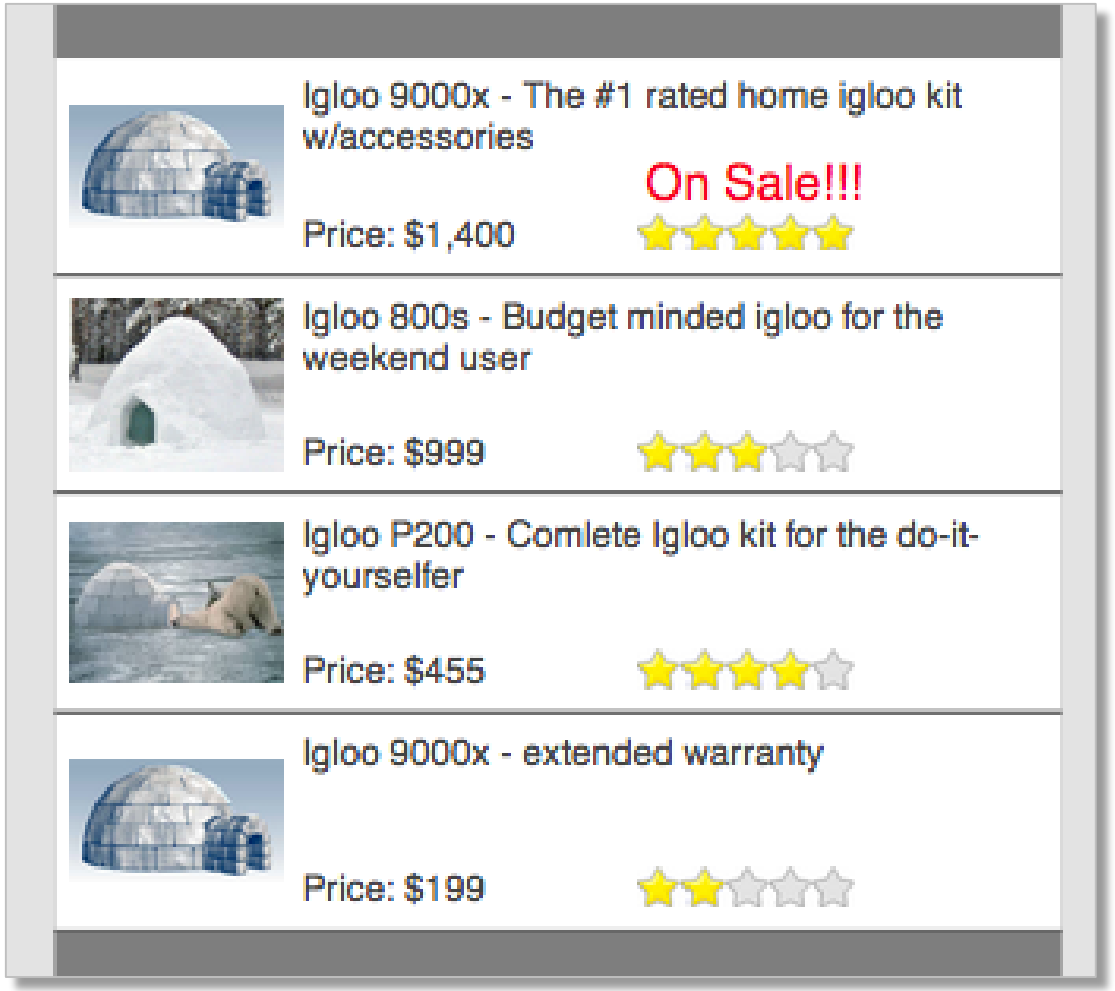
Like Flex, you can specify Free Form or Horizontal/Vertical

Like the Form or Flex, you can specify the grid for layout inside this widget

- You have the normal Look tab layout options to place the segment in your UI

Segment Layout - Exercise

- Let's try it by creating the following screen:
  - It's a segment with image on the left, 3 labels (name, price and on sale) and another image for the review
  - Skin everything to have the right font sizes and colors
    - Remember not to show any text, don't provide any master data - this is how we turn on/off the "On Sale!!!" text

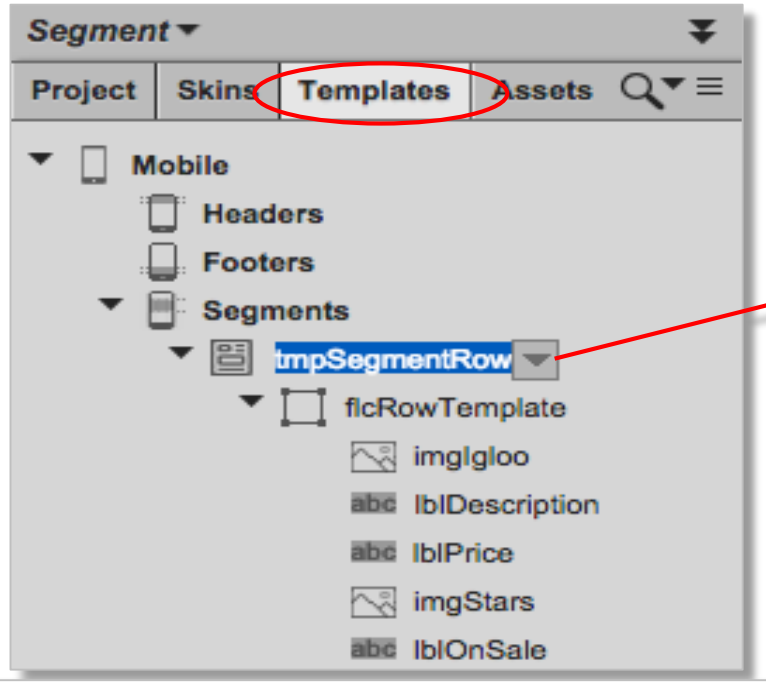


Segment Widget - Row Templates

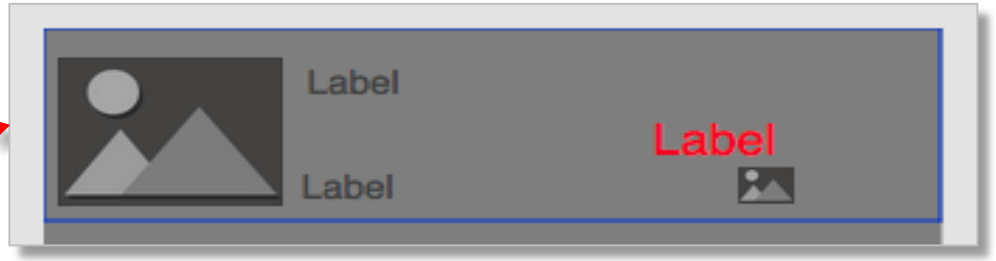
- In Visualizer, you're creating the UI elements and not necessarily the final business logic
- Row templates are VERY helpful when you have different displays depending on the data for a given row
  - In code, you can specify, on a row-by-row basis, which template to use and how the data maps to that template
  - The developer will implement the logic for using the right template
- In Visualizer, create all the row templates you want to use
  - Use a test form and test segment to test out each template to make sure it works
  - Note: you can only apply 1 template at a time in Visualizer

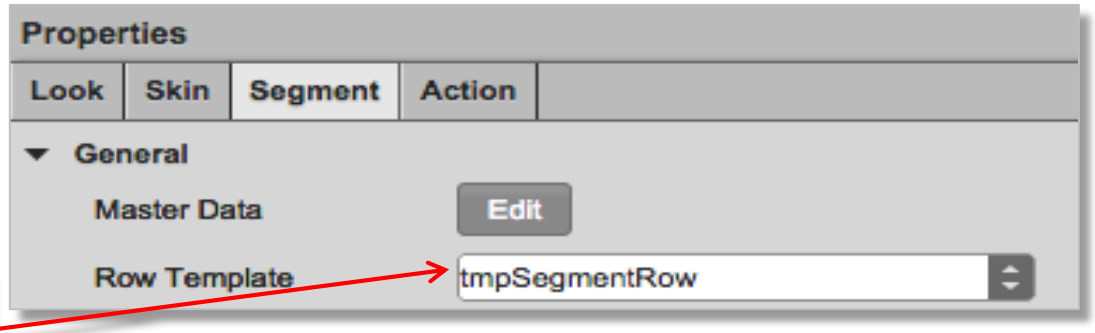
Segment Widget - Row Templates (cont'd)

- Rather than adding widgets to the segment, you can build a row template with your layout and refer to that:



Configure your template how you want each row to look:





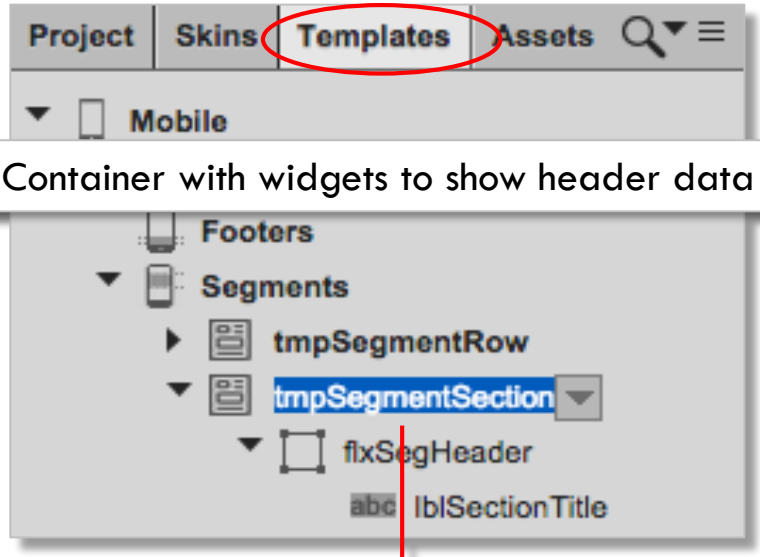
Now pick your row template and configure the data like before – in canvas or using Master Data

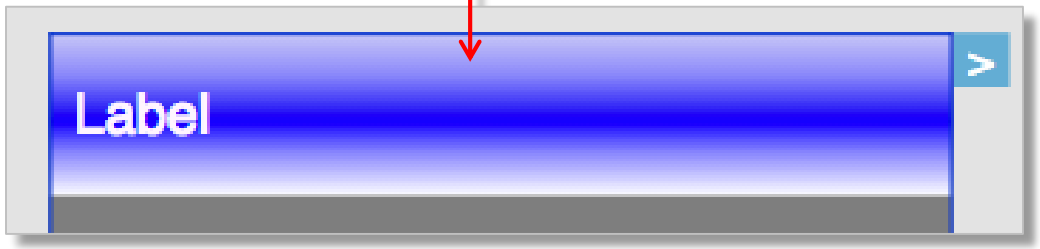
Segment Widget - Sections

- There are times when you want to create sections of rows:

Here are the section headers using a segment template

Template is a Flex Container with widgets to show header data

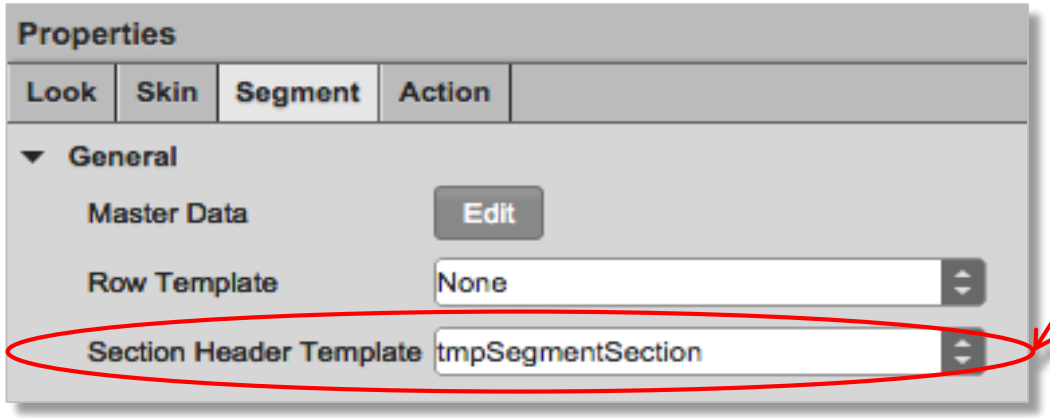




Note that you'll need to specify the header data and exactly which row data goes into which section

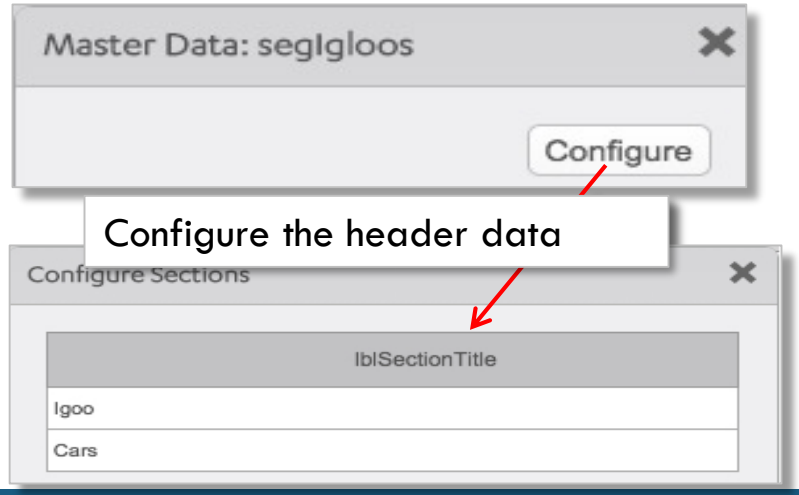
Using Segment Widget - Sections (cont'd)

- Sections are specified by first picking the section template:

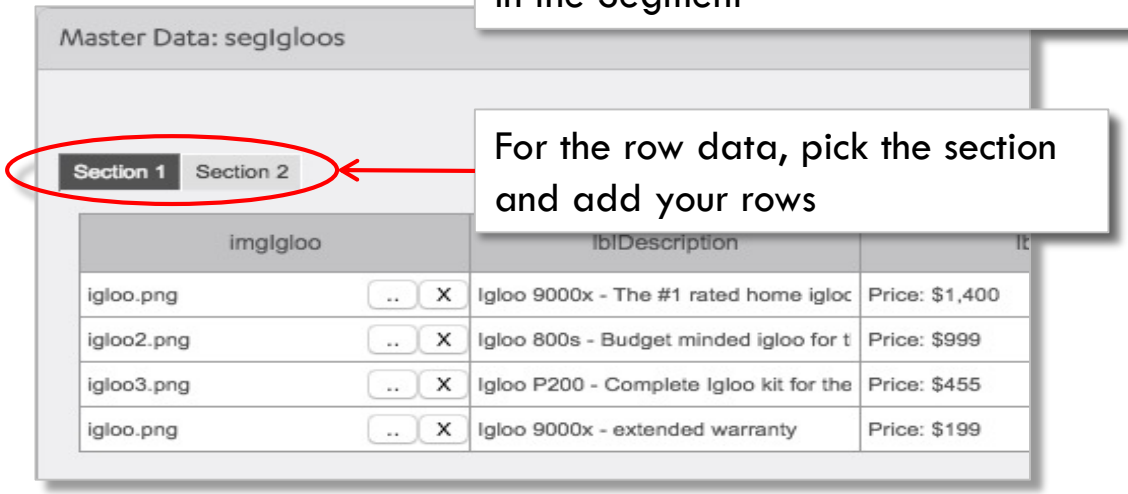


Pick the segment template you want to use as your section header

- Then, Configure the header data and row data accordingly:



Note: all the data is editable in canvas by double-clicking the widgets in the Segment



For the row data, pick the section and add your rows

Segment Widget - Other Properties

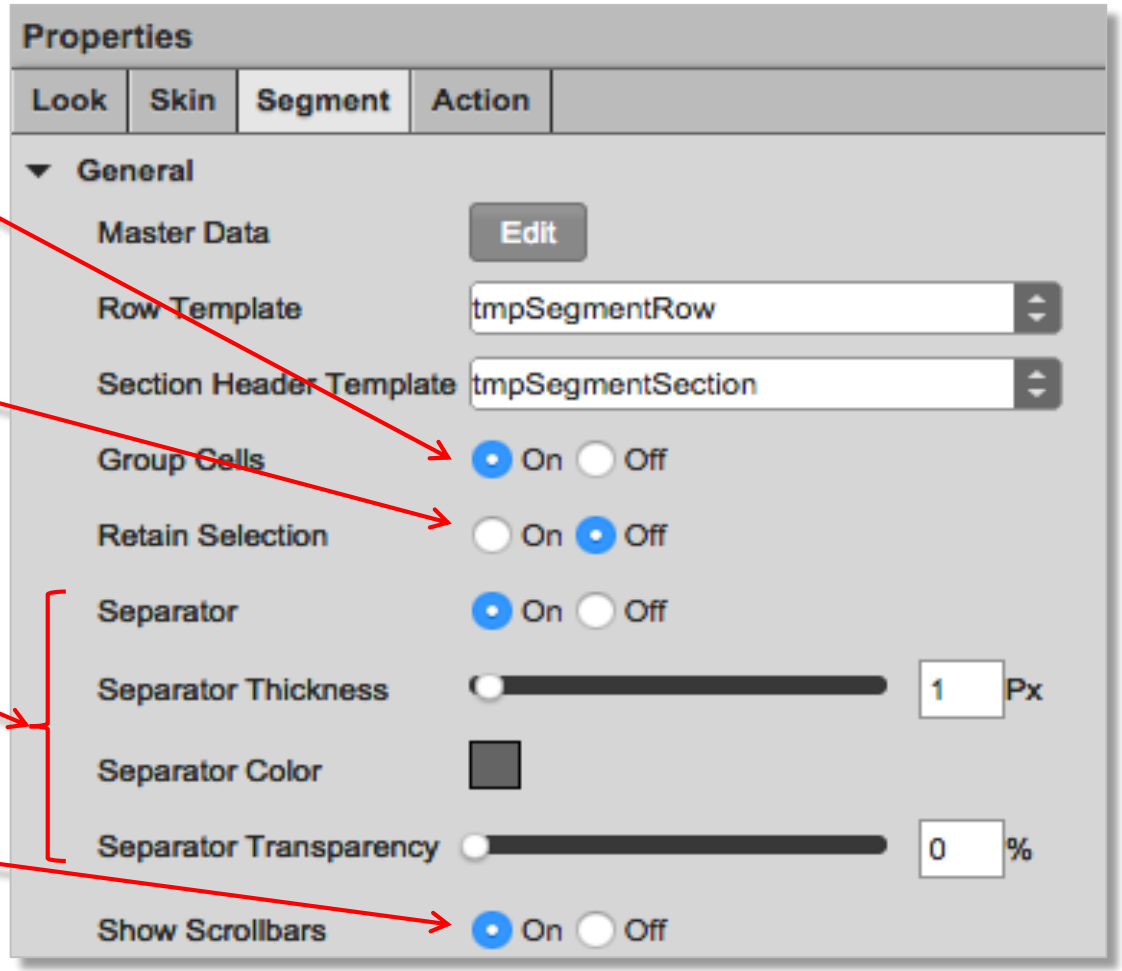
- Let's look at the other properties for configuring the Segment:

Applies a rounded corner treatment to the whole segment

If the user picks a value and you navigate to a new screen – this will keep that selected value selected when you come back

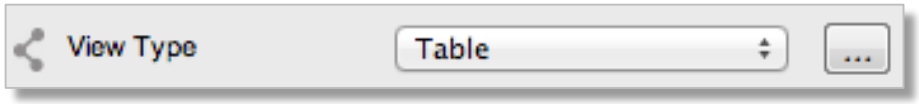
Configure the separator between the rows

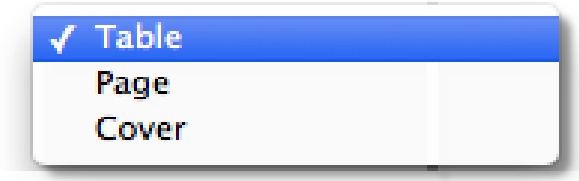
Turn Segment scrollbars on/off – it'll scroll anyway, this just turns the scrollbars on or off

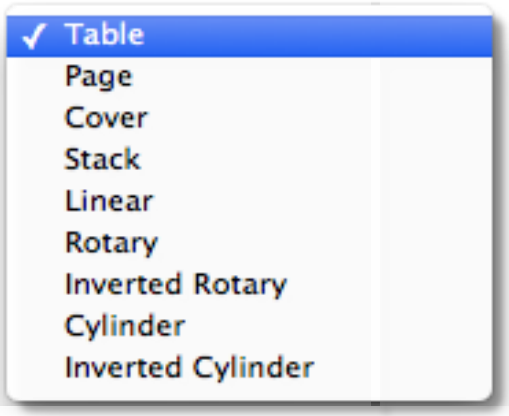


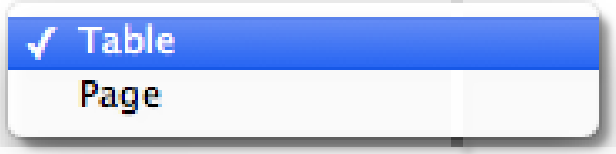
Segment Widget - View Type

- We'll now look at the different view types available...
- The default view for the Segment is "Table" - This shows all the rows vertically
  - When configuring this value, you have different choices depending on the device type
    - Here are your generic choices (applicable to all devices):
    - Here are your Android choices:
    - And, here are your iPhone choices:
    - First, let's look at Page and then, we'll see what those other ones are...



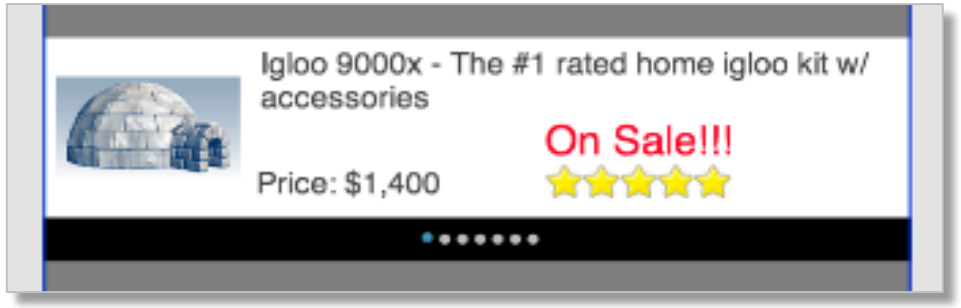






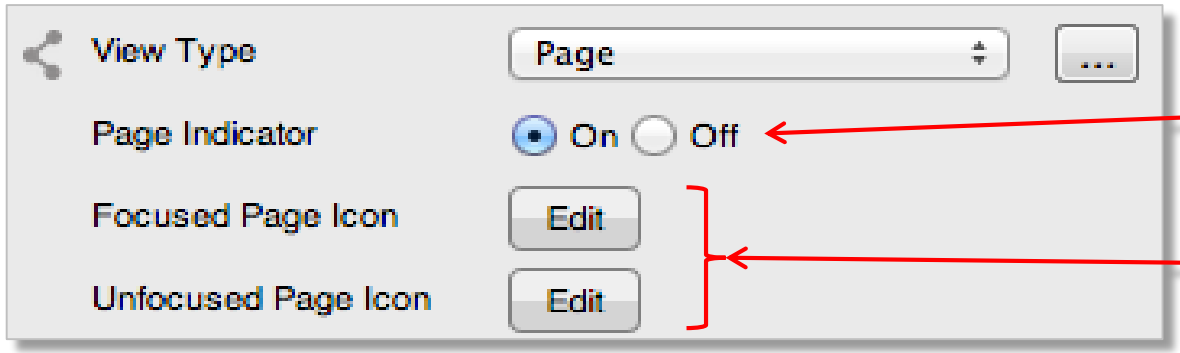
Segment Widget - Page View

- The Page view shows one "row" at a time
  - Scroll horizontally to move to the next/previous value
  - Use dot indicators to show what "row" you are on:
    - When you set the View Type to Page, you get more configuration options:
    - Note: Your layout changes since the Segment height is MUCH shorter in this mode



Swipe left/right to see the next/previous values

Note: sections are ignored and all rows are shown in order



Turn off the page indicator dots if you want to

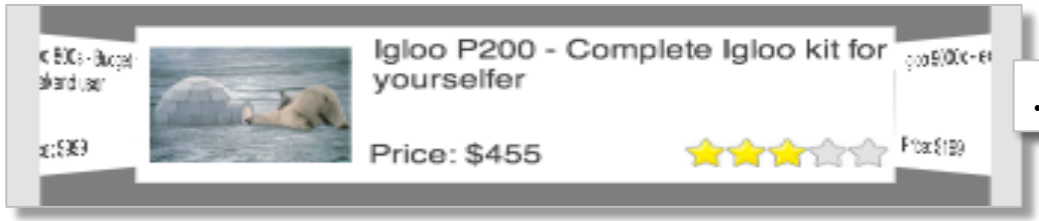
You can specify the dots if you want to use different images than the defaults

Segment Widget - Cover View

- The other views are for iPhone only except for the Cover view that works for iPhone and Android:
  - Cover - is for "Coverflow" which is an iPhone paradigm that shows the individual "rows" with animations to flip the row into view
    - In the canvas, the blue arrows let you simulate the user swipe action:

Click to move to the next one...

Here, we see it mid flip ready to come into view...



...and here's the next row

Segment Widget - Other Views

- All the other views are for iPhone only
- Depending on the view, you'll find that some look/work better than others depending on how the data and widgets are configured
- These views are mostly used for image based data – our last example with igloos doesn't work to well since there is lots of text:

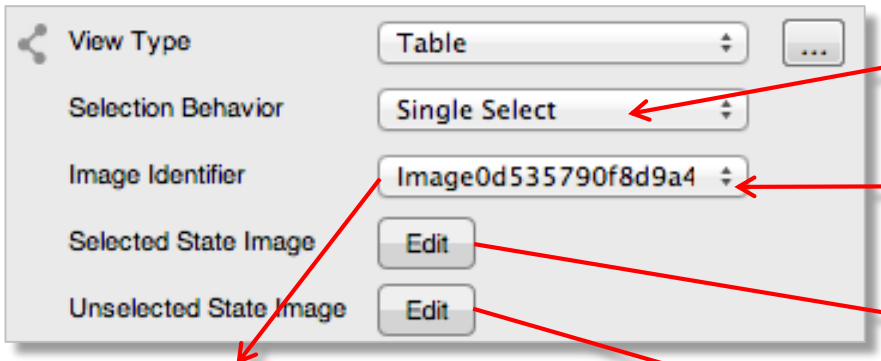
Shown here is the Stack view type that shows the album cover with the artist name above and the album name below

Segment Widget - Selection Options

- The whole point of the Segment widget is to not only show the data to the user, but let the user actually pick a row to do something
  - There are several options for the Selection Behavior property:
  - **Default** – this is where the whole row is clickable to select it with the expectation that the user will now see details or more info on that data
    - Used for picking a product to buy from a product search or picking a friend from the contact list to show his/her details
  - **Single Select** – this is where you are using the Segment to present a list of options to the user and the user will select one by using a visual indication (usually a checkbox)
    - In Single Select only one value can be picked
  - **Multi Select** – this is where you are using presenting a list of options and the user can pick more than one value – works like Single Select
    - Using a Select behavior means we have to provide that visual selection element…

Segment Widget - Single/Multi Select

- First, we must add an image widget to our segment - this will be used to indicate if that row is selected or not
  - I've created 2 images to use for my selected/unselected states
  - I've configured my new properties to use these images:



This works for both Single and Multi Select

Here we pick our new image widget on our Segment

Selected image...

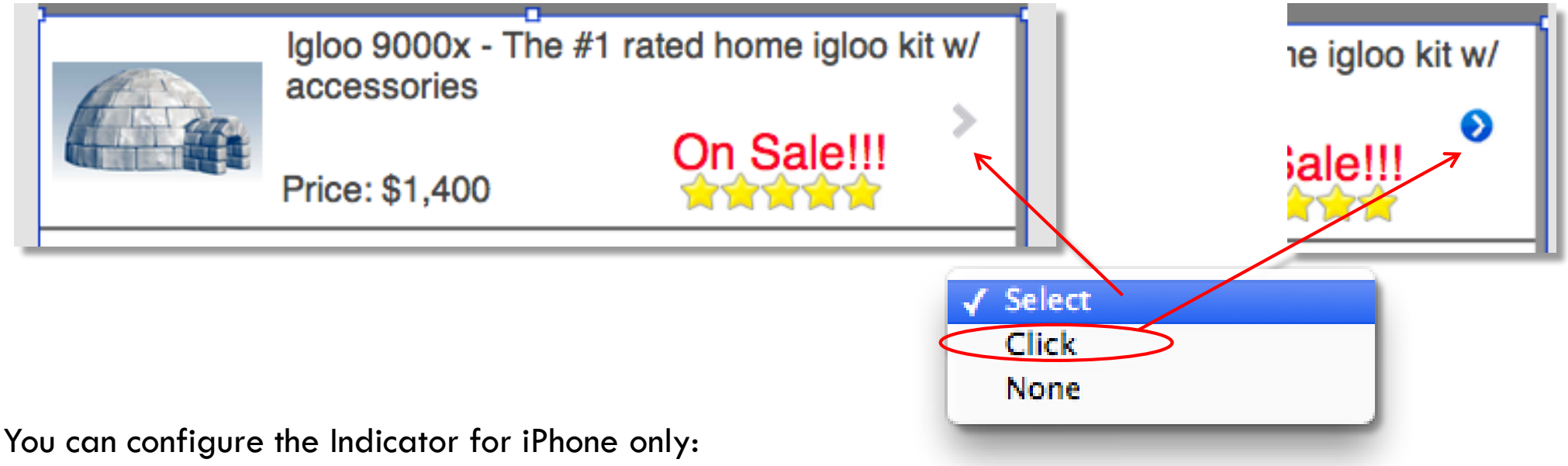
Unselected image...

Our new Image widget for selection:

Here is our segment with the first one unselected and the second row selected – click in the Canvas to change state

Segment Widget - Indicator

- iOS first started using the row indicator and that paradigm has caught on
  - This is most common in the Default view mode where the user expects to navigate to some more details about what they clicked:



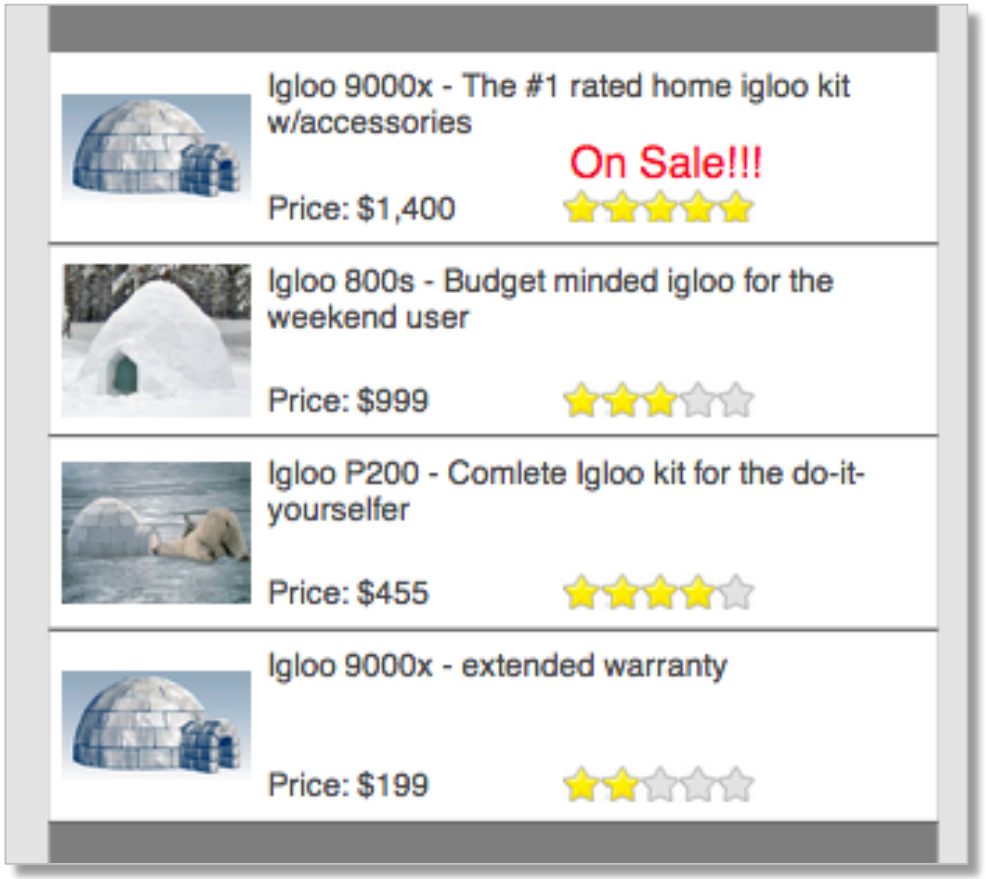
  - You can configure the Indicator for iPhone only:
  - What about other devices?
    - IF you want a selector, a better solution is to add an image widget and set it for ALL devices
    - Very common to not have a selector at all – even on iOS

Segment Views - Exercise

- This is a quick exercise to switch the view of our segment to test out the other views
  - Try the cover flow, page view, and other view options
  - For example:

Cover flow

Page View



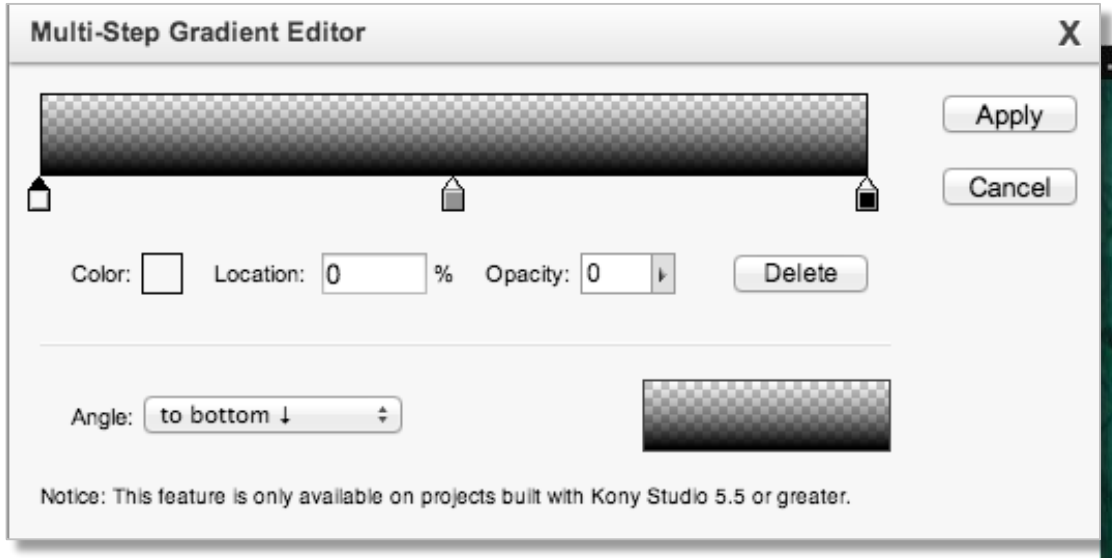
Design Techniques

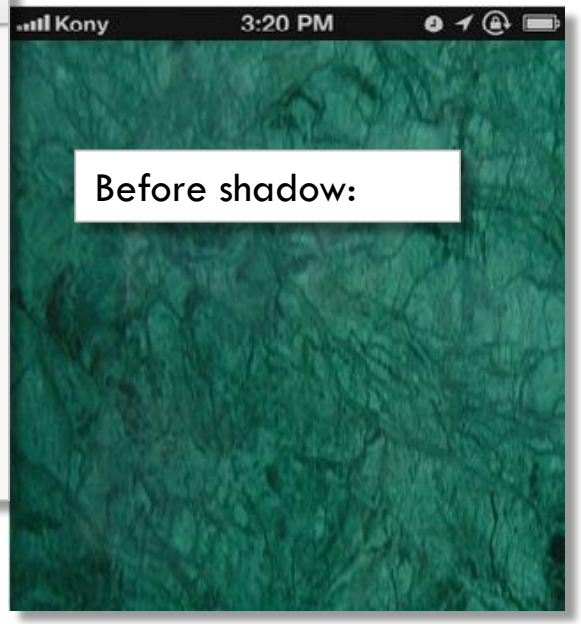
Design Techniques

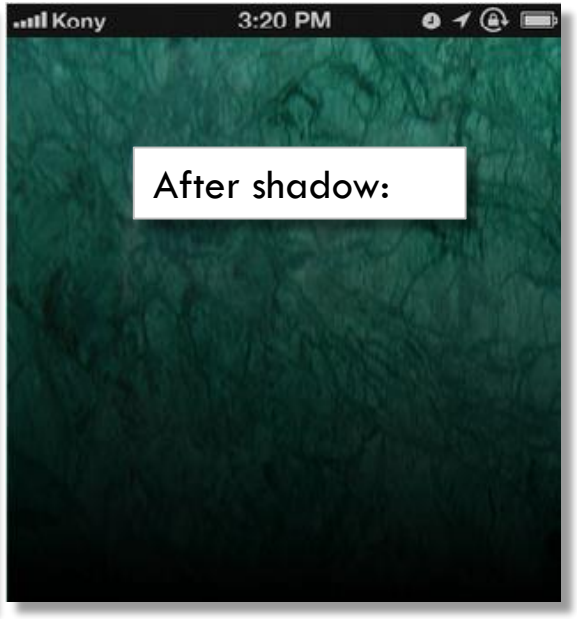
- In this module, we will cover common design techniques
  - While you might not use the exact technique, the idea is to show you the types of things you can do in Visualizer to create really great designs
- Here are some general things to keep in mind:
  - When designing, think of what you'll want to affect as a group – put all those widgets in a flex container
  - Use transparent buttons (Show Text set to off) to provide clickable targets over a background image
  - If you have 2 "states" and you have 2 pictures to show those 2 states – use one as static and then make the other one visible/invisible to toggle between them
    - Easier to implement than switching the picture for 1 image widget
    - Developer only needs to check visibility rather than the image name

Flex Container for Shadow Effects

- Applying shadow effects using Flex Container:
  - Add a Flex Container to cover the form
  - Configure it's skin as a multi-step gradient
  - Add the same form with the shadow flex container



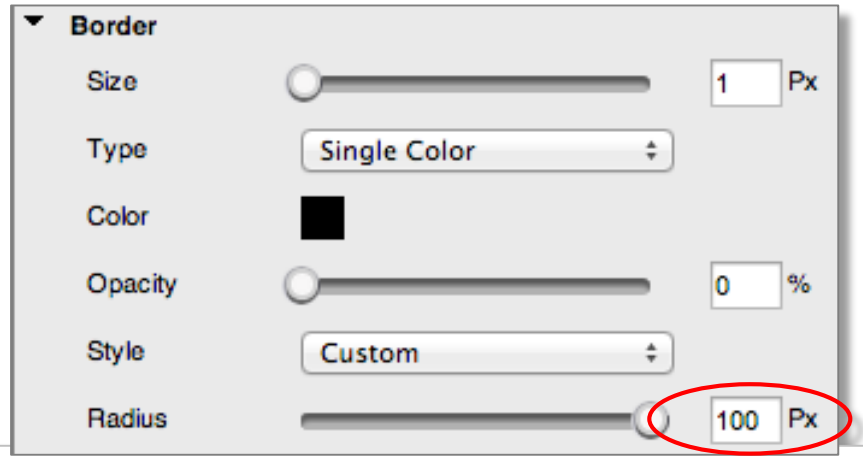




The gradient color is black with varying opacity settings

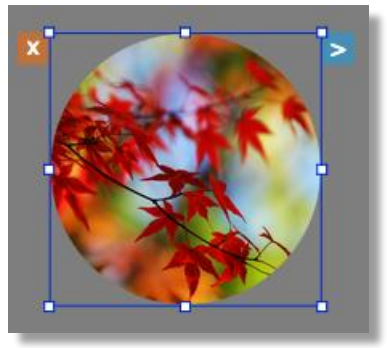
Circles

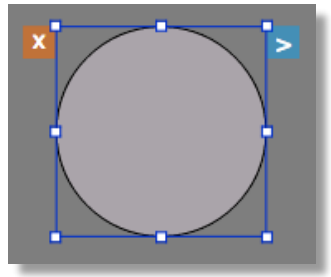
- There are many times when you want to have things in the shape of a circle
- Skinning allows for custom rounded corners:
  - For a circle, layout must specify Height = Width
  - Border configuration and final look below
  - Or just use an empty Flex to create just a circle



Note the max at 100px (Dp really) – this technique works for 200Dp or smaller widgets

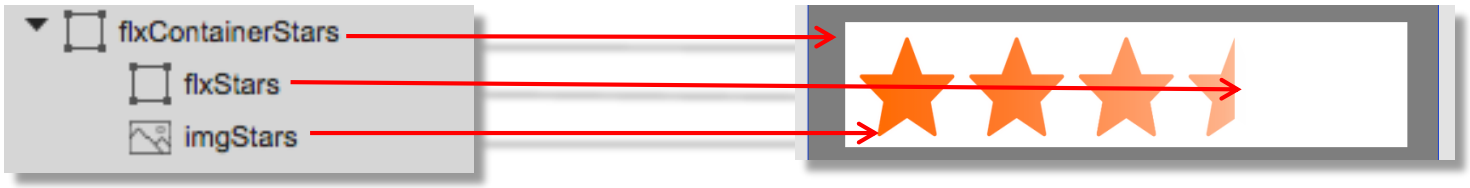
Original skin:





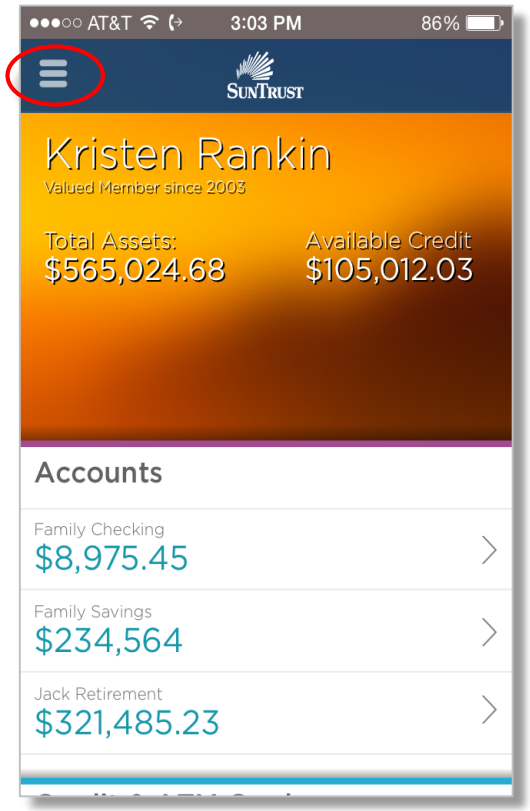
Animated or Static Progress Elements

- A common theme is to create visual indicators like:
  - Progress towards a goal or process
  - Average user reviews
- In each case, you'll want to be able to show different values without having a zillion images in your project to represent each level you need to show
- Answer – hollow shapes and flex!
  - In this example, the image is white with 5 star shaped "holes"
  - The orange flex container moves right/left to show more or less stars as it passes below the star "holes" to show through
  - Orange flex has a horizontal gradient – or you can just use an image that has the perfect edge effect
  - For progress or to show changes, just animate (move) the orange flex when the values change



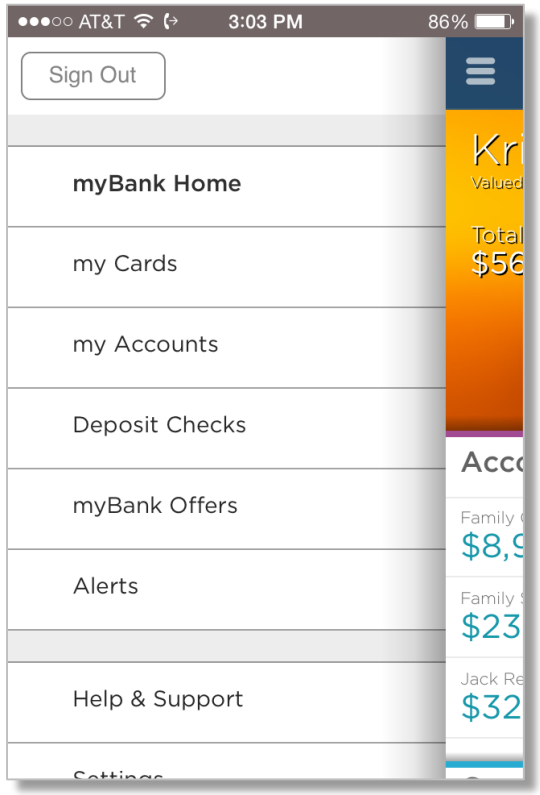
Creating a "Hamburger" Menu

- The “Hamburger" menu is pretty popular these days
  - It's those 3 bars you see in the header – click them to reveal a menu
  - Here's an example:



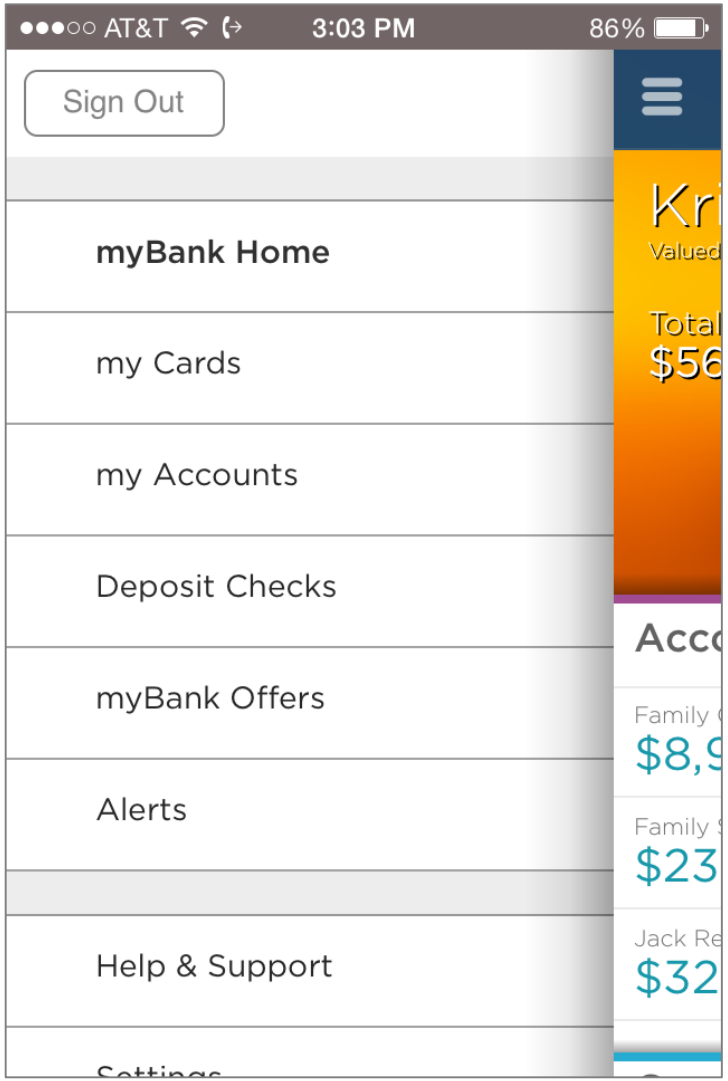
Click the 3 lines and the whole form slides to the right to reveal the menu below

Click the 3 lines again and it slides the form back to the left to the normal position



Creating a "Hamburger" Menu (cont'd)

- This is easy to set up:
  - The main contents of the form ALL on a flex container:
  - Clicking the 3 bars (transparent button over an image of 3 bars) simply animates the flex container to the right
  - That reveals another flex container, of lower Z index, that is stationary on the form
  - How to toggle the direction? Clicking the 3 bars again slides the form back – how do you know where you are?
    - Option 1: Use code and an IF statement
      - You can then just check either the left position of that flex container or set a variable to "remember"
    - Let's look at Option 2 on the next slide...



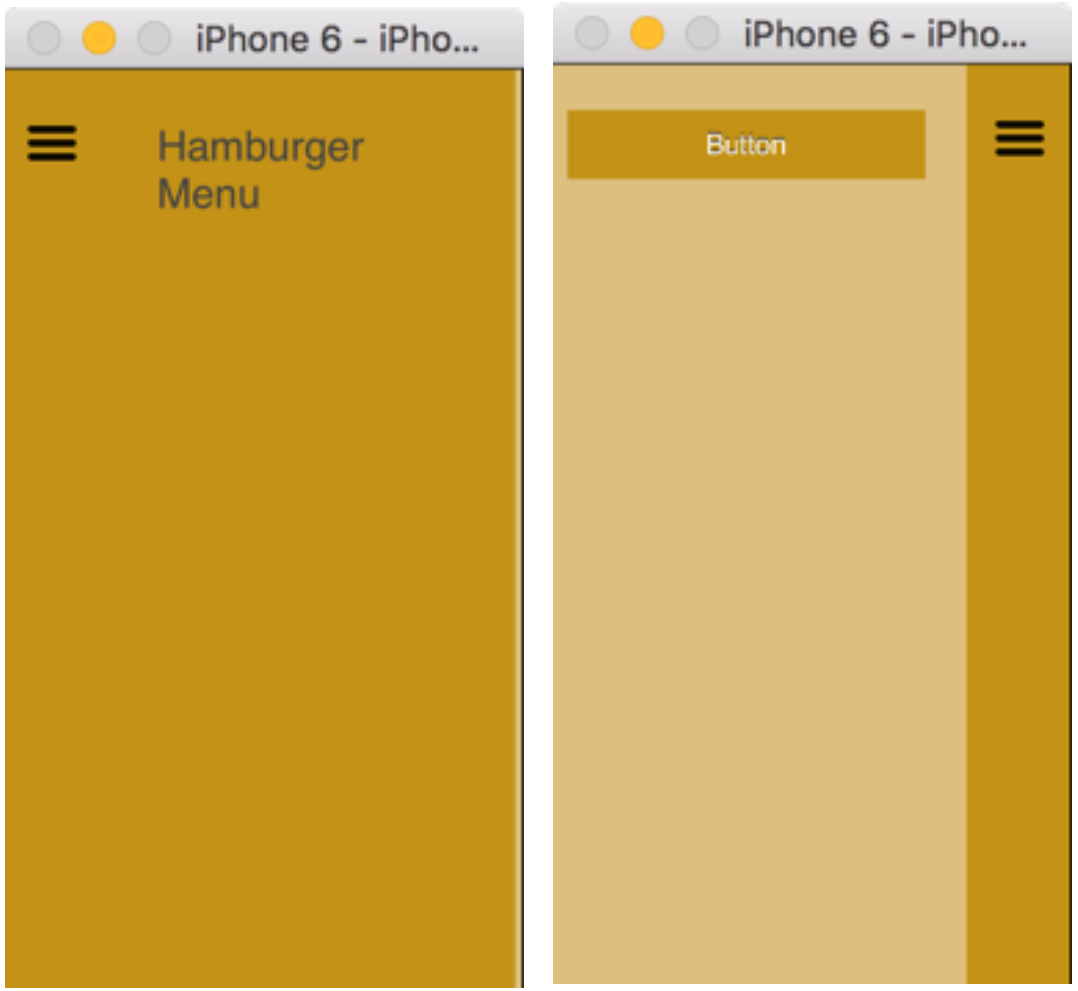
Toggling UI State

- Option 2 involves using 2 transparent buttons
- Here's how it works:
  - Both buttons are in the same place over the 3 bar images
  - The initial condition is set up so that button 1 is visible and button 2 is invisible
  - Create actions for each button click as follows:
    - Button 1 click action sequence:
      - Animate flex container to the right to reveal the menu
      - Make button 1 invisible (set visibility to false)
      - Make button 2 visible (set visibility to true)
    - Button 2 click action sequence:
      - Animate flex container to the left to cover the menu back up
      - Make button 1 visible
      - Make button 2 invisible
  - Invisible buttons don't work so you're just toggling which button is "active" and hence which action to take - no coding involved!

Exercise – Hamburger Menu

- Let's do an exercise to test out a hamburger menu
- There are a lot of little nuances in this exercise that I'll walk you through
  - Here's the app when you first run it:
  - When you click that hamburger menu it slides the top-level flex container over to show this:
  - When you click that hamburger menu again, it slides the top-level flex container back

- Simple, right?
- Give it a try! if you're not sure, the next few slides talk about how to do it...



Exercise – Hamburger Menu Steps

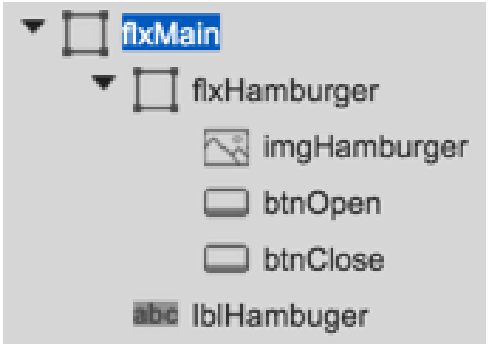
- The main part of the form:
  - Contains a flex container that will contain all the parts of our form, including our "hamburger menu", which is an image and 2 transparent buttons all on top of each other

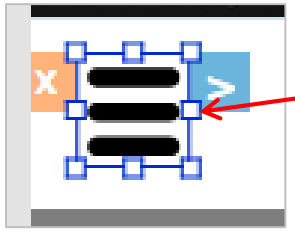
- All other items, such as the "Click Me!" button, are behind the top-level flex container and have a lower Z Index:
  - You can put the button (and anything else) anywhere on the form

- Don't forget
  - Make sure to set the Z Index of the top-level flex container BIGGER than the button and other widgets

Exercise – Hamburger Menu Steps

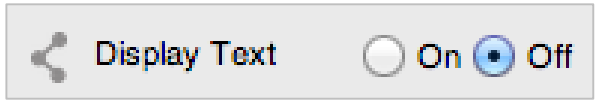
- The hamburger menu itself:
  - Use an image widget
  - Height/Width 40Dp

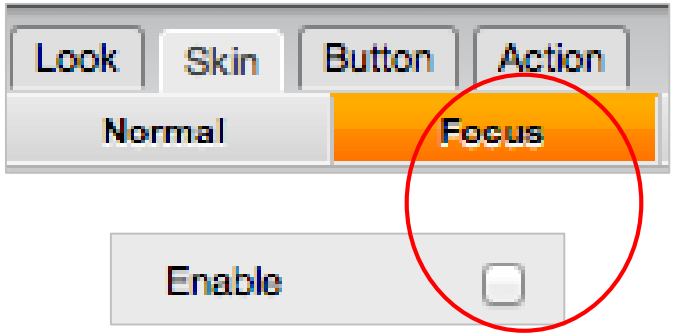




You can't see them because it's transparent – use hierarchy to select it

- Button 1 & Button 2 – used to open/close the menu:
  - Both (use copy) inside the flex container
  - height and width set to 40Dp (to match our hamburger menu size), set top and left to match the hamburger menu
  - Turn text off:
  - Turn skin opacity to 0% (make it transparent) and disable focus skin:
  - Set Button 2 to not be visible:

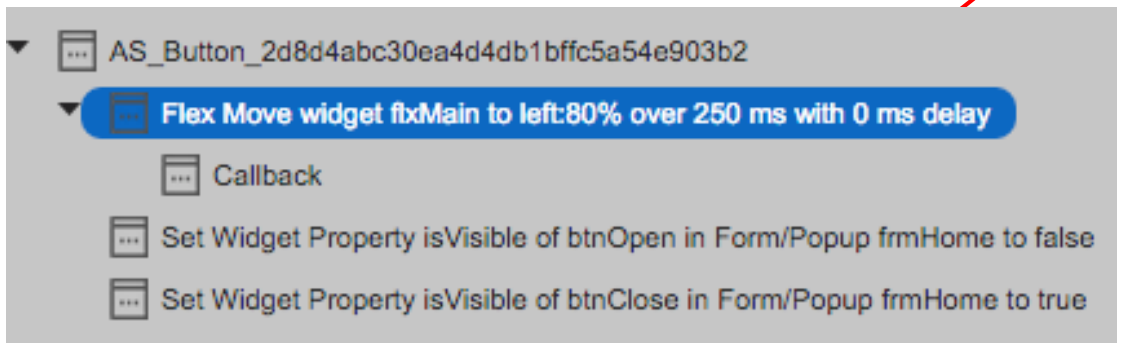


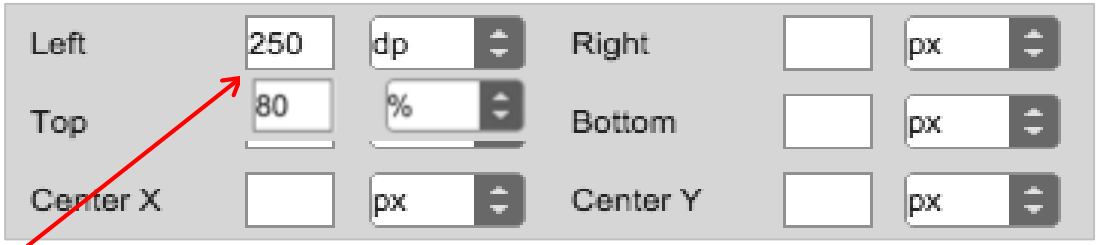


Exercise – Hamburger Menu Steps (cont'd)

- Button 1 animation:
  - The first button will "open" the menu – move the left to 80%
  - It will ALSO make itself invisible and make the OTHER button visible
    - Invisible widgets can't be clicked so we're effectively disabling this button
  - Tip: I renamed by buttons so I can easily tell what I'm doing – button 1 is called "btnOpen" and button 2 is called "btnClose"

Here are the 3 actions:



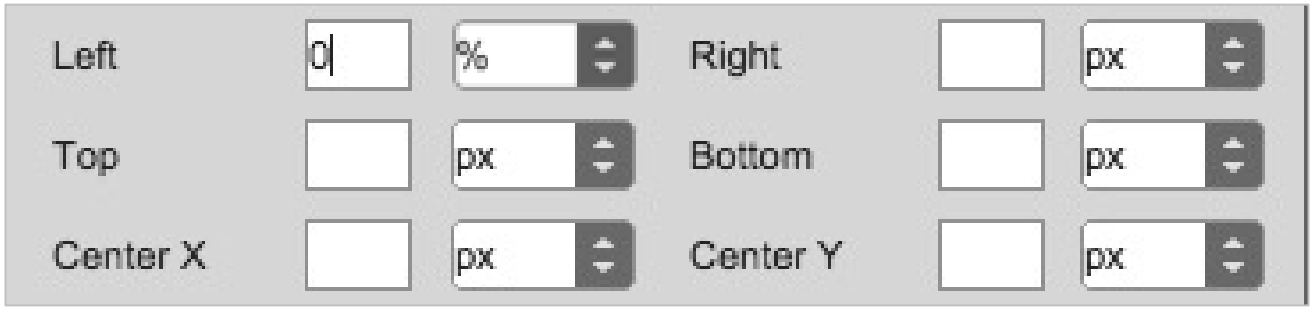


Set the flex container's left to 80% (final value)

Exercice – Hamburger Menu Steps (cont'd)

- Button 2 (btnClose) animation:
  - This second button will "close" the menu – move it back to the original spot – move it left to 0%
  - It will ALSO make itself invisible and make the OTHER button visible

Here are the 3 actions:

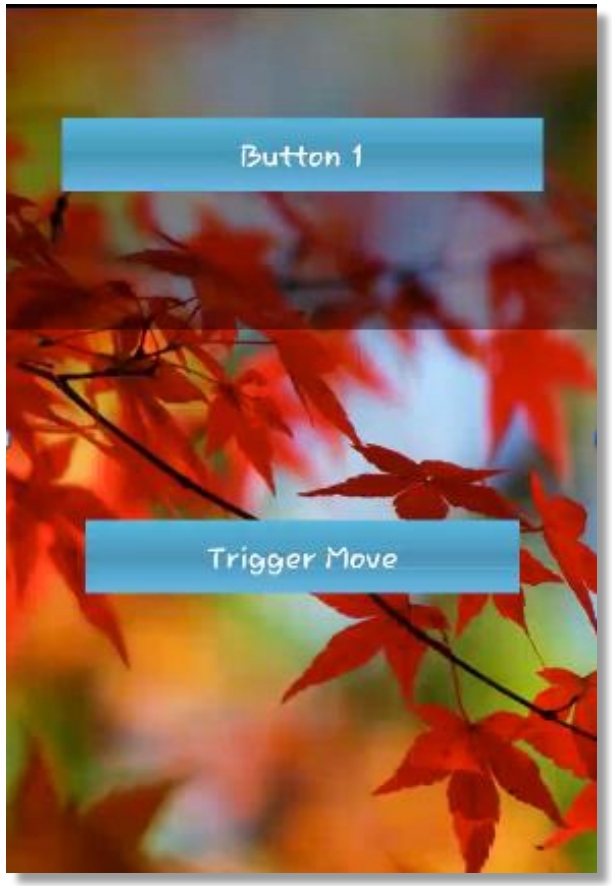


Set the flex container's left to 0% (original value)

- That's it! Now it's your turn to get it working

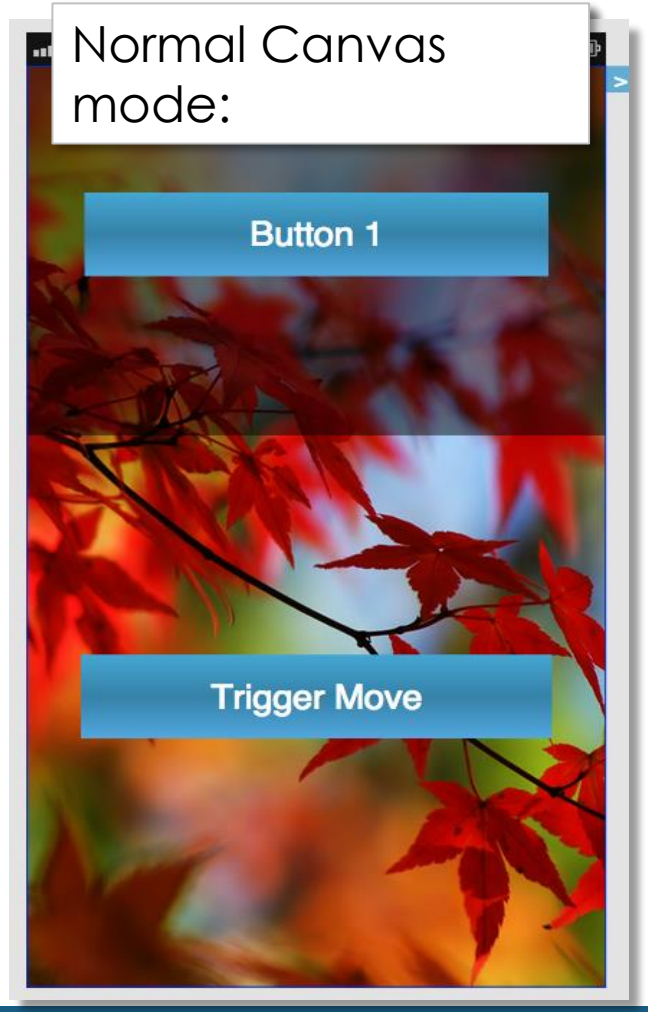
Parallax Effect

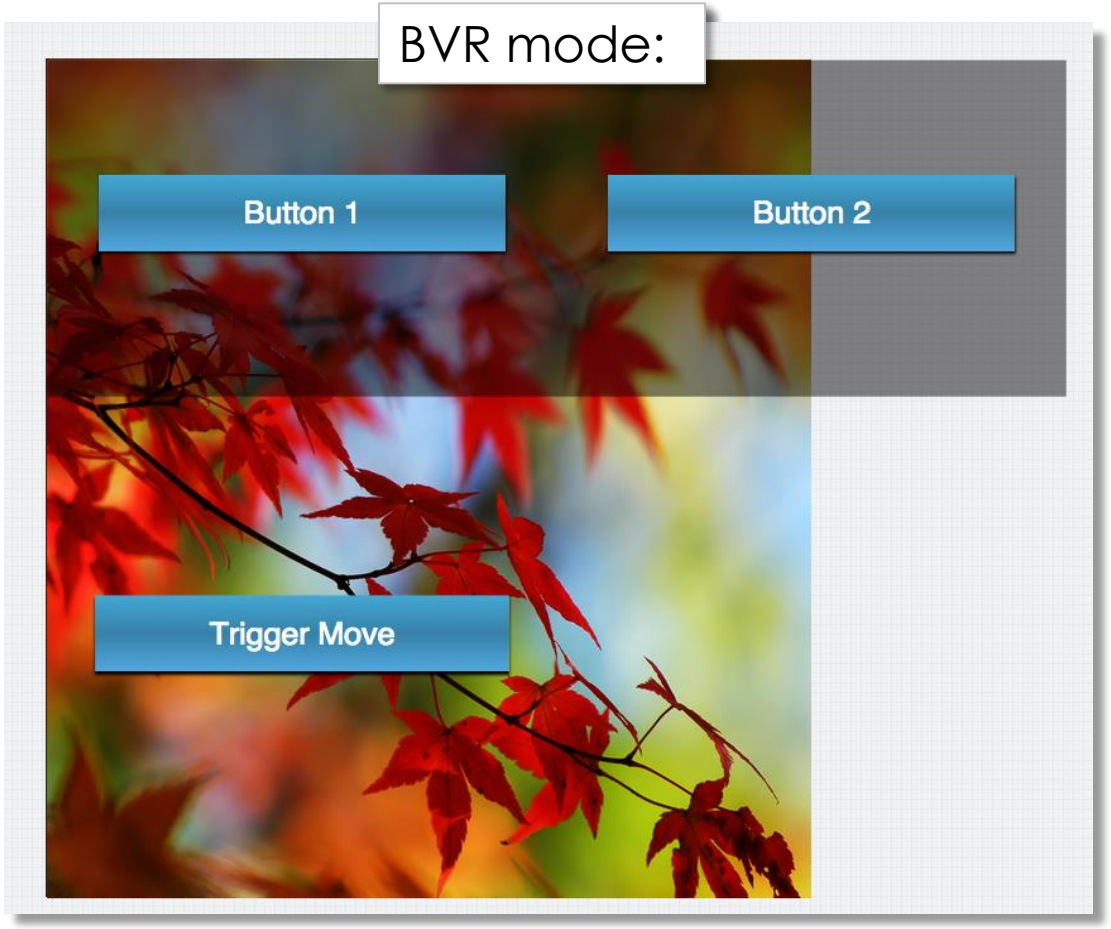
- A common effect is to use a parallax effect
  - Where the foreground moves at a different rate/distance than the background



Parallax Effect (cont'd)

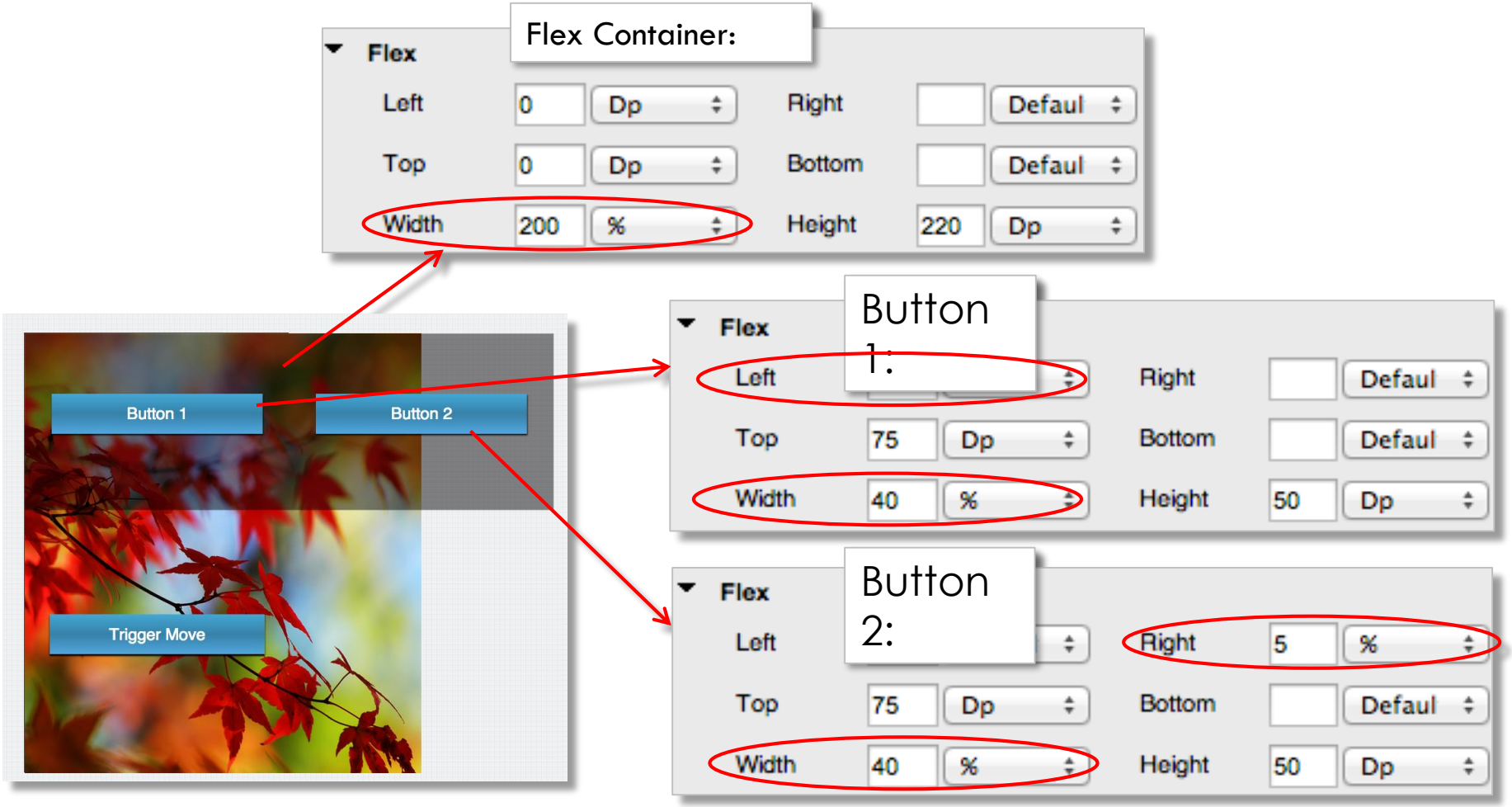
- Here is the canvas showing how this is set up:





Parallax Effect (cont'd)

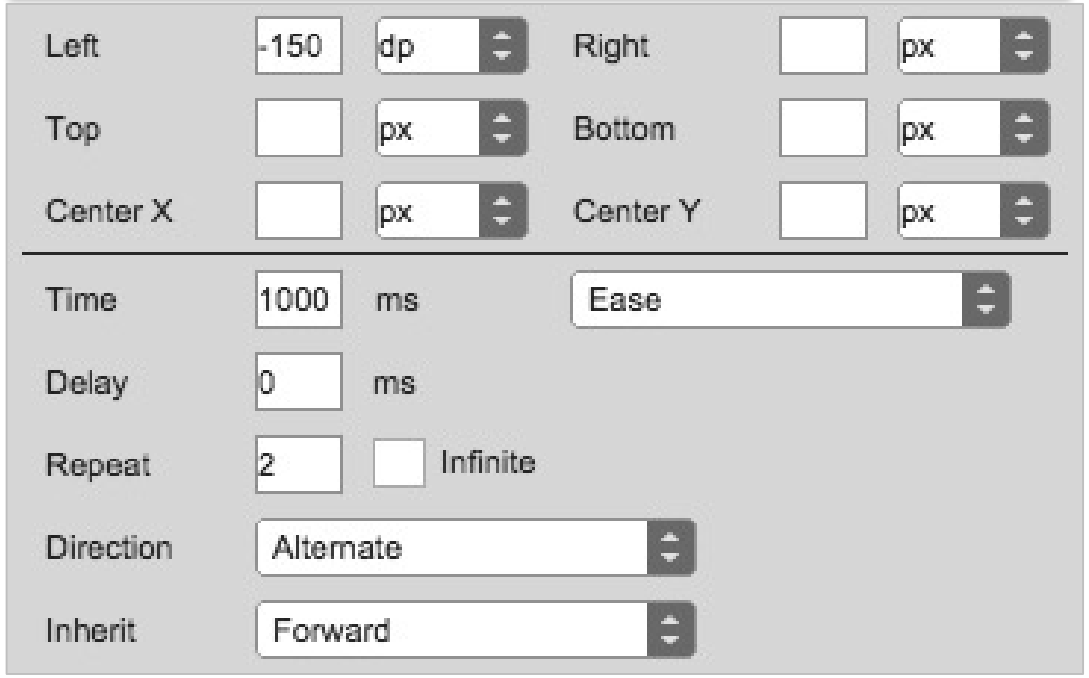
- Here is the Flex container, Button 1 and Button 2 layout configurations:



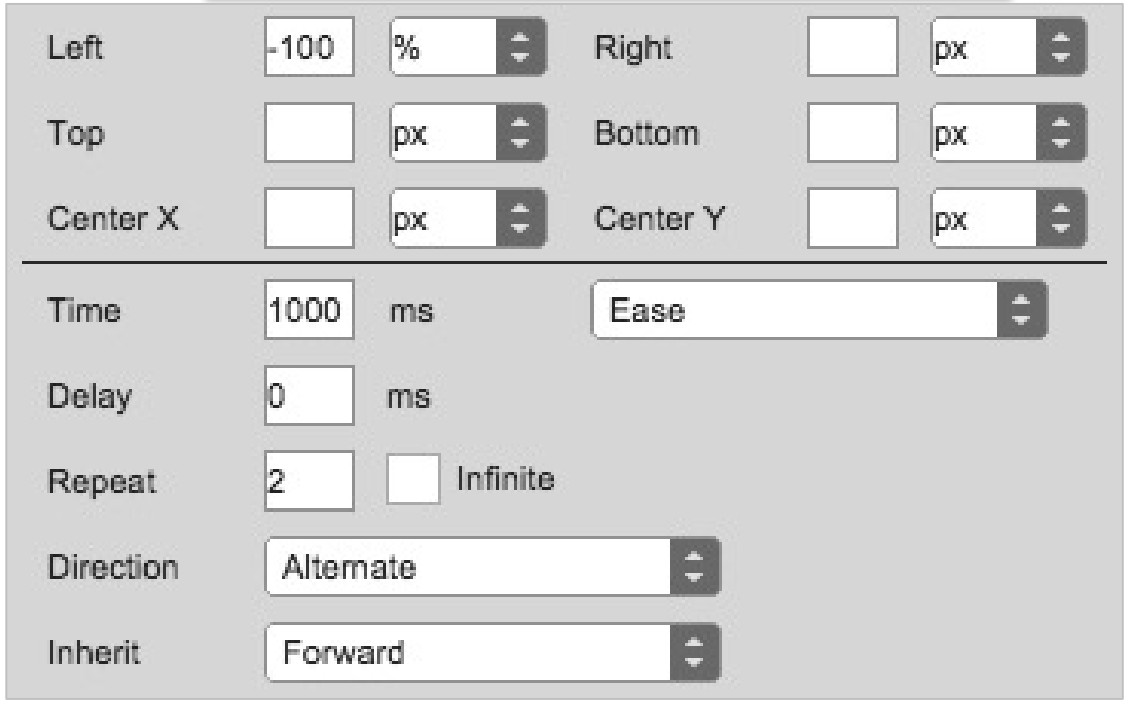
Parallax Effect (cont'd)

- Here are the animation actions:

Move the Flex Container with buttons to the right:

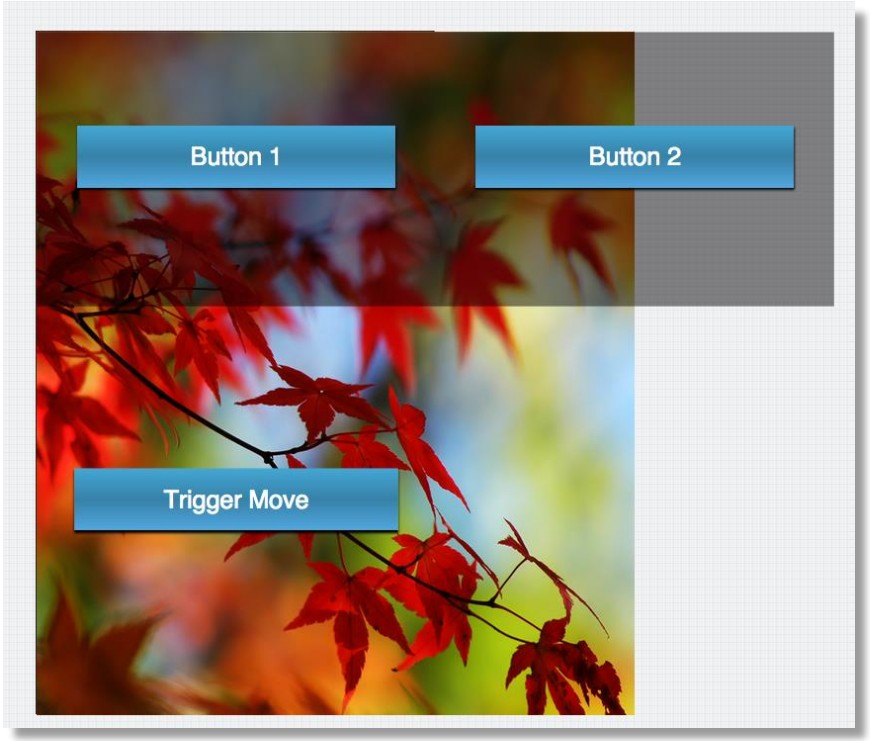


Move the background to the right:



Parallax Effect - Exercise

- Let's recreate that parallax effect
- You have all the information you need from the slides
- Remember the math:
  - Top flex is 200% the screen width
    - Each button is 40% total flex width (80% screen)
- If you set the animation Repeat to 2 and the Direction as Alternate, it'll return to the starting point so you can retest
  - Whenever testing animations, it's good to be able to re-test otherwise you have to preview it again



Outside the Design

Outside the Design

- So far, we've focused on using the tool to build up a UI
  - How to use the layout features
  - Some design tips and tricks
- Now, I want to focus on those things outside of the actual UI design that you'll need to know how to do with Visualizer:
  - Localization - Using other languages in your app
  - Libraries and Collections - Ways of saving your UI work for use on other projects

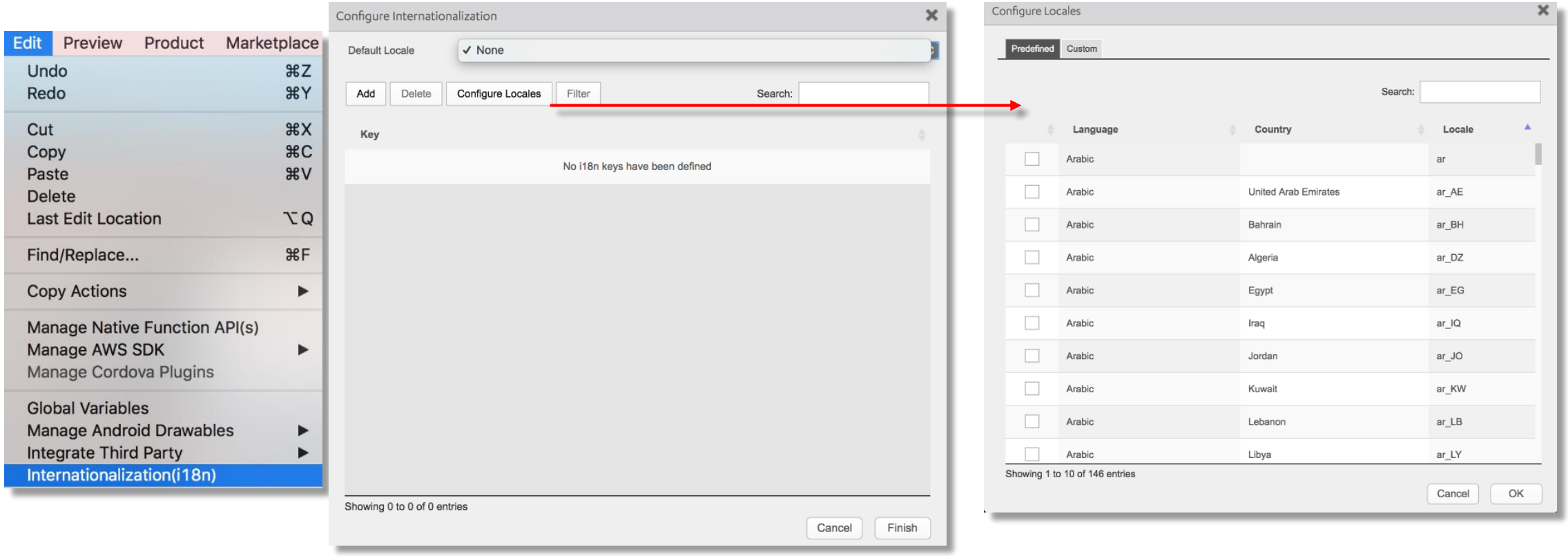
- Let's start with localization...

Localization/Internationalization

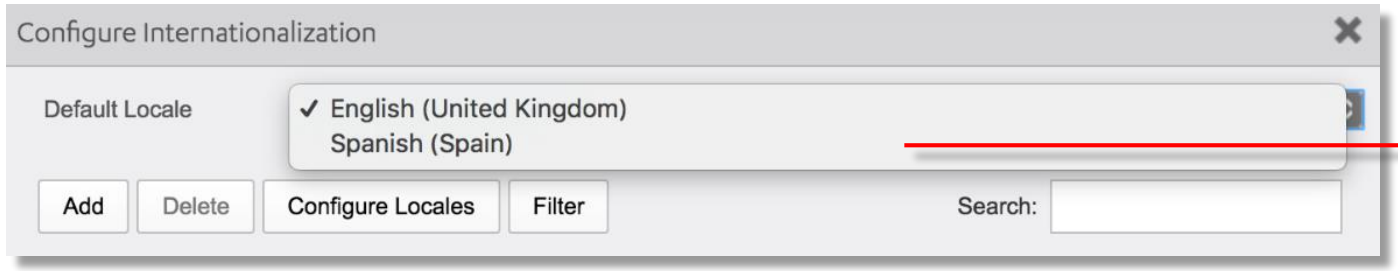
- One thing that any designer needs to be aware of is localizing the app to more than one language
  - Of course, text will change to be in the right language
  - Images MAY change as well if you need localized images
  - UI may change if other locale's have new text entry fields or different workflow
- For handling text, Visualizer supports localization so you can test what your app looks like in various languages
  - There is no "automatic" way to change images per locale or workflows per locale - that will have to be coded up which image/workflow to use where
  - Note: IF you have text embedded in your images, localization will require image changes. A better solution is to use no-text images and then superimpose text over the image (using label, button text, etc.,) so that the text can automatically handle localization

Configuring Localization

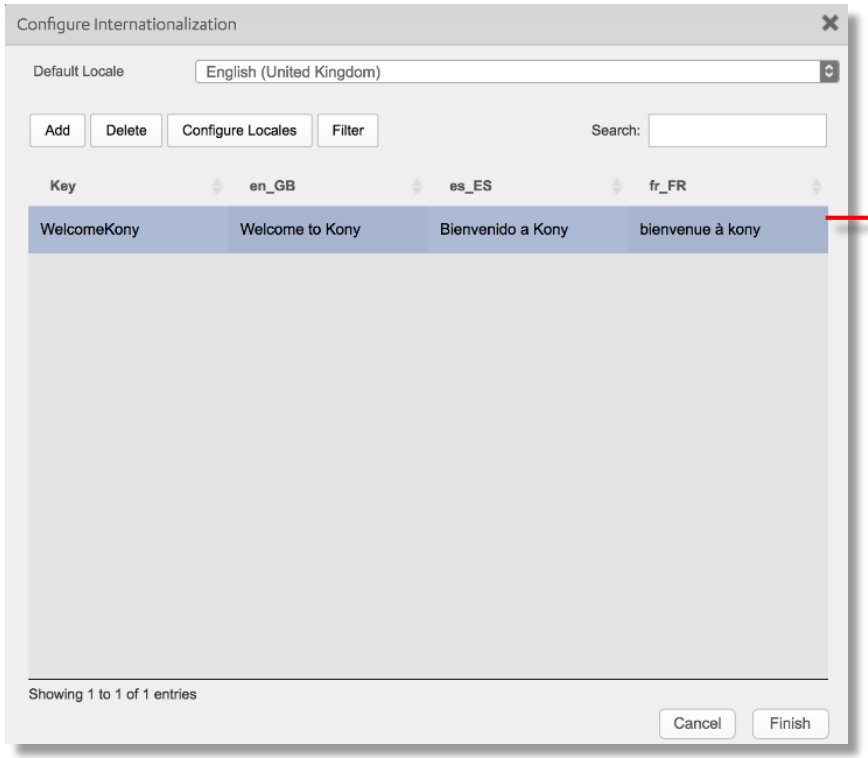
- Adding the required locales



Configuring Localization



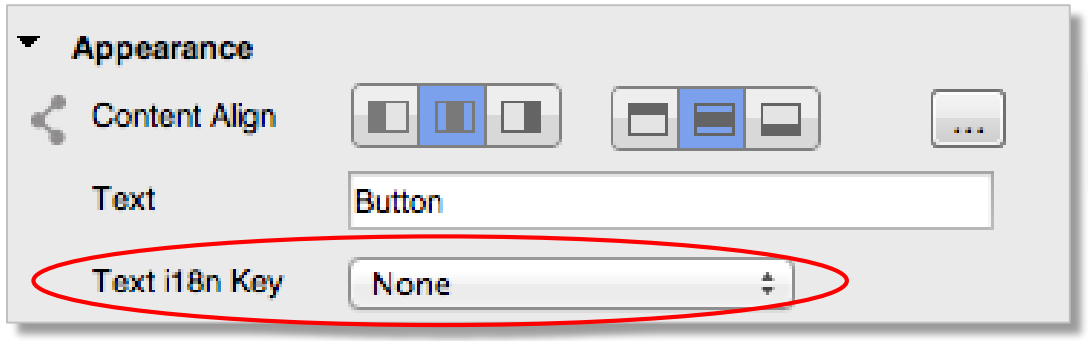
Default Locale can be changed using this window



Adding Key Value

How does Localization Work?

- Once you enable localization, now for all text widgets, you'll see a new property:
- Here's how it works:

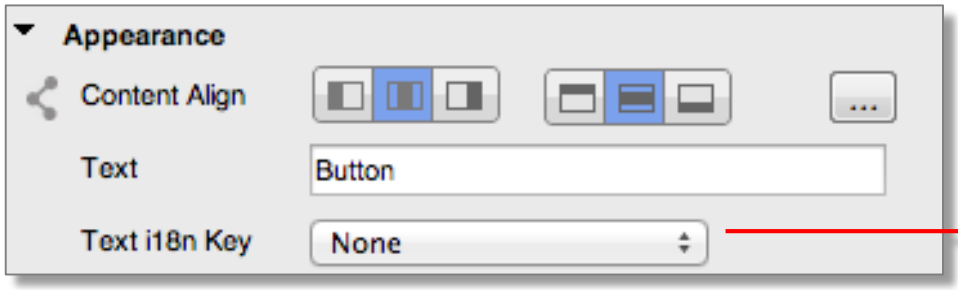


Using a Button for example

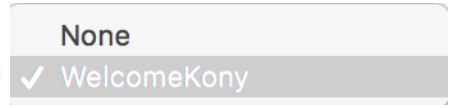
  - You'll first decide which languages you want to support
  - You'll create "keys" for all the text values in your whole app
  - Each key will have the appropriate text for each language
  - You'll then associate this key with each text widget as shown above – that dropdown will have ALL your keys so you'll pick the key that corresponds to the text you want to show in that widget
  - Let's go through an example so you can see how it works...

Using I18N Keys

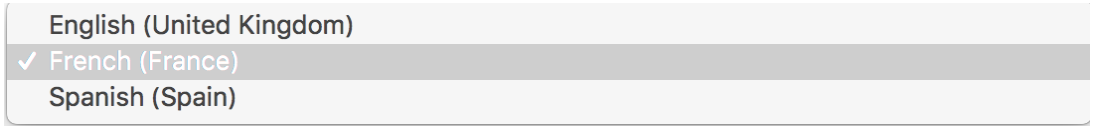
- For Button widget, we see the new I18N dropdown that we can use to pick the right key for this widget
  - It's possible 2 widgets share the same key IF they say the same thing
    - For example "OK" might be used in many places sharing the same key



This list will show all the keys you've configured – we only did the one, so we'll pick that one:

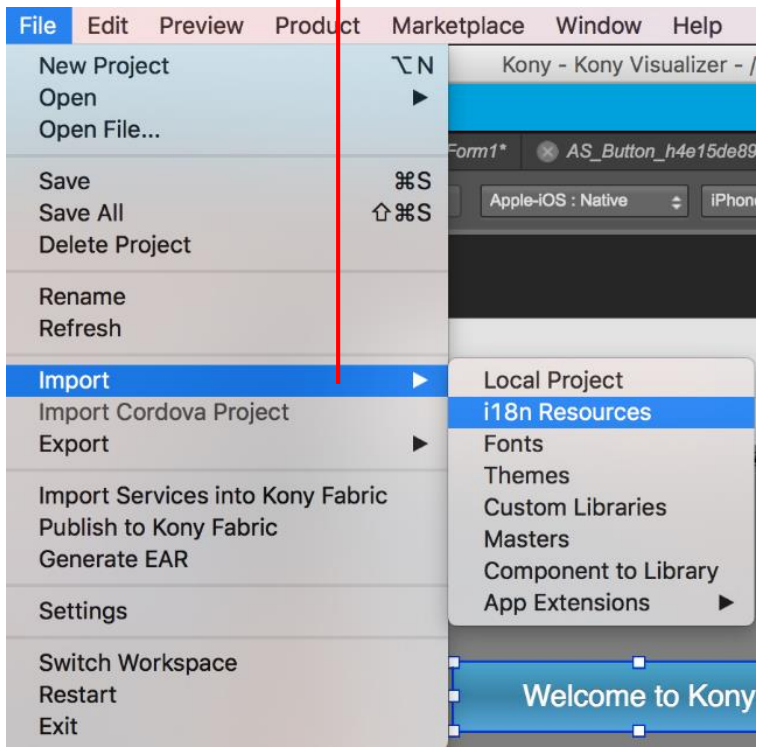


- Change the locale and observe that the value is changed

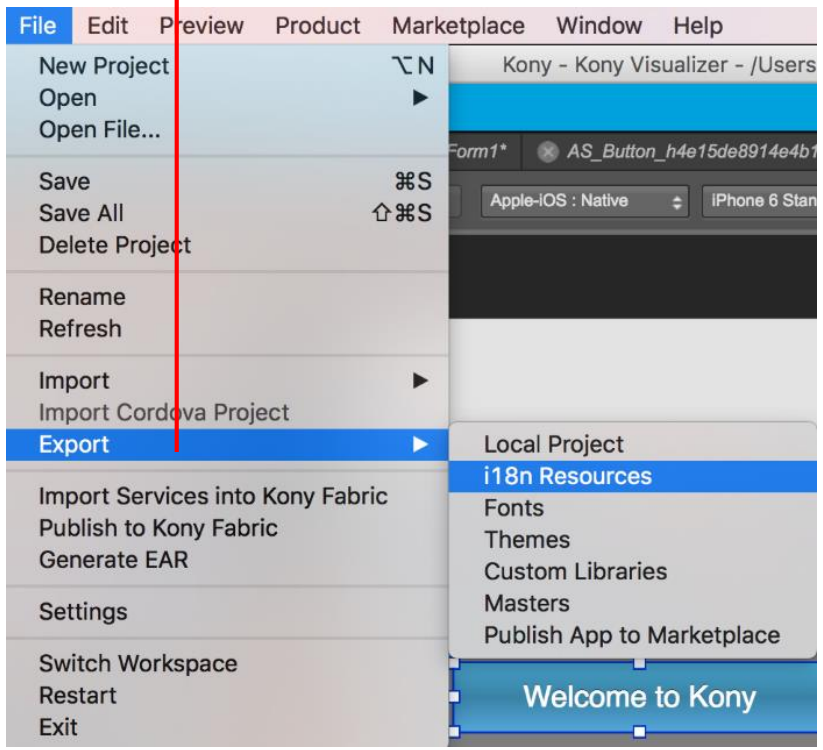


Import or Export i18n Keys

Import i18n

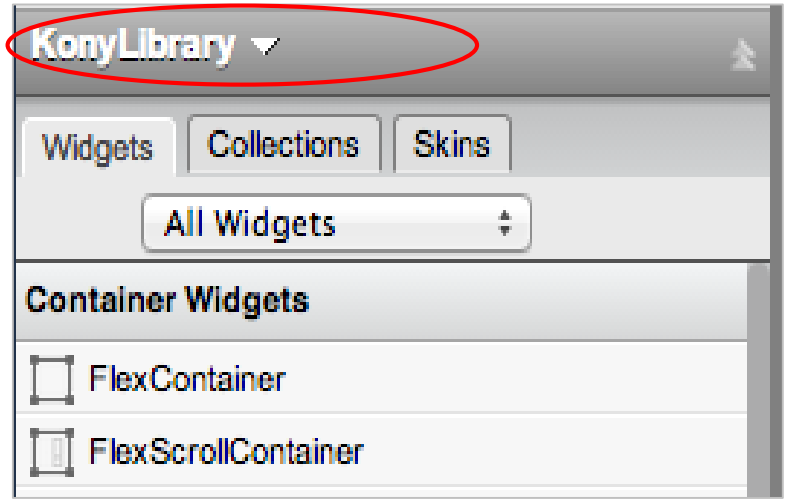


Export i18n



Libraries

- Visualizer supports the concept of libraries - ways of saving your work for future use in other projects
- By default, there is always a Kony Library that has Widgets, Collections and Skins
  - The Widgets are always shared by all libraries, but Collections and Skins are different
- What can I keep in libraries?
  - In Collections, you can keep UI elements that you can drag back onto a form in any project
  - In Skins, you can keep all your created Skins so that you can apply them to any project



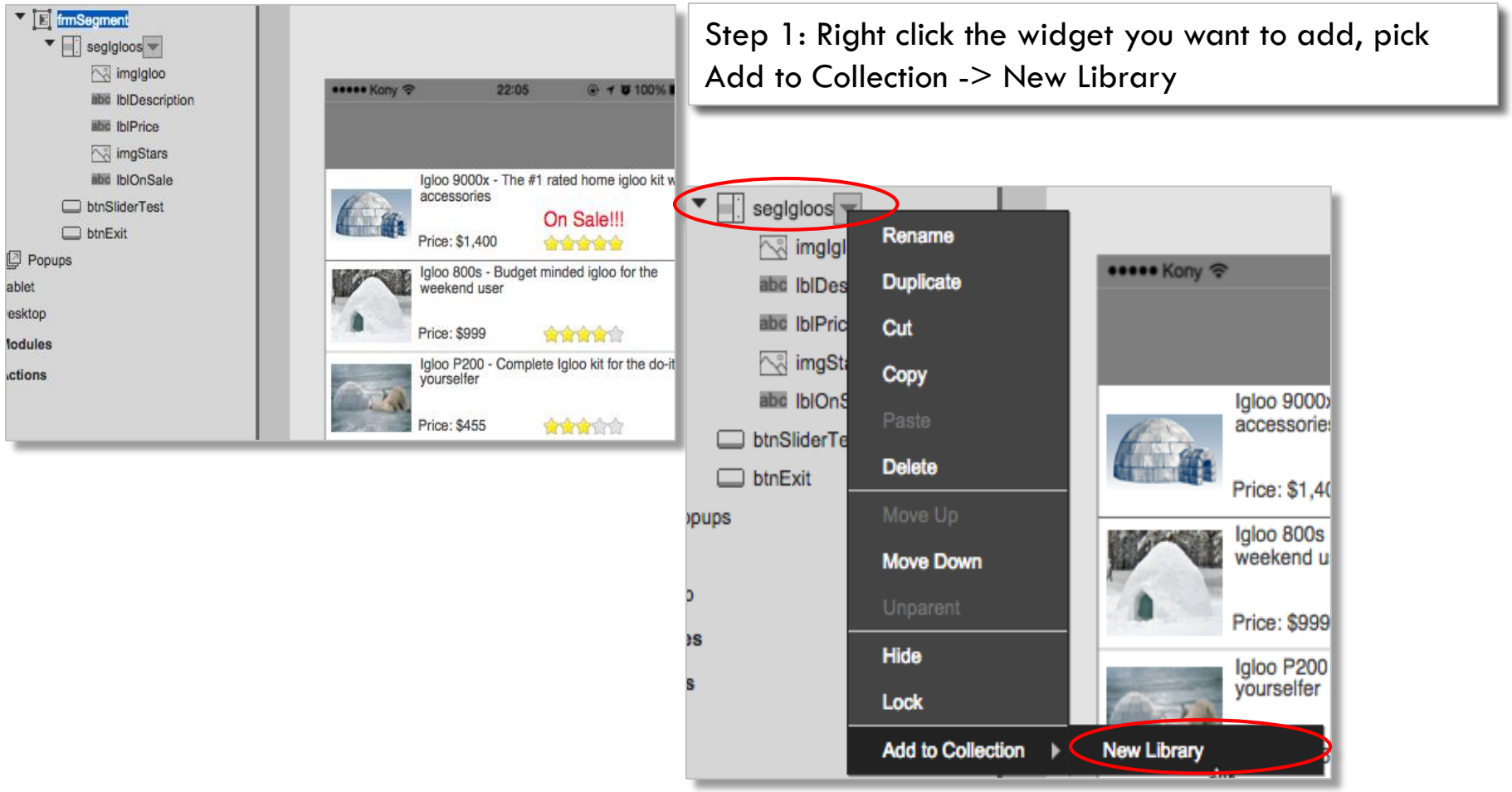
Libraries - Collections

Collections

- What's the best way to "work" with more than one widget at a time?
  - Put them in a flex container so you can move, hide/show and manage all the widgets inside at once
  - You'll have LOTS of flex containers in your app - you'll have one for each piece of the UI you want to affect as a unit
    - Can be for skinning purposes: To make a block of the UI stand out to the user
    - Can be for animation purposes: Animate the container and all the widgets inside go along – WAY easier than animating each widget
    - Can be for some UI paradigm you want to re-use: You have the username/login UI perfected and need to use it in more than one app so you want to save it all as one unit
- Whenever you want, you can take a widget (container or individual widget) and save it into a collection for future use

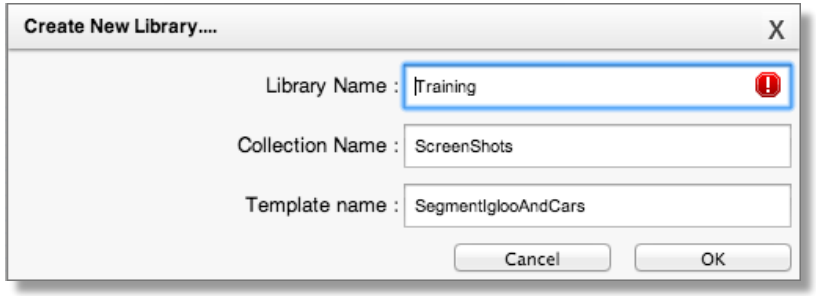
Creating a New Collection

- Here is a Segment that I want to save for use in other projects or even to use again for this project:

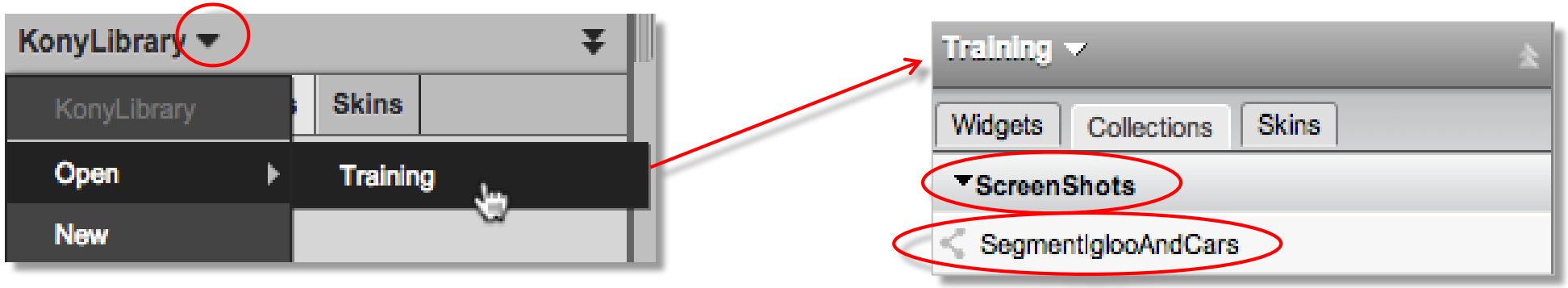


Creating a New Collection

- We can create a new Library
  - Libraries have one or more Collections
  - Collections have one or more templates - that widget you saved to the collection



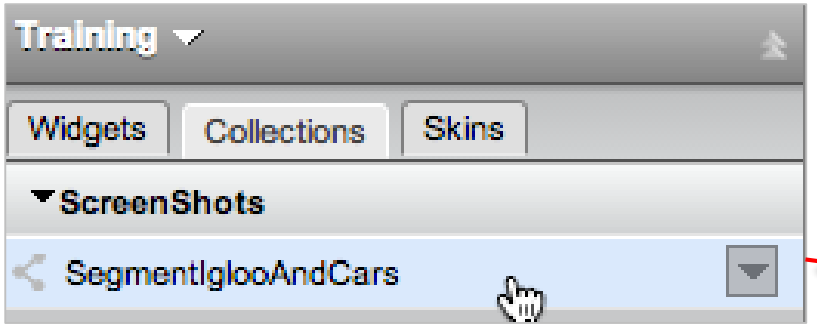
- Now in my Library section of Visualizer, I can switch over to my new library:



  - In the above image, I see my collection (ScreenShots) and my saved Segment (SegmentIglooAndCars)

Using a Collection

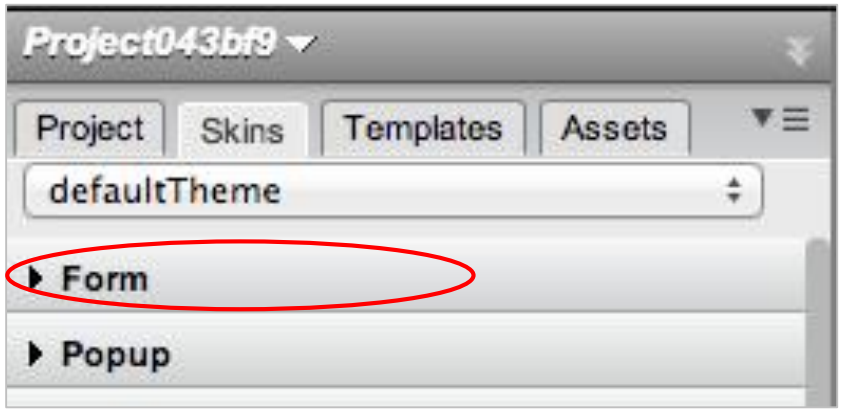
- How do you use that new item in a new form?
  - Just drag and drop like any widget!



- Some things to note about these collection items:
  - Actions do NOT get copied over
  - Dragging onto a new project ALSO copies over all the necessary skin and image resources that were used in the template
  - Best Practice: Save any key work elements as a safety as there is no Undo if you mess it up
    - TIP: Right-click any widget and choose Lock to avoid accidental changes

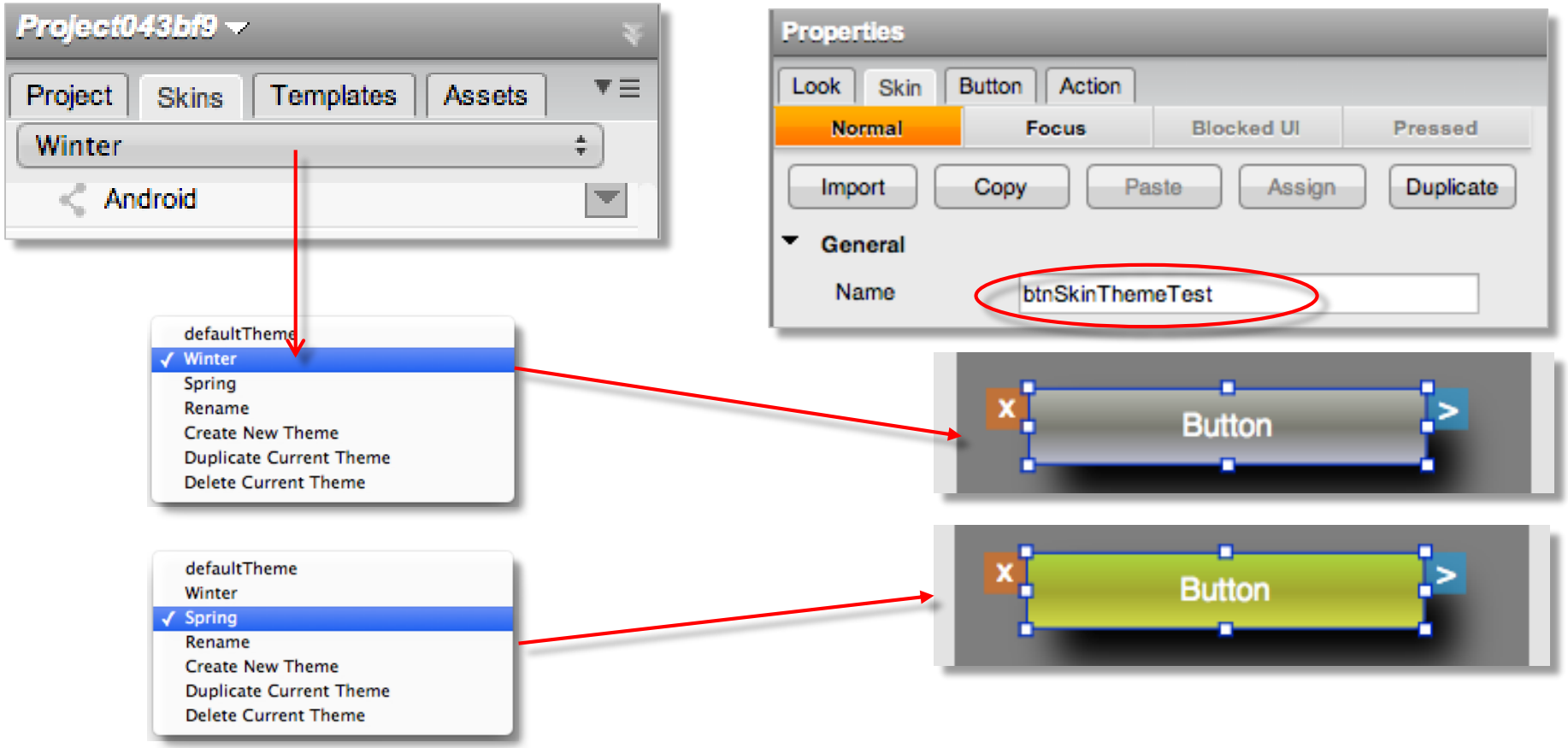
Themes

- Save skins to your library, so you can use them in other projects
- Skins are organized by widget but they are also defined as part of a Theme
  - By default, all skins use the defaultTheme
  - Theme is a collection of skin definitions
  - All skins belong to all themes
    - Each theme may have a different skin configuration
  - Themes are applied by picking the appropriate one and all those skin configurations will be used
    - You can switch themes at runtime in code (developer performs this)



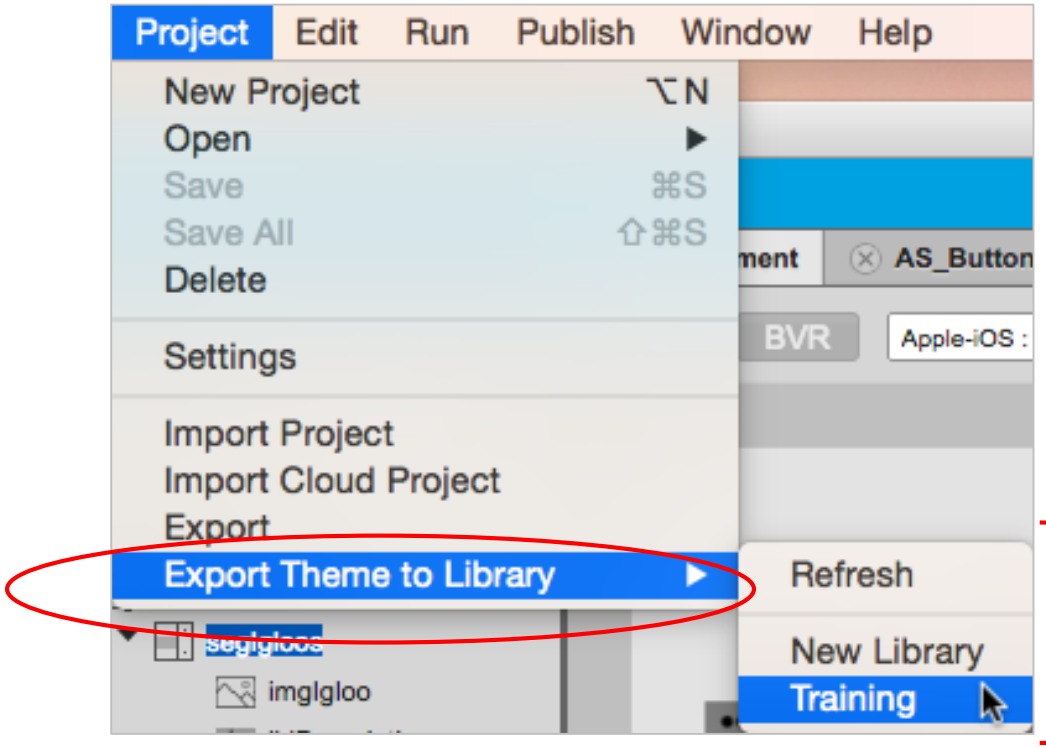
Theme - Example

- You might have 2 themes – Spring and Winter
  - Here are the 2 definitions for one skin in these themes - same skin, different themes:



Export Theme to Your Library

- For any theme you've created, you can archive it in your Library for use in other projects
- Note that this will move ALL the projects skins into the library

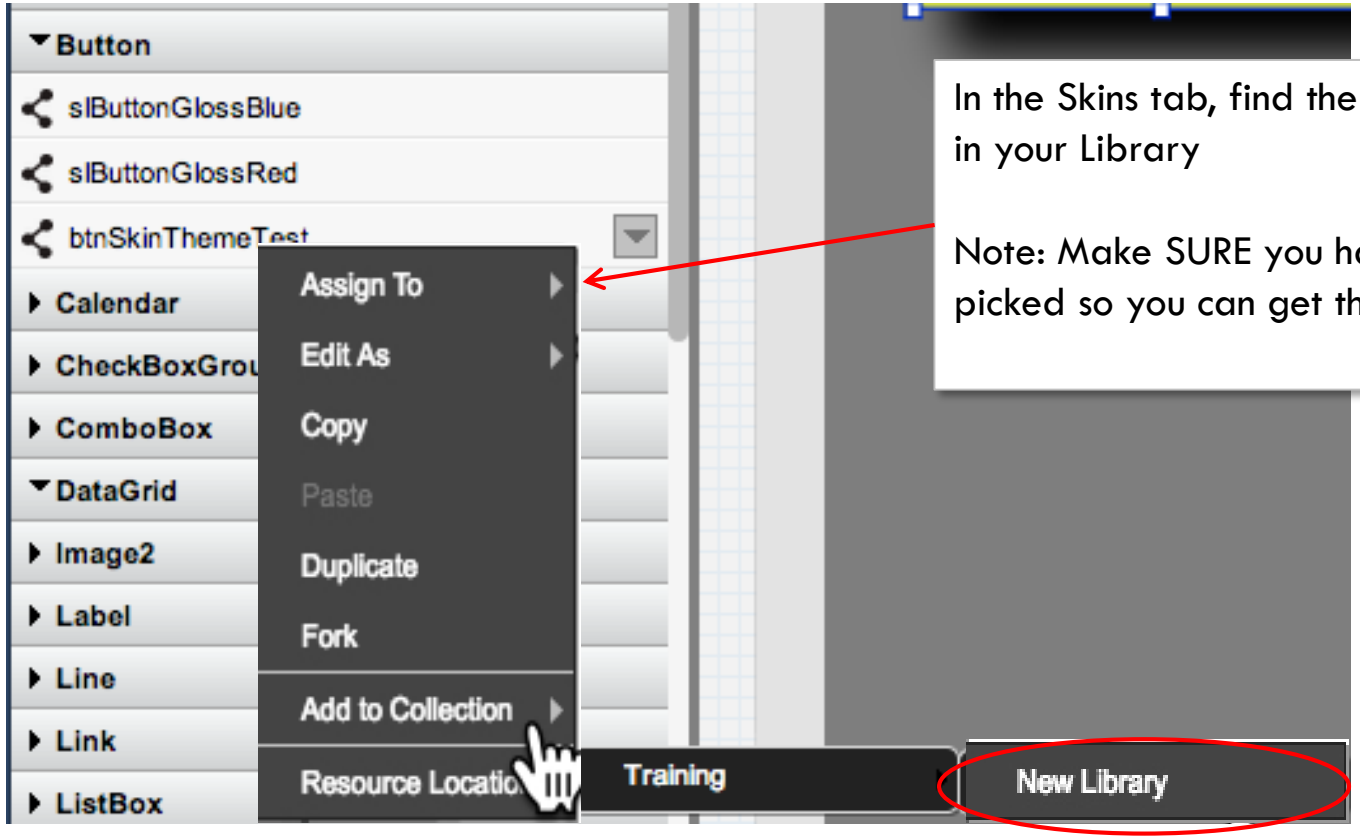


Note: Careful with this option, you'll get ALL the skins and ALL the themes in your Library

When saving, you can create a new library or use an existing one

Copying Skins to Your Library

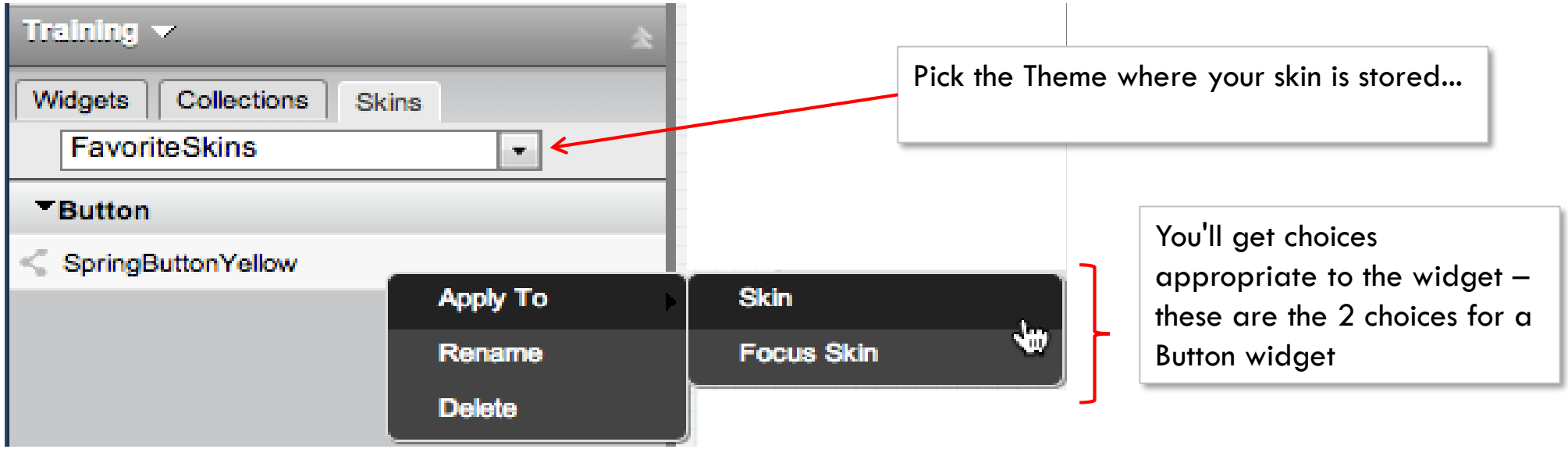
- Instead of the bulk export, you can save skins, one at a time, to your library:



Note: The word "theme" in this context is incorrect - it's a collection of skins, not a "real" theme you are creating here

Using Skins From Your Library

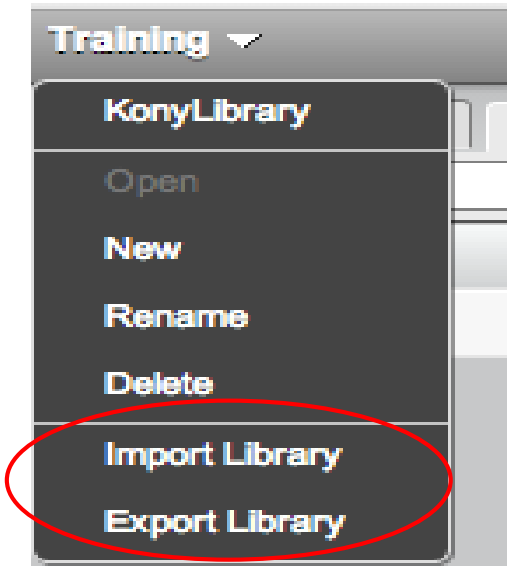
- Once you have your skin(s) in your library, you can now use any of these in your current project
  - First highlight the widget you want to apply to the skin
  - Then, you can right-click your library skin and apply it



- Once you apply the skin, then it becomes part of your current project

Using Libraries

- It is very important to save your work periodically – ESPECIALLY while you are learning/practicing
  - You can always go back and delete elements you don't need/want any more
  - It's very frustrating to mess up something and not have a way to get it back
- For saving more than one widget, you'll have to have your widgets in a container
  - Plan ahead and use a top-level container for your work to make it easy to save
  - If you forgot and later need to put all the widgets in a container, then cut/paste them onto your container, adjust the layout if necessary, and save in your collections
- You can import/export libraries to share with others
  - VERY useful in a distributed organization with many designers
  - Easy to create source controlled libraries



Programming In Visualizer

kony Stay Ahead

Programming in Visualizer

- We can create 2 types of Projects:
  - Kony Reference Architecture (MVC)
  - Freeform JavaScript
- MVC – Model View Controller
  - Uses RequireJS to load forms and code as needed to improve speed
- JavaScript is used for programming in Visualizer
  - RequireJS uses a define structure that is a JSON object with a key/value
- You can find the JavaScript code in the following 3 places
  - Controllers
  - Controller Actions
  - Modules

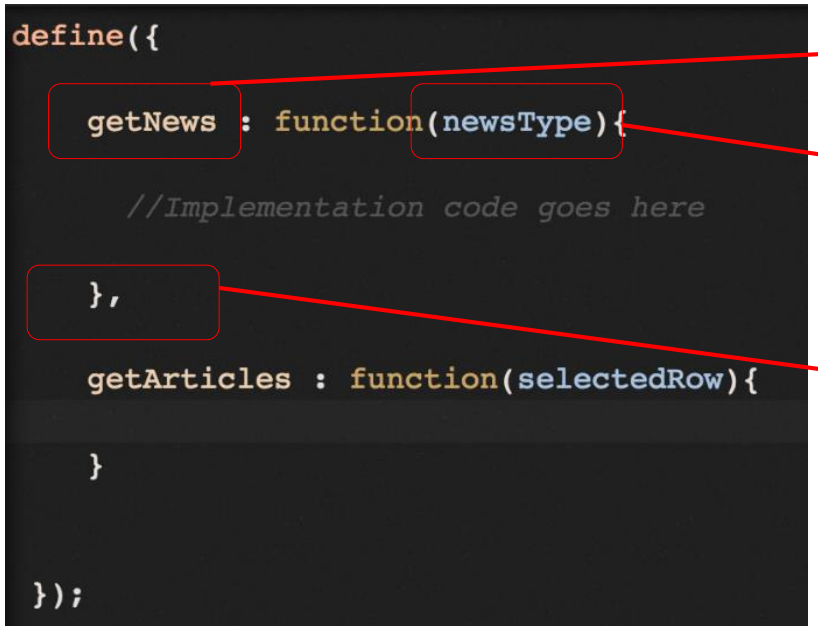
Programming in Visualizer

- You can find the JavaScript code in the following 3 places
  - Controllers
  - Controller Actions
  - Modules
- To access a widget's property in MVC, following the given format
  - "this" in the code refers to the current controller
  - "this.view" refers to the current form
  - "this.view.<widget>.<property>" to access property of a widget
  - For example if you have to access the selected key from the list box:

```
this.view.lstNewsType.selectedKey
```

RequireJS in Kony Context

- A method can be defined in
  - Controllers
  - Modules
- All the methods should be defined with in define module
- To define a method in controllers , use the following syntax:



Method Name / Function Name

Function or Method Input Parameters

Methods are separated by comma

RequireJS in Kony Context

- To navigate between one form to another, you need to create a navigation object and call navigate method on the navigation object

```
navigateToDetails : function(){

  var selItem = this.view.segNews.selectedItems[0];
  var navObj = new kony.mvc.Navigation("Form2");
  navObj.navigate(selItem);

}
```

Creating navigation object
and navigate to the next form"**Form2**".
The navigate method takes JSON object as input parameter.

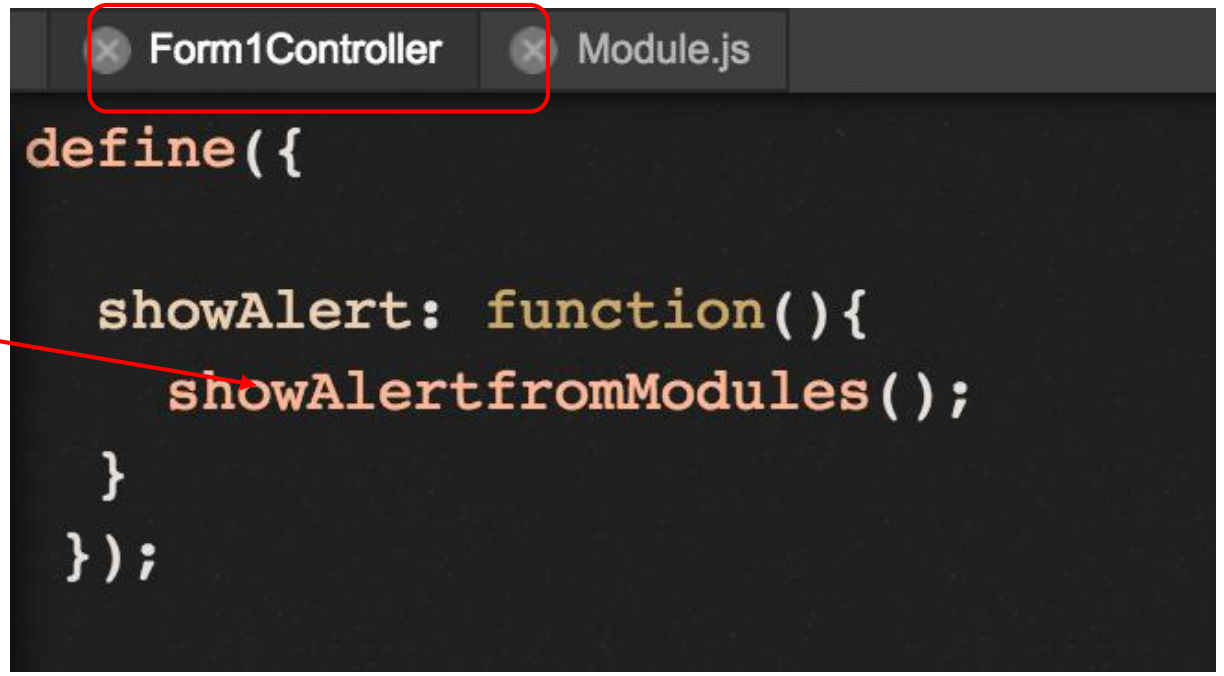
- When a form is navigated, the destination form controller's onNavigate method will get triggered. If the user has any data from the calling form, it can be handled here.

```
onNavigate : function(arg){
    this.view.lblTitle.text= arg.title;
    this.view.lblDescription.text = arg.description;
    this.view.lblPubDate.text = arg.pubdate;
}
```

onNavigate method needs to be implemented if you have any data sent from the calling form

RequireJS in Kony Context

- To access any module code in controller the function by it's name directly. As the modules code written under Project→Modules→ <Module>.js they are public by nature and you don't need to load them explicitly



RequireJS in Kony Context

- To define a asynchronous callback method to any SDK, use the following syntax.

```
getNewsArticles : function(){
  //KNYMobileFabric = kony.sdk.getCurrentInstance();
  serviceName = "FoxNewsServcie";
  integrationObj = KNYMobileFabric.getIntegrationService(serviceName);
  operationName =  "getNews";
  data= {"newsType": this.view.lstNewsType.selectedKey };
  headers= {};

  kony.application.showLoadingScreen(null, "Loading articles", constants.LOADING_SCREEN_POSITION_ONLY_CENTER, true, true, null
  integrationObj.invokeOperation(operationName, headers, data, operationSuccess.bind(this), operationFailure.bind(this));

  function operationSuccess(res){
    kony.application.dismissLoadingScreen();
    alert("sucess");
    this.view.segNews.widgetDataMap = {lblTitle : "title"};
    this.view.segNews.setData(res.articles);

  }

  function operationFailure(res){
    kony.application.dismissLoadingScreen();
    alert("failed");
  }
},
```

RequireJS in Kony Context

- Please note that in the callbacks 'operationSuccess' and 'operationFailure' are defined with in the same function and the form controller object 'this' will not be available. To get this, use 'bind'

```
integrationObj.invokeOperation(operationName, headers, data, operationSuccess.bind(this), operationFailure.bind(this));
```

- To invoke a method with in the controller, use the 'this'

```
define({

  showAlert: function(){
    this.displayButton();
  },
  displayButton: function(){
   //Code Implementation
  }
 });
```

Programming in Visualizer

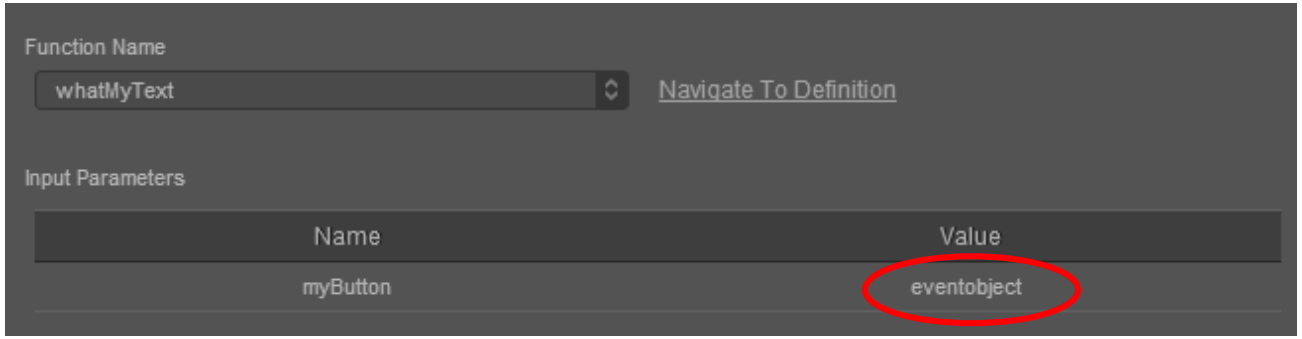
- Now you know how to add a function to your form controller
- You can define global functions in Modules
- You don't have to type all the code - Use IntelliSense
- How to invoke your function?
  - Use Code Snippet or Invoke Function Action from the Action Editor
- You can pass values to functions

EventObject

- When a function is called from a widget event, we can also pass the information to the function about that widget
- This is done by sending the eventobject is the "thing" that triggered the event – usually a widget
- For example, if I assign a function to a button onClick event, then the button object will be passed as eventobject

  whatsMyText(myButton):function{

  kony.print("button text is: "+ myButton.text)}
- In the Event Editor, I can use the eventobject to pass into my function call:



- In this case, the eventobject will be the button object

Logging

- Kony API to log information and objects to device logs is

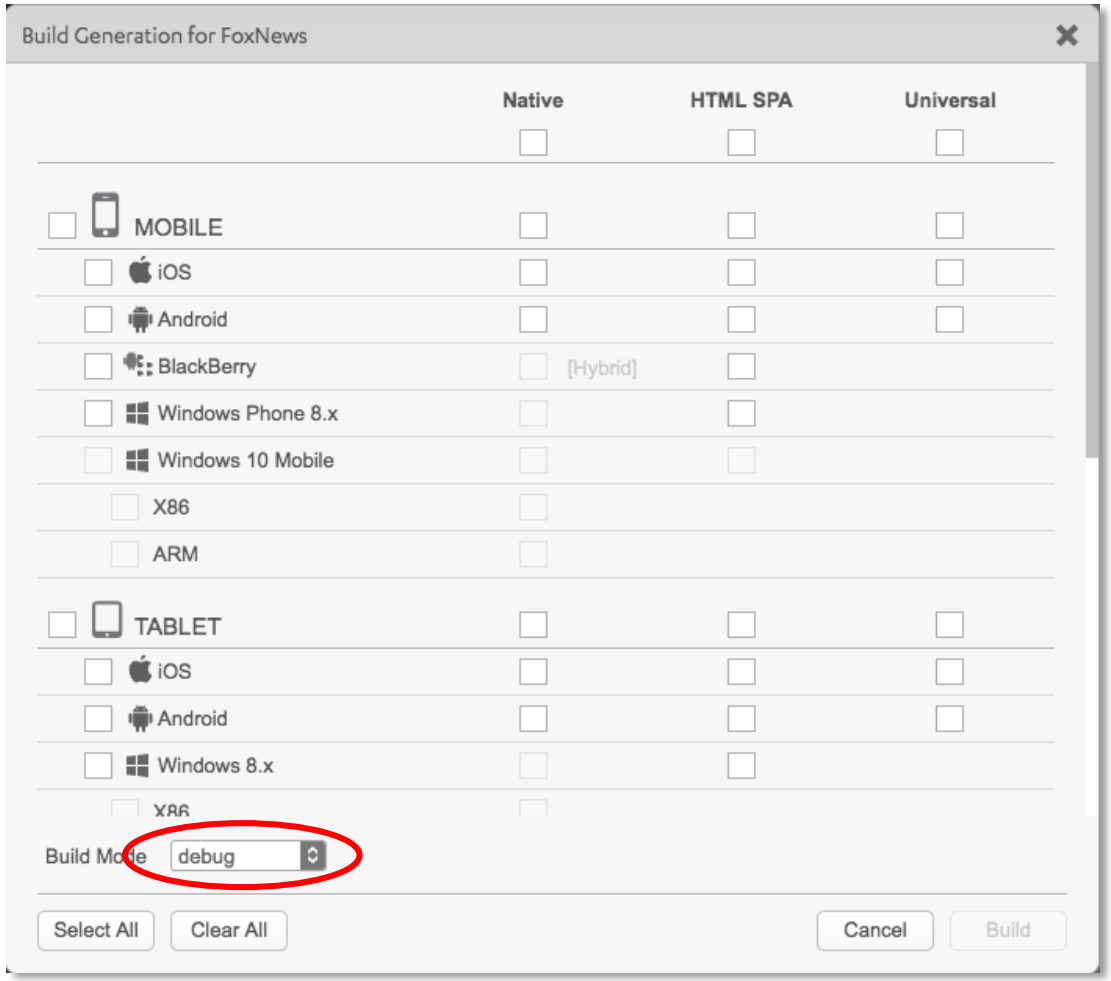
  kony.print(<text to print>)
- API to get the text representation of a JSON object

  JSON.stringify(obj)
- The Kony platform generates a lot of logging data, so it's incredibly helpful to see what's going when errors occur
- Use some special string to distinguish your messages like below

  kony.print("$$$$$$$$$$$$$$$$$$$ " + debugData)

Logging (cont'd)

- Ensure to build the application in debug mode to see logs



iPhone & Android Logging

- iPhone Logs
  - Let's see the logs in iPhone simulator
  - Simulator Menu – Debug > Open System Log
  - Filter the logs by search String

- Android Logs
  - Let's see the logs of android device
  - Launch Android Monitor
  - Filter by tag "StandardLib"

Inline Debugging

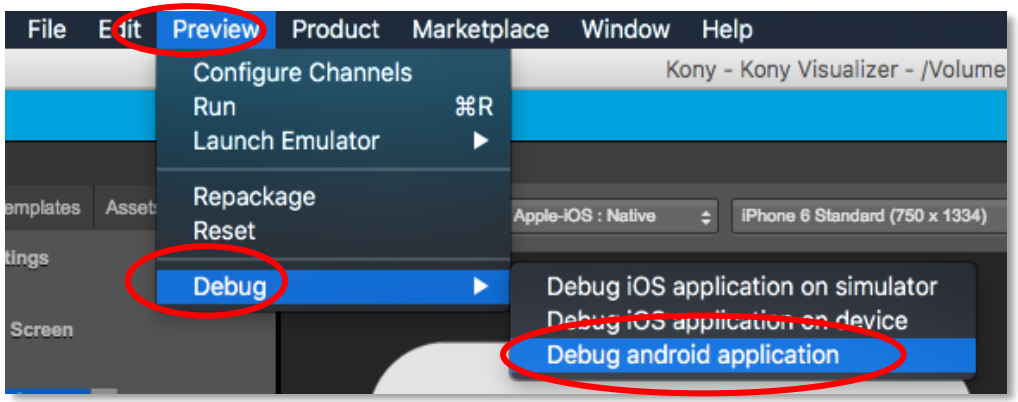
- Inline Debugging enables you to
  - Use breakpoints to halt execution
  - Examine the state of the app and examine all variables
  - Step through your code

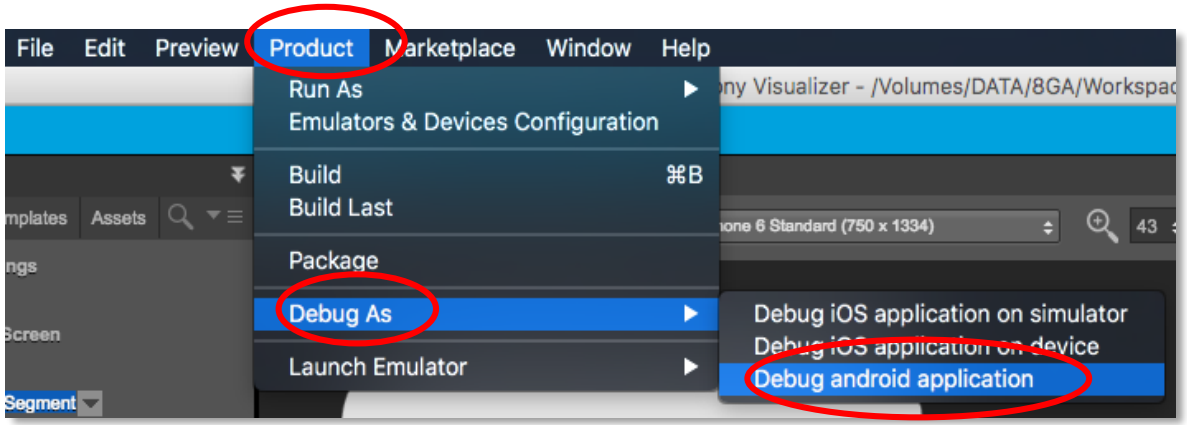
- Inline debugging can be done for
  - Android and iPhone Native apps
  - SPA and Desktop apps

- It is supported in both Preview and Build

Inline Debugging (cont'd)

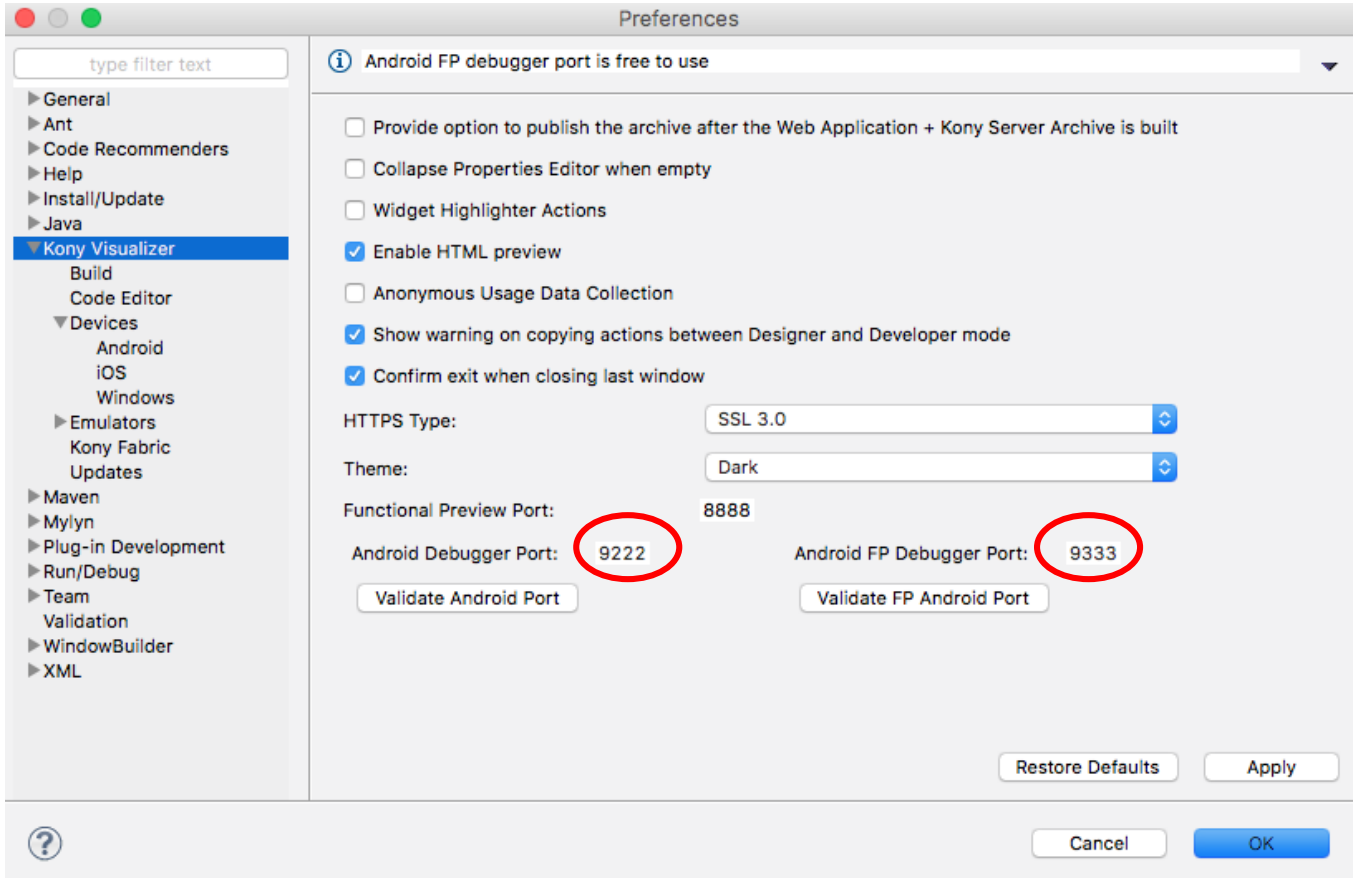
- Let's see inline debugging for Android Preview and Build
  - Run preview/build in debug mode
  - Connect device and launch the application
  - Click START on debug window
  - Launch Debugger from Visualizer





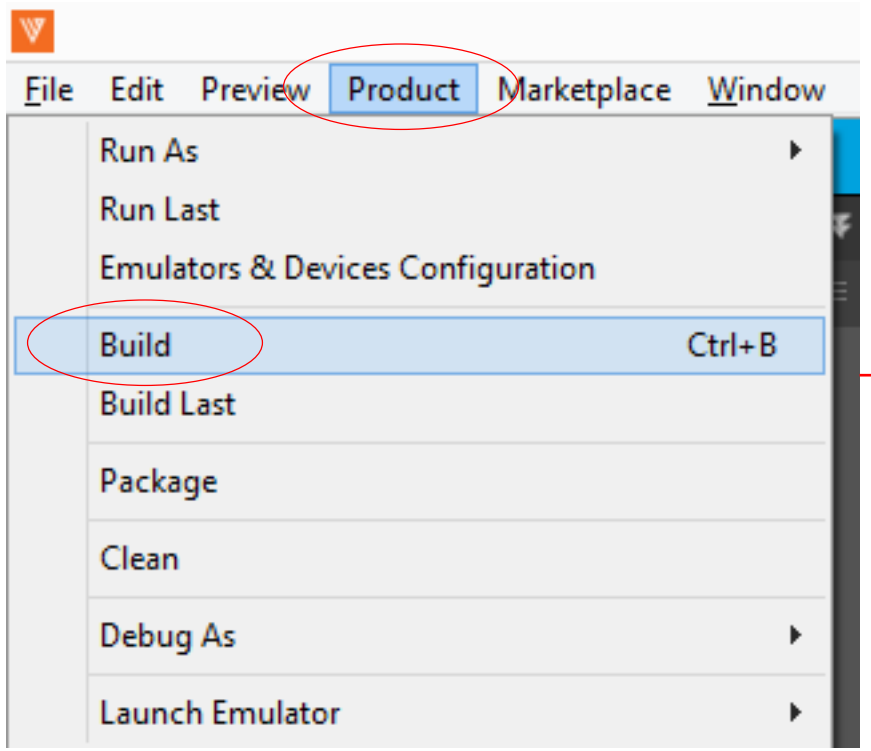
Inline Debugging on Android Native

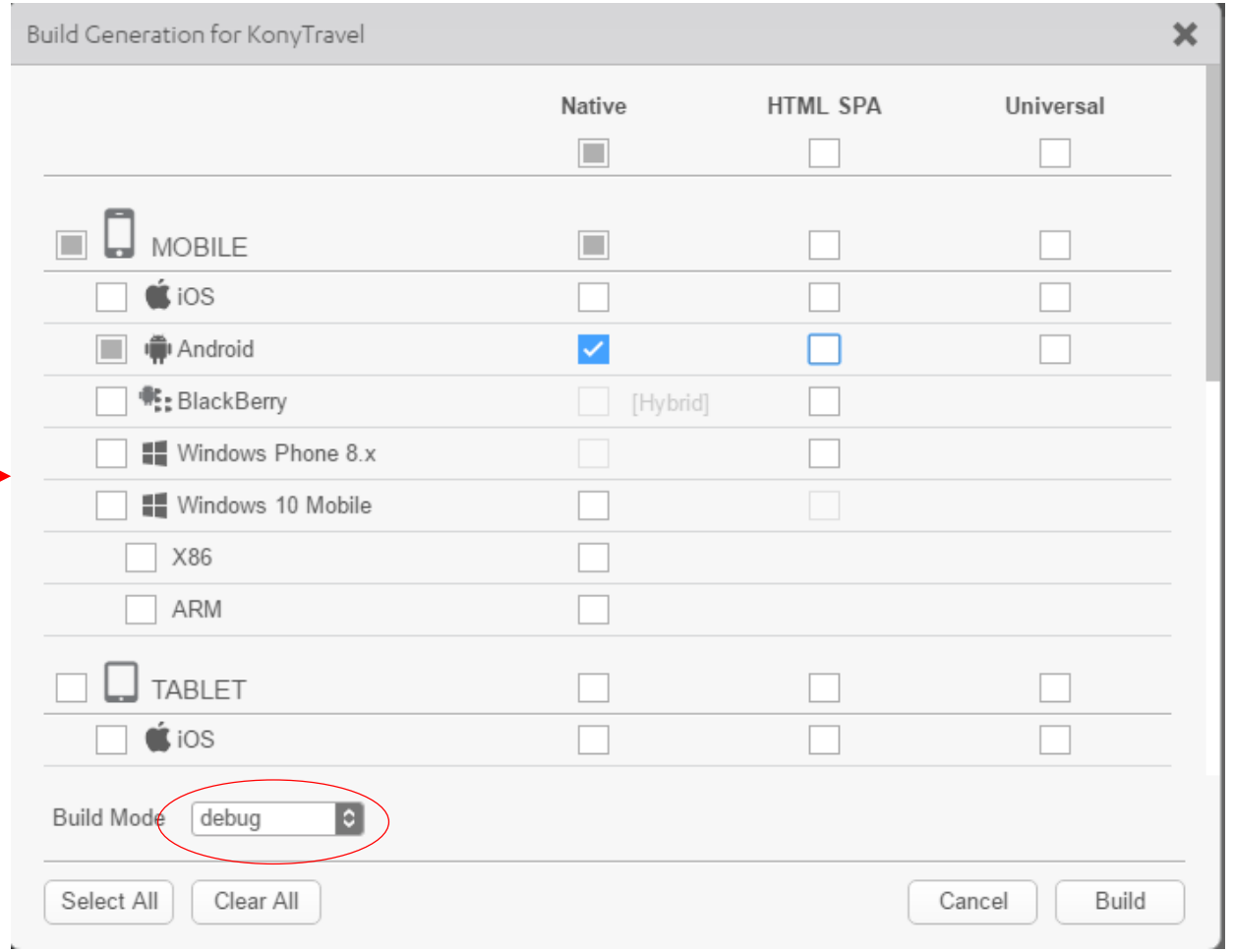
- For Android, the debugger will be launched in Chrome
- Ports used by the debugger…



Build the application For Android Native

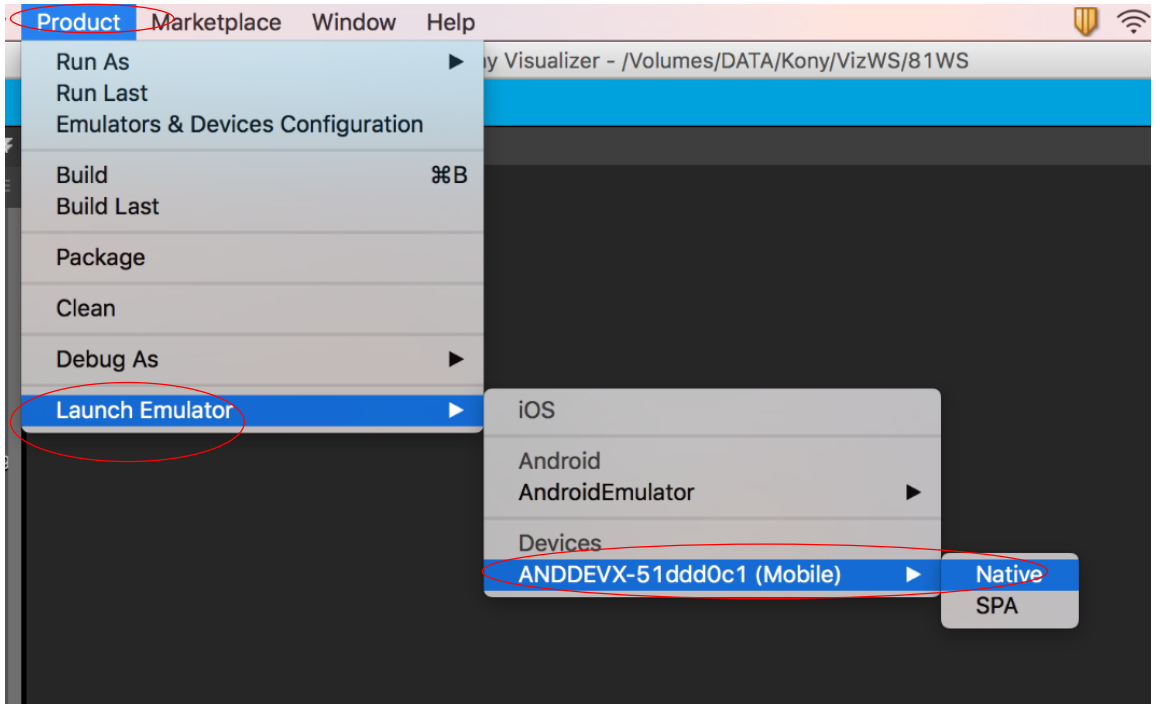
Build the application in debug mode



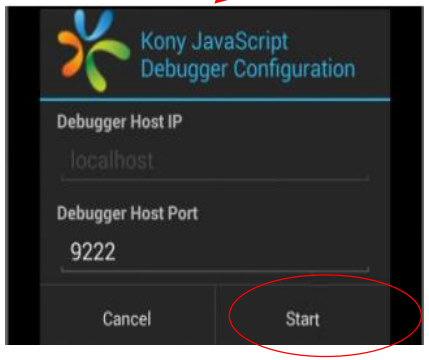


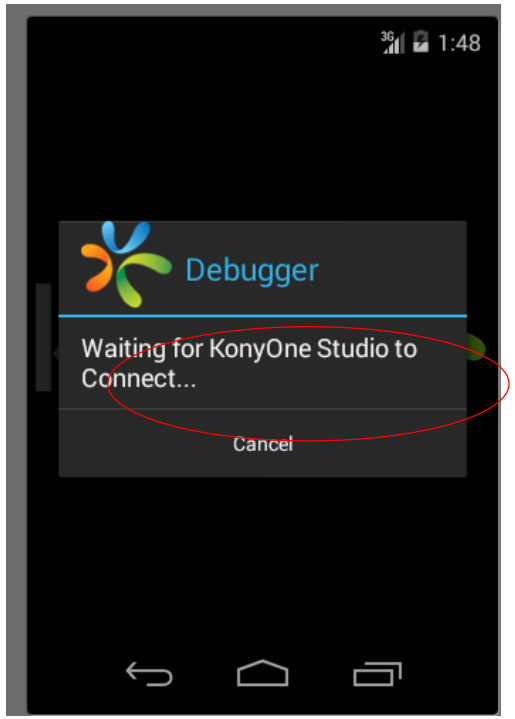
Launch the application on Android device

Launch the application onto android device



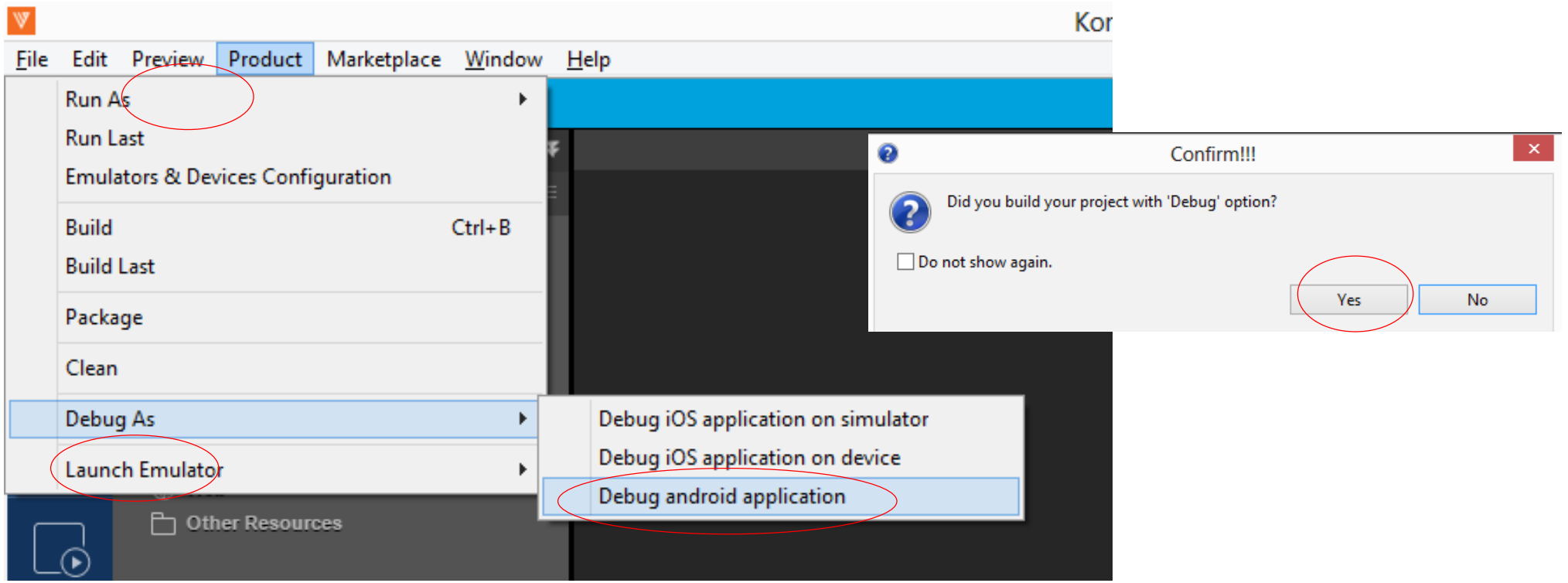
Application launched on android device/emulator. Click on start to start the debugging for your application





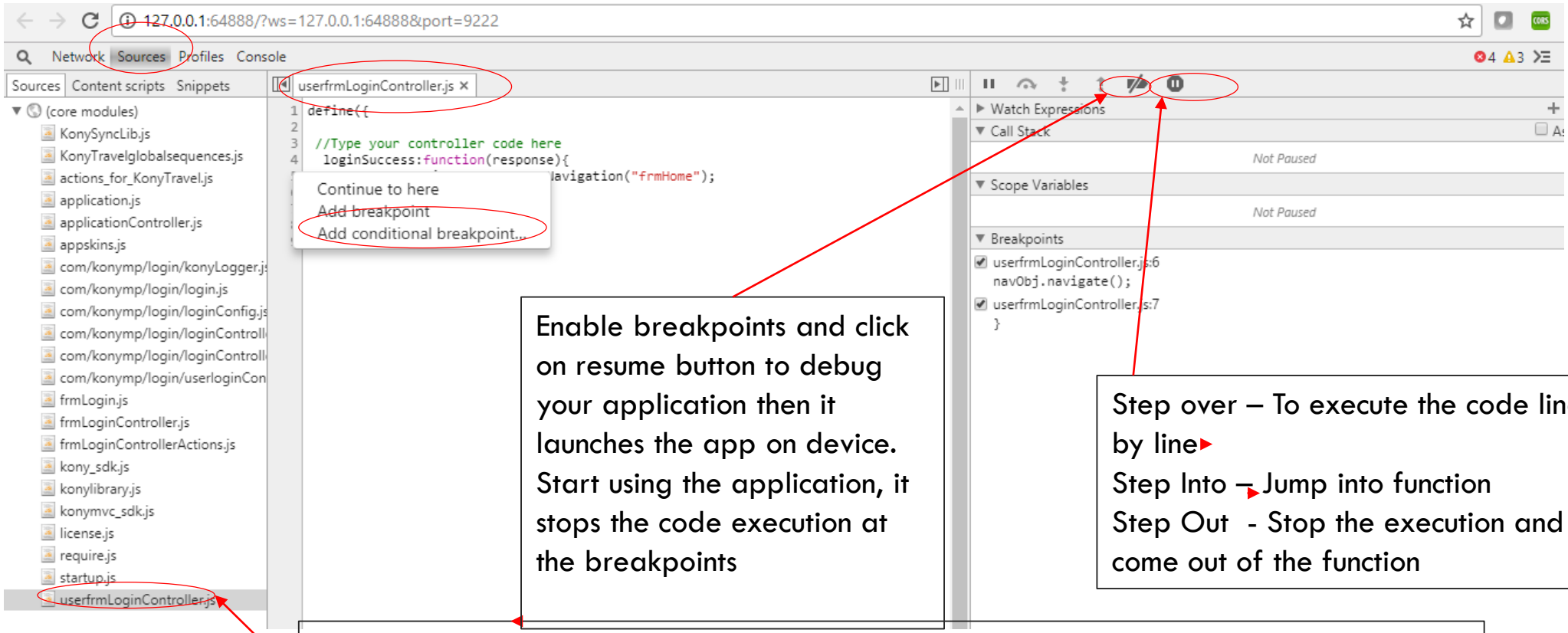
Launch the debugger from visualizer

In visualizer, navigate to Product -> Debug As -> Debug android application.
It uses chrome browser to launch the debugger console



Debugger Console On Chrome Browser

Add break points and debug the application in debugger console



Enable breakpoints and click on resume button to debug your application then it launches the app on device. Start using the application, it stops the code execution at the breakpoints

Step over – To execute the code line by line
Step Into – Jump into function
Step Out - Stop the execution and come out of the function

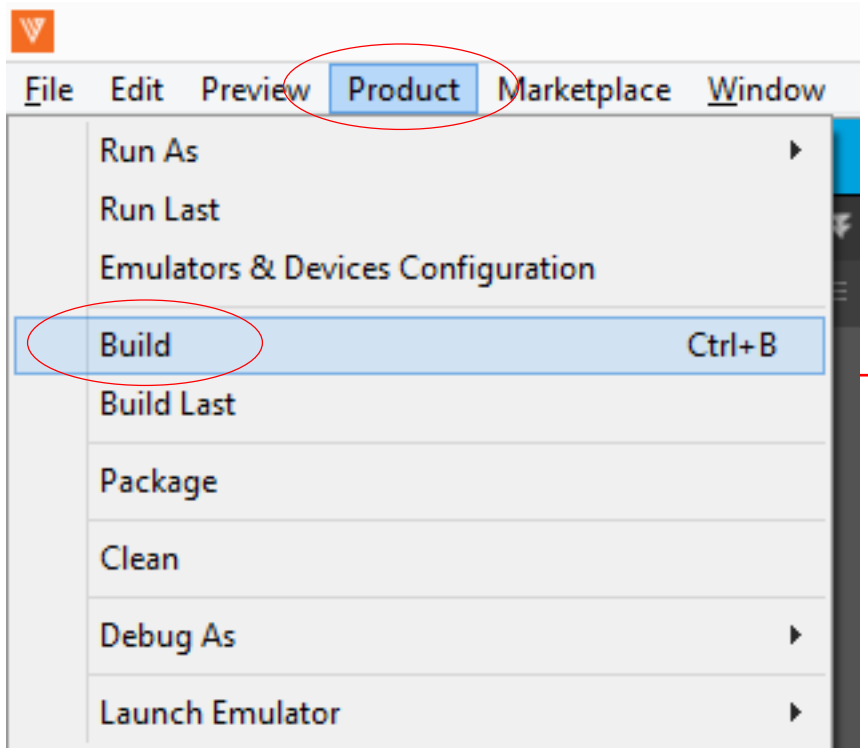
Double-clicking on any of these code files opens them up so we can put in our breakpoints by simply clicking on line number (or) right click on line number and click on add breakpoint to enable it

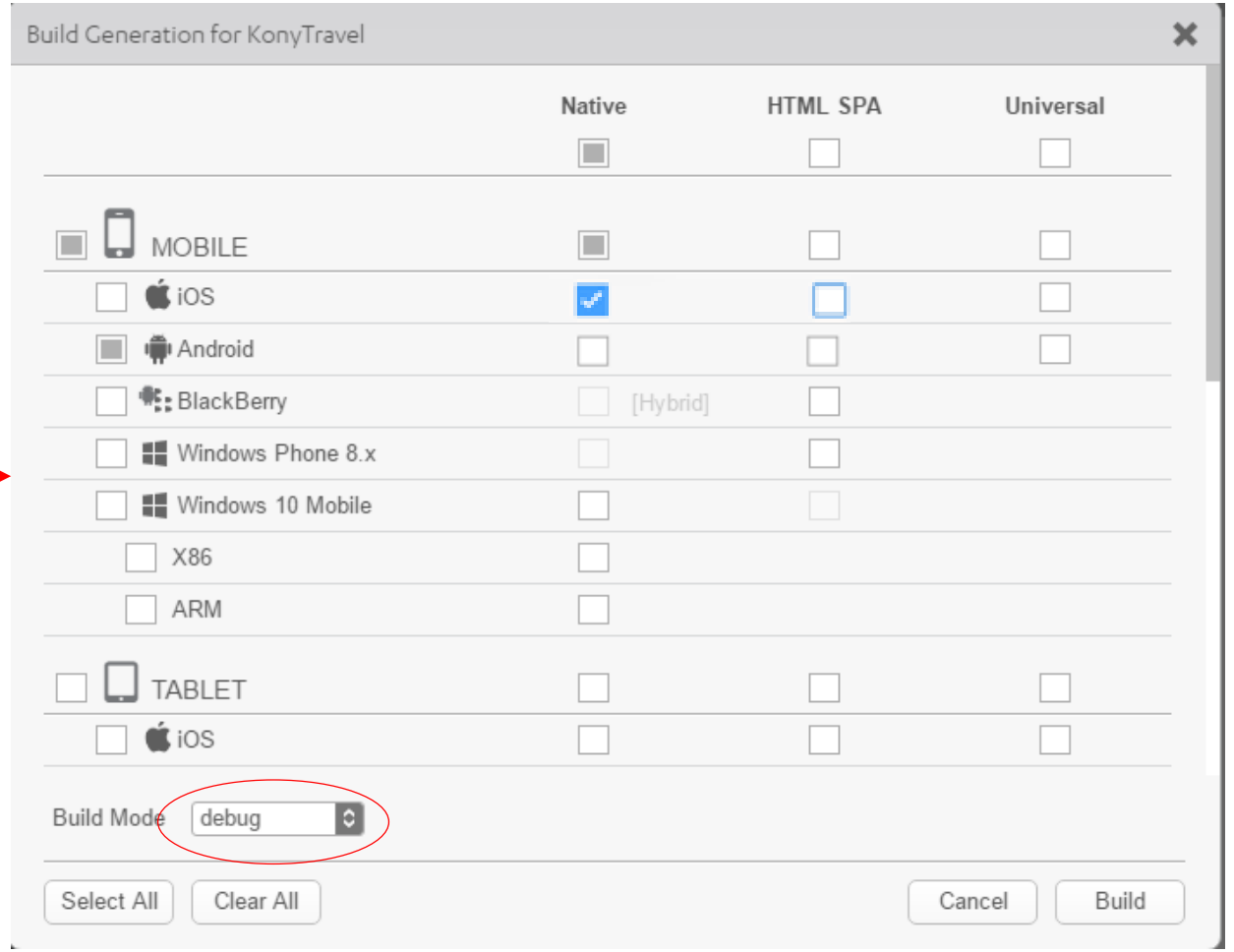
Inline Debugging iPhone Native

- Let's see inline debugging for iPhone Build on Simulator
  - Build the application in debug mode
  - Launch Safari browser on MAC
  - Enable develop menu in Safari Browser
  - Launch the debugger console from Safari
  - Debug the application by enabling breakpoints

Build the application

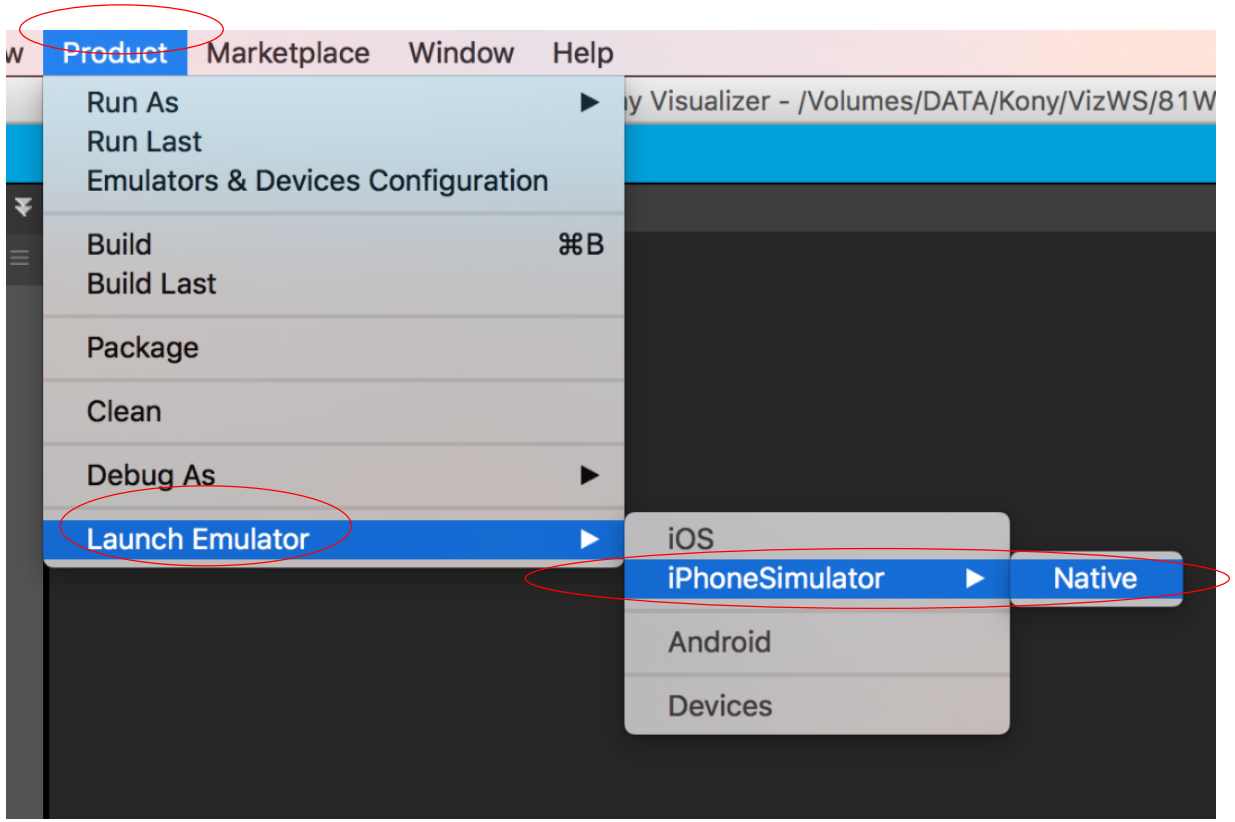
Build the application in debug mode



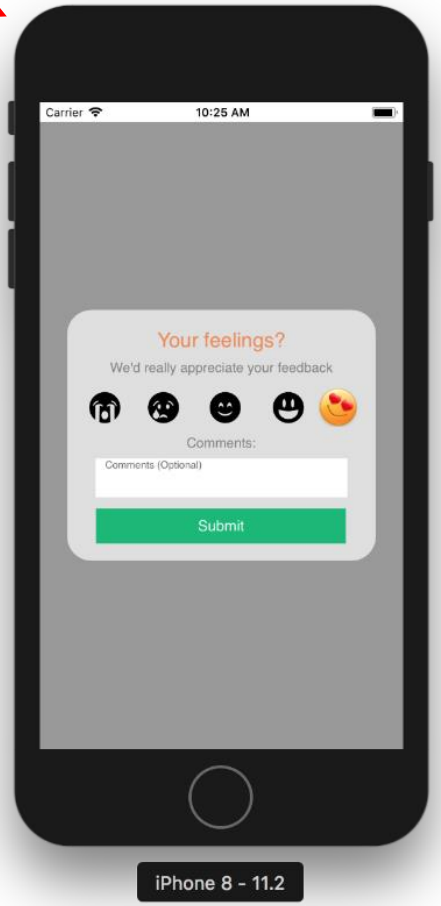


Launch the application on iPhone Simulator

Launch an application onto iPhone Simulator

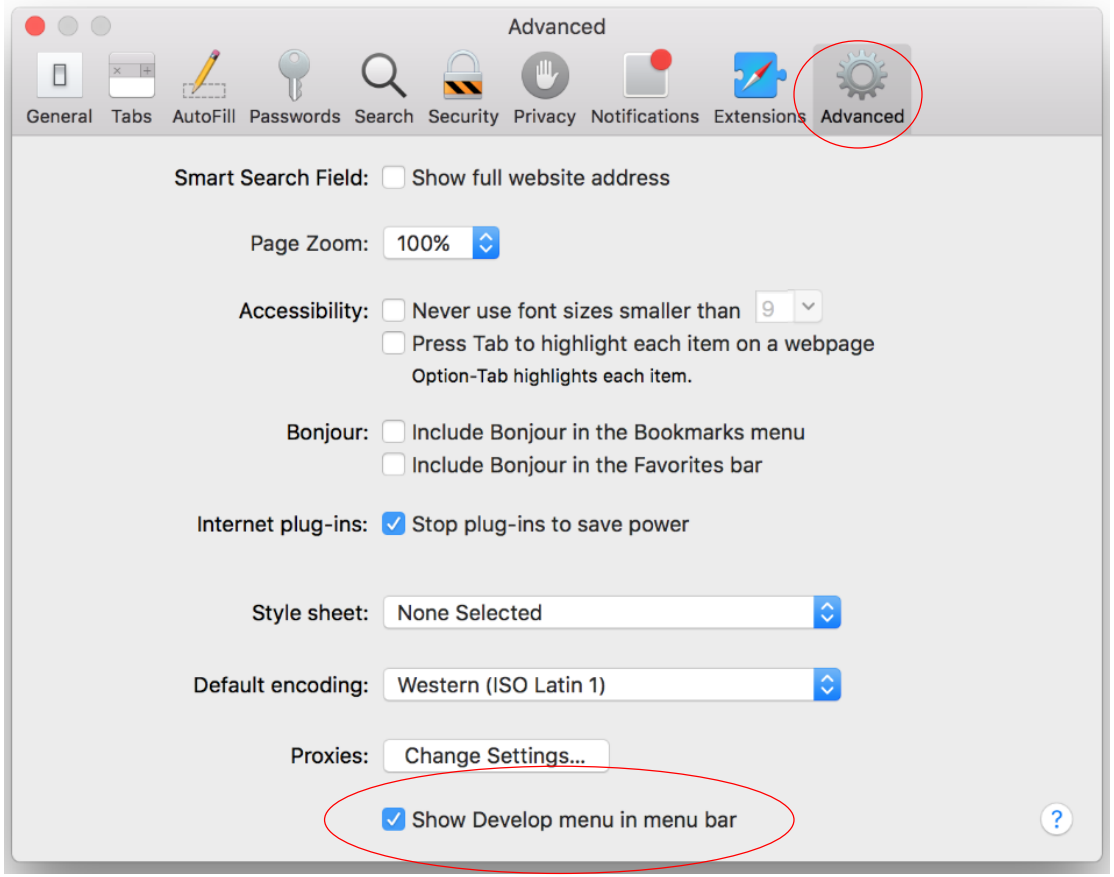


Application launched on iPhone simulator.



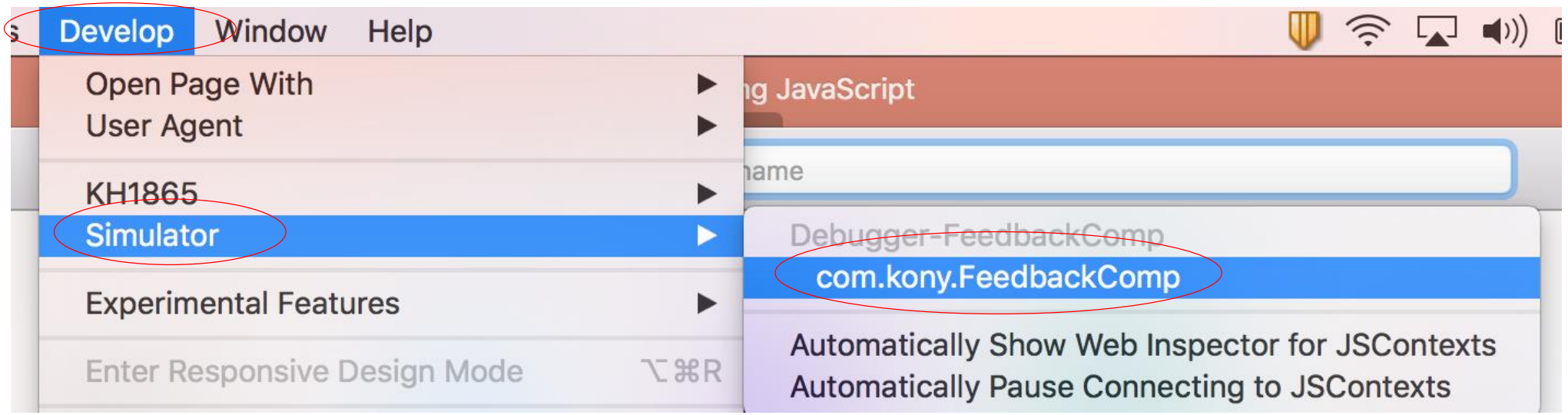
Enable Develop Menu In Safari Browser

- For iPhone, the debugger will be launched in Safari
- Open Safari browser on MAC and go to preferences -> Advanced tab -> Enable 'Show Develop menu in menu bar ' option

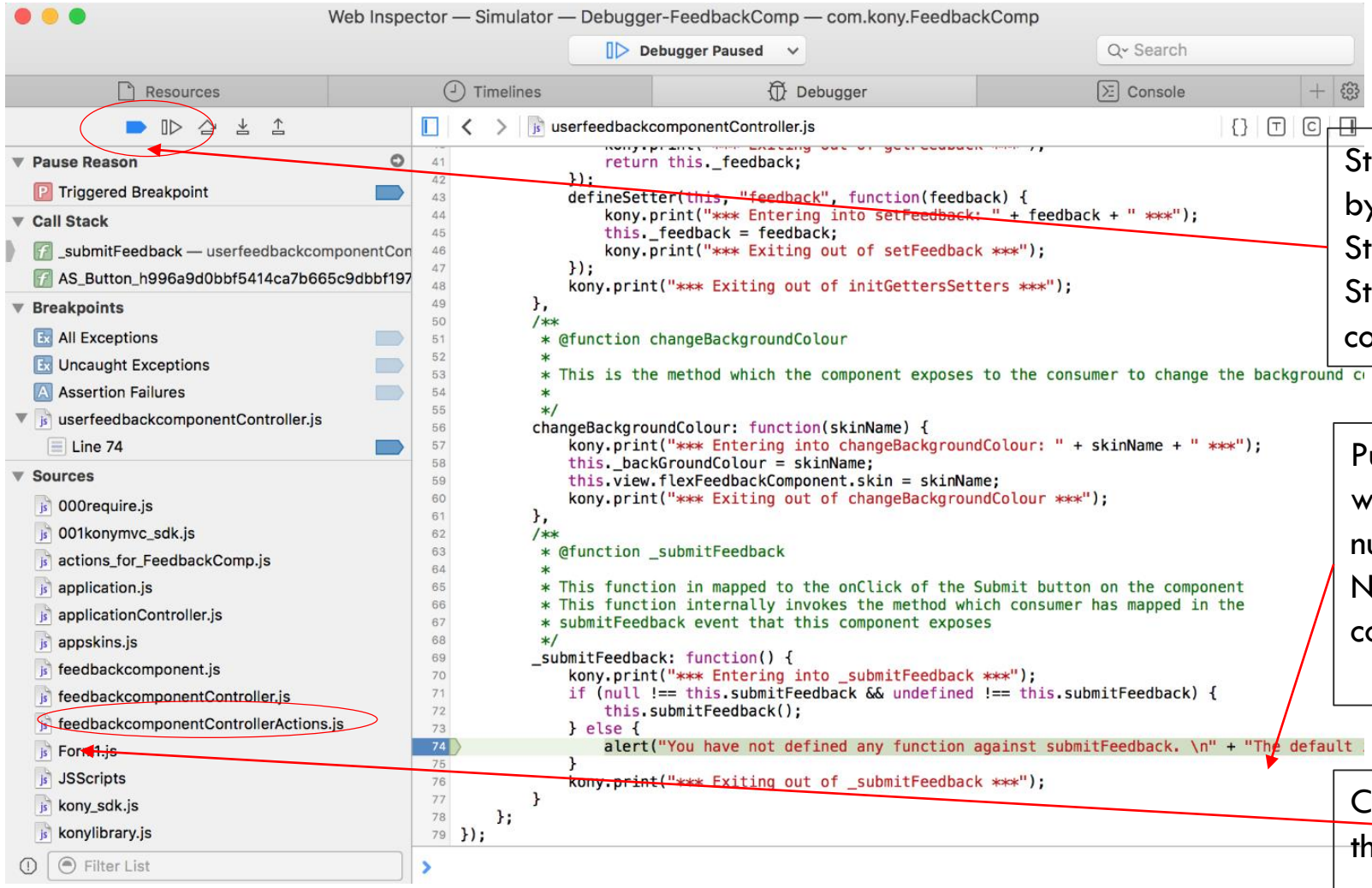


Launch Debugger Console On Safari Browser

Select Safari browser and click on develop menu -> Simulator -> Select the application which you want to debug



Debugger Console On Safari Browser

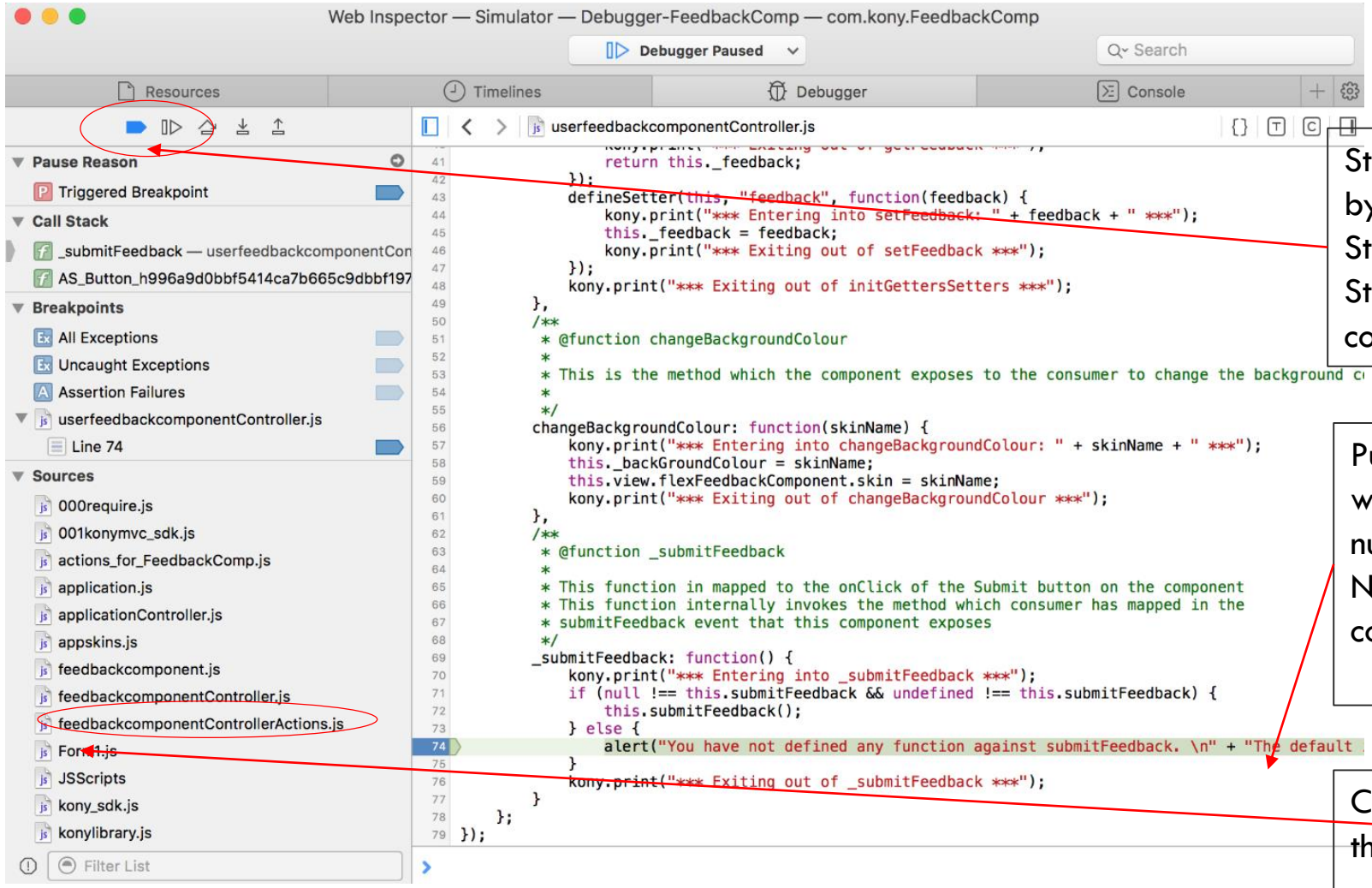


Step over – To execute the code line by line
Step Into – Jump into function
Step Out - Stop the execution and come out of the function

Put breakpoints wherever you want by simply clicking on the line number.
Now start using the application, control stops at this breakpoint

Clicking any of these code files opens them up so we can put in breakpoints.

Debugger Console On Safari Browser

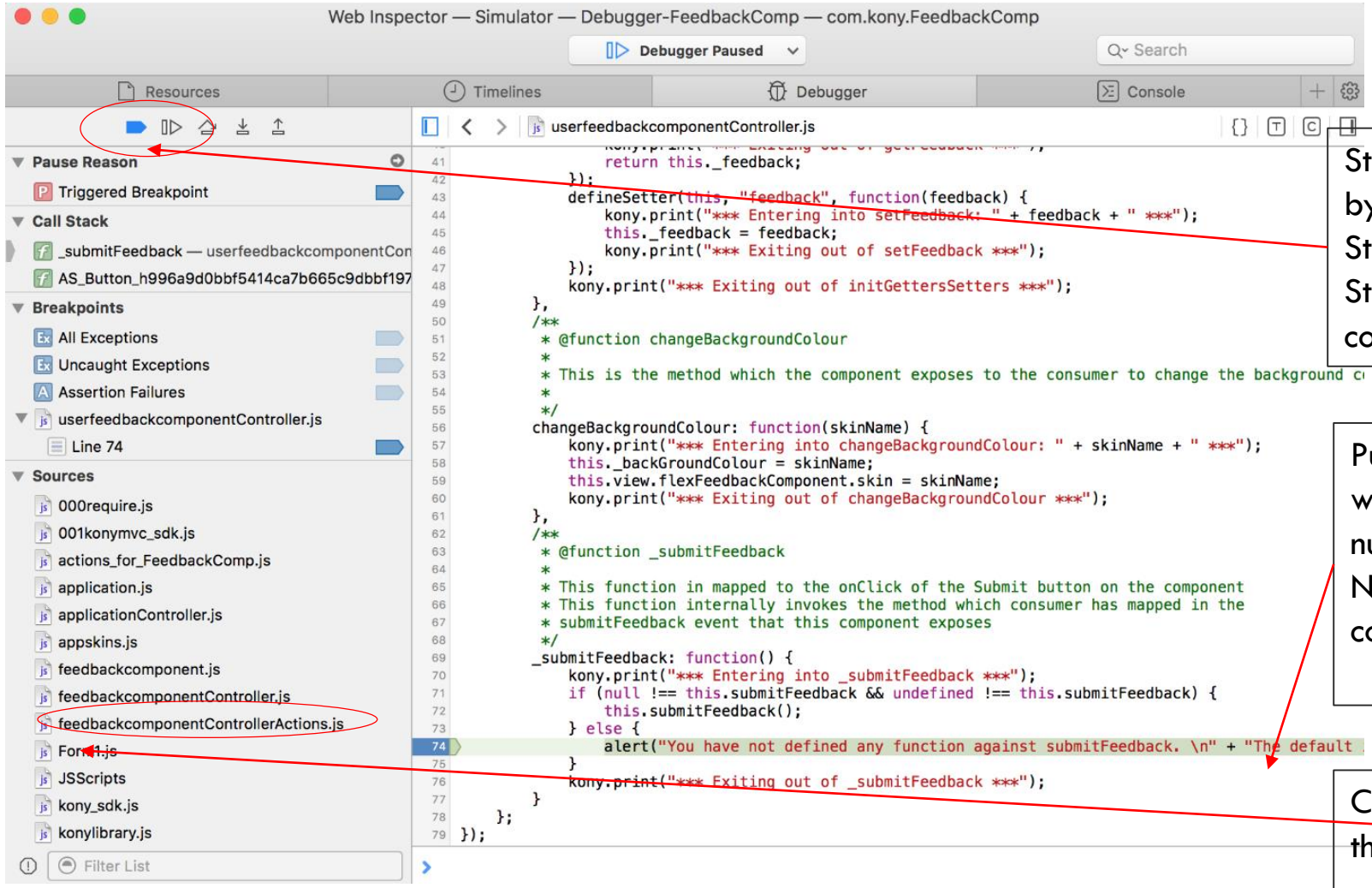


Step over – To execute the code line by line
Step Into – Jump into function
Step Out - Stop the execution and come out of the function

Put breakpoints wherever you want by simply clicking on the line number.
Now start using the application, control stops at this breakpoint

Clicking any of these code files opens them up so we can put in breakpoints.

Debugger Console On Safari Browser



Step over – To execute the code line by line
Step Into – Jump into function
Step Out - Stop the execution and come out of the function

Put breakpoints wherever you want by simply clicking on the line number.
Now start using the application, control stops at this breakpoint

Clicking any of these code files opens them up so we can put in breakpoints.

Kony API

kony Stay Ahead

Kony API

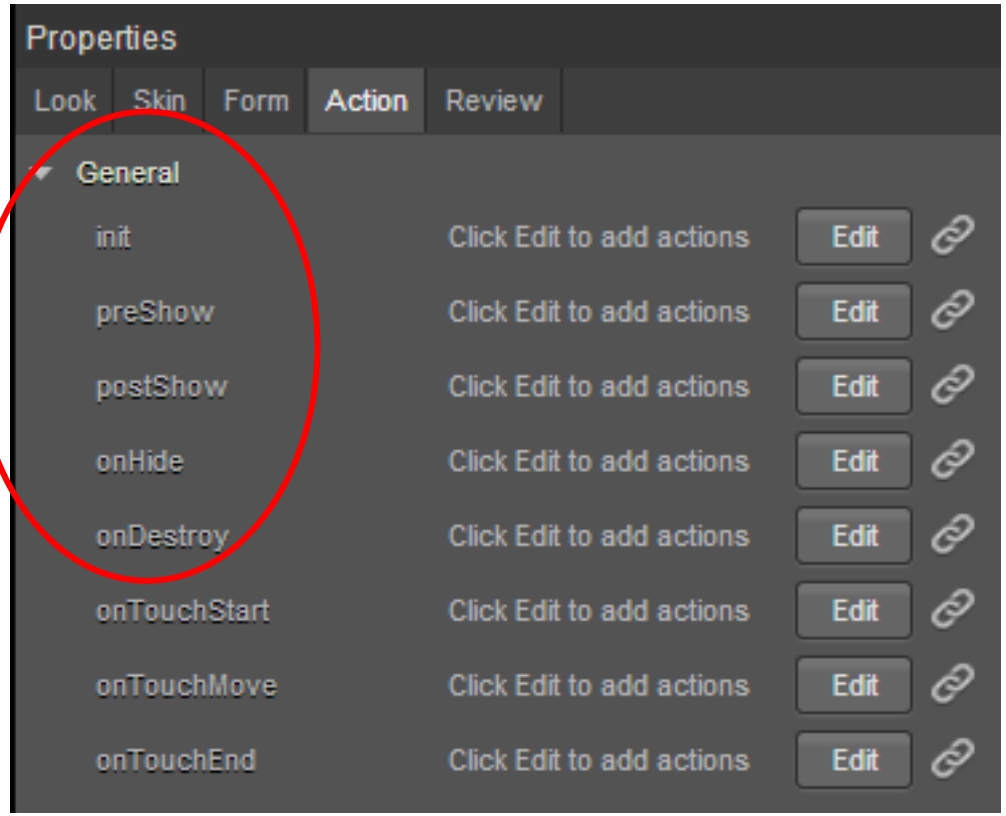
- The Kony API is very extensive and gives access to everything you need to do in a mobile app
  - For anything that is NOT covered in the Kony API, Foreign Function Interface (FFI) is Kony technology that lets you write native code and wrap it up so you can call it from your application
  - We cover FFI later on
- We will introduce you to some of the API methods now
- Note: Many parameters for the API are JSON objects (ex: {parm1:"v1", parm2:"v2"…}) Going forward, unless it's ambiguous, we'll refer to them simply as "objects"
- The following module is NOT meant to be an all-inclusive look at the entire API
- The goal is to expose you to some key features and encourage you to go to the Kony documentation set and use the API guide

Form - Overview

- Let's now take a closer look at what is going on at the form level
- Forms have a lifecycle:
  - Forms can be created
  - Forms can be shown/hidden
  - Forms can be destroyed
- We're now going to take a look at how to manage form lifecycle events
- Note: All of the code and events described in this module are applicable for all app types (Native, SPA, etc.)
- Remember we already used the form's preShow event to set up things on a form in Visualizer – we'll now see how it and other events are used

Form - Overview (cont'd)

- A Form is defined to have a life cycle method that gets called at appropriate events
- Form Life Cycle Methods :
  - Init
  - preShow
  - postShow
  - onHide
  - onDestroy
  - The form lifecycle BEGINS when you:
    - Access a form
    - Access a widget on a form
    - Show a form



Form - Life Cycle Events (cont'd)

- init
  - This event is invoked only once in the form's life cycle, when the form is ready with the widgets hierarchy
  - This is called after the addWidgets method, and is a good place to put code to initially set up the form – you'll have access to all the widgets
  - When form is destroyed and reused again, init gets called as a part of form's lifecycle

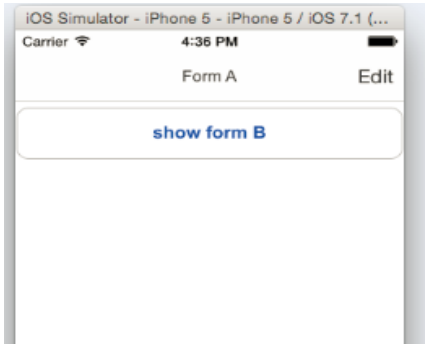
- preShow
  - This event is triggered just before the form is visible on the screen
  - This event is triggered every time the form is shown whether through code, the device back button, etc.
  - This is a good place to put code to prepare the screen for viewing

Form - Life Cycle Events (cont'd)

- postShow
  - This event is triggered every time after the form is rendered on the device's screen
  - This is a good place to put code that does things like make service calls to get data to display since postShow is not holding up the UI thread like preShow
  - Can be used to display other forms (e.g. interstitial screen) during service calls
- onHide
  - This event is called when the form goes out of view because some other form is displayed
  - TonDestroy
  - his event is called when the form and it's controller is removed from memory
  - You can call this using API kony.application.destroyForm("frmHello");
  - We will discuss application API's later

Exercise - Form Events

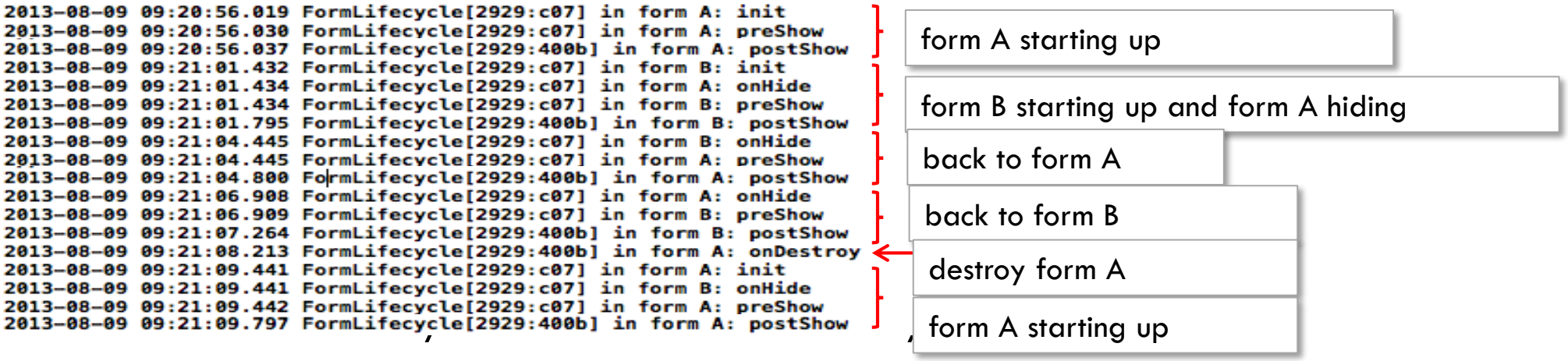
Let's create a 2 screen app that looks like this:



- Navigation is simple – the "show form B" button takes you to form B and the "show form A" button takes you to form A
- On form B, the second button calls the **destroyForm()** on form A
  - Use a snippet with: kony.application.destroyForm("frmA");
  - Navigation can be done using the navigation action

Exercise - Form Events (cont'd)

- On both forms, add a quick print statement for each of the form lifecycle methods:



  - When the app starts - form A does an init
  - When we go to form B for the first time
  - After we destroy form A and then navigate back

JavaScript - String Library Examples

Let's look at just a few useful Kony String API functions, so you can see how they work:

- kony.string.containsChars (inputstring, characterArray) - verifies if any of the specified set of characters is available in the given string where:
  - inputstring - the input string that will be checked
  - characterArray - the set of characters that need to be searched within the input string.
  - Return value:
    - true - if the given input string contains any one of the characters in the specified list.
    - false - if there is an error in the input or if the given input string does not contain any of the specified input characters.
  - Examples:

kony.string.containsChars("abdcdADV", ["|","-"]) returns false

kony.string.containsChars("abdcd|ADV", ["|","-"]) returns true

JavaScript - String Library Examples (cont'd)

- kony.string.isNumeric(inputstring) - checks if the passed in string represents a numeric value where:
  - inputstring - is the string to check
  - Returns: true/false
  - Examples:

    kony.string.isNumeric("10.34") returns true

    kony.string.isNumeric("10 pies") returns false
- kony.string.trim(inputstring) - trims off any leading and trailing spaces from the passed in string
  - inputstring - is the string to check
  - Returns: String with no leading or trailing blanks
  - Example: kony.string.trim(" Hello JS user ") returns "Hello JS user"

JavaScript - String Library Examples (cont'd)

- kony.string.isValidEmail (inputstring)- checks if the passed in string is a valid email address, where…
  - inputstring - is the string to check
  - Returns: true/false
  - Note: Here is what this method is checking for:
    - has at least 6 characters
    - has @
    - has at least 4 characters after @
    - has . (dot) followed by at least 2 characters
    - has at least 1 character before @
  - Examples: kony.string.isValidEmail("abcd@")) returns false

    kony.string.isValidEmail(abcd@abc.com) returns true

Phone Service API

- kony.phone.sendSMS(phoneNumber, text) - sends an SMS with the text to the number specified
  - Returns a 0 for success or a -1 for failure
- kony.phone.dial(number) - brings up the dialer and populates the specified number
- kony.phone.openEmail(toRecipients, ccRecepients, bccRecepients, subject, messageBody, isMessageBodyhtml, attachments) - opens a new email message in the native email client with the fields filled out according to the information passed in
  - For example:

```
var to = ["training@kony.com"];
var cc = ["myself@gmail.com"];
var bcc = ["mymanager@gmail.com"];
var sub = "I love Kony training!!!";
var msgbody = "Just wanted to tell you how much I enjoyed the class";
kony.phone.openEmail( to, cc, bcc, sub, msgbody, false );
```

Alert API

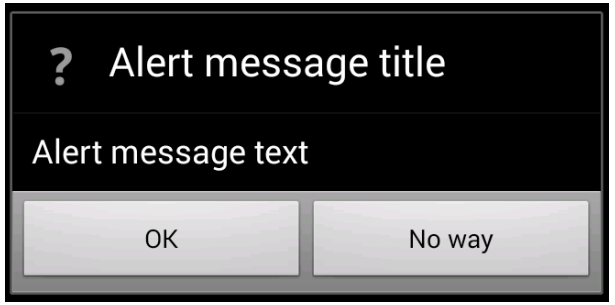
- The basic syntax is: kony.ui.Alert (basicconfig,deviceconfig) where:
  - basicconfig is an object for configuring the alert (see example below for all the fields)
  - deviceconfig is an object configuring device info - not implemented yet
- For example (with no callback):

```
kony.ui.Alert(
    {   message     : "Alert message text",
        alertType   : constants.ALERT_TYPE_CONFIRMATION,
        alertTitle  : "Alert messages title",
        yesLabel    : "OK",
        noLabel     : "No way",
        alertIcon   : null,
        alertHandler: null
    }, {});
```

other options are:
ALERT_TYPE_ERROR
ALERT_TYPE_INFO

no icons on iOS

deviceinfo is passed as an empty object



Alert API - Callbacks

- But what if wanted to run code after the user clicks the button? Where does that code go?
- The Kony API makes extensive use of callbacks
- A callback is necessary when an activity must be triggered asynchronously
  - The call to the API triggers the activity to start and identifies the function name of the callback
  - When the activity is complete*, the specified callback function will be called

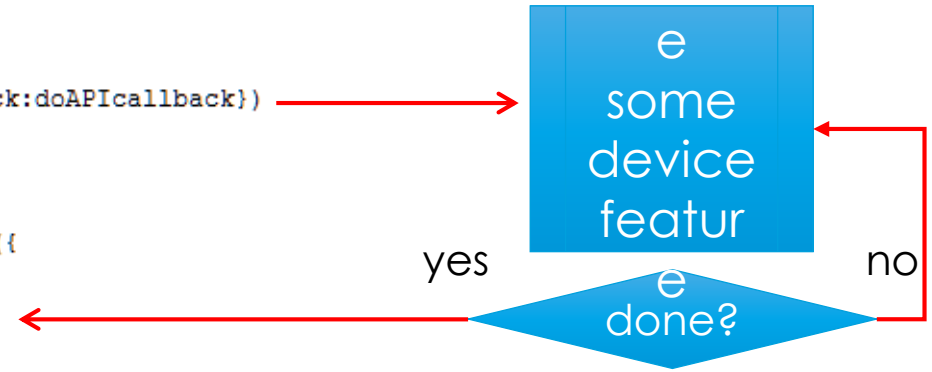
```
function doAPI(){

    someAPI.do({callback:doAPIcallback})

}

function doAPIcallback({

    //do something
}
```

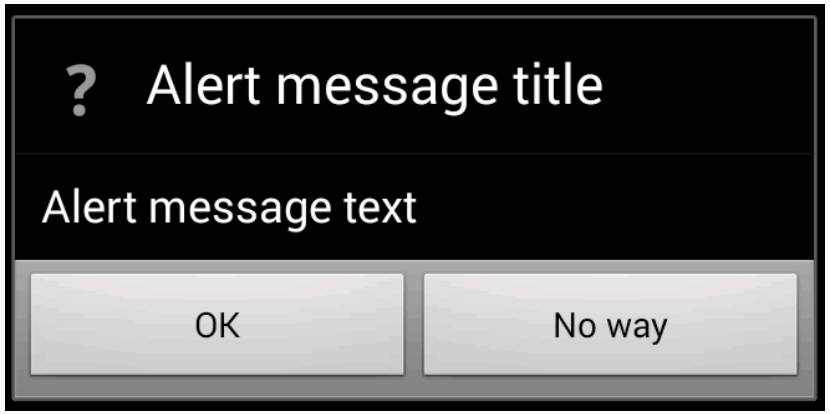


- Note: Some API calls invoke the callback for steps in the process rather than only upon completion

Alert API

- You can also specify a callback function that will be called when the user dismisses the dialog by pressing any button
- The callback will be passed 1 Boolean value: true if "yes" button is clicked, false if not
- For example:

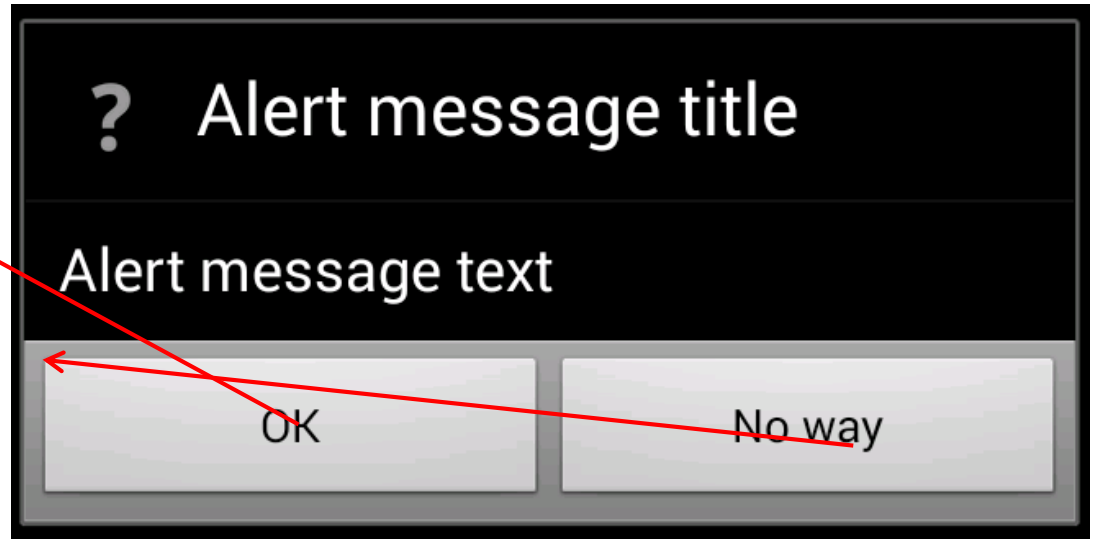
```
kony.ui.Alert(
 {  message     : "Alert message text",
    alertType   : constants.ALERT_TYPE_CONFIRMATION,
    alertTitle  : "Alert message title",
    yesLabel    : "OK",
    noLabel     : "No way",
    alertIcon   : null,
    alertHandler: this.alertcallback}, {});
```



- The callback alertcallback should have a signature with 1 parameter
- For example: Function alertcallback(response) where response will be true or false

Alert Callback Example

```
alertcallback:function(response) {
    if (response == true)
        {
            kony.print("you clicked OK")
        }
        else
        {
            kony.print("you clicked No way")
        }
}
```



- Clicking the first button will set response to be true whether or not there is a second button
- Clicking the second button (if configured) will set response to be false
- Note: whenever specifying a callback, do NOT use quotes in the parameter definition, it is not a string

Geolocation API

- The Geolocation API has methods to retrieve location data from the device
- kony.location.getCurrentPosition(successcallback, errorcallback, positionoptions) - is used to take a single GPS reading
  - successcallback - Receives one parameter containing the location object
  - errorCallback(optional) - Receives one parameter containing the error object
  - Positionoptions(optional) - Used to configure how the location reading is taken
- The object has the following keys:
    - enableHighAccuracy - When set to true will try to get best possible fix
    - timeout - a numerical value specifying, in milliseconds, how long to wait
    - maximumAge - a numerical value specifying how "old" the location data should be before triggering a new reading - basically this is the sample rate in milliseconds
    - Example: {enableHighAccuracy:true, timeout:250, maximumAge:2000}

Geolocation API

- If the getCurrentPosition call is successful, the location object will be returned to the specified callback function; The object has a key, coords that has the following data:
  - latitude – the latitude in degrees
  - longitude – the longitude in degrees
  - altitude – altitude in meters
  - accuracy – accuracy of latitude/longitude reading in meters
  - altitiudeaccuracy – accuracy of altitude in meters
  - heading – direction of travel specified in degrees measured clockwise from due North
  - speed – current ground speed specified in meters/second
  - timestamp – a number representing the time the measurement was taken
- If the call is unsuccessful, then the error object will be returned with -
  - message – the error message
  - code – the error code

error codes from API guide:

Error Code	Description
101	Network Related Error
102	Location API access is denied by user
103	Unknown location or Location not found
104	Unknown error
105	Location Services Disabled

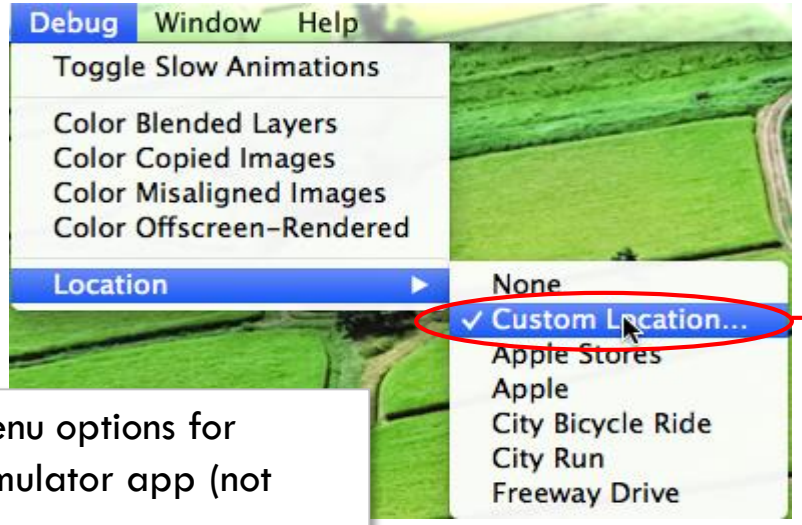
Geolocation API - Example

```
getlocation:function(){
        kony.location.getCurrentPosition(this.gpsworked, this.gpserror,
                {enableHighAccuracy:true, timeout:250, maximumAge:2000});
},
gpsworked:function(location) {
        kony.print("latitude: "+location.coords.latitude);
        kony.print("longitude: "+location.coords.longitude);
        kony.print("altitude: "+location.coords.altitude);
        kony.print("accuracy: "+location.coords.accuracy);
        kony.print("altitudeaccuracy: "+location.coords.altitudeaccuracy);
        kony.print("heading: "+location.coords.heading);
        kony.print("speed: "+location.coords.speed);
},
gpserror:function(err){
        kony.print("message: "+err.message);
        kony.print("code: "+err.code);
}
```

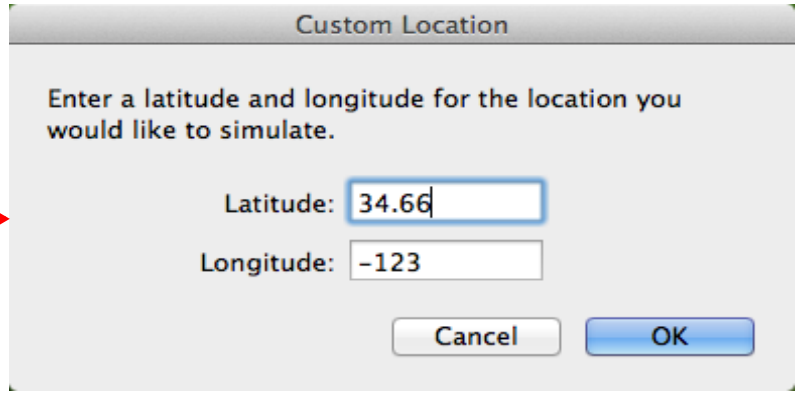
notice the object structure – coords is the object in the location object in the passes in result

Testing GPS on Emulators/Simulators

- Testing GPS on emulators can be tricky
- The iPhone works great and easy for single readings but doesn't work for the watch
- Android works fine but you have to use Telnet to set GPS coordinates
- Let's start by showing you how to simulate GPS coordinates in iPhone:



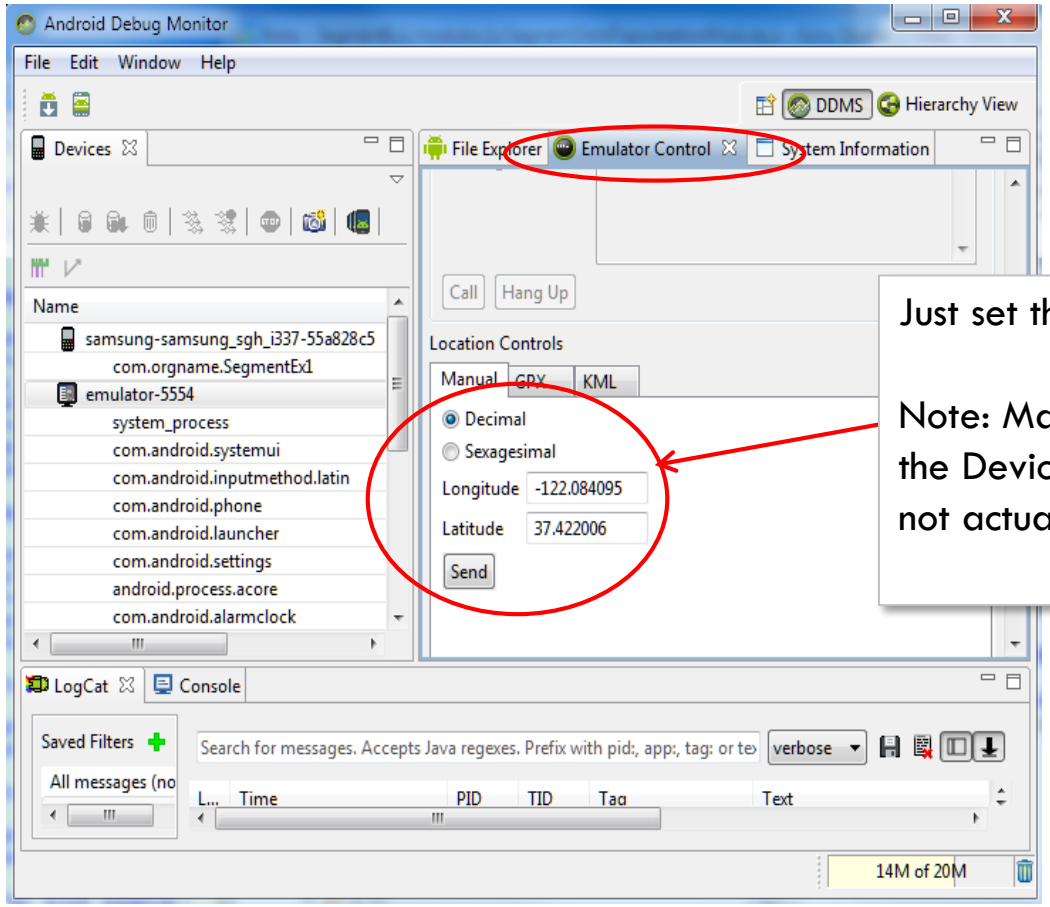
Debug menu options for iPhone Simulator app (not XCode)



enter any valid coordinates – the iPhone will now use those values when reading GPS

Testing GPS on Emulators/Simulators (cont'd)

- For Android, we'll use the monitor.bat tool at:
  <android SDK>\tools\monitor.bat

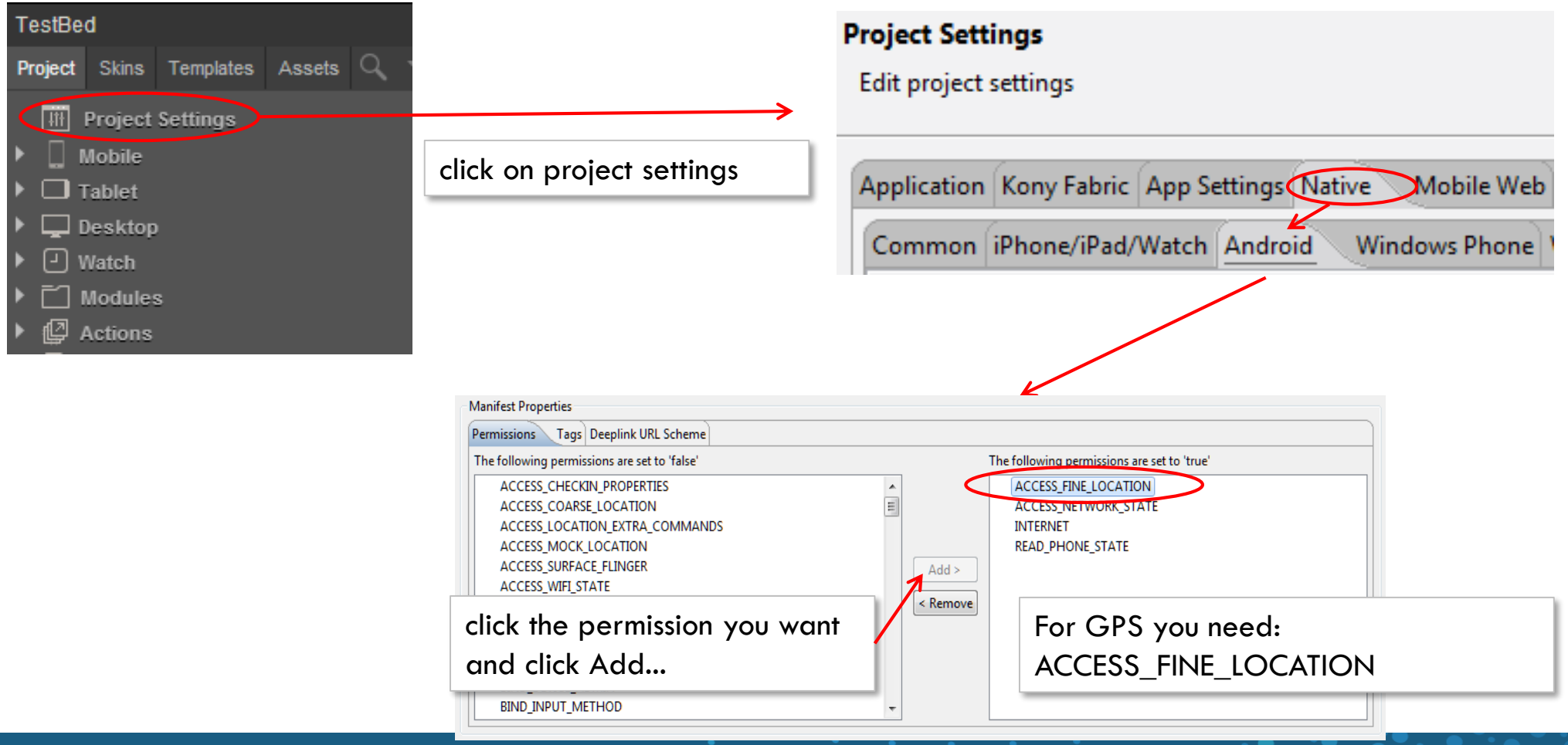


Just set the values you want and click **Send**

Note: Make sure your emulator is selected in the Devices tab, this only works for emulators, not actual devices

Android Permissions and Setup

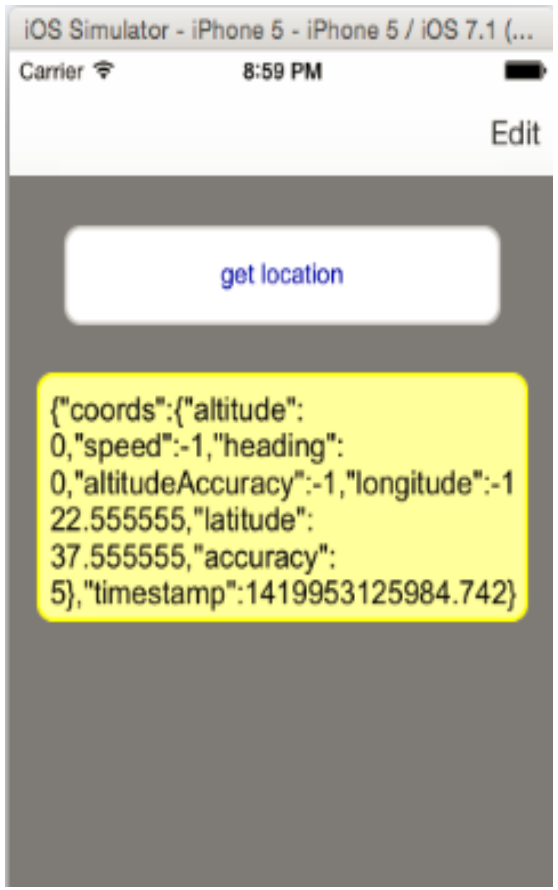
- Many of the device features are protected on Android and you must give the app explicit permissions to access those functions



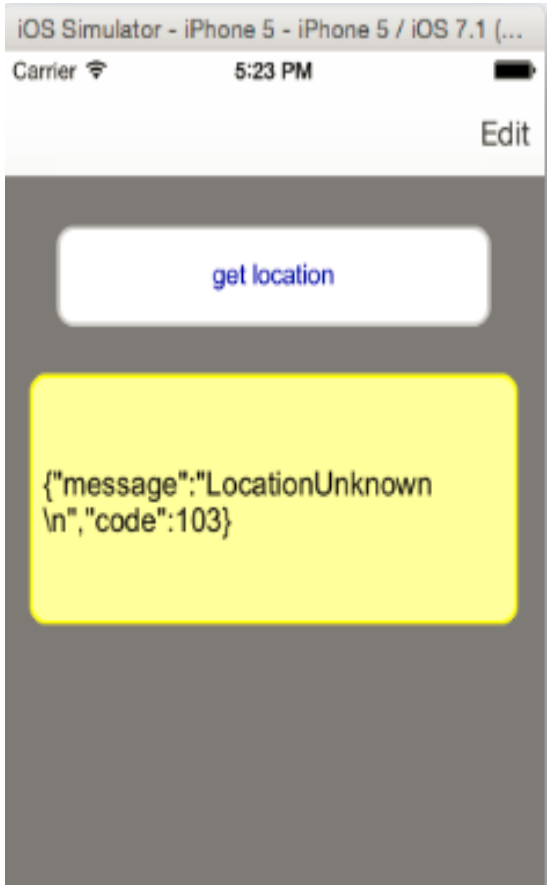
Geolocation - Exercise

- Ok, let's try the geolocation API
- In a new app/screen, have a button and a label
- When you click the button, trigger a location read and display the entire returned result set in that label
- Change the device location values and try again
  - Make sure you do it for both success and failure of the call

success ☺



failure ☹



Operating System API

- This library is implemented through the "os" namespace
- Below are some examples – please refer to the docs for the full list and full details:
  - kony.os.userAgent() - returns a unique identifier for the device used by the Kony Server to identify the device. This unique ID represents the device model and the manufacturer
  - kony.os.freeMemory() - returns the number of bytes available for use
  - kony.os.hasCameraSupport/hasGPSSupport/hasOrientationSupport/hasTouchSupport/hasAccelerometerSupport()- returns true or false depending on whether or not the device supports that feature
  - kony.os.getDeviceCurrentOrientation()- returns a 1 or 0 to represent the current orientation of the device. 0 for landscape and 1 for portrait (doesn't distinguish between upside down or rotated left/right)
  - kony.os.deviceInfo()- returns a lot of information about the device including: name, model, version, width/height, device id, etc.

Operating System API - Example

- Here is a very simple example running on the iPhone simulator

```
kony.print("touch? - "+kony.os.hasTouchSupport());
kony.print("accelerometer? - "+kony.os.hasAccelerometerSupport());
kony.print("camera? - "+kony.os.hasCameraSupport());
kony.print("orientation? - "+kony.os.hasOrientationSupport());

kony.print("current orientation: "+kony.os.getDeviceCurrentOrientation());
kony.print(kony.os.toCurrency(44.33));
kony.print(kony.os.userAgent());
kony.print(kony.os.deviceInfo());
```

results in xCode console:

```
2013-07-26 10:47:33.762 HelloWorld[25054:400b] touch? - true
2013-07-26 10:47:33.763 HelloWorld[25054:400b] accelerometer? - true
2013-07-26 10:47:33.763 HelloWorld[25054:400b] camera? - false
2013-07-26 10:47:33.763 HelloWorld[25054:400b] orientation? - true
2013-07-26 10:47:33.763 HelloWorld[25054:400b] current orientation: 0
2013-07-26 10:47:33.764 HelloWorld[25054:400b] $44.33
2013-07-26 10:47:33.764 HelloWorld[25054:400b] iPhone
2013-07-26 10:47:33.765 HelloWorld[25054:400b] { deviceid : 16b50d18771ce7f6e15094ca5ebc8f0f,
hastouchsupport : true, hasorientationsupport : true, customedeviceid :
16b50d18771ce7f6e15094ca5ebc8f0f, hasgps : false, osname : i-phone, hasaccelerometer : true,
customdeviceid : 16b50d18771ce7f6e15094ca5ebc8f0f, hasCamera : false, osversion : 6.1,
version : 6.1, deviceWidth : 320, model : iPhone Simulator, name : iPhone, deviceHeight :
480,  }
```

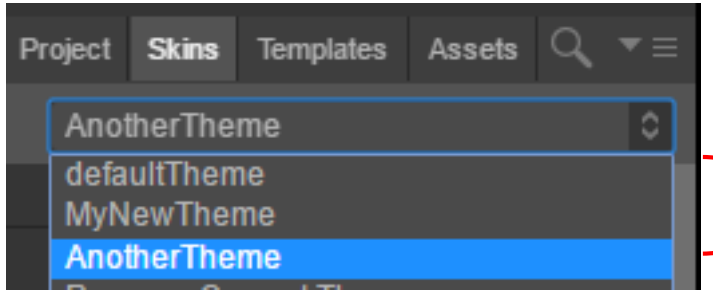
note the features supported here also

Application API

- The Application API has a lot of methods that help control the application programmatically
- Below are some examples - please refer to the docs for the full list and full details
  - kony.application.openMediaUrl(URL) – opens the specified http URL string that points to an audio/video file. This method ASSUMES that the media can play on the device.This API is not applicable on SPA
  - kony.application.openURL(URL) – opens the native browser with the provided URL, and the application goes into background. For web pages, it needs the http:// for a completely qualified URL
  - kony.application.exit() – is used to shut down the app. This is non-standard functionality for many devices, but is a GREAT debugging tool if you need to test your app startup functionality
  - kony.application.getCurrentForm()/kony.application.getPreviousForm() – will return the form

Theme API

- The Theme API allows you to manage (query, create, delete, switch, etc.,) themes programmatically
- Below are some examples - please refer to the docs for the full list and full details
  - kony.theme.getAllThemes()- returns an array of all available themes (specified by their string ID name - what you see in the IDE)
  - kony.theme.setCurrentTheme(themeid, successcallback, errorcallback)- allows you to apply the theme specified by it's string themeid
    - kony.theme.getCurrentTheme() - returns the current theme's string ID
- Let's look at an example:
  - Consider an app that has these themes:



our app shows 3 themes: MyNewTheme, AnotherTheme and defaulttheme.

Theme API (cont'd)

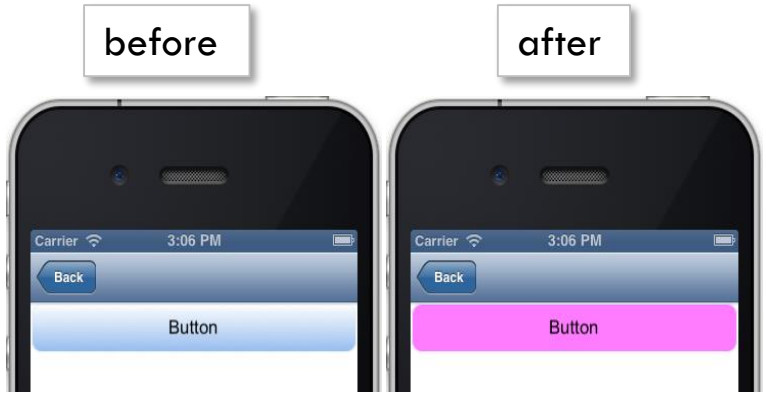
- The following code will print out all the themes on the device, the current theme and then switch to the AnotherTheme

```
apiTest:function(){
    var themelist = kony.theme.getAllThemes();
    var currenttheme = kony.theme.getCurrentTheme();
    kony.print(themelist);
    kony.print(currenttheme);
    kony.theme.setCurrentTheme("AnotherTheme",this.successcallback,this.errorcallback)
}

successcallback:function(passparams) {
    kony.print("success!");
}
```

- here are the logs and before/after screenshots:

```
2013-07-26 14:22:10.919 HelloWorld[27357:400b] [ AnotherTheme ,default ,MyNewTheme , ]
2013-07-26 14:22:10.919 HelloWorld[27357:400b] default
2013-07-26 14:22:10.920 HelloWorld[27357:400b] success!
```

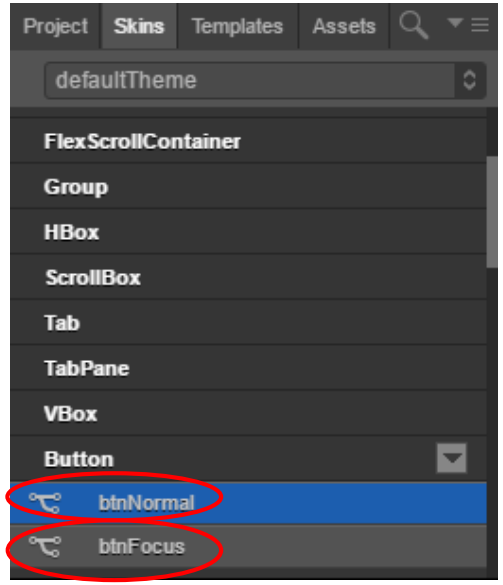


Dynamic Skinning

- Changing the theme applies to all the skins - what if we only want to change one widget's skin? Dynamic skinning!
- Just like themes, changing a skin is nothing more than setting the appropriate skin property to the skin you want
- Each widget has a skin property but may also have other skin properties
- You can dynamically assign skins to ANY skin property for ANY widget
- Let's look at an example:
  - to change a button's skin:

    this.view.btnHello.skin = btnNormal;
  - to change a button's focus skin:

    this.view.btnHello.focusSkin = btnFocus;
  - Note: you can put the skin name in quotes or not, it doesn't matter

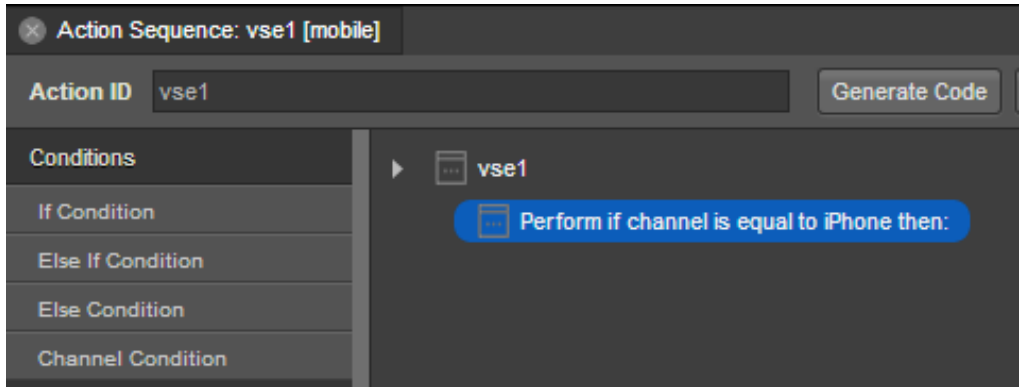


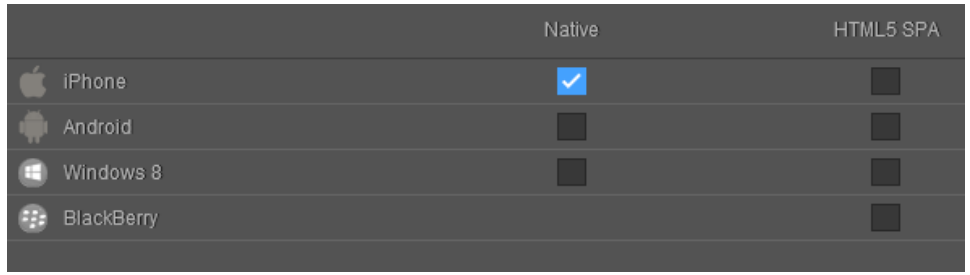
Channel Condition

- There are times when you need to write code that applies to only a specific platform
- You can use the kony.os.deviceInfo() and use that in an IF statement, but the statement will be compiled for every platform and be part of the code
- ifdef is a compiler directive to selectively compile code only for a certain platform
- Here are all the commands (the working example on the next slide will clear it up):
  - `//#define <identifier>` - creates a new identifier that we can use in an ifdef
  - `//#undef <identifier>` - removes an identifier. It will no longer work for an ifdef
  - `//#ifdef <identifier>` - creates a pre-compiler IF statement for that identifier
  - `//#ifndef <identifier>` - checks for an identifier and if NOT present, creates a pre-compiler IF statement
  - `//#else` – the ELSE in the pre-compiler IF statement
  - `//#endif` – the END in the pre-compiler IF statement

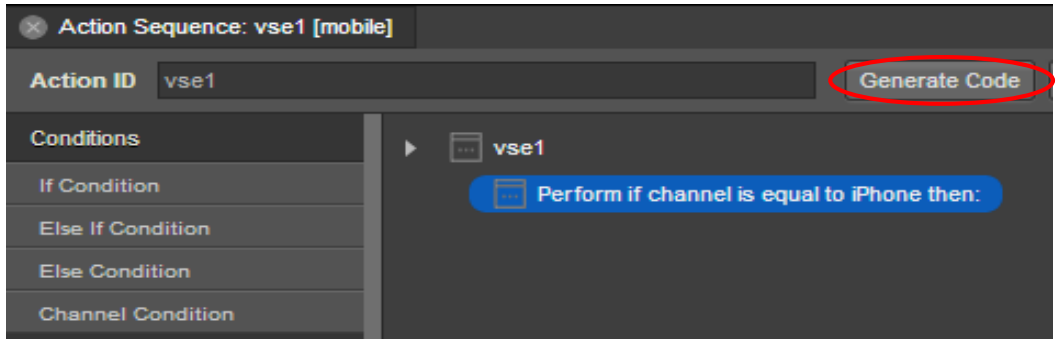
Channel Condition (cont'd)

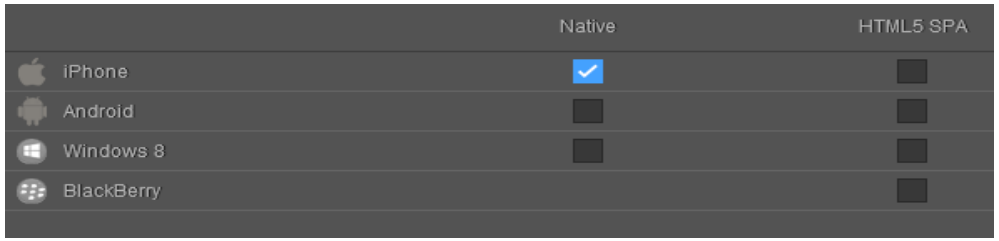
- There are a ton of pre-defined device identifiers that the system already uses:
  - Devices: winphone8, winmobile, android, iphone
  - SPA: spawinphone8, spabbnth, spabb, spaip, spaan, spawindows, spaipad, spatabandroid, spatabwindows
  - Tablet: windows8, ipad, tabrcandroid
- The easiest way to generate this list is to add a pre-processor in the event editor (choose what you want) and then, generate to see the code:





Channel Condition (cont'd)





- It'll all make sense when you see the example on the next slide…

Channel Condition Example

- For our example, we'll use a screenshot of the code to show how it's formatted (note the preprocessor statements are treated as comments in green):

```
var teststring = "initial value"

//#ifdef iphone
    teststring = "changed for iPhone only";
    //#define expensivephone
//#endif

//#ifdef android
    teststring = "changed for android only";
    //#define expensivephone
//#endif

//#ifdef expensivephone
    teststring = "changed by an EXPENSIVE phone";
//#endif

//#ifndef expensivephone
    teststring = "changed by a reasonable phone";
//#endif
```

this //#ifdef...//#endif block will ONLY be compiled for iPhone

Note: we also create a NEW identifier called expensivephone

this block is the same as above for android

if it's an iphone or android, then this identifier got defined so it will do this block

if it's NOT an iphone or android, then this identifier didn't get defined so it will DO this block because it's an ifndef

Channel Condition Example (cont'd)

- When we look at the generated code, we can see what happened…
- This code was compiled for iPhone, Android, and Windows Phone 8:

iPhone code

```
var teststring = "initial value"
teststring = "changed for iPhone only";
teststring = "changed by an EXPENSIVE phone";
```

Note: we'll look into where you can find this generated code a bit later on…

Android code

```
var teststring = "initial value"
teststring = "changed for android only";
teststring = "changed by an EXPENSIVE phone";
```

Windows Phone 8 code

```
var teststring = "initial value"
teststring = "changed by a reasonable phone";
```

- Now, there is no overhead in the code that was generated for each platform
- Defining new identifiers is better than complex ifdef test conditions

Skinning with ifdef - Exercise

Let's practice some of this stuff:

- Create a new screen that has a button on it
- Create 2 new button skins: a red one and a green one
- Assign the button the default skin (just not red or green)
- When the button is clicked:
  - If it's an iPhone, change the button skin to red
  - If it's an Android, change the button skin to green
  - If the skin is already red or green, exit the app

Store API - Overview

- There are times when you need to store data on the device
- For example:
  - Store an airline boarding pass barcode
  - Store login credentials for "remember me" functionality

- The Store API allows the users to store and retrieve data in persistent store on the device
  - This is the code behind the DataStore keys we talked about in the services tab

- This storage technique uses simple key-value pairs to store and retrieve data
- The Store API lets us store, retrieve and clear values
- Let's look at how this works…

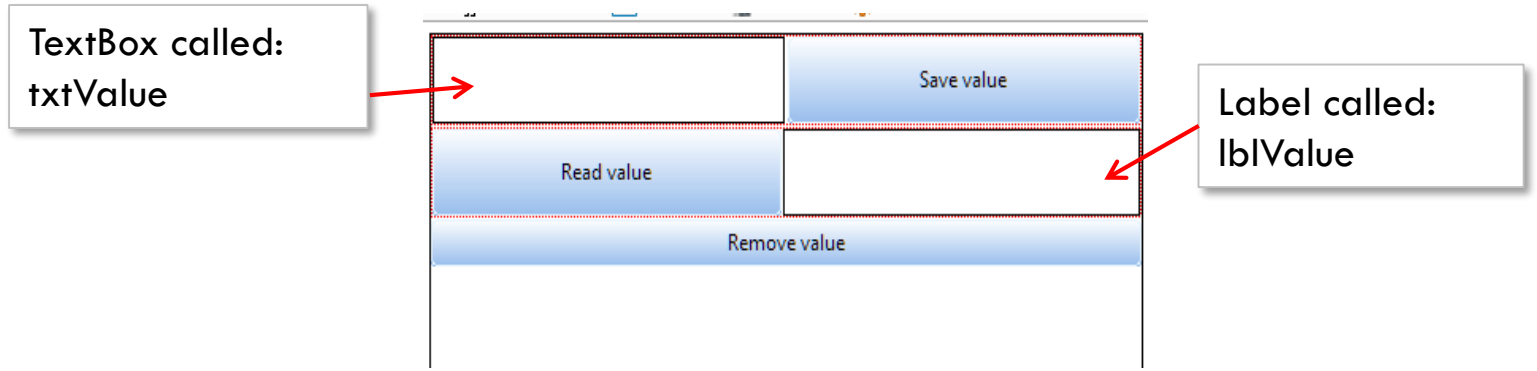
Store API - Overview

- kony.store.setItem(key, value)- will store the value for the specified key. If that key already exists, the value is updated. If not, the key is created and the value stored
  - key - specifies the key name as a string
  - value - specifies the value to store.
  - The value can be any simple type (Boolean, number, string) or complex type (array, JSON object, etc.)
  - There is no return value
- kony.store.getItem(key) - returns the value stored for that key. If that key doesn't exist, null is returned
  - key - specifies the key of the data to be retrieved
  - Returns the data saved using the specified key or null if key doesn't exist
- kony.store.removeItem(key) - removes the item identified by the key, if it exists.
  - key - Specifies the key name for which the item needs to be removed and there is no return value

Store API - Example

- Here is an example of saving, reading and deleting values stored using the Store API:

```
savevalue:function(){
        kony.store.setItem("test123", this.view.txtValue.text);
},
readvalue:function(){
        this.view.lblValue.text = kony.store.getItem("test123");
},
Removevalue:function(){
        kony.store.removeItem("test123")
}
```



Store API (cont'd)

- kony.store.length() - returns the number of keys in local storage

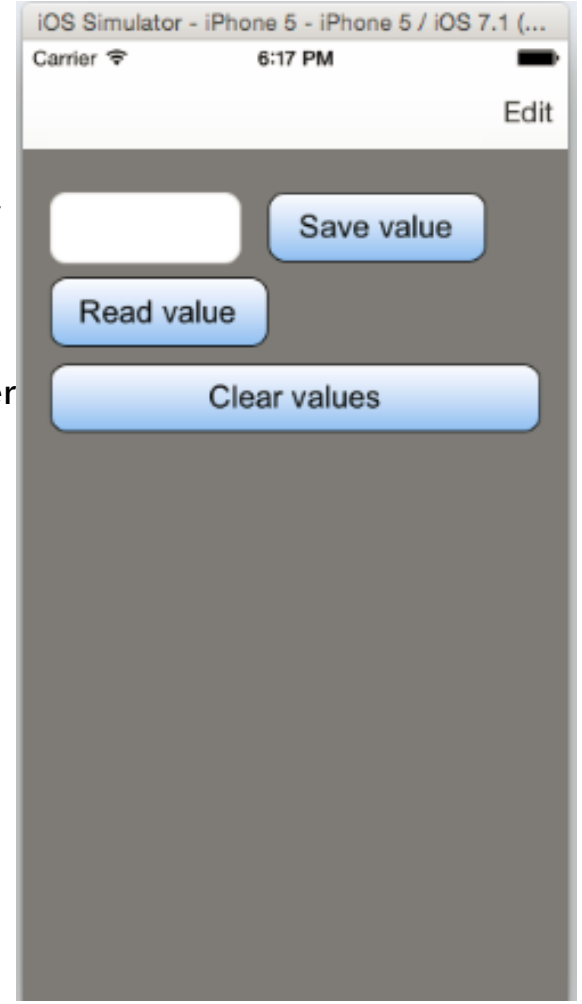
- kony.store.key(index) - keys are given an index when created

- Regardless of the value, keys are stored in the order created. You can retrieve a key at a given index using this method. This is a 0-based index!
  - index - Specifies the index for which the key name is to be returned.
  - Returns the key, as a string, or null if there is no key at that index

- kony.store.clear() - wipes out all the data for all the keys. The local storage will be empty after this call
  - No return values for this API

Store API - Exercise

- Build a screen that looks something like as shown:
- Here's how it works:
  - Clicking the Save value button will store whatever was typed in using some key
  - Each time you click Save value, it stores whatever was typed in using a NEW key
    - You are effectively creating a "stack" of key-value pairs
  - Clicking the Read value button will:
    - read the key AND value from the top of the "stack" i.e., it will show the LAST value enter formatted as <key>:<value>
    - it will also remove this key from the Store
    - if there are no keys, display a nice message
  - Clicking the Clear values deletes all keys

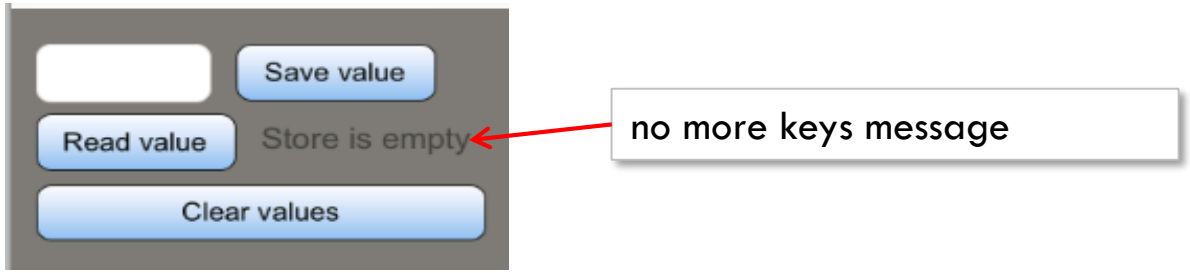


Exercise - Store API (cont'd)

- Let's look at an example:
  - Enter "1111" and click Save value and then enter "2222" and click Save value again
    - you should have 2 key-value pairs stored in the local storage
  - Click Read value and you should see:

  - Click Read value again, and you'll see:

  - Clicking again should show:



Animation API

Animation API

- Go to Actions exercises. Click Generate Code of the Move animation action.
- Do you see something like this?

```
this.view.Button05970073bc7df4a.animate(
      kony.ui.createAnimation(
              {"100":
                      {"left":"50dp",
                       "stepConfig":{"timingFunction":kony.anim.EASE}}}),
      {"delay":0,"iterationCount":1,"fillMode":kony.anim.FILL_MODE_FORWARDS,"duration":0.25},
      {"animationEnd" : function(){}})
```

using the animate() widget method that method takes 3 parameters:

#1

- #1 - Animation object - Defines the type of animation

#2

- #2 - Animation configuration - Defines duration, repetition and final state

#3

- #3 - Callbacks

Animation Object

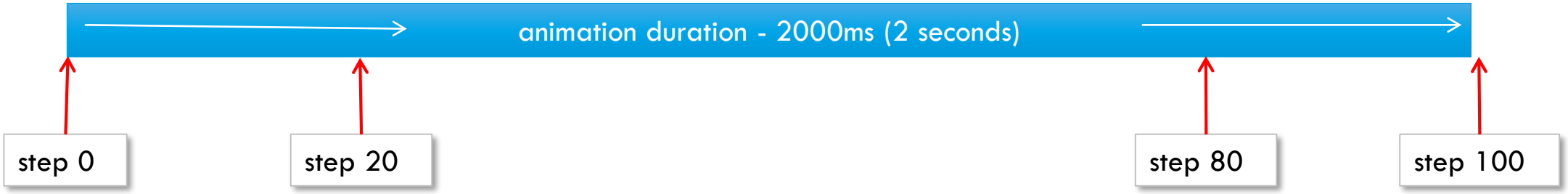
- Animation object is created using kony.ui.createAnimation(animationDefinition)
- animationDefinition is a collection of animation steps

- Animation Steps
  - In Action Editor, we can mention only one step
  - In the Animation API, you can create as many as 100 steps for each action
  - Each animation object consists of 1 or more steps

- 2 steps have special meaning:
  - Step 0: this is the initial step - you might not always have a step 0
  - Step 100: this is the final step - you WILL always have a step 100

Animation Definition

- Let's look at an example where the animation object has steps at 0, 20, 80 and 100 and duration 2000ms
  - Step 0 defines the initial state of the widget
    - May involve changes but they will be applied instantly at this step
  - Step 20 defines the animation for the first 20% of the time - 400ms
  - Step 80 defines the animation for the next 60% of the time - 1200ms
  - Step 100 defines the animation for the remaining 400ms



Animation Definition

- Here is the syntax for a step:

```
<step #>: { "<property>":<property value>,"stepConfig":{"timingFunction":<timing curve>}}
```

- First part defines the target widget properties
  - Positional properties can be used for move animation
  - Dimensional properties can be used for scaling
  - Widget skin properties like background color and opacity can also be set
- Second part defines the timing curve of animation
  - kony.anim.EASE
  - kony.anim.LINEAR
  - kony.anim.EASIN_IN
  - kony.anim.EASIN_OUT
  - kony.anim.EASIN_IN_OUT

Animation Object

- Let's say the widget's starting left value is 20Dp
- You want to move it to the right by 10Dp in each step
- The number steps are 4 and they are 0,20,80,100
- Here is the animation object creation code

```
kony.ui.createAnimation(
        {0:{"left":"20dp", "stepConfig":{"timingFunction":kony.anim.LINEAR}},
        20:{"left":"30dp", "stepConfig":{"timingFunction":kony.anim.LINEAR}},
        80:{"left":"40dp", "stepConfig":{"timingFunction":kony.anim.LINEAR}},
        100:{"left":"50dp", "stepConfig":{"timingFunction":kony.anim.LINEAR}}})
```

Animation Object

- Each step can set more than one property
- If you add the Top property along with left you can do it
- Here is the animation object creation code

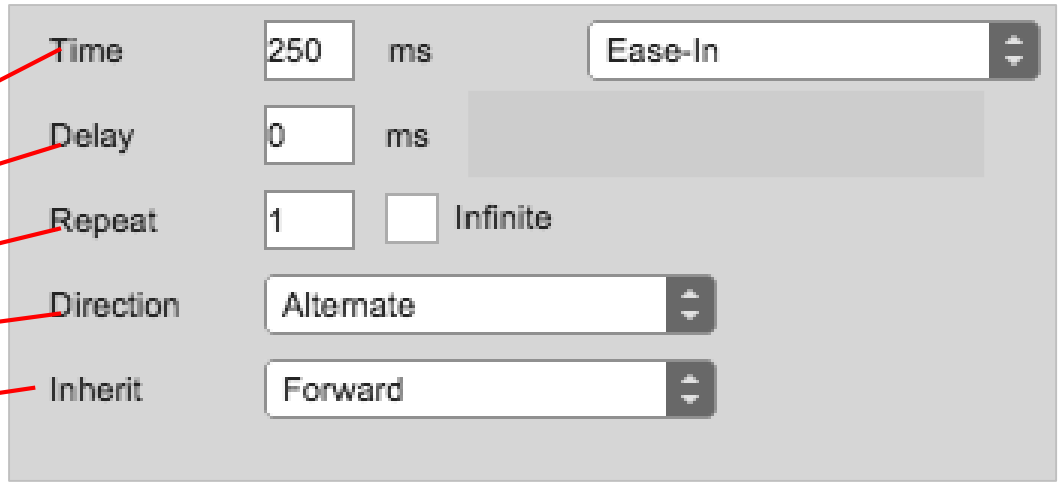
```
kony.ui.createAnimation(
        {0:{"left":"20dp","top":"20dp","stepConfig":{"timingFunction":kony.anim.LINEAR}},
         20:{"left":"30dp","top":"30dp","stepConfig":{"timingFunction":kony.anim.LINEAR}},
         80:{"left":"40dp","top":"40dp","stepConfig":{"timingFunction":kony.anim.LINEAR}},
         100:{"left":"50dp","top":"50dp","stepConfig":{"timingFunction":kony.anim.LINEAR}}})
```

- The stepConfig is optional - the default is kony.anim.EASE

Animation Configuration

- Remember our animate method syntax:
  - <widget>.animate(animationObject, configObject, callbackObject)
  - We just talked about animationObject, let's talk about configObject now:
  - configObject will be a JSON object that contains all the animation timing and configuration items
    - The syntax is:

```
{"duration":<duration in sec>,
 "delay":<delay in sec>,
 "iterationCount":<repeat>,
 "direction":<direction>,
 "fillMode":<fill mode>}
```



Animation Configuration

- Like the animation object, the timing configuration object also uses Kony constants to represent values you picked in the Action editor:
  - direction can be one of the following values:
    - kony.anim.DIRECTION_NONE (the default value)
    - kony.anim.DIRECTION_ALTERNATE
  - fillMode can be one of the following values:
    - kony.anim.FILL_MODE_NONE (the default value)
    - kony.anim.FILL_MODE_FORWARDS
    - kony.anim.FILL_MODE_BACKWARDS
    - kony.anim.FILL_MODE_BOTH

Animation Configuration

- fillMode specifies the state of the widget after the completion of animation
  - kony.anim.FILL_MODE_NONE (the default value) - widget returns to it's pre-animation state and ignores anything set in the animation
  - kony.anim.FILL_MODE_FORWARDS - widget inherits the final values of the animation as it's state after the animation is done
  - kony.anim.FILL_MODE_BACKWARDS - widget returns to it's step 0 state
    - that's right - even if step 0 represents a change to the pre-animation widget state, that step 0 will be the final state of the widget after the animation
  - kony.anim.FILL_MODE_BOTH - this is where the INITIAL state of the widget is set to step 0 immediately AND when the animation is done, the widget inherits the final values of the animation state after the animation is done
    - This is an odd case - the best way to see this is put a big delay on your animation - step 0 will happen BEFORE the delay, then the delay, then the other steps in the configuration

Animation Callbacks

- Remember our animate method syntax again:
  - <widget>.animate(animationObject, configObject, callbackObject)

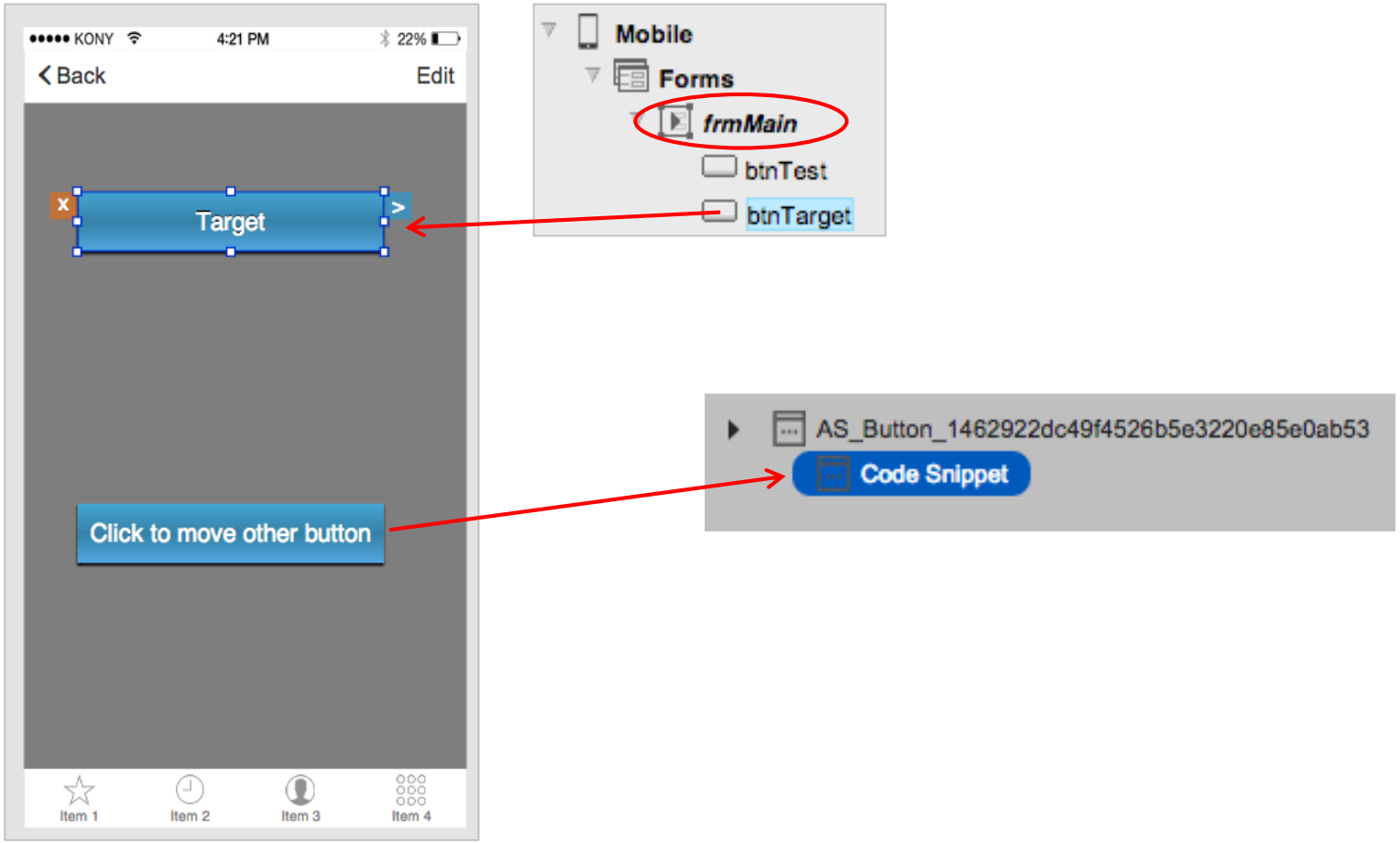
  - callbackObject is a JSON object that specifies what functions to run when the animation starts and when it ends.
    - Syntax

      {"animationStart":<function name>,"animationEnd":<function name>}
    - Example

      {"animationStart":myStartFunc,"animationEnd":myEndFunc}
    - "do nothing" entry looks like this:

      {"animationEnd":function(){}}

Animation in Code Exercise



Animation in Code Exercise (cont'd)

- Let's do a 4 step animation on the target button
  - Step 0 - Set left=0 and top=0 and change the background color to "ff7f50"
  - Step 25 - Set left=50Dp and top=50Dp, opacity to 0.75
  - Step 50 - Set the opacity to 0.50
  - Step 100 - Set the opacity to 0.0
  - Duration is 3 seconds, all steps linear and the fillMode will be NONE (to reset everything)

Animation in Code Exercise (cont'd)

- Here is the snippet showing the button's animate method and 3 parameters:

```
this.view.btnTarget.animate(
        kony.ui.createAnimation(
      {"0":{"left":"0dp","top":"0dp","backgroundColor":"ff7f50",
                    "stepConfig":{"timingFunction":kony.anim.LINEAR}},
    "25":{"left":"50dp","top":"50dp","opacity":0.75,
                    "stepConfig":{"timingFunction":kony.anim.LINEAR}},
    "50":{"opacity":0.5,"stepConfig":{"timingFunction":kony.anim.LINEAR}},
    "100":{"opacity":0.0,"stepConfig":{"timingFunction":kony.anim.LINEAR}}}),
  {"delay":0,"iterationCount":1,"fillMode":kony.anim.FILL_MODE_NONE,"duration":3.0},
  {"animationEnd":function(){}});
```

#1

#2

#3

Animation in Code Exercise (cont'd)

- Once you have it working, change the delay to 3 seconds, change the fillMode to BOTH and retest
  - You should notice Step 0 happens immediately, then the delay, then the rest of the animations happen to end up on the Step 100 state

- Change the fillMode to FORWARD and retest
  - You should now notice the delay happens first, then step 0, and then the rest of the animation

Transform Object

- Transform object lets you do a composite animation
- That is move, and/or scale and/or rotate a widget
- Creating Transform object

```
var xfrm = kony.ui.makeAffineTransform();
```

Transform API

- Here is how to configure each type of animation on that xfrm transform object:
  - xfrm.translate(x,y) is used to move the widget x Dp and y Dp where positive values move the widget to the lower left and negative values move the widget to the upper right. All values are in Dp
    - For example: xfrm.translate(100,100) will move the widget 100Dp down and 100Dp to the right
      - Note this is different than setting the top or left property since that represents a final value - translate is used to move the specified amount from wherever the widget is currently
  - xfrm.rotate(deg) is used to rotate the widget to degrees. The same rules we had for the rotate action apply - specify values that result in < 180 degrees in one step
    - Positive turns counter - clockwise and negative values turns clockwise
    - For example: xfrm.rotate(45) rotates the widget to 45 degrees in a counter-clockwise direction from it's current rotation position

Transform API

- `xfrm.scale(scaleX,scaleY)` is used to scale the widget's dimensions by the value specified where `scaleX` scales the widget width and `scaleY` scales the widget height
  - The `scaleX` and `scaleY` values will just be multiplied by the widget's width and height to get the final value
  - For example to shrink a widget by 50% on both height and width, we have:

    ```
    xfrm.scale(0.5,0.5);
    ```

  - To make a widget twice as big we'd have: `xfrm.scale(2,2);`
- The different transformations can be used in conjunction with each other for a particular step
  - For example, to rotate, scale AND translate (move) a widget, we can do the below:

    ```
    var xfrm = kony.ui.makeAffineTransform();
    xfrm.translate(100,100);
    xfrm.scale(2,2);
    xfrm.rotate(45);
    ```

  - That `xfrm` object now will do all 3 animations in the step it uses

Exercise

- So, we can now create a transform and replace our original step 50 with our new transform:

```
var xfrm = kony.ui.makeAffineTransform();
xfrm.translate(100,100);
xfrm.scale(2,2);
xfrm.rotate(45);
this.view.btnTarget.animate(
        kony.ui.createAnimation({"0":{"left":"0dp","top":"0dp","backgroundColor":"ff7f50",
        "stepConfig":{"timingFunction":kony.anim.LINEAR}},
    "25":{"left":"50dp","top":"50dp","opacity":0.75,
                "stepConfig":{"timingFunction":kony.anim.LINEAR}},
    "50":{"transform":xfrm,"stepConfig":{"timingFunction":kony.anim.LINEAR}},
    "100":{"opacity":0.0,"stepConfig":{"timingFunction":kony.anim.LINEAR}}}),
  {"delay":0,"iterationCount":1,"fillMode":kony.anim.FILL_MODE_NONE,"duration":3.0},
  {"animationEnd":function(){}});
```

Exercise

- Look at how we use that transform in step 50
  - We just specify the step "property" to be `"transform"` and then the value is our `xfrm` object with the 3 animations

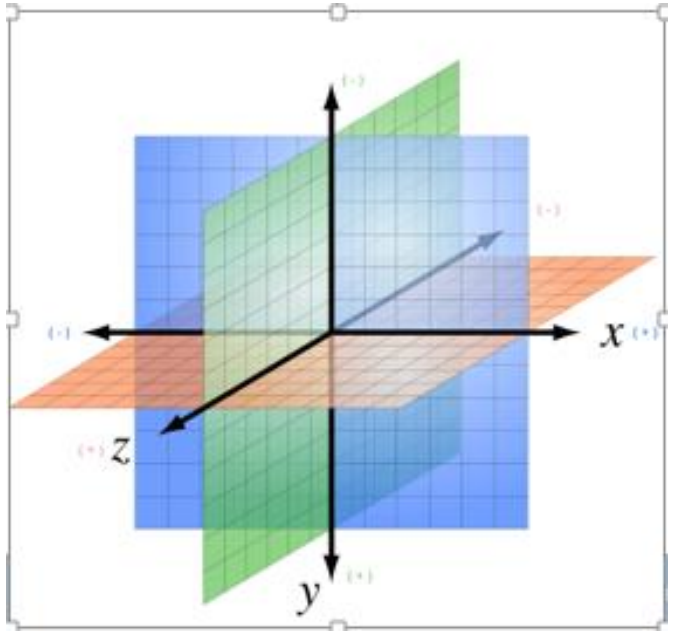
- Ok, let's do it again!

- Repeat the example I just showed you by replace step 50 with a transformation

- Now is your time to experiment with this API

3D Animations

- Before we get into 3D transformations, lets understand x,y and z axis directions
  - X-axis is towards the right of the widget
  - Y-axis is towards the bottom of the widget
  - Z-axis is towards the viewer i.e., coming out of the screen
- Below image showcases all the x,y and z axis directions



3D Transformations

- 3D Transformations can be achieved with the following API's
  - rotate3D
  - translate3D
  - scale3D
  - setPerspective
- Supported platforms
  - iOS
  - SPA
  - Android (only rotate3D)

3D Transformations - Perspective

- The value of perspective determines the intensity of the 3-D effect. Think of it as a distance from the viewer to the object
- The greater the value, the further the distance, so the less intense the visual effect
- Perspective 2000 yields a subtle 3-D effect, as if we were viewing an object from far away
- Perspective 100 yields a tremendous 3-D effect, like a tiny insect viewing a massive object

3D Transformations - setPerspective (cont'd)

- This method sets the perspective and sets the vanishing point at the center of the widget.
- API for setting perspective

  setPerspective(distanceOfViewerToPlane)
  - distanceOfViewerToPlane: The distance between the viewer and object. Always the value of this parameter should be greater than zero
  - The perspective has to be set in combination with other transforms i.e., rotate, translate or scale. The perspective set by itself will not have any effect.
  - For the iOS platform, the value of the distanceOfViewerToPlane parameter should be greater than max (width, height) values of the widget view's frame.
  - For example, if the value of (width, height) is (100, 50), the parameter value should be greater than 100.
  - The effect of this parameter vary visually on different platforms for the same value. The units of the distanceOfViewerToPlane parameter is platform-specific.
  - In the Android platform, when perspective is not specified, the default perspective is applied.

3D Transformations - rotate3D

- This method rotates the widget by angle on the unit directional vector formed by rx, ry, and rz
- API for setting 3D rotation

  rotate3D(angle,rx,ry,rz)
  - angle:Specify the angle in degrees, by which a widget to be rotated around rx, ry, and rz axis.
  - Angle range can be in between 180° to -180° and any value greater or lesser than range will result into platform-specific behavior.
  - rx:Specify the x-axis value on which rotation to happen
  - ry:Specify the y-axis value on which rotation to happen
  - rz:Specify the z-axis value on which rotation to happen
  - The values of rx, ry, and rz should be in the range of 0 - 1. If the (0,0,0) vector is specified, the behavior is platform-specific.
  - In the Android platform, the values between 0 - 1 are not accepted. Only '0' or '1' is accepted.

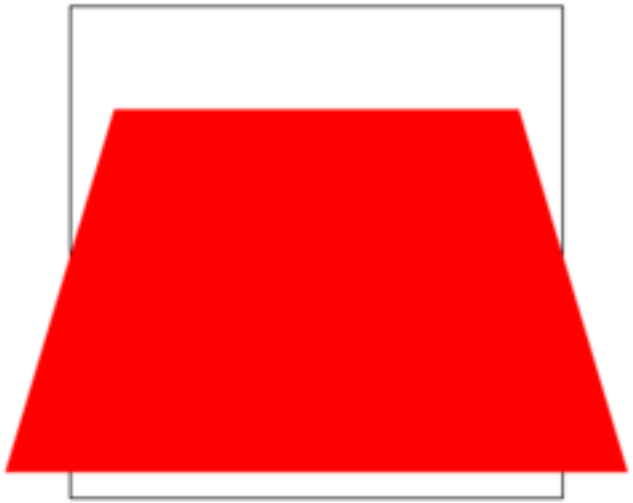
3D Transformations - rotate3D (cont'd)

- Here is the code for the example

```
var transformObject= kony.ui.makeAffineTransform();
transformObject.setPerspective(500);
transformObject.rotate3D(140,1,0,0);
var animationObject = kony.ui.createAnimation(
      {"0":{"left":"0dp","top":"0dp","stepConfig":{"timingFunction":kony.anim.LINEAR}},
      "100":{"transform":transformObject,"stepConfig":{"timingFunction":kony.anim.LINEAR}}});
var animationConfig = {duration:5,fillMode: kony.anim.FILL_MODE_FORWARDS};
var animationCallbacks = {"animationEnd":function(){kony.print("animation END")}};
this.view.flexInside.animate(animationObject , animationConfig, animationCallbacks);
```

3D Transformations - rotate3D (cont'd)

- In the below screenshot, observe that the widget is rotating along x-axis with 3-D effect.



e any effect on z-axis rotation

3D Transformations - translate3D

- This method translate the widget from present location to new location by x, y, z amount
- API for setting 3D translation

  translate3D(tx,ty,tz)
    - tx:Specify the value to be moved in the x direction from present location
    - ty:Specify the value to be moved in the y direction from present location
    - tz:Specify the value to be moved in the z direction from present location
- Note: Perspective will not have any effect on x,y axis translation

3D Transformations - scale3D

- This method scales a widget in three dimensions (x, y, z) coordinate system
- API for setting scale.

  scale3D(sx,sy,sz)

  - sx:Specify the value to be scaled in the x direction
  - sy:Specify the value to be scaled in the y direction
  - sz:Specify the value to be scaled in the z direction
  - Any value with in the 0 - 1 range scales down the widget and the value greater than '1' scales up in the specified directions.
  - The scale3D method should not be applied on zero dimension widgets. If applied, the behavior is undefined
  - In the Android platform, the values between 0 - 1 are not accepted. Only '0' or '1' is accepted
- Note: Perspective will not have any effect on scale

Selection Widgets API

Selection Widgets

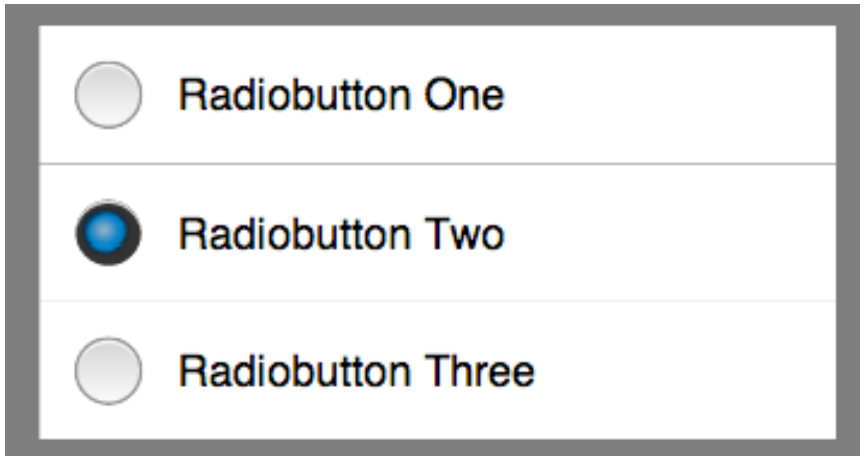
- The following widgets are classified as the Selection Widgets:
  - ListBox
  - RadioButtonGroup
  - CheckBoxGroup

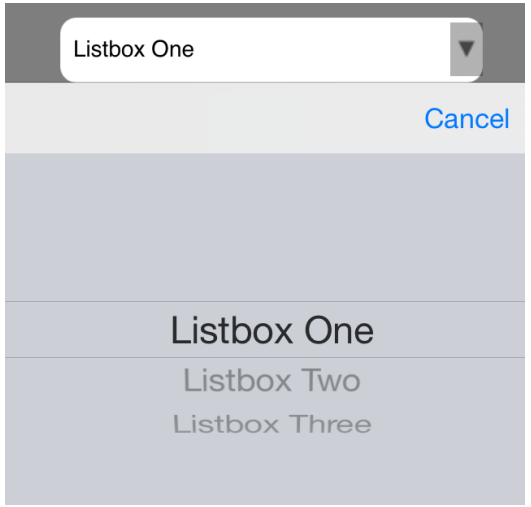
- These widgets help us gather some user Inputs by selecting one or more of the available options that are presented by this widgets

- They also have common methods and properties for getting and setting information

Selection Widgets - UI Review

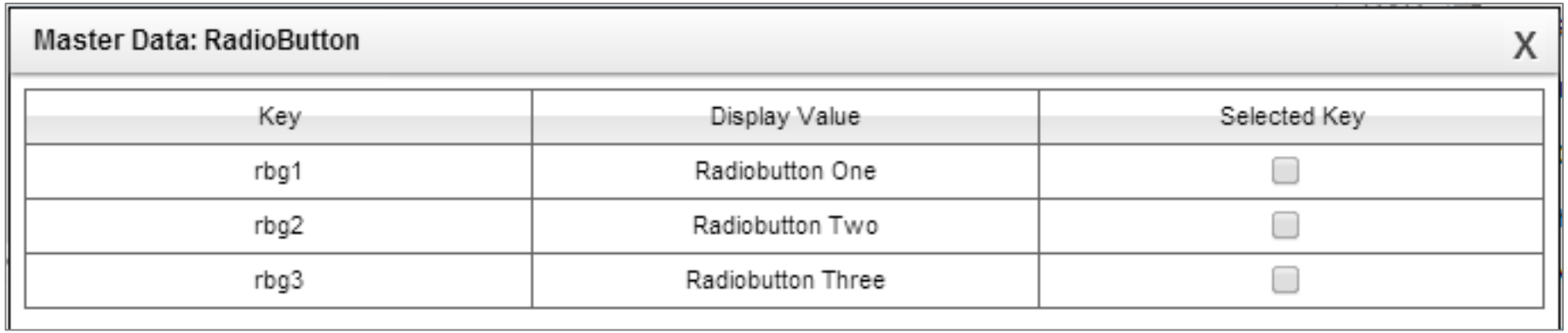
- As we saw in Visualizer, there is a lot of similarities in how these widgets can be configured to offer choices to the user:





Selection Widgets - How to Populate Data

- To populate data to most of these widgets from the Visualizer use the masterData property.
- Using the masterData property dialog we can:
  - Set the Keys and Display Values (what the user sees)
  - Determine default selection(s)



- Checking the Use as test data in preview mode to ONLY use these values during preview
- With all selection widgets, both the Key and the Display Value are available for the selected item(s)

Selection Widgets - Populate Controls

- In JavaScript, we can populate the selection widgets using the masterData property.
- masterData takes sets of arrays where the first value is the key and the second value is the display value.
  - For example:

```
this.view.myListBox.masterData = [
                                    ["a","very much"],
                                    ["b","some what"],
                                        ["c","not so much"],
                                        ["d","not at all"]]
```

- To set a default selection, we'll use the selectedKeys or selectedKey property (depending on if the widget is configured for multi-select)
- selectedKeys takes an JS table of keys and selectedKey takes one item. For example:

```
this.view.myCheckBoxGroup.selectedKeys = ["b","c"]
this.view.myRadioButtonGroup.selectedKey = "a"
```

- Note: there are no update data methods, you'll have to just set masterData to new data to show new/updated values

Selection Widgets - What Was Selected?

- Use the selectedKeyValue property if the widget was set as single select. It returns the array of the selected key-value pair (singular) . If nothing is selected, the return value is null.
  - For example, in our RadioButtonGroup example, I pick "somewhat":

    var selectedItem = this.view.myRadioButtonGroup.selectedKeyValue

    selectedItem --> ["b","somewhat"]

    Get the key with: selectedItem[0]

    Get the value with: selectedItem[1]

- Use the selectedKeyValues property if the widget was set as multi-select. It returns the array of selected key-value pairs . If nothing is selected, the return value is null.
  - For example, in our CheckBoxGroup example, I pick "apple" and "kiwi":

    var selectedItems = this.view.myCheckBoxGroup.selectedKeyValues

    selectedItems --> [["a", "apple"],["d", "kiwi"]]

    Get the first key with: selectedItems[0][0]

    Get the first value with: selectedItems[0][1]

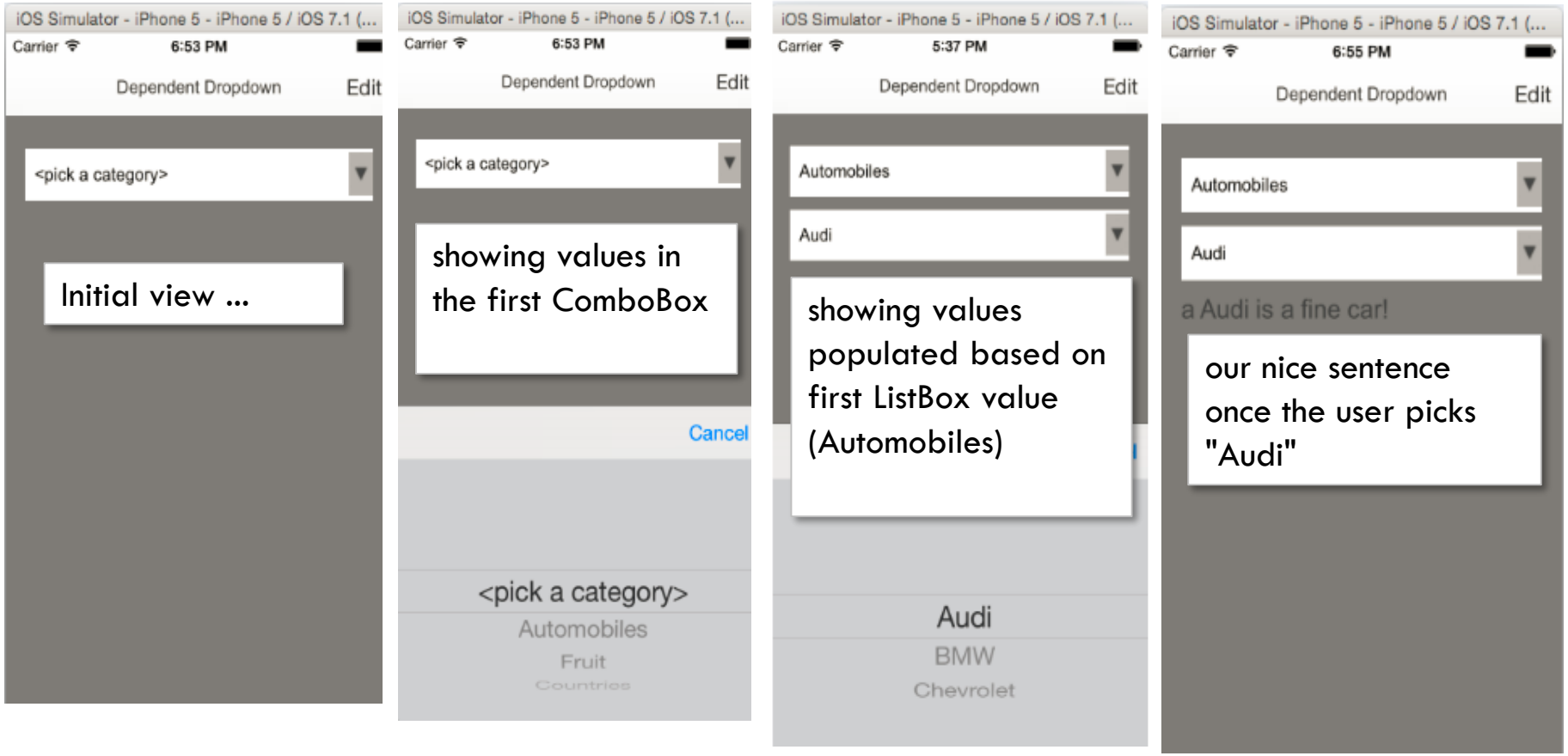
Selection Widgets - onSelection Event

- The selection widgets all have an onSelection event that fires when the user picks a value
- You can do whatever you need to do when a user picks a value
  - On single select widgets, typically the user is done so you can do what you want
  - On multi-select widgets, the event will fire EACH time the user picks an item - they might NOT be done after the first onSelection event
- Note that it's very common to have a "pick a value" item shown by default
  - Showing a selection widget with nothing selected creates an awkward UI - best to show something
  - Always check to see if the user picks that default value because typically this is a "no selection" state

Exercise - Selection Widgets

- Ok, we'll create some dependent dropdowns - this is where there are 2 ListBox widgets. Picking a value from the first will determine what values are shown in the second ListBox - they are dependent on each other.
- In the first ListBox, put at least 3 categories of things. For example, "Automobiles", "Fruit" and "Countries". Do this using the IDE.
  - The first item, selected by default, should say something like "pick a category"
- The second ListBox is initially hidden until the user picks a category (that ISN'T the default "pick a category"). Once the category is picked, show at least items that belong to that category. Do this in code.
  - Don't forget to show the second ListBox!
- When the user picks an item (from the second ListBox), show what they picked in a nice sentence on a label. Let's take a look at how it works...

Exercise - Selection Widgets

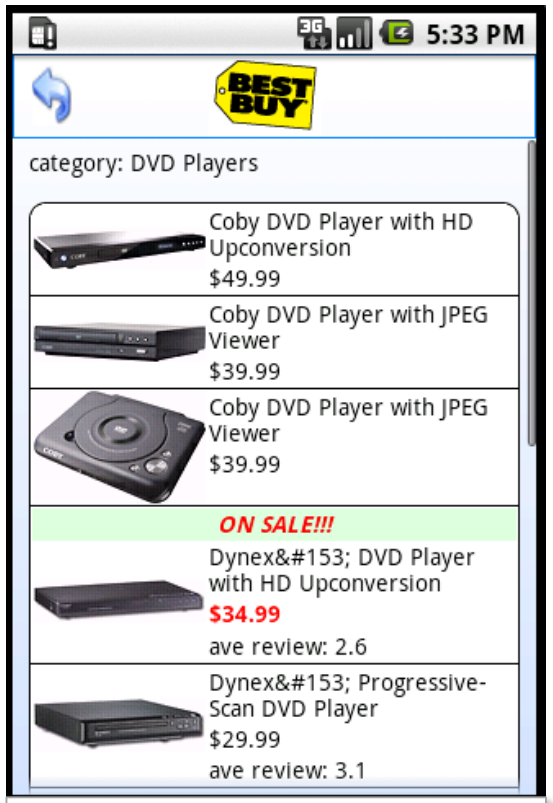


- Make sure that nothing happens when "<pick a category>" is selected
- Hide/show the label and second ListBox appropriately

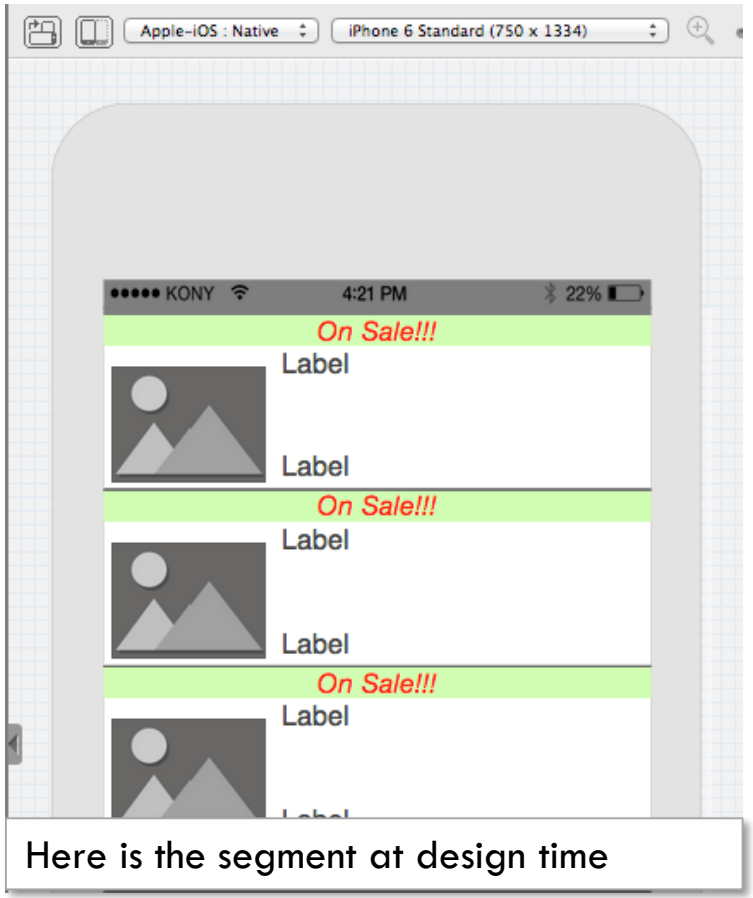
Segment API

Segment - Review

- The Segment acts as a predefined template to populate multiple rows or records of similar type of data at runtime.
- Segments are typically used to show a list that can be:
  - Static: for example, used as a menu
  - Dynamic: to show service or transactional data like a list of products
- For example, here is a screen showing products:



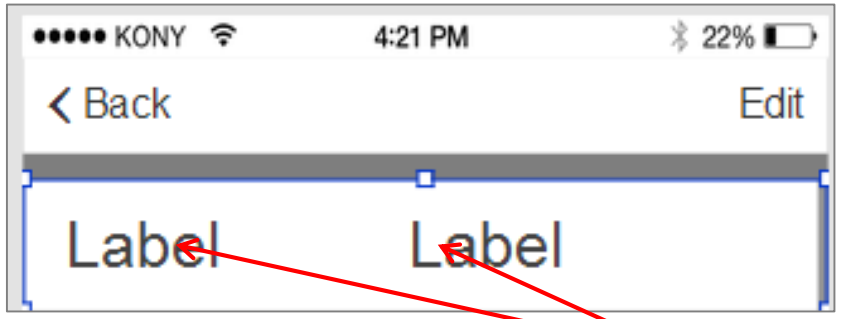
Here is the segment at runtime



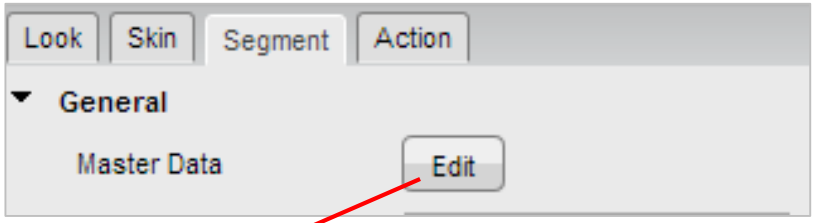
Here is the segment at design time

Segment - Static Data Population

- You can use the Master Data property in Visualizer to populate the segment with data:



Segment property sheet:



A segment with 2 labels: lblAccountName and lblAccountBal

Master Data: segmentAccounts

lblAccountName	lblAccountBal
Checking 494	$110.00
Checking 490	$1900.20
Checking 495	$1200.16
Savings 567	$1600.00

The data for the labels will be displayed in the segment

Segment - Dynamic Data Population

- To set the data dynamically, we need to provide a set of key-value pairs for each "row" of data - each row is a JSON object
- The key should match the widget name and the data will be displayed in that widget
- For the example on the previous slide, our data might look like:

```
mySegData =     [
                {lblAccountName : "Checking 494", lblAccountBal : "$110.00"},
                {lblAccountName : "Checking 490", lblAccountBal : "$1900.20"},
                {lblAccountName : "Checking 495", lblAccountBal : "$1200.16"},      {lblAccountName : "Savings 567",
                lblAccountBal : "$1600.00"}
                ]
```

- Use the setData method of segment API to assign the data to the segment.
  For example: this.view.mySegment.setData(mySegData)
- The segment will update as soon as it's data is set - no need to redisplay the form

- Ok, but what if my data has different keys from my backend data source?

Segment - WidgetDataMap

- You can map ANY dataset to a segment
- To do this, you must set the segment's widgetDataMap property.
- This property maps the data keys to the segment's widgets
- Setting up this must be done BEFORE calling the setData method
- The syntax is: this.view.SEGMENT.widgetDataMap = <Object Array> where:
  - <Object Array> is a collection consisting of key-value pairs where the key is the widget name and the value is the key in the dataset.
- Using our previous example, if the data were:

```
mySegData =     [{acct : "Checking 494", bal : "$110.00"},
                 {acct : "Checking 490", bal : "$1900.20"},
                 {acct : "Checking 495", bal : "$1200.16"},
                 {acct : "Savings 567" , bal : "$1600.00"}]
```

- Our widgetDataMap would be:

  this.view.mySegment.widgetDataMap={lblAccountName: "acct", lblAccountBal: "bal"}
- We could now call our setData method and the segment would know where to get each piece of data and which widget to assign it to.

Segment - Hidden Columns

- Hidden columns of a segment allow you to store more data, for each row, than is displayed in the segment
- When the user selects a segment row, you have access to ALL the data (displayed and hidden) for that row
- For example: you want to show the account name and balance but you also need to store the accountID as a hidden piece of data:
  - Define your segment as we've done before with the 2 labels for account and balance
  - Set up your data to include this new column of data
  - For the selected row, you can now access account name and balance (both displayed to the user) and accountID, the hidden piece of data

Segment - Hidden Columns (cont'd)

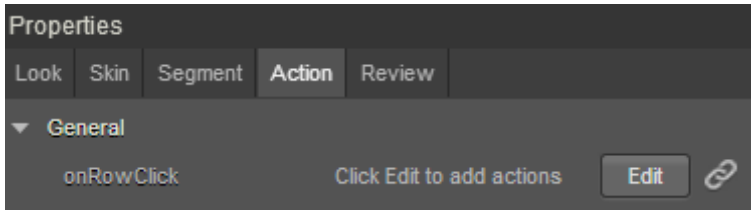
- If you are populating your segment from code, you DON'T need to do anything special
- When populating data in code, you can add as many extra pieces of data as you want!
- First populate the required data into a JS array (with extra data):

```
mySegData =     [{acct:"Checking 494", bal:"$110.00",  accountID:"101"},
                 {acct:"Checking 490", bal:"$1900.20", accountID:"102"},
                 {acct:"Checking 495", bal:"$1200.16", accountID:"103"},
                 {acct:"Savings 567" , bal:"$1600.00", accountID:"104"}]
```

- Then map your data (the extra data is not mapped so it's not displayed):
  this.view.mySegment.widgetDataMap={lblAccountName:"acct",lblAccountBal:"bal"}
- Finally, populate your widget with data
  this.view.mysegment.setData(mySegData);
  - The segment contains ALL the data including the extra data

Segment - accessing the selected row(s)

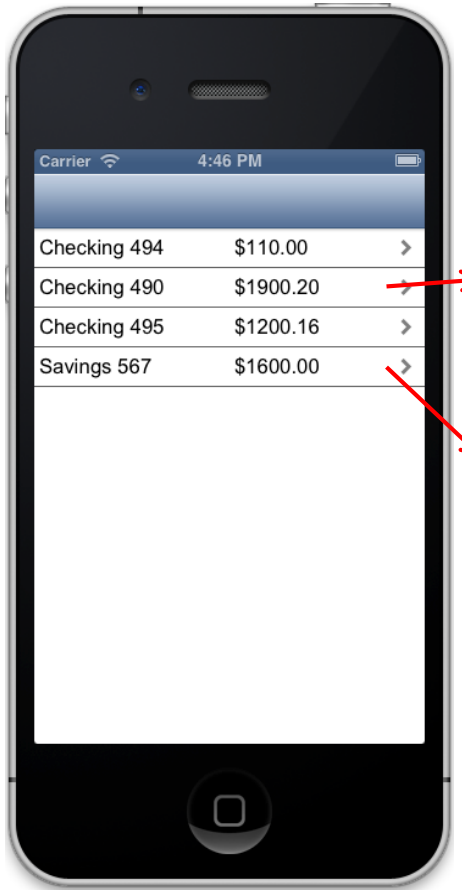
- The segment has an onRowClick event that you can use to trigger code when the user picks a row
- The segment exposes 2 properties to retrieve the data
- These properties are:
  - selectedRowItems - returns the actual data of the selected row(s)
    - Array of JSON objects (based on Selection Behavior
  - selectedRowIndex - returns [sectionIndex,rowIndex]
    - sectionIndex = 0 if there are no sections
  - Note: selectedRowIndex is a special use property - you typically want the data from the selected row exposed by selectedRowItems
- Let's see what data these properties contain by doing a quick test and showing the results



Selected data for single select mode

- We will simply output to the logs the values of these 2 properties when a segment row is clicked:

selectedRowIndex returns [sectionIndex,rowIndex] - our example has sectionIndex as 0 (no sections defined) and rowindex as 1 (second row - 0 based)



All Output | Clear

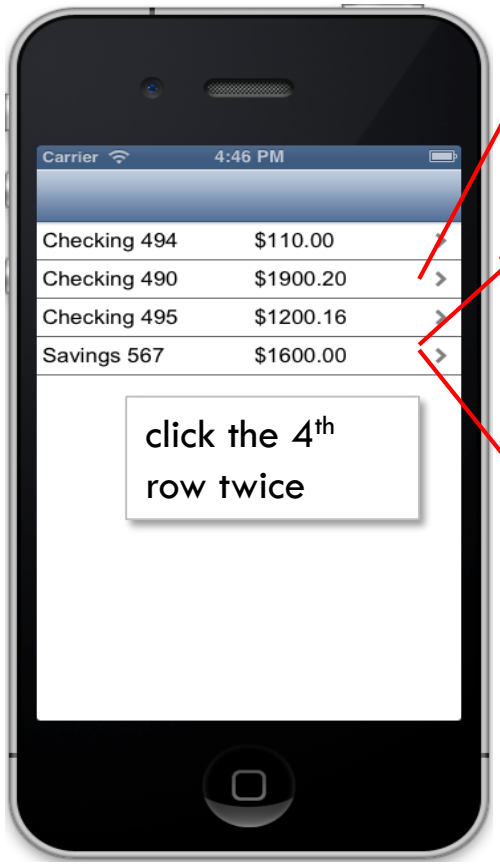
```
2013-08-01 17:09:10.841 HelloWorld[61261:400b] selectedIndex: [0,1]
2013-08-01 17:09:10.841 HelloWorld[61261:400b] selectedItems:
[{"lblAccountName":"Checking
490","lblAccountBal":"$1900.20","accountID":"102"}]
2013-08-01 17:09:10.841 HelloWorld[61261:400b] selectedIndices: [[0,[1]]]
2013-08-01 17:09:13.281 HelloWorld[61261:400b] selectedIndex: [0,3]
2013-08-01 17:09:13.281 HelloWorld[61261:400b] selectedItems:
[{"lblAccountName":"Savings
567","lblAccountBal":"$1600.00","accountID":"104"}]
2013-08-01 17:09:13.281 HelloWorld[61261:400b] selectedIndices: [[0,[3]]]
```

selectedRowItems returns [rowDataSet] - with single select we get one row's worth of data back as a JSON object

Selected data for multi-select mode

- For multi-select mode:

Note: selectedRowIndex never changes after the first click - we'll dig into that next slide…



All Output Clear

```
2013-08-01 17:23:40.437 HelloWorld[61495:400b] selectedIndex: [0,1]
2013-08-01 17:23:40.438 HelloWorld[61495:400b] selectedItems:
[{"lblAccountName":"Checking
490","lblAccountBal":"$1900.20","accountID":"102"}]
2013-08-01 17:23:40.438 HelloWorld[61495:400b] selectedIndices: [[0,[1]]]
2013-08-01 17:23:41.290 HelloWorld[61495:400b] selectedIndex: [0,1]
2013-08-01 17:23:41.291 HelloWorld[61495:400b] selectedItems:
[{"lblAccountName":"Checking
490","lblAccountBal":"$1900.20","accountID":"102"},
{"lblAccountName":"Savings
567","lblAccountBal":"$1600.00","accountID":"104"}]
2013-08-01 17:23:41.291 HelloWorld[61495:400b] selectedIndices: [[0,[1,3]]]
2013-08-01 17:23:41.898 HelloWorld[61495:400b] selectedIndex: [0,1]
2013-08-01 17:23:41.898 HelloWorld[61495:400b] selectedItems:
[{"lblAccountName":"Checking
490","lblAccountBal":"$1900.20","accountID":"102"}]
2013-08-01 17:23:41.898 HelloWorld[61495:400b] selectedIndices: [[0,[1]]]
```

selectedRowItems contains the collections BUT note that re-clicking a row REMOVES that row or index from the collection (Savings row was removed in the final result set)

Segment - accessing the selected row

- In single-select mode (when segment has no sections):
  - selectedRowItems[0] is the segment row data
  - selectedRowIndex[0] is the section and [1] is the index
- In multi-select mode:
  - selectedRowItems[] is the collection of row data (ex. selectedIRowtems[1] is the 2nd selected row)
    - Note: the ORDER of items is NOT the order they are clicked
  - selectedRowIndex didn't change in our test…why?
    - Note how the collection properties added/removed values as rows were clicked…
    - selectedRowIndex will the lowest index in the selected items collection
    - Don't use selectedRowIndex in multi-select mode unless this information is useful to you

Segment - accessing the selected row (cont'd)

- Ok, so the data will be in the segment's selectedRowItems collection.
- How do we access the data, what does it "look" like?
  - Example #1: let's say that we populated our segment with this dataset:

    [{lblAccountName:"Checking 494", lblAccountBal:"$110.00", accountID:"101"},
    {lblAccountName:"Checking 490", lblAccountBal:"$1900.20", accountID:"102"},
    {lblAccountName:"Checking 495", lblAccountBal:"$1200.16", accountID:"103"},
    {lblAccountName:"Savings 567" , lblAccountBal:"$1600.00", accountID:"104"}]
- We could access each piece of data by it's key, namely "lblAccountName", "lblAccountBal" and "accountID"
- In our single-select example, when the user clicked on row showing "Checking 490", then:
  - this.view.mySegment.selectedRowItems[0] will contain the data specific to the selected row:

    selectedData = this.view.mySegment.selectedRowItems[0]

    selectedData.lblAccountName will return "Checking 490"

    selectedData.lblAccountBal will return "$1900.20"

    selectedData.accountID will return "102"

Segment - accessing the selected row (cont'd)

- Example #2: let's say that we populated our segment with this dataset:

  [{acct:"Checking 494", bal:"$110.00", accountID:"101"},
  {acct:"Checking 490", bal:"$1900.20", accountID:"102"},
  {acct:"Checking 495", bal:"$1200.16", accountID:"103"},
  {acct:"Savings 567" , bal:"$1600.00", accountID:"104"}]

  - AND we also had a widgetDataMap set to:

    {lblAccountName:"acct",lblAccountBal:"bal"} Note: no map for accountID
  - In this case, we'd access the data using the keys "acct", "bal" and "accountID" - WHY? because we ALWAYS use the keys in the dataset regardless of widgetDataMap
- Now, in our new single-select example, when the user clicked on row showing "Checking 490", then:
  - this.view.mySegment.selectedRowItems[0] will contain the data specific to the selected row:

    selectedData = myForm.mySegment.selectedRowItems[0]

    selectedData.acct will return "Checking 490"

    selectedData.bal will return "$1900.20"

    selectedData.accountID will return "102"

Segment - accessing the selected row (cont'd)

- In our multi-select example, how do you know how many values are selected?
  - selectedRowItems is an array so selectedRowItems.length will tell you.
  - There are many times when we'll be iterating through data.
  - Here's an example of printing out all the selected account names:

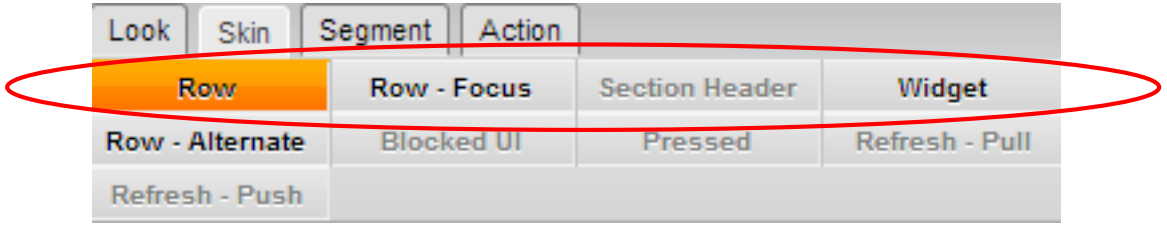
```
function segselectedvalues(seg){
        for (var i=0; i<seg.selectedRowItems.length; i++){
          kony.print("selected value #" + (i+1) + " : " + seg.selectedRowItems[i].acct);}
}
```

  - **Note: don't forget that everything is 0 based**

```
2013-08-02 08:32:47.684 HelloWorld[65439:410b] selected value #1 : Checking 494
2013-08-02 08:32:47.685 HelloWorld[65439:410b] selected value #2 : Checking 490
2013-08-02 08:32:47.685 HelloWorld[65439:410b] selected value #3 : Savings 567
```

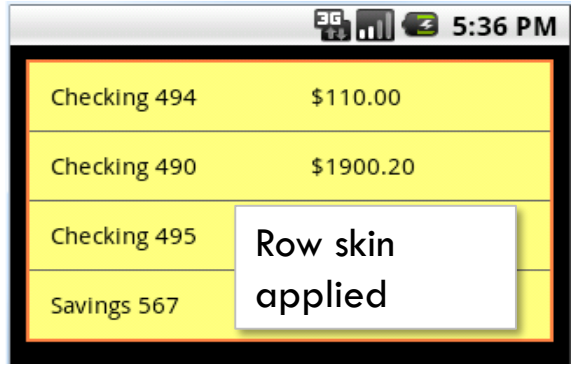
Segment - Skins

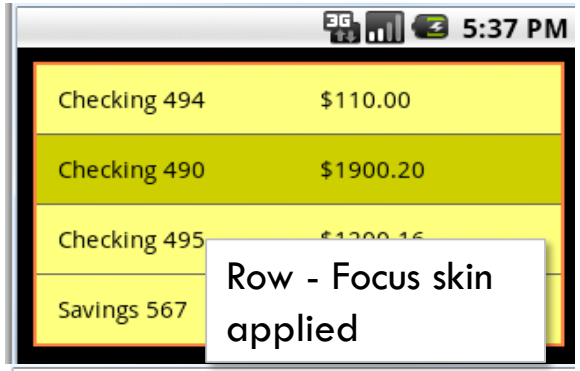
- Like all widgets - Segments have skinning options:

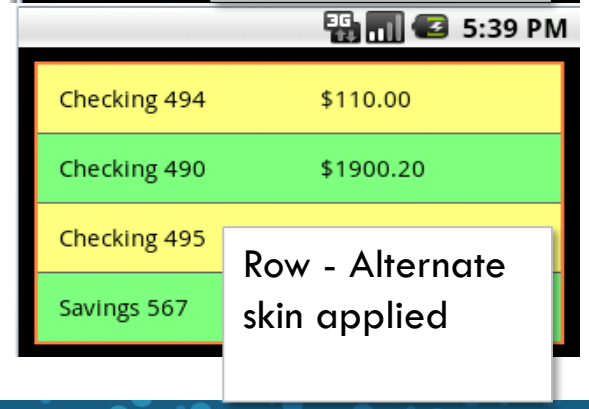


- Row skin will override the Widget skin and apply this skin to every row
- Row-Focus skin specifies the skin when user clicks on a row
- Section Header skin is used for skinning section headers -we'll cover this later on
- Widget specifies the skin to be applied to the entire segment
- Row - Alternate specifies a skin to be used for every other row of the segment. Other rows get the Row skin applied

- Note: Visualizer lets you change or set skin values



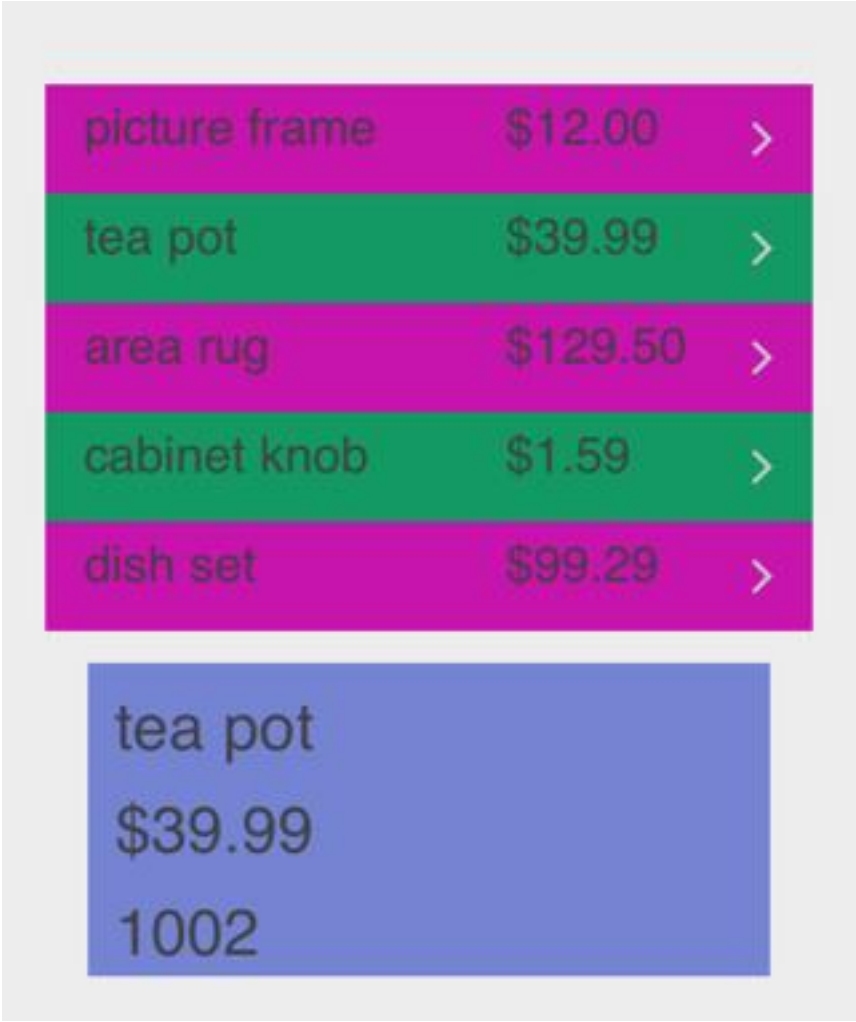




Exercise - Segment Data

- Ok, let's practice what we've learned
- Here is the specification for this exercise:

- Populate a segment with data to show a product's name and price
- Also populate the segment with the productID field, but don't show it on the first screen
- When the user clicks a row, show the 3 values: name, price and productID
- Play around with Segment Skin and set the alternate row skins



Segment Drag and Drop

- We can drag & re-order rows of the segment in the application
- enableReordering is the property that allows you to enable or disable reordering the rows in a Segment
- To allow the users reorder (drag and drop) the rows in a segment, set the property to true
  - eg:this.view.Sgmnt1.enableReordering = true;
- The reordering of the rows works only when the Segment's view type is table view
- In case of iOS the reordering works only when the segment editstyle is set

Segment Drag and Drop (cont'd)

- onDragCompleted event is invoked when the drag and drop of a row in the Segment is complete.
  - Syntax

    onDragCompleted(this,dragstartcontext,dragendcontext)

    this: SegmentUI reference

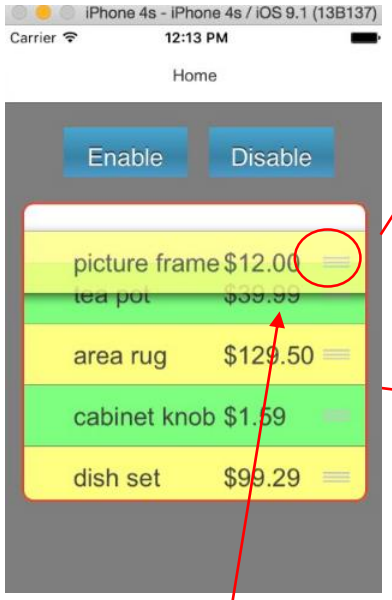
    dragstartcontext: Is a table, which contains row and section information when dragging of a row is started.

    dragendcontext: Is a table, which contains row and section information when dragging of a row is ended.
- You can call the onDragCompleted event when the enableReordering property is true to allow the app users to drag and drop a row within a segment
- enableReordering and onDragCompleted are supported in iOS and Android

Segment Drag and Drop Exercise

- Let us have a set of records on the segment and two buttons i.e., Enable button to enable reordering and disable button to disable reordering

When the user clicks on Enable button then reordering is enabled



Observe that we can reorder the records by dragging it

In iOS reordering works only when the segment editstyle is set, here we have set it to icon

After reordering the records

Creating Services With Kony Fabric

kony Stay Ahead

Kony Fabric Server - Overview

- Kony Fabric provides enterprise security and complex system integration services and allows developers to focus on building app experiences
- This is accomplished by providing a powerful set of services to handle identity, integration, orchestration, data sync, and messaging
- When these services are configured within Kony Fabric Server, they can easily be incorporated into a mobile application using any third-party app development tool using our SDKs or direct REST API interface
- Following are some of the platforms supported for Kony Fabric
  - Kony Visualizer Enterprise
  - Android Native
  - iOS Native
  - JavaScript

Kony Fabric - Overview (cont'd)

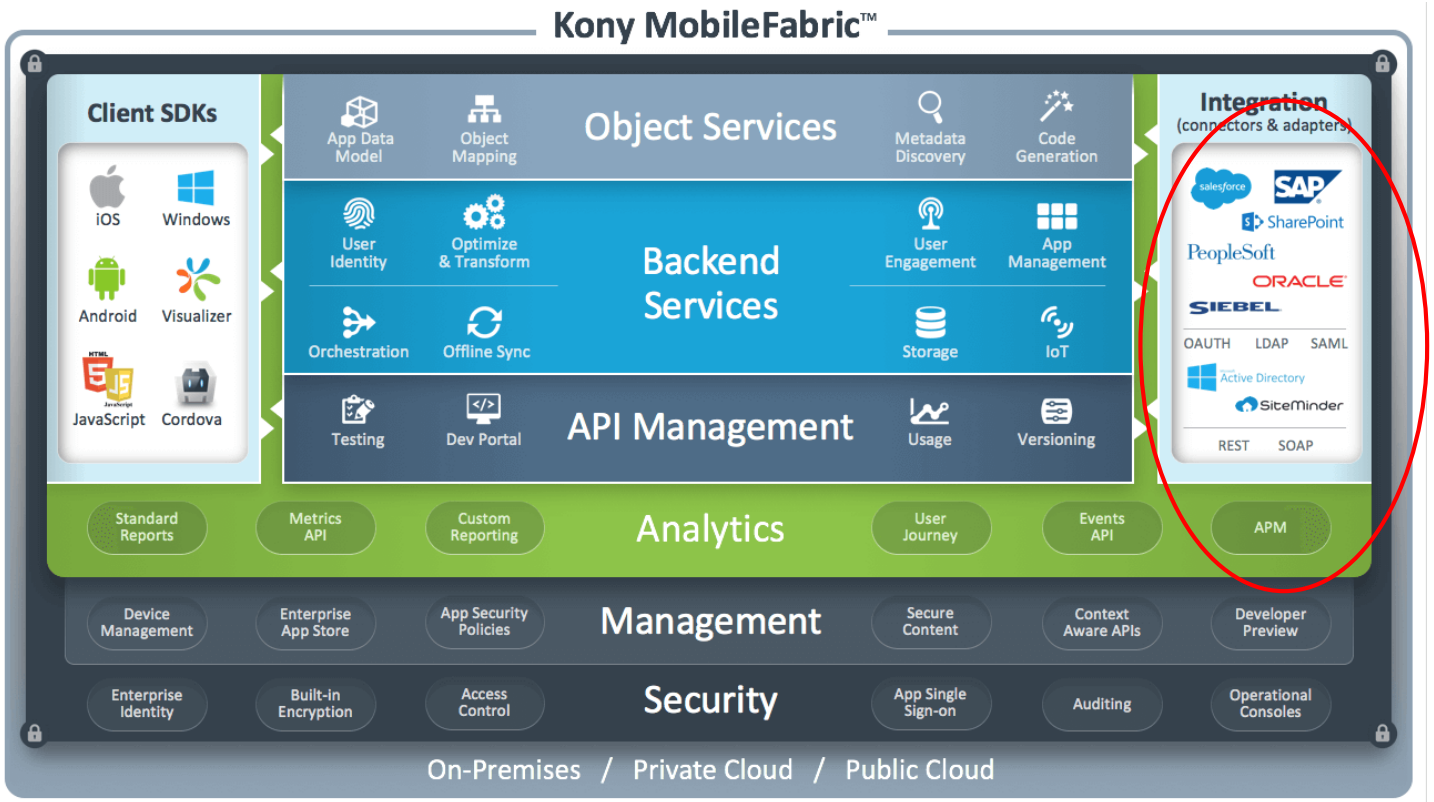
- Kony Fabric takes the headache out of authentication and data services for mobile applications
  - "Typical" mobile applications have to have all the information built into the application to be able to:
    - Authenticate to backend systems
    - Configure and call web services and manage the returned data
    - Call multiple web services to aggregate the resulting data
    - Manage a mobile database for data access when the application is offline AND managing reconciling all the changes on the client once connectivity is returned
    - Managing push message subscriptions and handling received messages
  - In these applications, if anything changes, the whole app must be re-submitted to the app store for release to the users
  - Each mobile application (iOS, Android, Windows, etc.,) needs it's own codebase to do all these things - you end up writing that code for as many phone types as you support

Kony Fabric - Overview (cont'd)

- Kony Fabric takes the headache out of authentication and data services for mobile applications by providing all of these services in a middleware solution
  - For each service type, there is a single definition in Kony Fabric
  - All device types can call that service in a standard way
  - Any changes happen on Kony Fabric so client applications don't necessarily need to change

Application Data Flow

- Remember, the Kony Fabric server will retrieve the data from the "backend" source and pass it on to the client



- We'll need to define these integration points that we call *Services*

Kony Fabric - Features

- The following are the six major services offered by Kony Fabric:
  - Identity - who can access the app and use it
  - Integration - accessing backend data
  - Orchestration - combining integration services
  - Synchronization - managing offline data
  - Engagement - incorporating push messaging
  - Reporting - measuring app usage and user info

Kony Fabric - Features (cont'd)

- You can create "apps" in Kony Fabric where you can have one or more of each of those features
  - On the client, in the application, your code can consume the features of one or more "apps" hosted by Kony Fabric
  - Think of a Kony Fabric "app" as a logical grouping of features - for example, you might have a Facebook "app" that has features for logging in and accessing Facebook data through web services
- We will use Identity and Integration services now. We will discuss others later

Accessing Kony Fabric

So…where is the Kony Fabric and how do we access it?

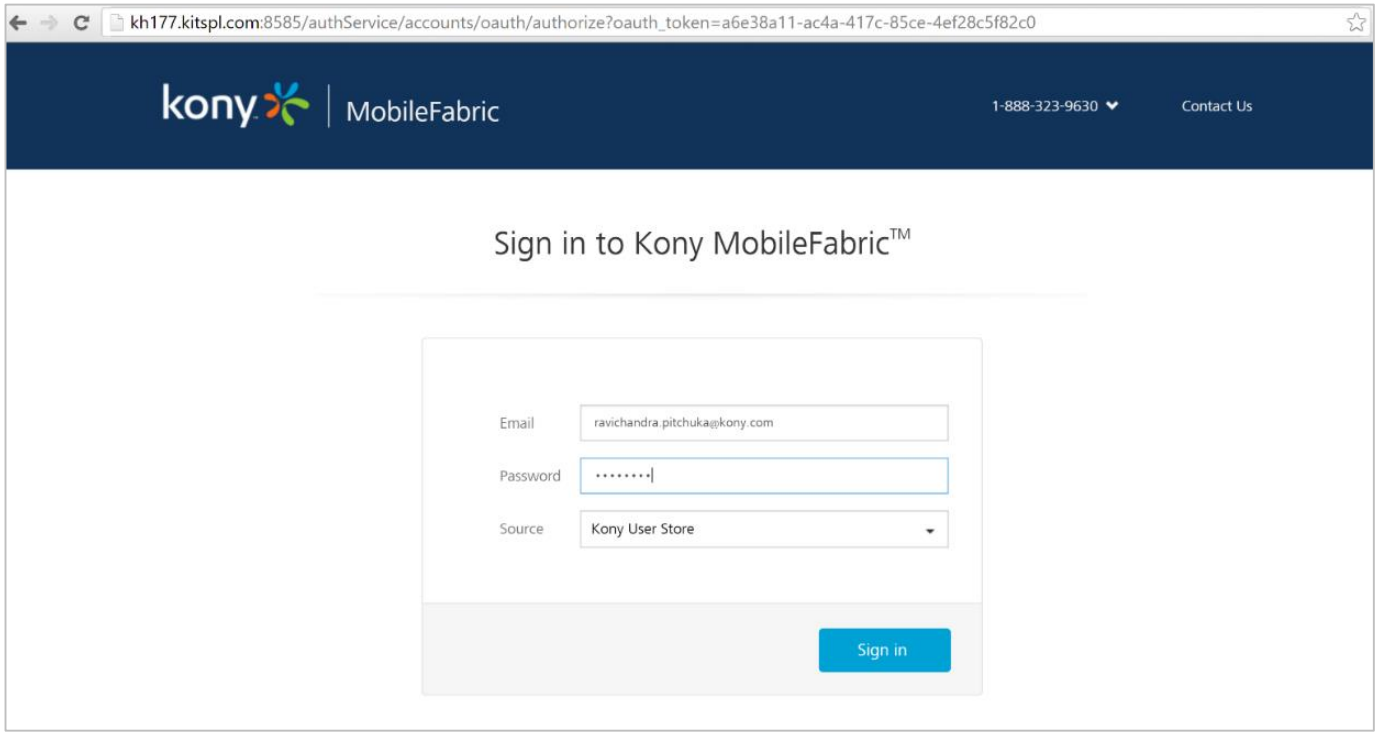
Kony Fabric is available in 2 ways

- As a local on-premise server
- As a cloud service

Regardless which you choose, the admin user interface and functionality is the same

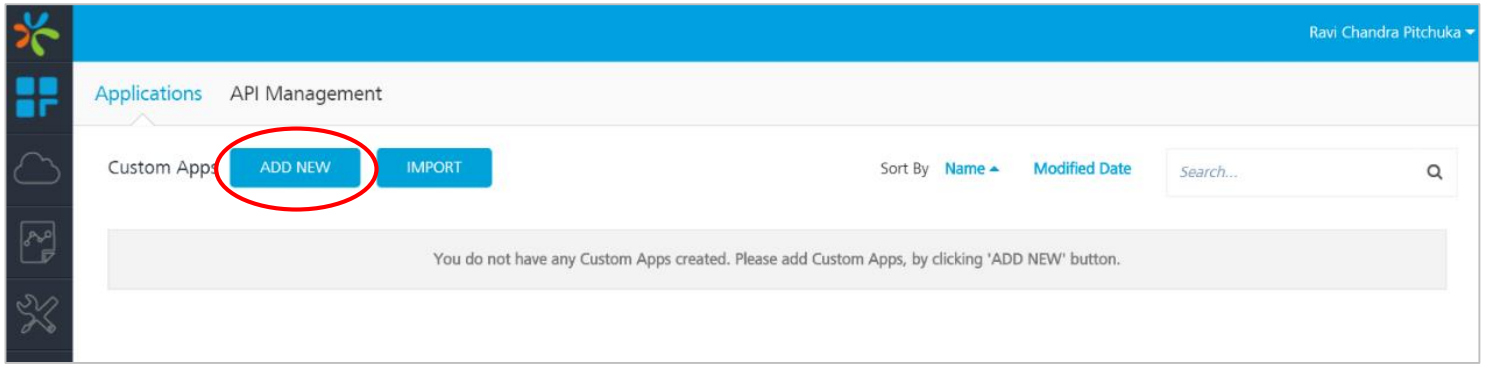
Kony Fabric - Creating an App

- Login to the Kony Fabric console
- Accessing the on-premise console - http://<IP address/domain name>:port/Fabricconsole
- Accessing the Cloud console: http://manage.kony.com

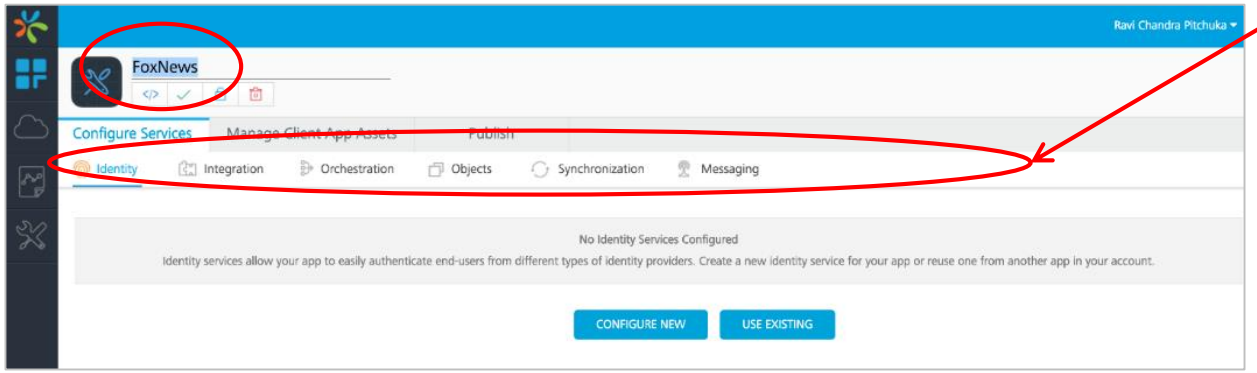


Kony Fabric - Creating an App (cont'd)

- Click on **ADD NEW** button to create a new app in Kony Fabric



- Rename the app to “FoxNews”



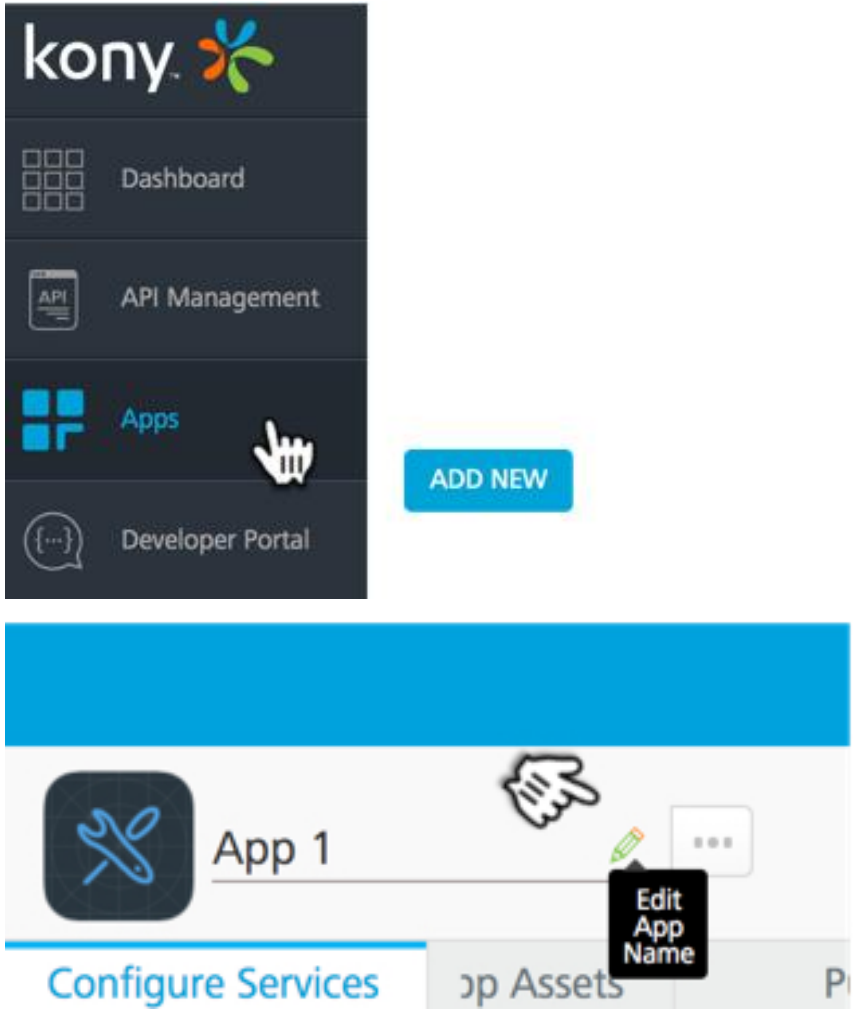
You can configure different types of services for this app by clicking on the corresponding tab

Exercise: Create Kony Fabric Service

Ok, let's go and do this now and create our own Kony Fabric Service

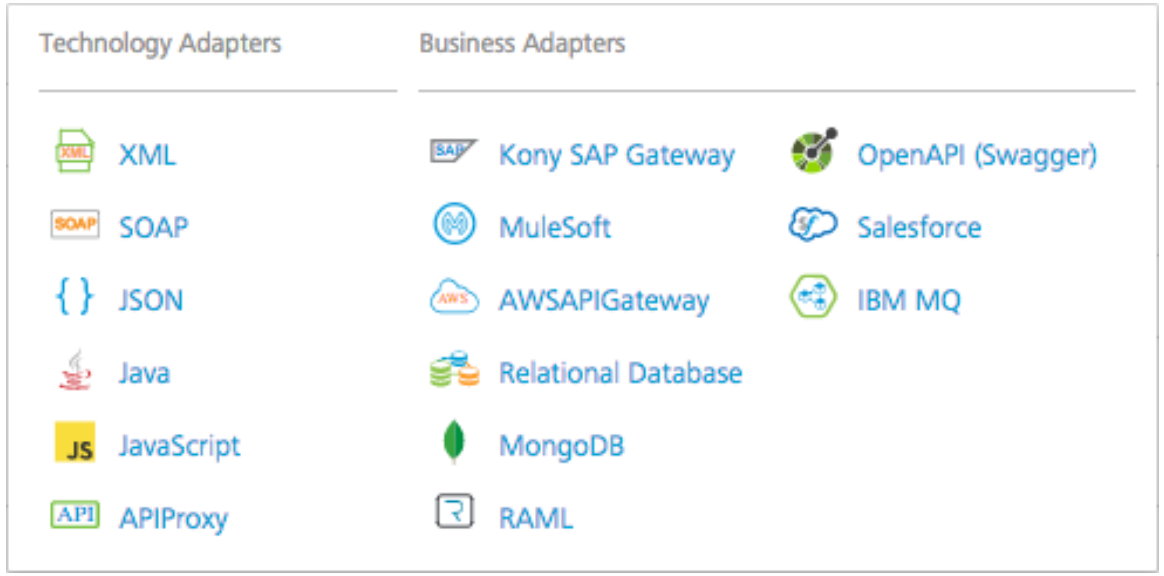
- Login into Kony Fabric
- Select Apps
- Click on “ADD NEW” to create a new application
- Edit the name so its meaningful

That's it for now!!!



Integration Service - Overview

- The Kony Fabric Integration Services can consume data from any back- end system
- Here is the list of current supported integration types



- The business adapters support back-end object discovery to facilitate integration tasks
- The technology adapters can consume web services (XML, SOAP & JSON) as well as give you the ability to build your own adapter in Java
- MuleSoft is a 3rd party connector provider with over 100+ available connectors

Integration Service - Uses

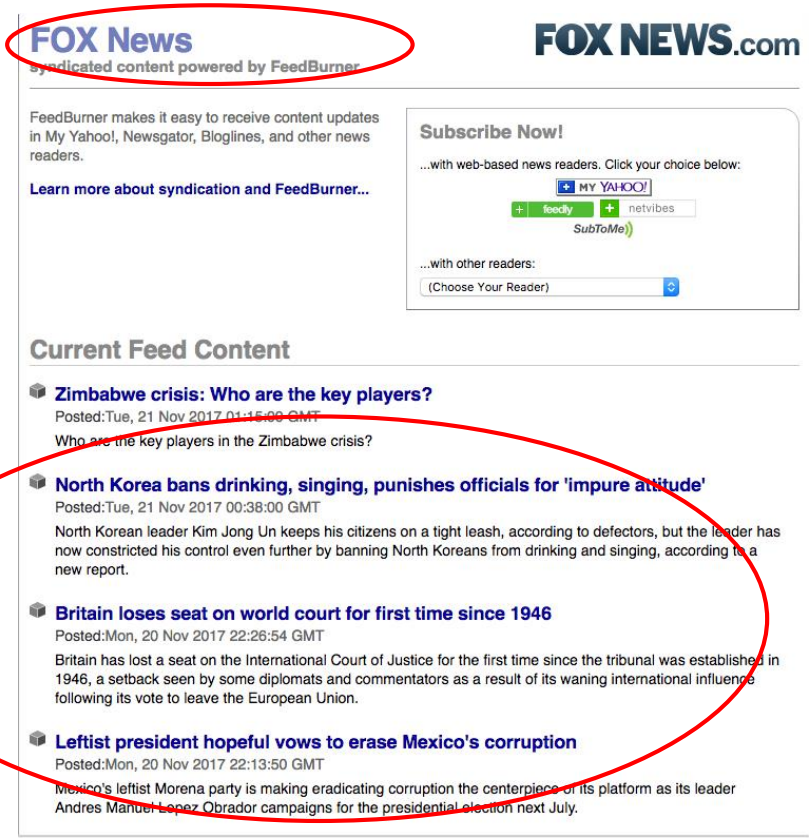
- Moving the Integration Service to the middleware allows the developer unique advantages:
  - When the data is returned to Kony Fabric, it's filtered such that ONLY the fields you need on the mobile device are sent back to the mobile application
  - You can perform data pre- and post- processing on the server rather than on the mobile application
    - You can store data in Kony Fabric's user cache for use in other Integration Service calls - no need to manage that data on the client application
    - You can process returned data before sending to the client application
  - You can create a robust library of configured Integration Services for use in more than one app
  - Fixes/changes/improvements can be made in Kony Fabric without affecting your mobile application - user's don't have to upload a new app just because the backend service changed
- Kony Fabric exposes these Integration Services through a standard API to simplify the mobile application code

XML Web Service

- We're going to walk through the entire process for configuring and using an XML Integration Service
- The basic steps we'll follow are:
  - Identify the web service - we'll use a Fox News public RSS feed
  - Create our Integration service base
  - Create one operation - getArticles
  - Test from Kony Fabric
  - Test from mobile application
- Here's the website we'll use as an example:

  http://feeds.foxnews.com/foxnews/world

Here is a list of returned articles



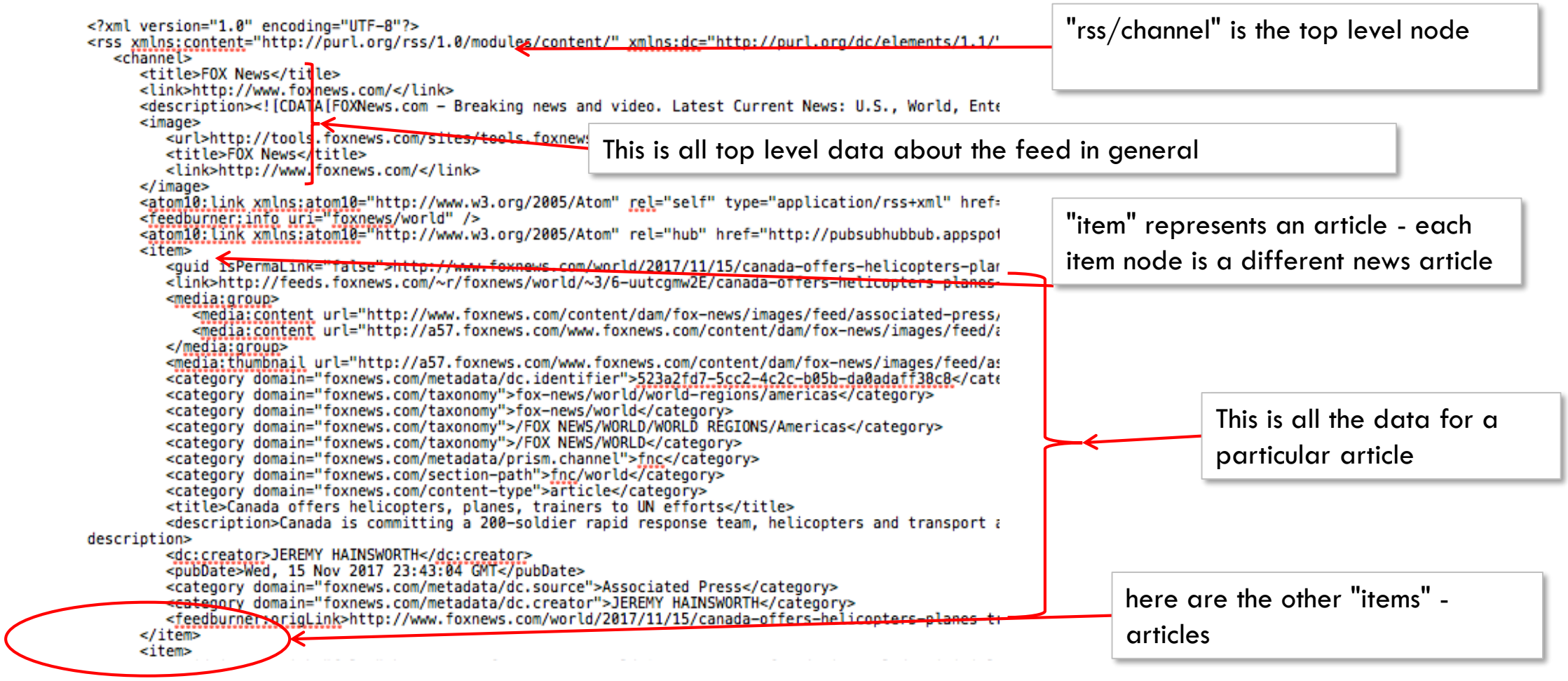
Know Your Data!

- Whenever we work with data it's CRITICAL that we understand the nature of the data that our web services return
  - Is the data structure fixed? is the same node always returned? Is no data a valid use case? is there a maximum amount of data returned? etc...
  - When we're testing our Integration Services, Step 1 is ALWAYS - Know your data

  - With that in mind, let's examine our Fox News web service:
    - On the previous slide, we saw the web page showing the results - what ARE those results?
    - We're talking XML web services, so we'll see that the URL actually returns an XML doc
    - The Fox News Feeds page is just presenting that XML data in a web page to make it easy to read
    - The raw data is available by calling that URL as a web service
    - Let's take a look....

Fox News Service Data

- If we run Developer Tools in Chrome (or any browser), you can see the returned doc:



Fox News Service Data (cont'd)

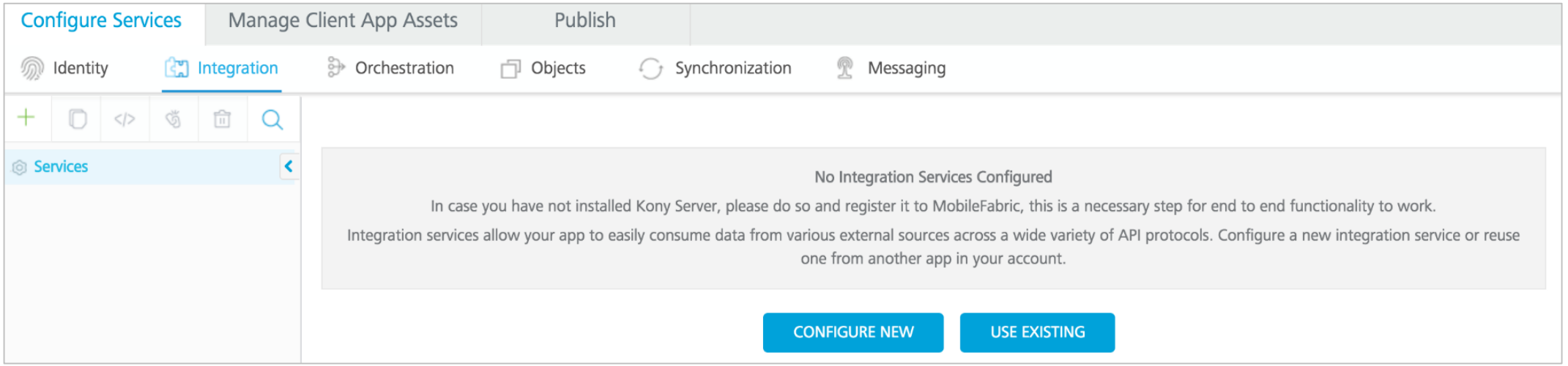
- Before we configure an Integration Service in Kony Fabric, let's talk about what our mobile application needs for data
- For the entire returned set of articles, there are a few pieces of information returned. For each article, there is a lot of info returned - do we need all that?

```
<item>
   <guid isPermaLink="false">http://www.foxnews.com/world/2017/11/15/canada-offers-helicopters-plar
   <link>http://feeds.foxnews.com/~r/foxnews/world/~3/6-uutcgmw2E/canada-offers-helicopters-planes-
   <media:group>
      <media:content url="http://www.foxnews.com/content/dam/fox-news/images/feed/associated-press,
      <media:content url="http://a57.foxnews.com/www.foxnews.com/content/dam/fox-news/images/feed/a
   </media:group>
   <media:thumbnail url="http://a57.foxnews.com/www.foxnews.com/content/dam/fox-news/images/feed/as
   <category domain="foxnews.com/metadata/dc.identifier">523a2fd7-5cc2-4c2c-b05b-da0adaff38c8</cate
   <category domain="foxnews.com/taxonomy">fox-news/world/world-regions/americas</category>
   <category domain="foxnews.com/taxonomy">fox-news/world</category>
   <category domain="foxnews.com/taxonomy">/FOX NEWS/WORLD/WORLD REGIONS/Americas</category>
   <category domain="foxnews.com/taxonomy">/FOX NEWS/WORLD</category>
   <category domain="foxnews.com/metadata/prism.channel">fnc</category>
   <category domain="foxnews.com/section-path">fnc/world</category>
   <category domain="foxnews.com/content-type">article</category>
   <title>Canada offers helicopters, planes, trainers to UN efforts</title>
   <description>Canada is committing a 200-soldier rapid response team, helicopters and transport a
description>
   <dc:creator>JEREMY HAINSWORTH</dc:creator>
   <pubDate>Wed, 15 Nov 2017 23:43:04 GMT</pubDate>
   <category domain="foxnews.com/metadata/dc.source">Associated Press</category>
   <category domain="foxnews.com/metadata/dc.creator">JEREMY HAINSWORTH</category>
   <feedburner:origLink>http://www.foxnews.com/world/2017/11/15/canada-offers-helicopters-planes-tr
</item>
```

specified by node name: title, description and pubDate

Integration Service Fox News

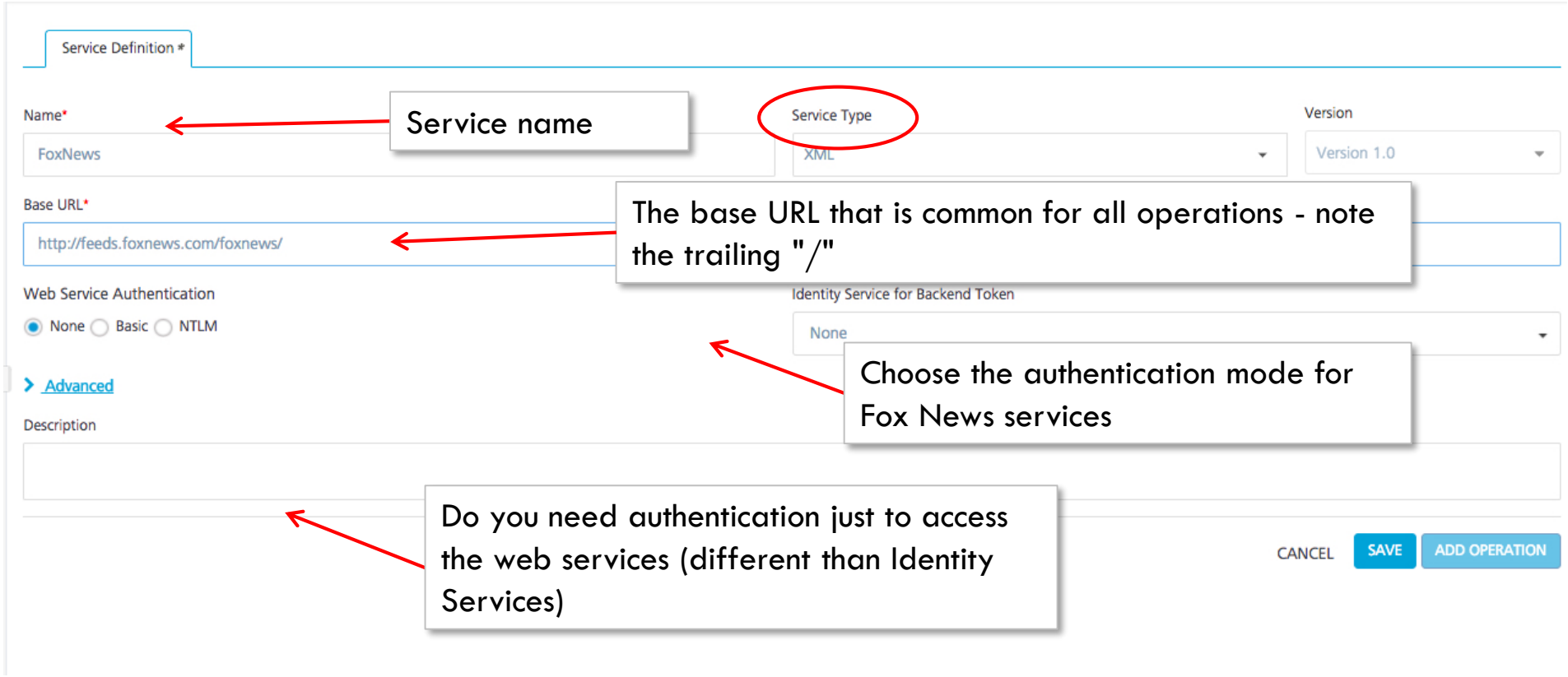
- We can create a Fox News Integration Service in our App:



- All services (Identity, Integration, etc.) that are created inside an app are copied to the library
- You can access those using the USE EXISTING button
- We'll start by creating our new Fox News Integration service with CONFIGURE NEW

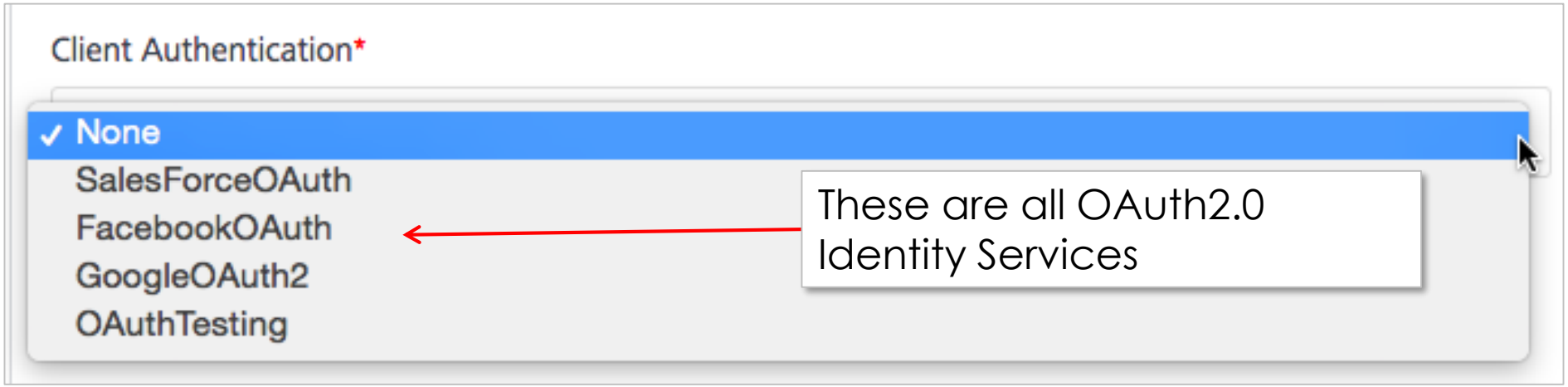
Fox News Configuration

- The first step is to create our base Integration Service configuration:



Client Authentication Options

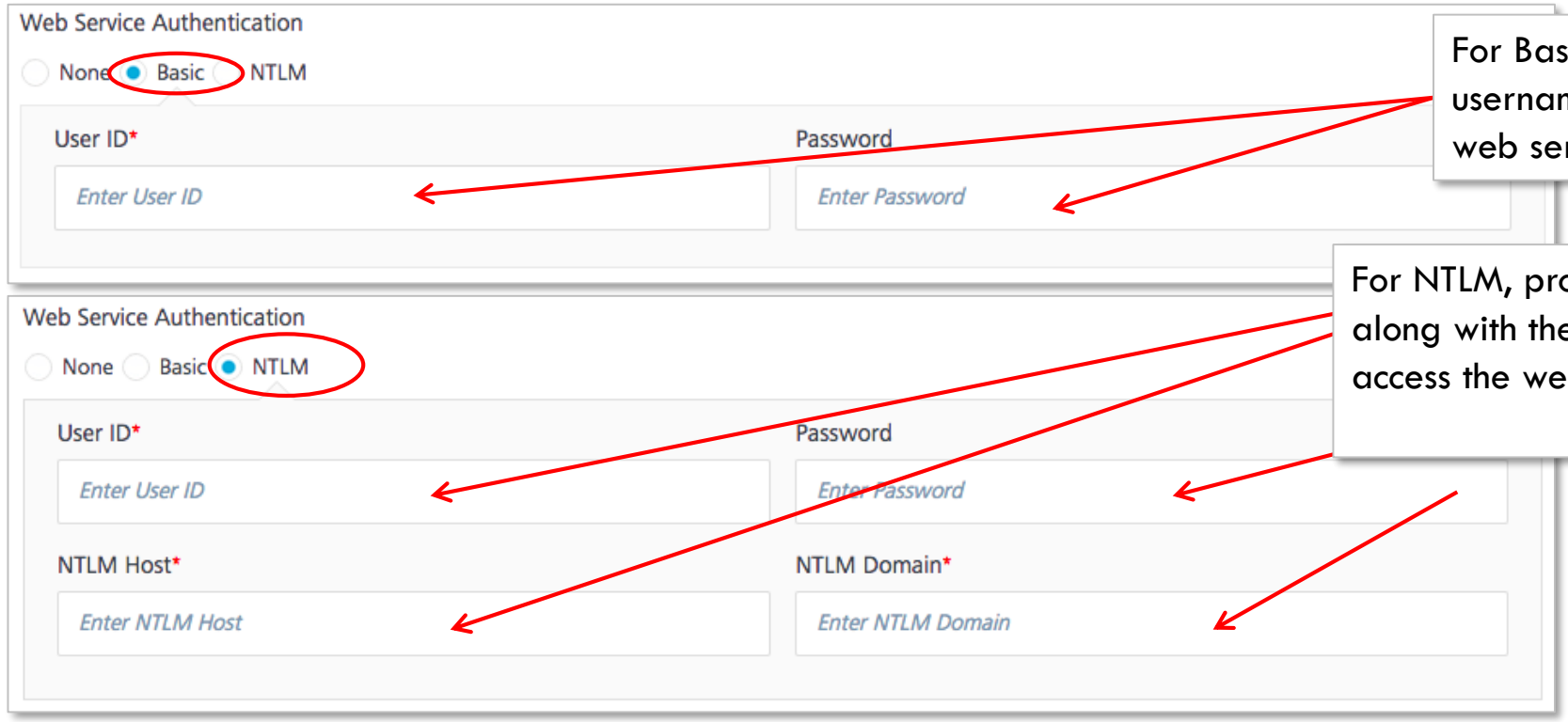
- For any Integration Service, you have to pick your Client Authentication scheme
  - Your options will be "none" and a list of Identity Services that:
    - We'll discuss these later
  - Let's take a look at an example of the options for the Client Authentication:



    - You can still call an Identity Service in any mobile application to authenticate the user
  - In our example, Fox News requires no Identity Service so we'll pick ***None***

Web Service Authentication Options

- There are times when you need authentication just to access the web service - this is different than Identity
  - Some service are protected by an additional security layer
  - This is what we're configuring with the Web Service Authentication options:



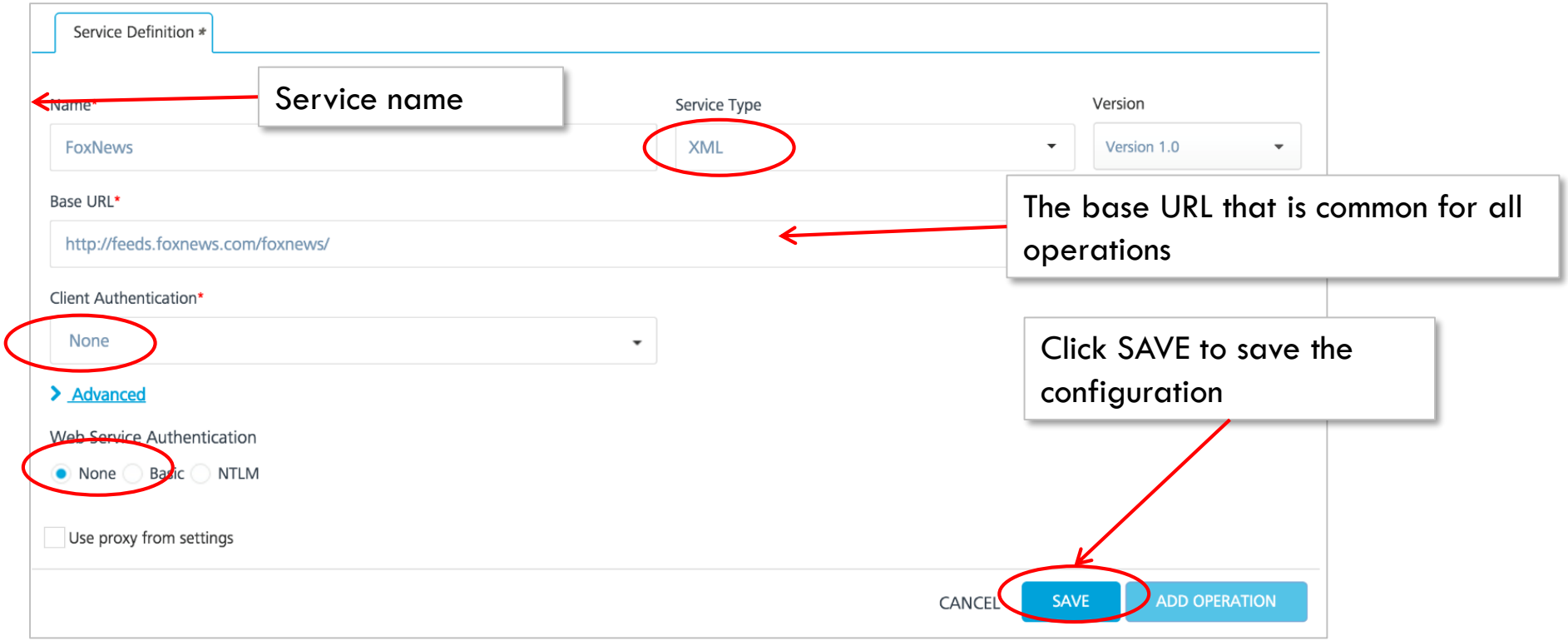
For Basic, provide a username/password to access the web service endpoint URL

For NTLM, provide a username/password along with the NTLM host/domain info to access the web service endpoint URL

Note: For our Fox News example, we can pick None

Fox News Configuration

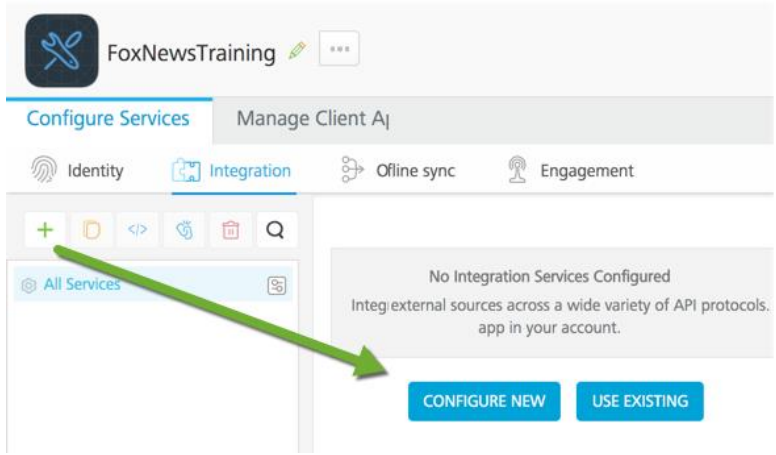
- Here is our final configuration



- Right now this Integration Service doesn't do anything - we need to add operations for each web service we need to call
  - Let's now add an operation to our Integration Service...

Exercise: Create a Service for Kony Fabric

- Let's go back to our Kony Fabric app and add a Service
- With your FabricApp open, select the Integration tab
- Select "Configure New"

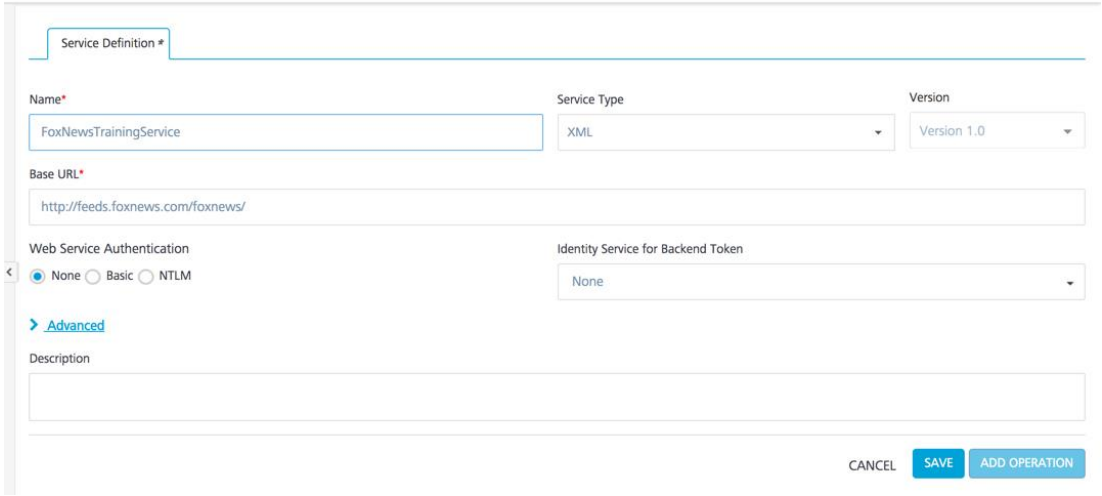


- Make sure to give it a meaningful name and save your work

Create a Service for Kony Fabric (cont'd)

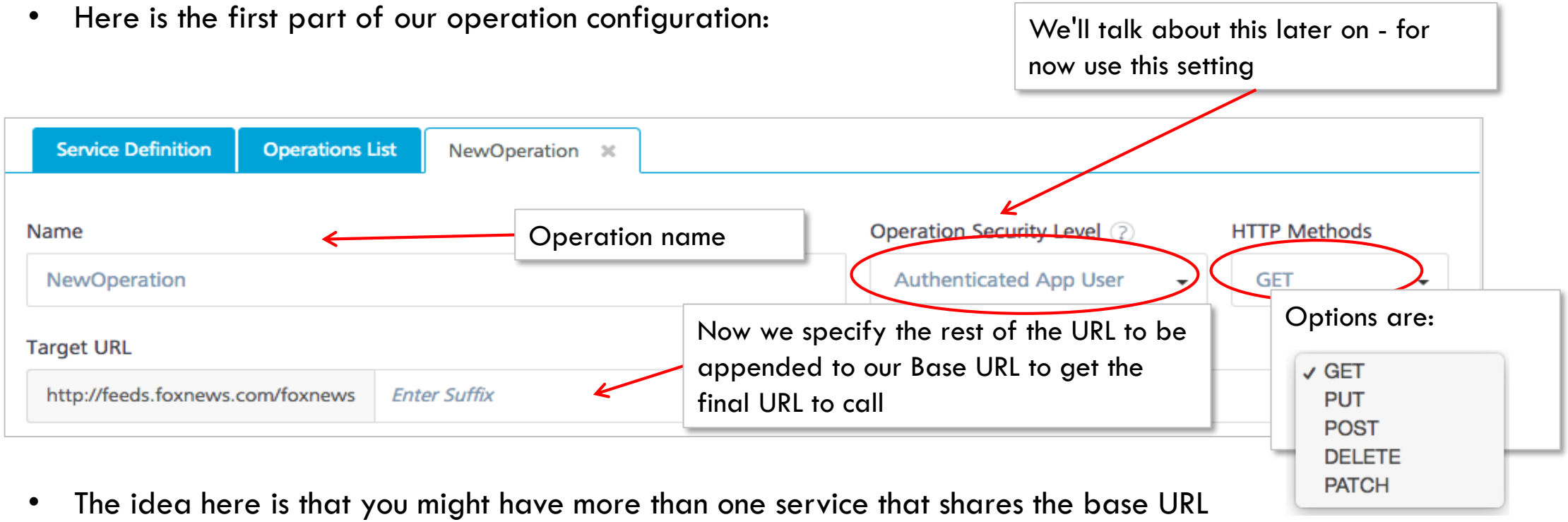
- Use the following URL for the Service (and don't forget the training "/" !!!

  - BaseURL : http://feeds.foxnews.com/foxnews/ -Don't forget the training "/" !!!
  - Service Type : XML
  - Keep the other default values
  - Nothing else is needed at this point



Adding an Operation

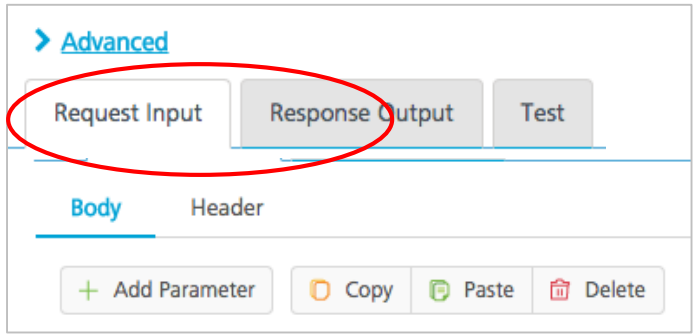
- Using ADD OPERATION we'll now create our **getArticles** operation - this will return the articles data
  - Here is the first part of our operation configuration:



  - The idea here is that you might have more than one service that shares the base URL
  - Each operation appends the specified suffix to the base URL and that is what is called
    - This is why we appended the "/" to our base URL - so we can add "world" as our suffix rather than "/world"

Adding an Operation (cont'd)

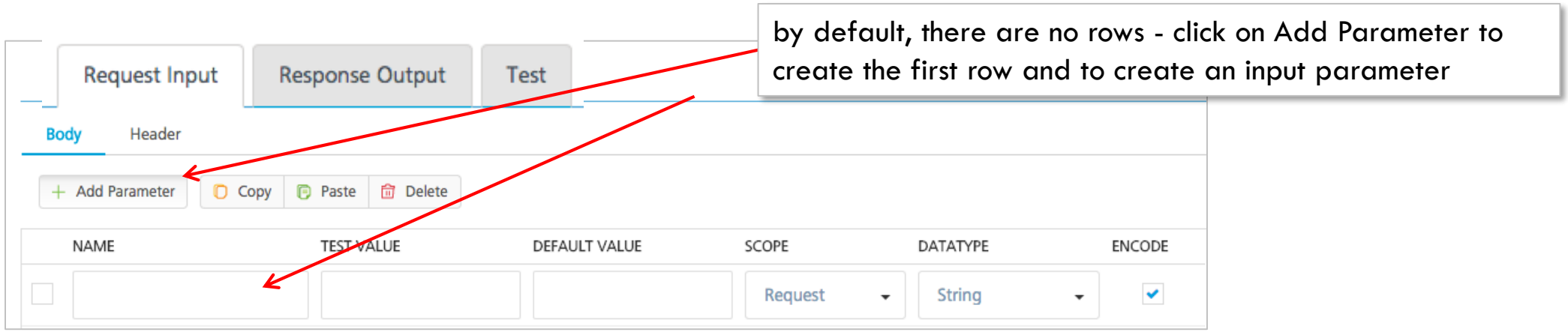
- Here is the second part of the operation configuration:



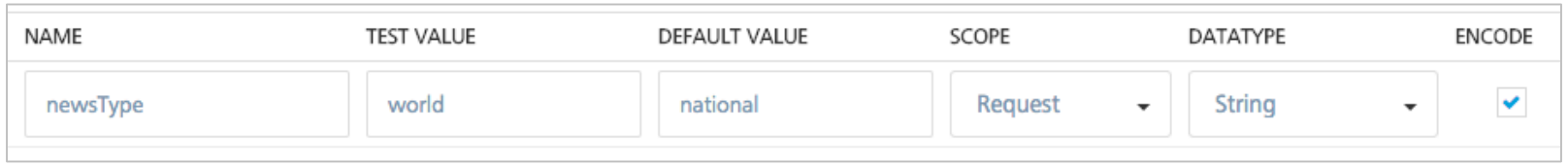
- Our URL is: http://feeds.foxnews.com/foxnews/world to return world news
  - BUT Fox News also has entertainment, national, science & technology and more!
  - Do we need to create an operation for each news type?
    - NO! we'll use input parameters instead so we can pass whatever we want in our mobile application
- Request Input parameters are your variable input values
  - There are 2 "tabs": Body and Header - you can specify the input parameters for each of these - depends on where your service needs them

Operation Request Input

- The input parameters are specified the same way (header or body) using a tabular input tool:



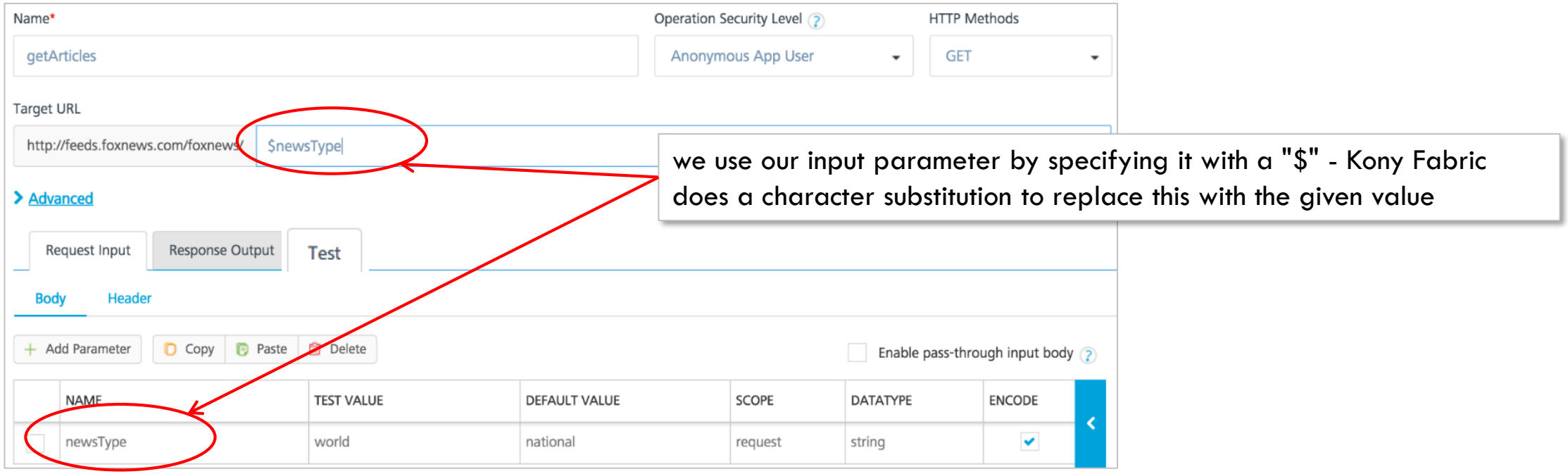
- For our operation, we'll create an input parameter for our news type as follows:



  - My input parameter is called "newsType" and for testing here in Kony Fabric I want to use "world" as my value
  - I didn't choose a default - the value used if I don't send any data from my mobile application
  - The value is a string that is encoded in the URL - we'll leave Scope to "Request" for now

Operation Configuration

- This is our operation configuration so far:



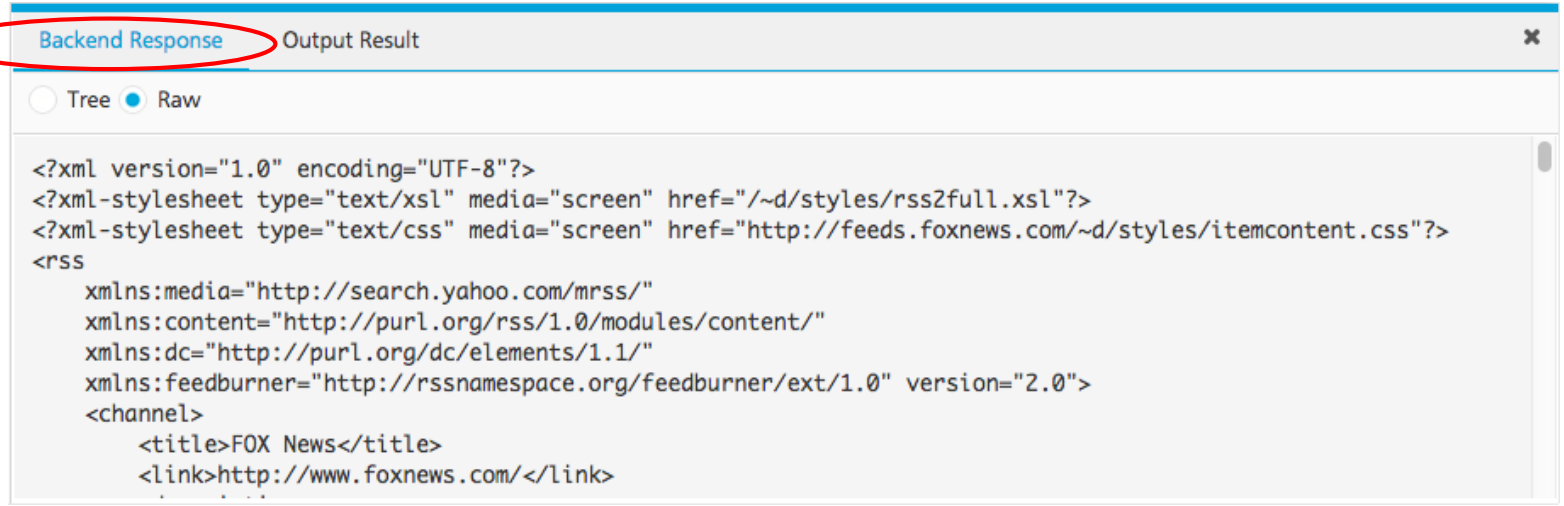
- At the bottom of the page we can Save our work:

- We can also test it by selecting the Test tab and then selecting:

- Let's see what gets returned...

Operation Testing

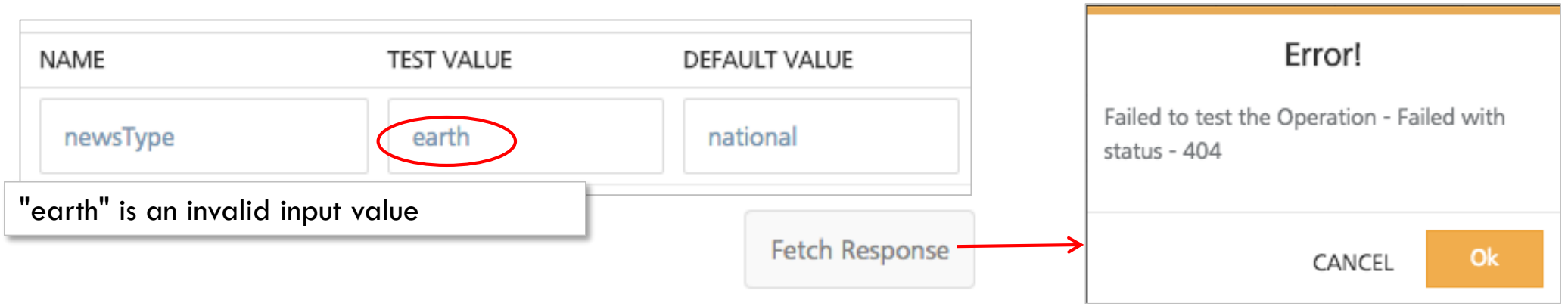
- Below our operation definition, we can see the returned results:



- The Backend Response "tab" contains the returned data from the web service to Kony Fabric
  - Switching to the tree view helps you identify the node values:
  - This is helpful in the limited real-estate
  - Hint: I like to copy/paste the response data into a text tool

Know Your Errors!

- Remember my plea that you Know Your Data? The same is true for error conditions - Know Your Errors!
- What if we pass a test value that Fox News can't interpret?
  - Let's change the test value and see!



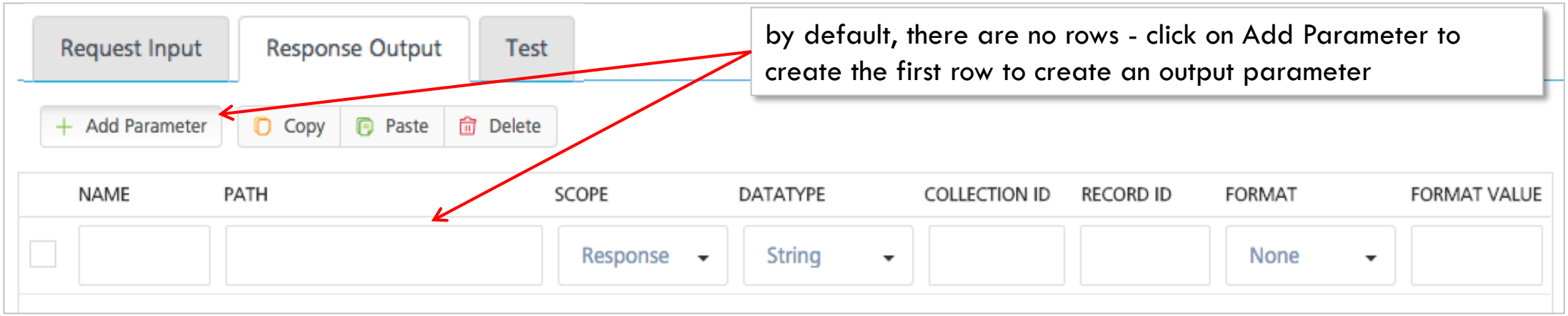
  - When we use this service, it's important to pass valid values - good to know!
- In general, depending on the type of invalid condition, the web service may or may not return valid data
  - In our case it didn't return anything
  - In other cases, the web service might work and return the error in the data - important to know how your service works

Operation Output Parameters

- Remember when we looked at the data with Developer Tools?
- Does all that response get sent back to the mobile application?
  - No! We, as integrators, can now choose exactly what data we want to return from to the client
  - Remember our business case - we wanted only to return some of the data, not all of it
  - We control what gets sent back by configuring output parameters:
  - We'll have to identify the data by specifying xPath to retrieve that specific piece of data from the XML response
  - We can create a hierarchy by specifying a Collection ID and we can alter the "shape" of the data by specifying a Record ID
  - Let's take a look at how this works...

Operation Output Parameters (cont'd)

- Does all that response get sent back to the mobile application?
  - No! We, as integrators, can now choose exactly what data we want to return from to the client
  - Remember our business case - we wanted only to return some of the data, not all of it
  - We control what gets sent back by configuring output parameters:



  - We'll have to identify the data by specifying xPath to retrieve that specific piece of data from the XML response
  - We can create a hierarchy by specifying a Collection ID and we can alter the "shape" of the data by specifying a Record ID
  - Let's take a look at how this works...

Picking Output Values

- Here is PART of our response showing the top level data and the first article out of many:

Here is the article's title node
xPath: /rss/channel/item/title

Here is the article's description node
xPath: /rss/channel/item/description

Here is the article's pubDate node
xPath: /rss/channel/item/pubDate

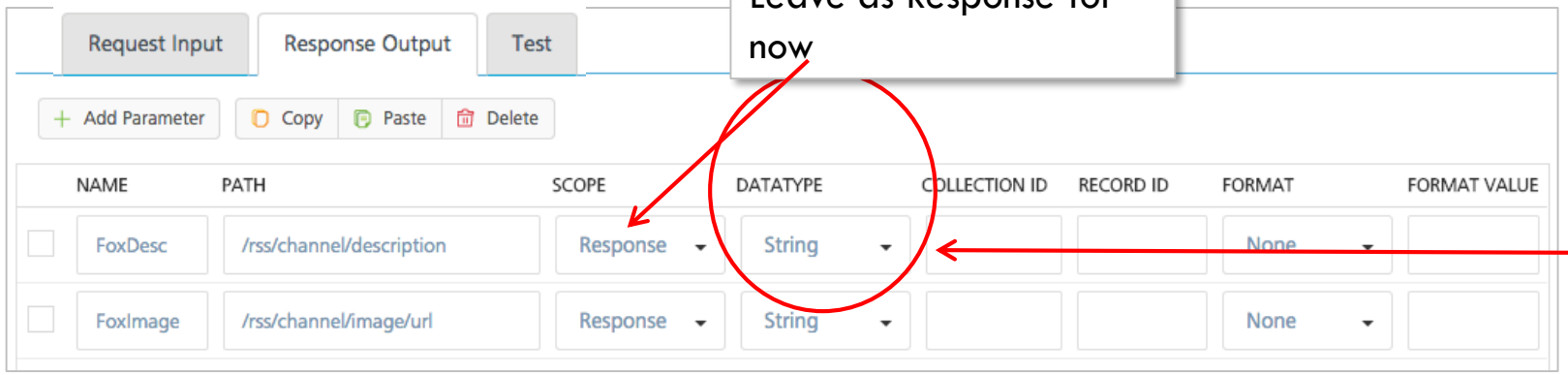
Configuring Output Parameters

- We'll start by configuring the output parameters for the top level description and image URL:
- Here are the output parameters:

Here is the description node we want
xPath: rss/channel/description

Here is the image URL node we want
xPath: rss/channel/image/url

Leave as Response for now

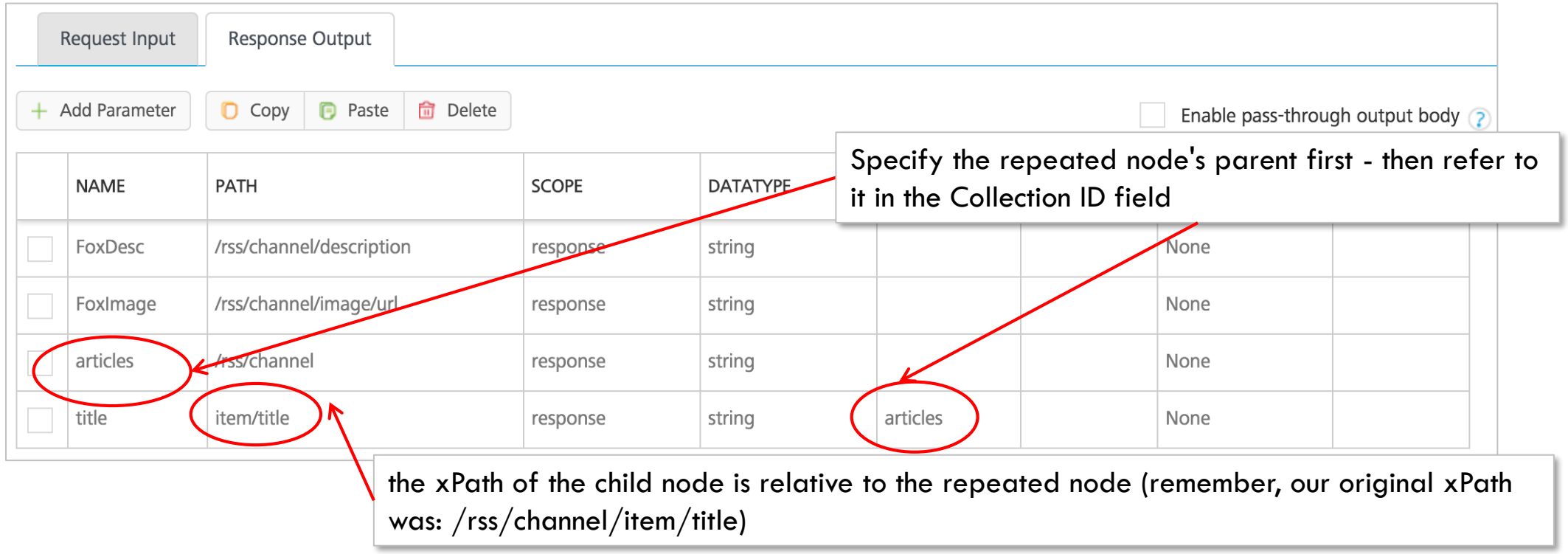


Under DATATYPE - pick String, Boolean or Number (we'll talk about Collection and Record in a bit...)

Below the output parameters is a button we can use to test our output parameters:
And we see our results in the Output Result tab:

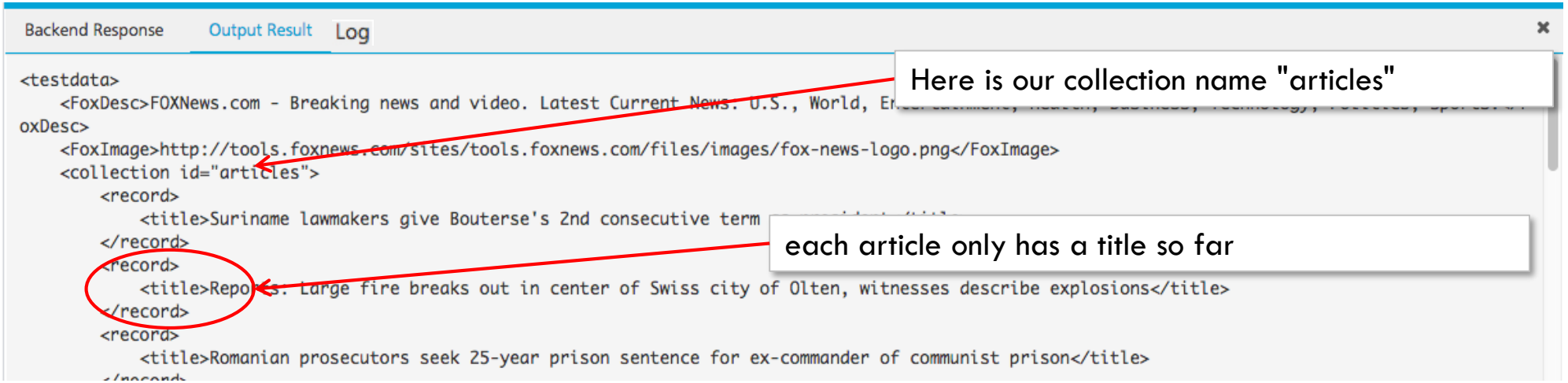
Configuring Collections

- The "Collection ID" field for the output parameter lets us create a collection of data - just what we need for our articles
- The Collection ID value will point to an output parameter that specifies the repeated node
  - In our case the repeated node is item, and this is how we'll configure only the article title to start:



Testing Collections

- If we look at our results:

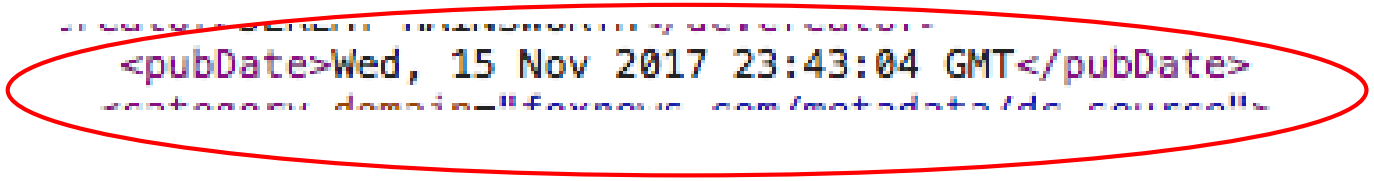


- We can see our list of articles
  - Each title is in a "record"
  - As we add more data for each article, there will be more data inside each record
    - For now there is just a title - here are the output parameters for the rest:

☐	description	item/description	response	string	articles		None	
☐	pubdate	item/pubDate	response	string	articles		None	

Format and Format ValueCollections

- Formatting is available using standard Java format strings. They are:
  - java.text.DecimalFormat for currency and number
  - java.text.SimpleDateFormat for dates
- Let's look at our example:



- The format for pubDate is configured as:

- What we should now see in results is: <pubDate> Wed, 15 Nov 2017</pubDate>

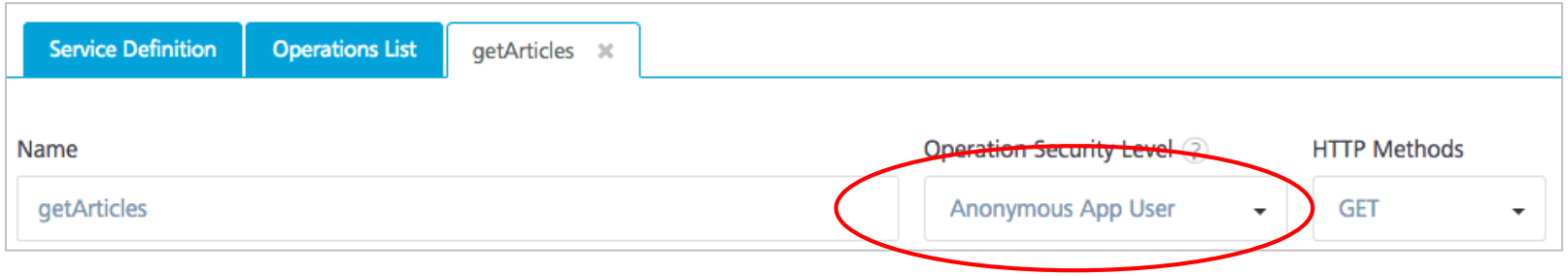
xPath Considerations

- While xPath can do a lot for you, you might find some limitations in the completeness of the implemented parser
- Consider how the description data comes back in our results:

- One tool I find useful is: http://www.freeformatter.com/xpath-tester.html
  - This site lets you enter any XML you want and then test xPath against it
  - Great for seeing what should work before testing in Kony Fabric

Operation Security Level

- Whenever we configure ANY operation, we have to pick an Operation Security Level:



- There are always 3 choices:
  - Authenticated App User (Default) - this means that in order to call this operation, the user must have authenticated first using the specified Identity service
    - Our only choice for the Service's Client Authentication was None or the OAuth2.0 services we created
    - Choose this Authenticated App User option, means that Kony Fabric will automatically provide that returned OAuth token in this operation's request header
  - Anonymous App User - this means that you only need to initialize the client instance with the Kony Fabric App's key and secret
  - Public - means that you don't even need the App's key and secret to call this service

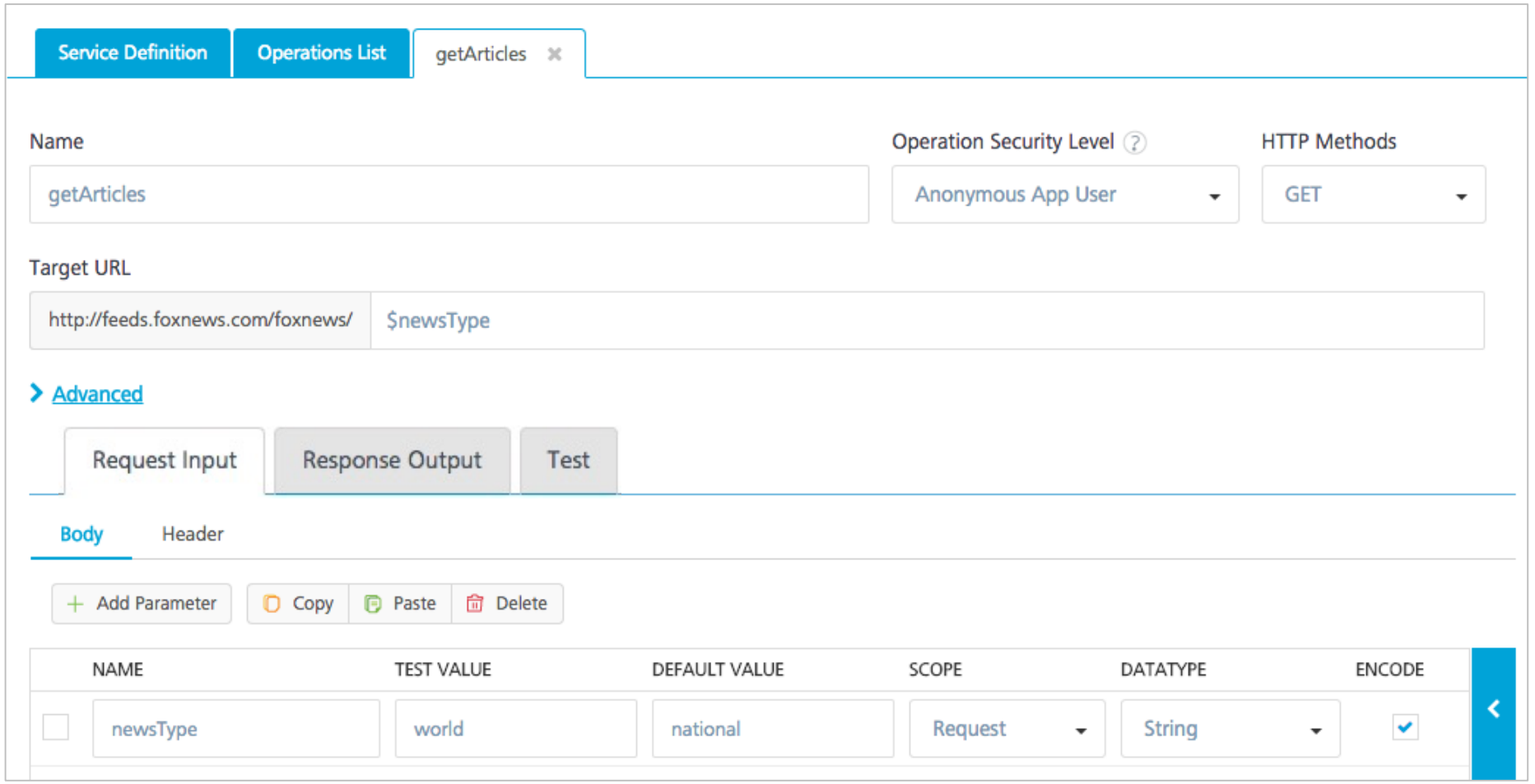
Summary So Far...

- Before covering more materials, let's review where we are at:
  - The XML Integration service consumes XML web services at Kony Fabric
  - You create an Integration service definition using the base URL
  - For the Service, you create one or more Operations that:
    - Specify a URL Suffix that is appended to the base URL for the web service call
    - Specify input parameters (for header and/or body) to pass data into the web service
    - Specify output parameters to determine what data is going to be sent back to the mobile device from Kony Fabric
      - Output parameters use xPath to identify the data elements in the web service's response
      - Output parameters can use Collection IDs to create a collection of data - multiple records of information
      - You can optionally format your output parameters

  - Let's now look at our finished XML service...

Summary So Far... (cont'd)

- Here is our getArticles operation configuration including input parameter:



Summary So Far... (cont'd)

- Here is our getArticles operation output parameters:

Request Input | Response Output | Test

+ Add Parameter | Copy | Paste | Delete | Enable pass-through output body

	NAME	PATH	SCOPE	DATATYPE	COLLECTION ID	RECORD ID	FORMAT	FORMAT VALUE
☐	FoxDesc	/rss/channel/description	response	string			None	
☐	FoxImage	/rss/channel/image/url	response	string			None	
☐	articles	/rss/channel	response	collection			None	
☐	title	item/title	response	string	articles		None	
☐	description	item/description	response	string	articles		None	
☐	pubdate	item/pubDate	response	string	articles		None	

Summary So Far... (cont'd)

- And finally here is our result based on our output parameter configuration:


```
"articles": [
    {
        "thumbnail": "http://a57.foxnews.com/www.foxnews.com/content/dam/fox-news
        "link": "http://feeds.foxnews.com/~r/foxnews/entertainment/~3/GNhRmliAf_8
        "title": "Surprisingly low movie star salaries",
        "desc": "Stars who made less than they should have<img src=\"http://feeds
        "pubdate": "Tue, 12 Dec 2017 10:00:00 GMT"
    },
    {
        "thumbnail": "http://a57.foxnews.com/www.foxnews.com/content/dam/fox-news
        "link": "http://feeds.foxnews.com/~r/foxnews/entertainment/~3/sn2tMahWdRw
        "title": "FOX411's snap of the day",
        "desc": "<img src=\"http://feeds.feedburner.com/~r/foxnews/entertainment/
        "pubdate": "Tue, 12 Dec 2017 10:00:00 GMT"
    },
    {
        "thumbnail": "http://a57.foxnews.com/www.foxnews.com/content/dam/fox-news
        "link": "http://feeds.foxnews.com/~r/foxnews/entertainment/~3/ZxMG1YxF9VQ
        "title": "CNN ridiculed for accusing Trump of bullying ?dumbest man on te
        "desc": "CNN has accused President Trump of bullying Don Lemon over a twe
        "pubdate": "Mon, 11 Dec 2017 21:30:00 GMT"
    },
    {
```


- Note: this is looking at copy/pasted results from the Kony Fabric results window (not all records shown here)

Exercise - XML Integration Service

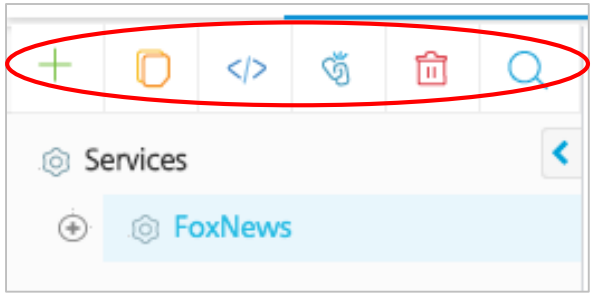
- Ok...let's try it!
- Create a new Integration Service for Fox News: http://feeds.foxnews.com/foxnews
- Create a new Operation for your Fox News service (set your security settings)
- Configure the input parameter to accept a value for the type of article
  - Some valid values are: national, world, entertainment, science, health, and politics
- Configure the output parameters to return:
  - top level data: title, description and the image URL data
  - For each article: title, description and pubDate

- Make sure that you can test and run this service successfully and you see the results formatted like shown in the previous slide
  - Make sure you have a collection of articles where each article's data is inside a record rather than a record for each piece of data and/or multiple collections

- Good luck! when done, we'll continue talking about other configurations...

Managing Integration Service

- Once your service is created, we can use the icons to manage it/them:

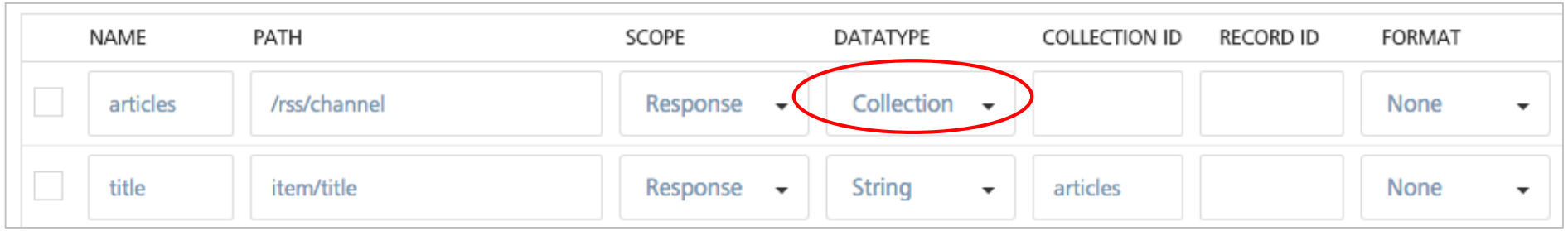


  - Expand the tree - highlight a service or operation

- Here is what each icon does (a service must be highlighted):
  - + pops a menu to choose the action you want: Add New Service, Use Existing or Add New Operation - you pick what you want to do
  - clone is used to make a copy of the current service/operation
  - </> shows you the sample code - we'll cover this next...
  - is used to unlink this Integration service - doesn't delete it, it's just not associated with the App any more
  - is used to delete the highlighted service or operation
  - is used to search your service and operations - useful if you have a huge list and want to find something
    - Hint: use common prefix/suffix text to isolate like services and operations using search

Configuring Collections & Records

- When you save and refresh the configuration, you'll see that the Datatype for the collections' parent is changed to Collection:



  - Feel free to specify Collection as the Datatype when configuring the parent
  - If you forget, Kony Fabric will check the relationships when the operation is reopened
- There is also a Record ID column (and Record Datatype):
  - This is used to group output parameters inside a JSON Object in the results
  - Note: the Datatype will be set to Record if you use that parameter as a Record ID - just like collections above - it's ok to leave the Datatype for the "parent" to String - it'll be changed for you

Configuring Records

- Let's compare the results between these 2 configurations of our "top level" parameters:
  - Here is what we had before - FoxDesc and FoxImage are data at the top level of the results:

NAME	PATH	SCOPE	DATATYPE	COLLECTION ID	RECORD ID
FoxDesc	/rss/channel/description	Response	String		
FoxImage	/rss/channel/image/url	Response	String		

  - ...and here we've configured this information to be packaged up as "info": using the Record ID field
    - The FoxDesc and FoxImage data is now packaged as "info"

The xPath for the children is relative to the parent - just like collections

info	//channel	Response	Record		
FoxDesc	description	Response	String		info
FoxImage	image/url	Response	String		info

Configuring Records (cont'd)

- Here are the JSON results returned to the mobile application (looking at the runtime console results):

showing the very end of the results in the console:


```
        "link":"http://feeds.foxnews.com/~r/foxnews/national/~3/7ulVT-9u3oE/"
      }
   ],"httpStatusCode":200,"FoxImage":"http://tools.foxnews.com/sites/tools.foxnews.com/files/images/fox-news-logo.png",
    "opstatus":0,"FoxDesc":"FOXNews.com - Breaking news and video. Latest Current News: U.S., World, Entertainment, Health, Bu
}
```


Notice that FoxImage and FoxDesc are at the same level as opstatus and httpStatusCode

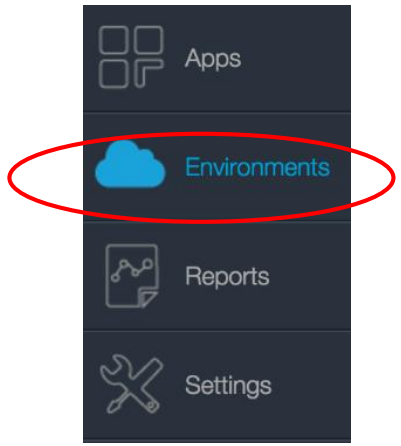
  - Using JS, you'd access the FoxDesc as: response.FoxDesc
- Here are the JSON results using a Record:

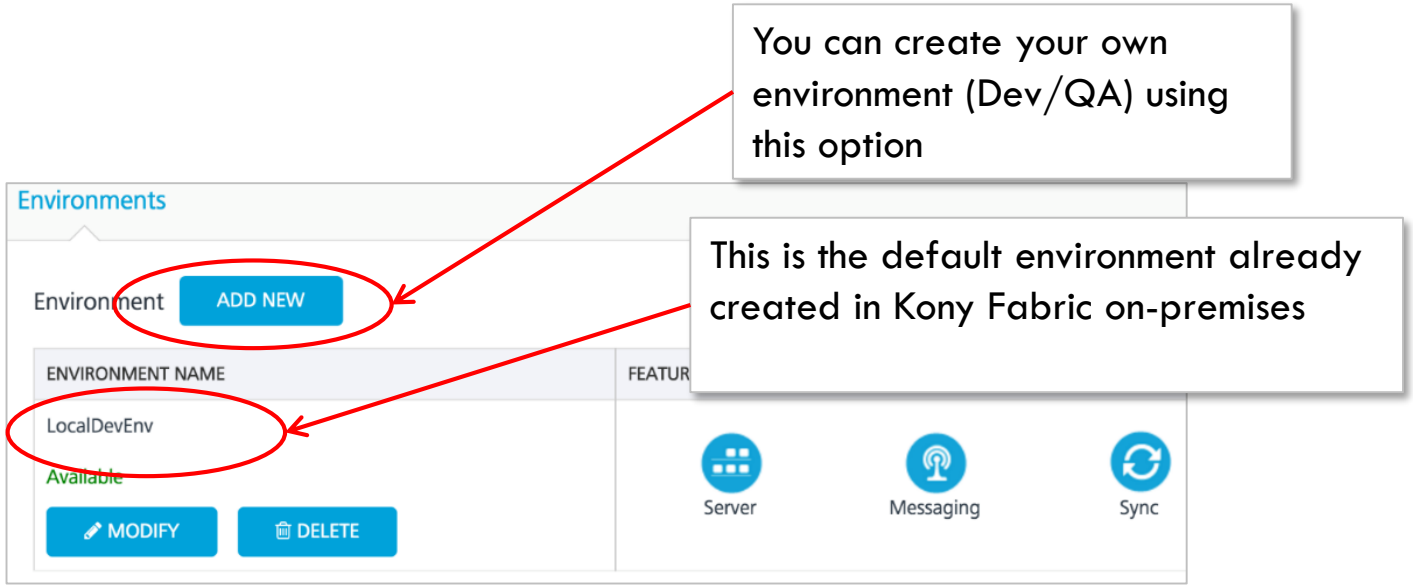

```
      "link":"http://feeds.foxnews.com/~r/foxnews/national/~3/DLfZgQtzopk/"
    }
],"httpStatusCode":200,"opstatus":0,"info":
    {
      "FoxImage":"http://tools.foxnews.com/sites/tools.foxnews.com/files/images/fox-news-logo.png",
      "FoxDesc":"FOXNews.com - Breaking news and video. Latest Current News: U.S., World, Entertainment, Health, Bu
    }
```


Notice that FoxImage and FoxDesc are now packaged as an object called info

Publishing the App

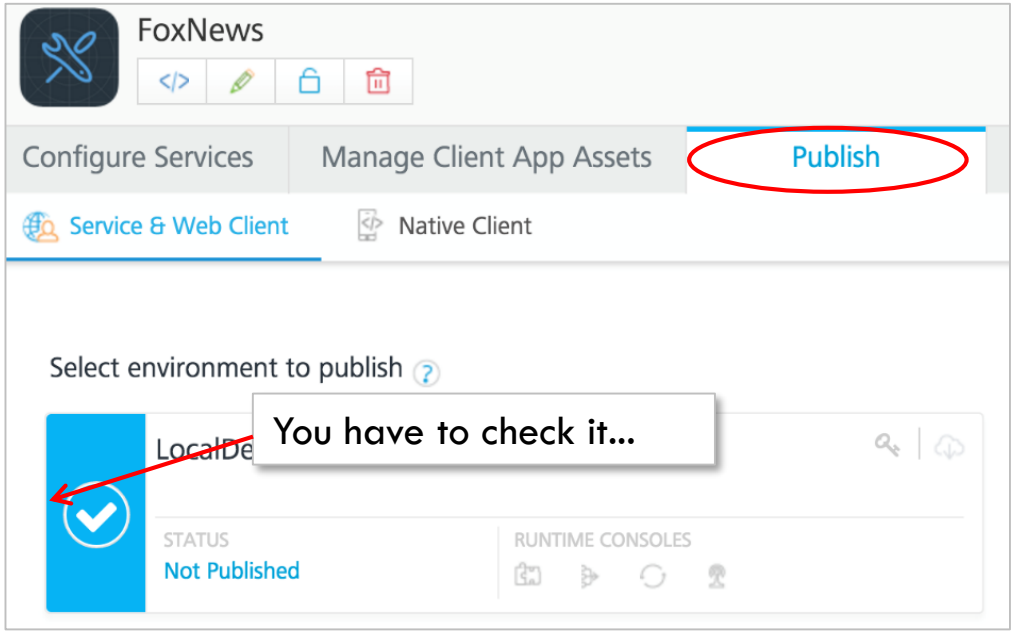
- Until you **Publish** your app, it's not available for use by the mobile application
- Publishing will put all your services into active production in the environment you select
  - You can publish dozens of apps - resources are only used when the app services are in action - the actual app doesn't consume resources just because it's published
- To publish an app
  - First, you need to create an environment or use existing.





Publishing the App (cont'd)

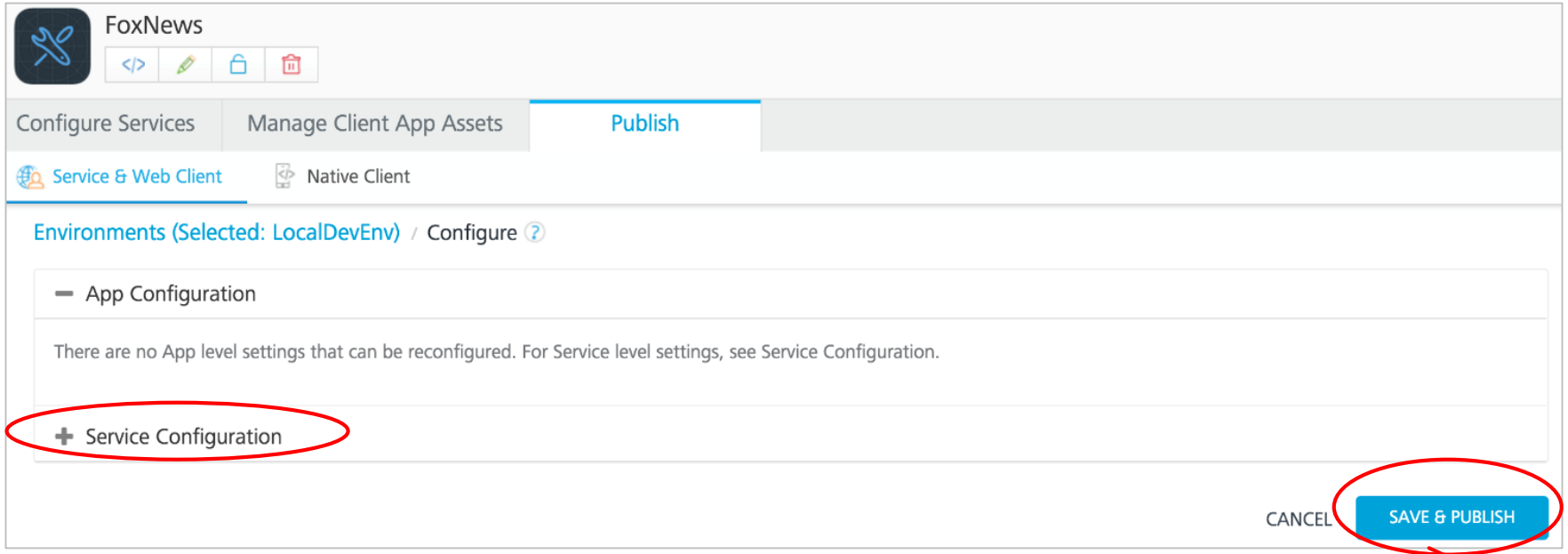
- Lets publish our **FoxNews** app to existing **LocalDevEnv** environment. Go to our FoxNews app, click on **Publish** tab and then select the environment.



- Click on NEXT button.

Publishing the App (cont'd)

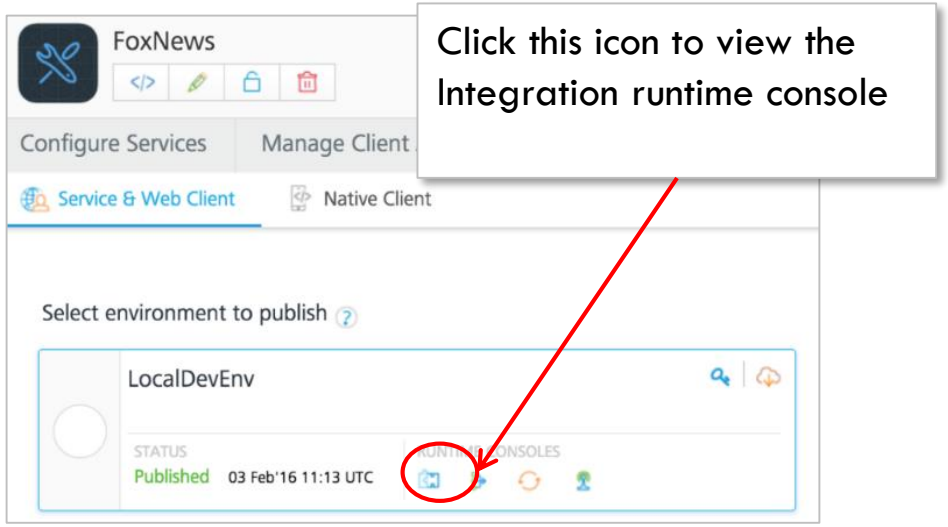
- You have a chance to edit some values in your Service Configurations...



Click here to finish publishing.

Integration Services Runtime Console

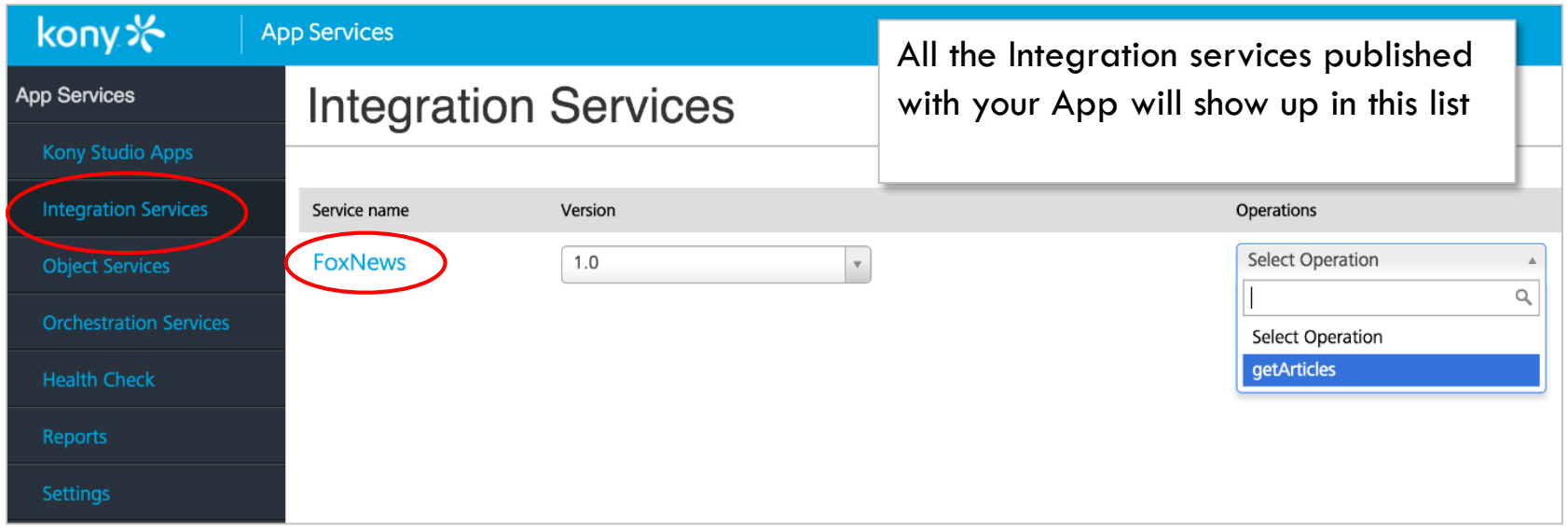
- You can check if your integration services are published or not by opening Integration Services runtime console. On the Publish screen, you can access all your App's Integration services runtime:



- The runtime console is launched in a new browser window
  - Note: currently it doesn't land on the specific feature selected - that'll change in time
- Let's take a look...

Runtime Console

- On the left is the menu - we want Integration Services:



Exercise - Publish

- Go ahead and publish your app now
- When you're done, test this using your Admin Console
- Next we're going to consume this with your client application

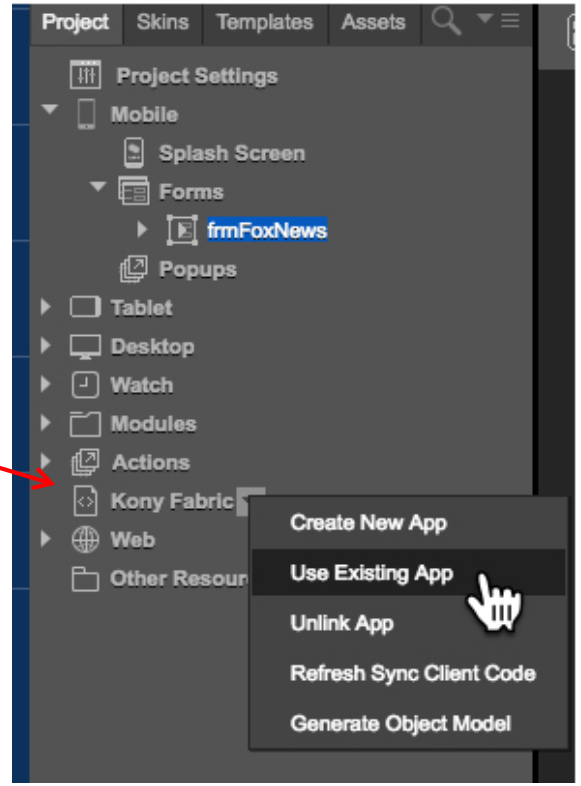
Kony Fabric Client SDK

- So far, we have configured the FoxNews XML integration service in Kony Fabric.
- How do we use this in our mobile application?
- We use Kony Fabric SDK and its APIs to call these identity and integration services from mobile application.
- Use the Kony Fabric documentation - it also has the API guide and more info about how to use it

Using the SDK - Initialization

- As we said earlier, we are not going to cover all different versions of the SDK - we'll use the Kony SDK in all our examples
  - The object model and concepts are identical between SDKs

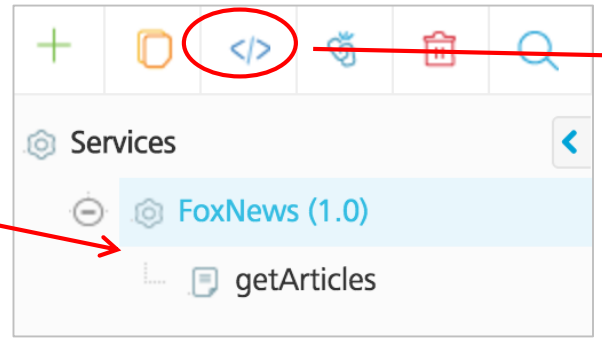
- The first step is to initialize an instance of the SDK that identifies the Kony Fabric App and URL
- For initializing the SDK, you simply link the Fabric app and Visualizer Enterprise

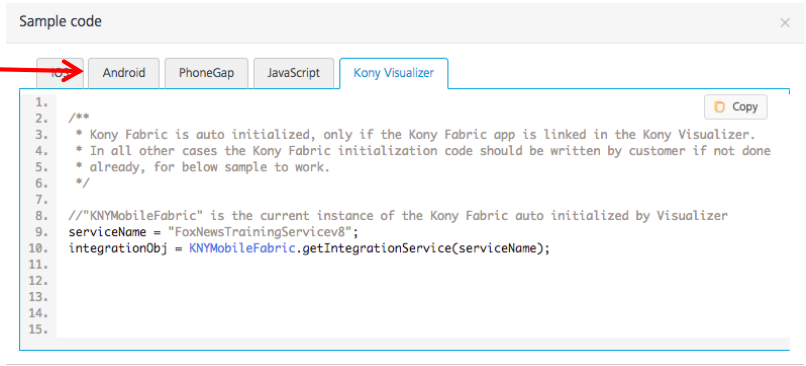


Calling our Integration Service

- In order to call a specific Integration Service's operation, you have 2 steps:
  - Create an instance of the Integration Service
  - Invoke the Operation
- In Kony Fabric, the sample code is provided for you!
- When you highlight your Service (not Operation), click on the sample code icon and you can see your sample for each SDK type:

With the service selected...





- Note: in the sample code, it "assumes" that you have a valid client instance (in the example above, it's KNYMobileFabric)
  - What is "valid"? it means that you linked Visualizer and Fabric

Invoking Operation

- Now that we've created our integrationObj with our service definition, we can call our operation:

```
//here is where we left off - getting our service definition
integrationObj = client.getIntegrationService("FoxNews");
//here is the invoke
operationName =  "getArticles";
data= {"newsType": "<place-holder>"};
headers= {};
integrationObj.invokeOperation(operationName, headers, data, operationSuccess, operationFailure);
```

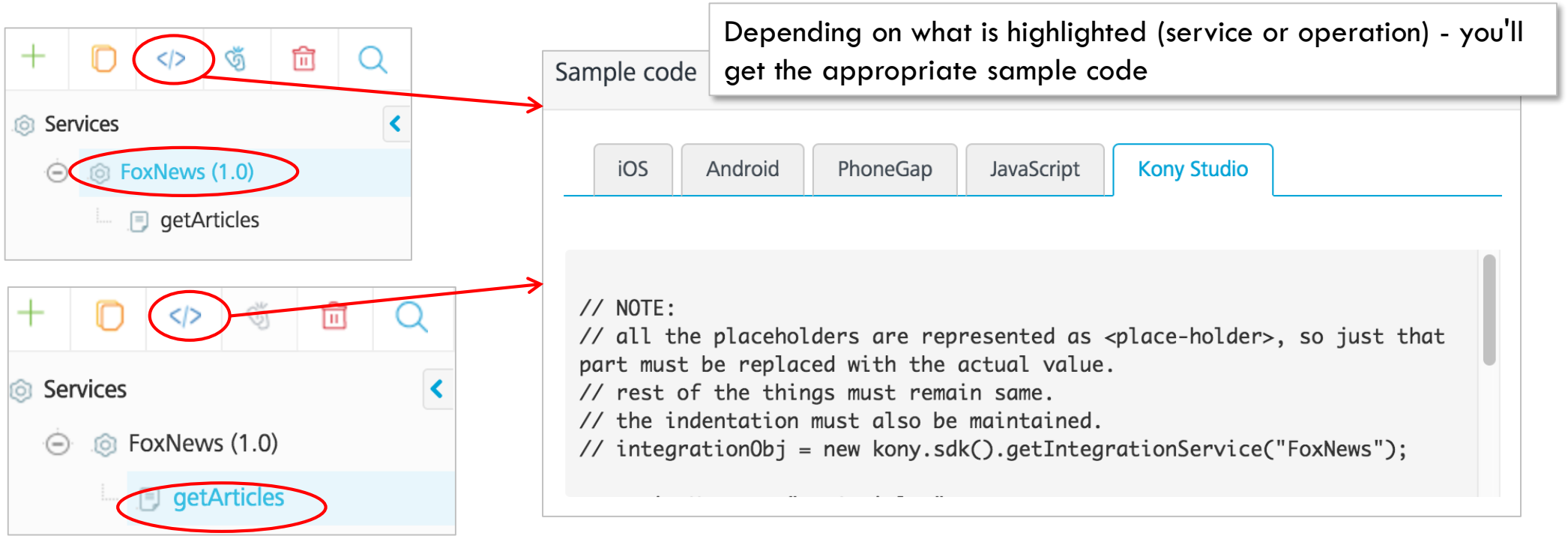
- Where our callbacks look like:

```
operationSuccess:function(res){
        //code for success call back
},
operationFailure:function(res){
        //code for failure call back
}
```

- How do we know how to call this? Sample code!
  - Let's take a look...

Service/Operation Sample Code

- In the Integration service tree, highlight the service or an operation to access sample code:



- The MAIN thing to keep in mind is to make sure that you've invoked the proper Identity service prior to calling the Integration service
  - Then make sure you pass values for all input parameters (body and/or header)

Preparing your front-end app

- Using Kony Fabric, select your Published application and find the sample code for the service and the operation

- "Wrap" the calling code with a function that we can call from our client application

- You should have 3 functions when you're done
  - The Service sample code and the InvokeOperation
  - The Success Callback
  - The Failure Callback

- We'll use this in our application controller shortly

Client Results

- When you call an Integration Service from the client, the data is always returned in JSON
- Here is what you get back:
  - httpStatusCode - tells you the HTTP status of the web service request - 200 means it worked successfully
  - opstatus - is the error code - returns 0 for a successful call, or a non-zero value if there was an error and then you also get:
    - errmsg - a error message
  - Your data specified in the output parameters
- Here's an example (showing simplified article results):

```
{
 opstatus:0,
 httpStatusCode:"200",
 FoxDesc:"FOXNews.com - Breaking News and Video",
 FoxImage:"http://tools.foxnews/sites/tools.foxnews.com/files/images/fox-news-logo.png",
 articles:[
            {
             title:"Something wonderful happened today",
             description:"This article talks about something lovely that happened today"
            },
            {
             title:"Nothing bad happened today",
             description:"This article doesn't have any bad news to report"
            }
          ]
}
```

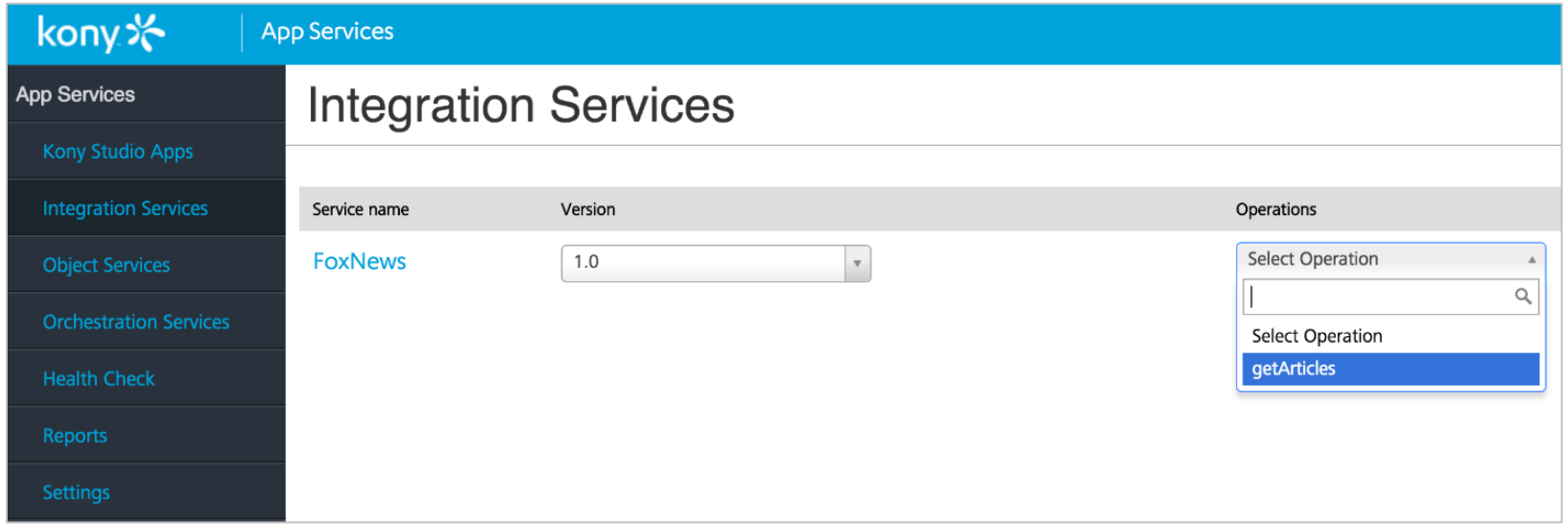
Testing an Integration Service Operation

- Once we have our Integration service instance, we can now call the individual operation(s) for that service
- Testing the results in the Integration service definition is NOT the same as testing using a mobile application client
  - Testing in the definition window is not running it from the production engine - it's a test mode for validating your configurations
- Before you can test outside of the definition window, you'll need to PUBLISH your application
  - ANY time you change/add/delete any services (integration, identity, etc.), you MUST publish to put them into production before you can consume them with an app
- Once published, we can now go build a mobile application and test it out by calling the service with the SDK

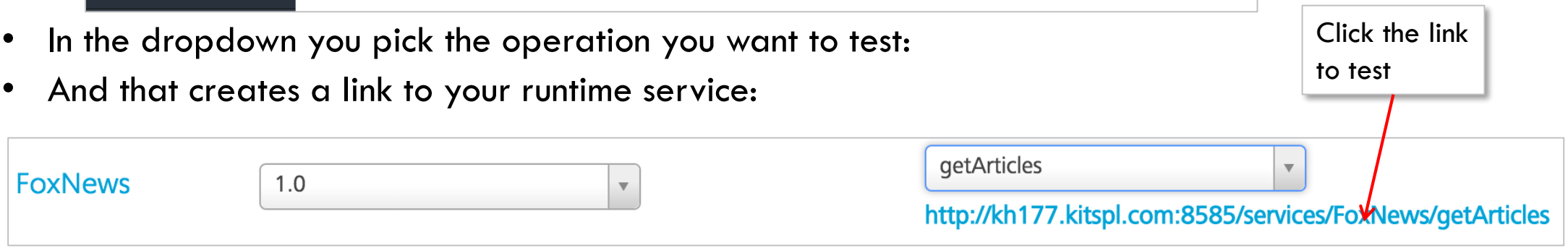
  ...OR...
- You can test by calling the runtime version directly
  - Let's take a look at that..

Testing in Runtime Console

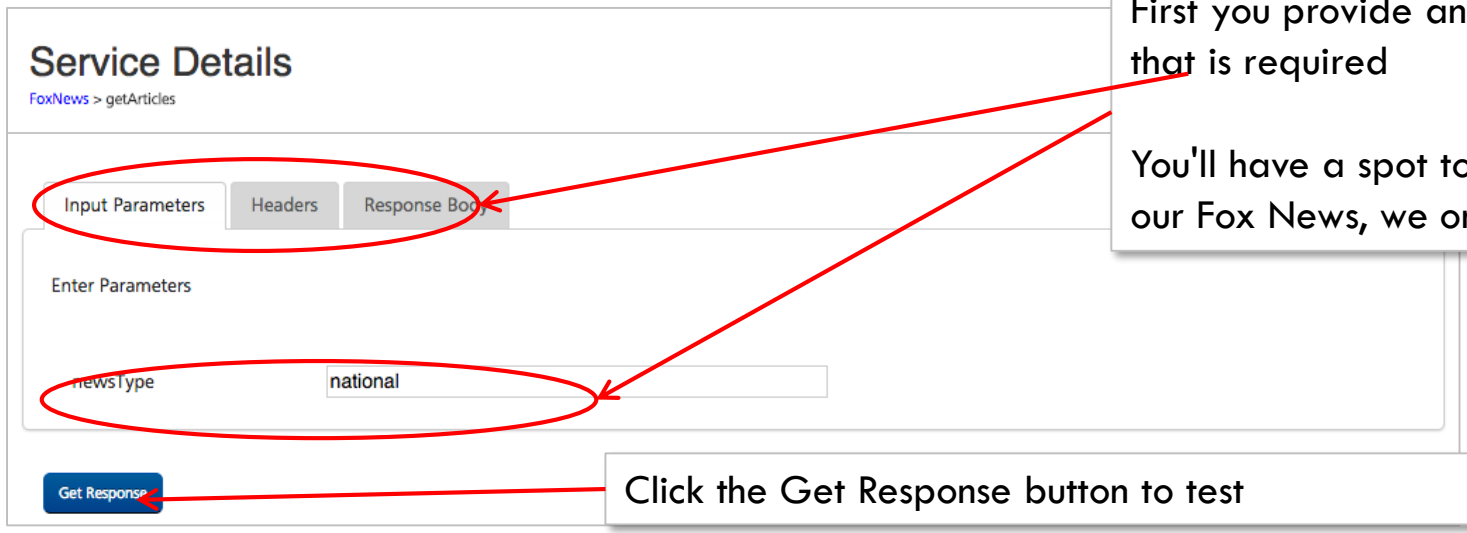
- Remember the Integration services runtime console where you will see all of your app's integration services after publishing your app



- In the dropdown you pick the operation you want to test:
- And that creates a link to your runtime service:



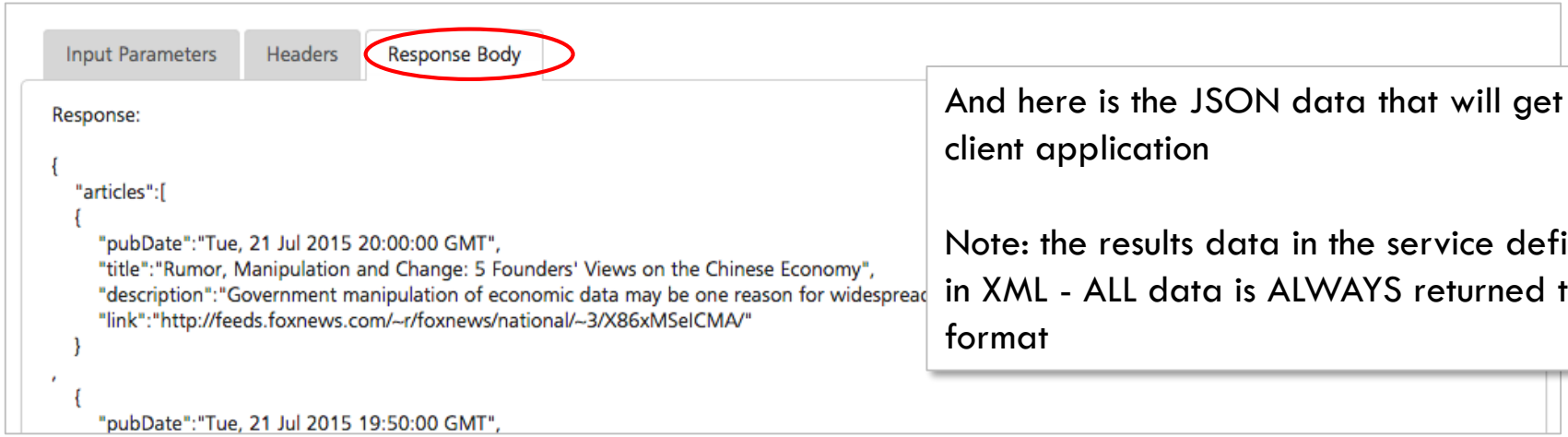
Testing in Runtime Console (cont'd)



First you provide any input parameter/header data that is required

You'll have a spot to enter each piece of data - for our Fox News, we only have one input - the news type

Click the Get Response button to test

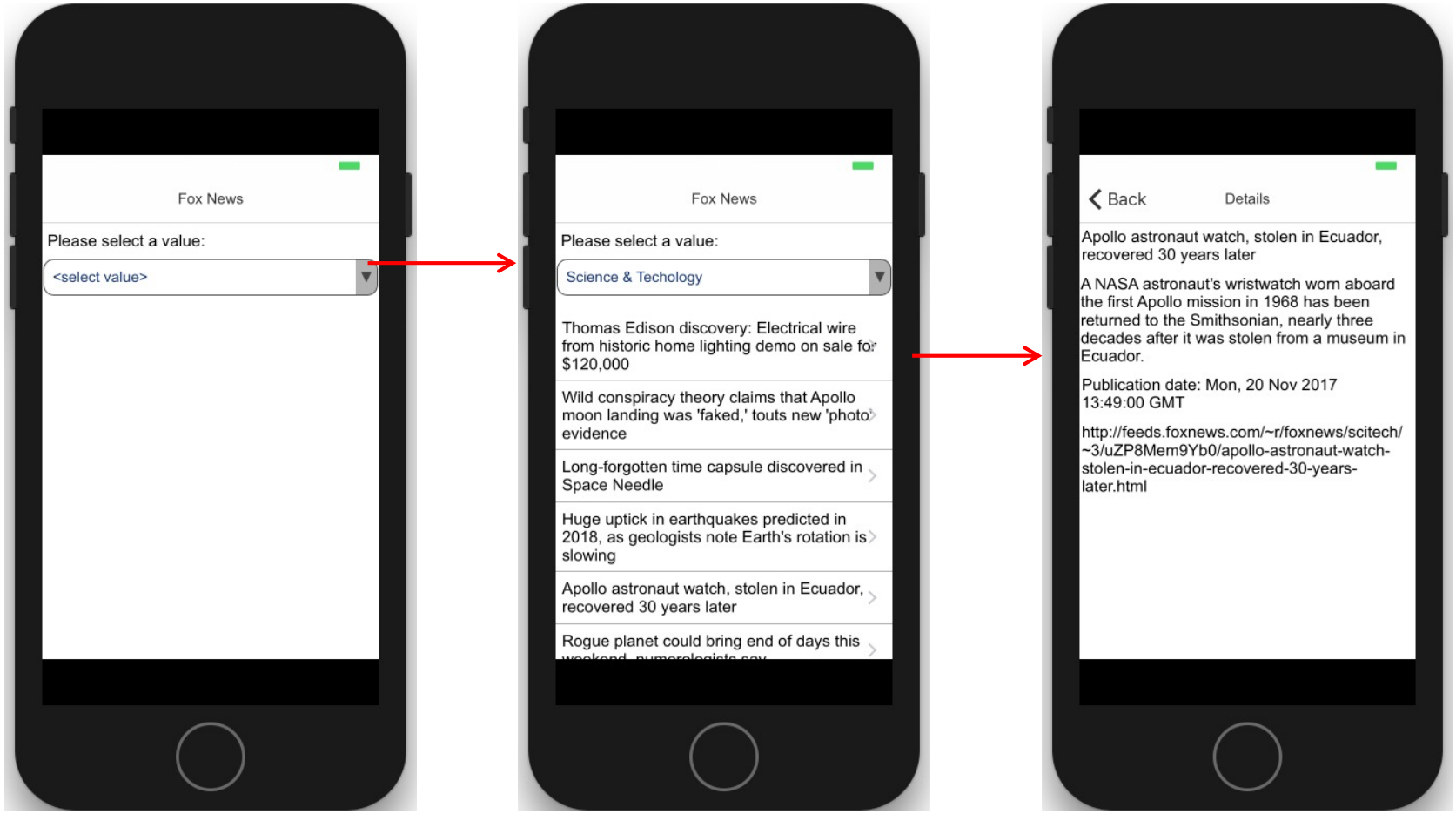


And here is the JSON data that will get returned to the client application

Note: the results data in the service definition showed data in XML - ALL data is ALWAYS returned to the client in JSON format

Calling Services - A Sample App

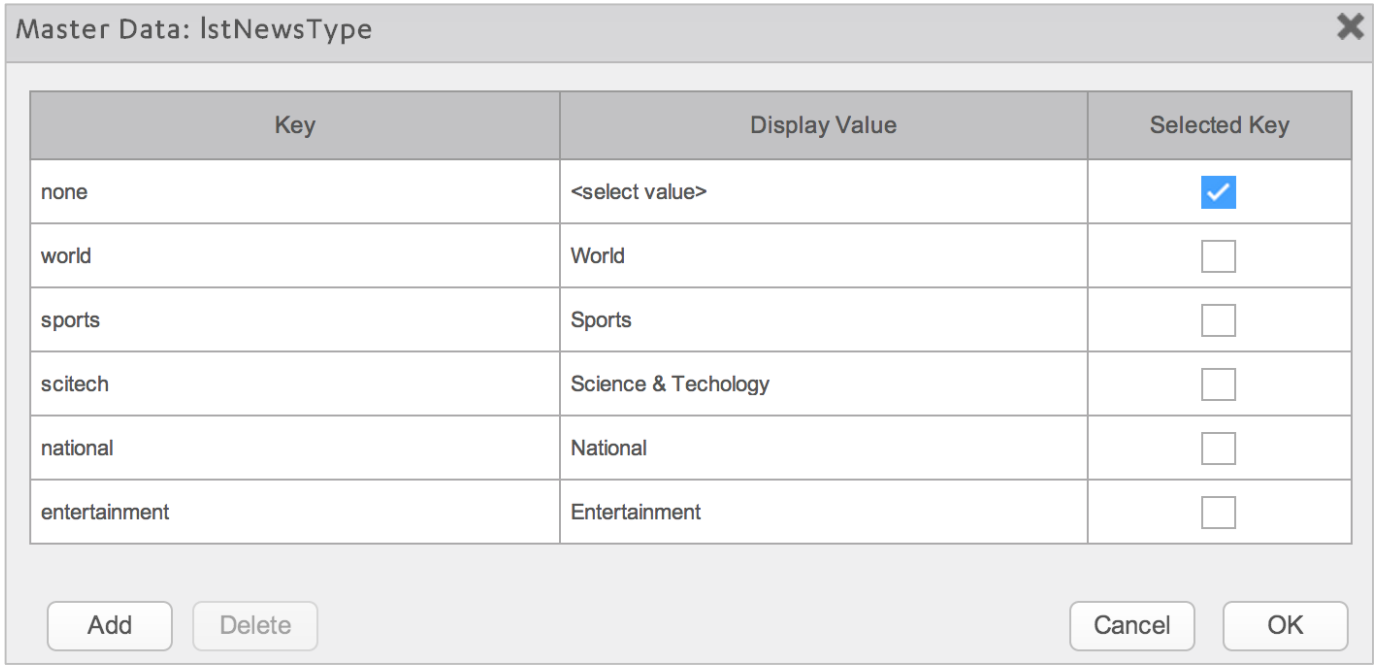
- Here is a sample 2 screen news app



Notice the description - the HTML <img src….> is embedded in the text - use JavaScript string functions search() and slice() to strip that text off

Creating the App

- The first screen is a label, ListBox and segment (with one label)
- ListBox should load with the list of news types, and that has been populated using the Master Data property :



- Note: for our example, the key is the value we actually pass to the service - this allows us to have a nicer display value (for example: "Science & Technology" is better than showing "scitech")

Creating the App (cont'd)

- When the user selects a value from the ListBox we then need to:
  - Make our service call using the value that was selected
  - Populate the segment with the titles in the results
- We know how to make the service call with the callback…right? Remember we configured Kony User Repository identity service and FoxNews integration service. Below are the steps for calling the services
  - Initialize the Kony Fabric via Visualizer
  - We can now call the actual integration service **FoxNews** and its operation **getArticles**

- We already discussed the code and the APIs to implement the above steps. Lets go ahead and write the code…

Creating the App (cont'd)

- To populate the segment :
  - We'll need to do a widgetDataMap for the label ID mapping to the output parameter for the article title
  - For example: this.view.segTitles.widgetDataMap={lblTitle:"title"};
    - Note: OR we can give our label ID the same name as the output parameter and not need the widgetDataMap - your choice
  - We'll then set the data by passing in the articles collection
    - Here's an example: this.view.segTitles.setData(response.articles);
    - This assumes that response is passed into the callback and your collection was called articles

- Note: the segment will "disappear" when the form is loaded because by default it has no data. When you call setData it will automatically refresh and show the results

Creating the App (cont'd)

- Now the segment is populated with news article titles. What happens when the user clicks a title?
- We want to populate the 3 labels on the second screen with the article's title, description and publish date
- How do we do that?
  - The segment's `selectedRowItems[0]` will contain all the data for the selected row
    - Remember how that works? By default the segment is in single-select mode so we'll grab the first item in the collection (i.e. `[0]`)
  - We just take each attribute from selected record and set the label's text property and then show the form
    - Since we sent the segment the entire results collection, we effectively have 2 "hidden" values (description and publish date) - those values that are stored in the segment but not displayed
    - To access the selected description (assume your output parameter is desc), you'd use:

```
this.view.segTitles.selectedRowItems[0].desc
```

Integration Services - Best Practices

- For all the services it is a best practice to check the opstatus value
  - If opstatus is zero, consider the service as a success
  - If opstatus is non zero, then it is a failure. The reason for the failure will be present in errmsg in the results
- If the service is a success:
  - Make sure you check that data was returned before assuming it was
  - Many times the service call is successful but returns no data
    - Example: searching for something that yields no results
- Don't forget to wrap your service code in try…catch blocks to handle any errors appropriately
- KNOW YOUR DATA! will all the fields come back with data? is null a valid value? It's critical to know what type of data will be returned

Server Error codes

- Some of the common server error codes (returned as opstatus) you are likely to run across in development are:
  - 10101 - Application does not exist with the AppID
    - Typically caused by having the wrong AppID or a simple misspelling in your service call input parameters
  - 10102 - Service does not exist with ServiceID
    - Typically caused because you have a typo in our ServiceID input parameter OR you forgot to publish your service to the server (so it doesn't know about it)
  - 8007 - Error parsing the XML response
    - Typically caused if non-XML data comes back. Check the service definition, many times it might default to a JSON result if XML is not specified.
- The Kony Server Troubleshooting Guide has the complete list of these error codes
  - Note: many times looking in the server's middleware.log will give you more info on what might have gone wrong

Services - Testing Considerations

- Let's review:
  - Services are DEFINED in the Kony Fabric console and RUN on the Kony Fabric server
  - Any time you need to change a service, you MUST republish!

- Let's say that you messed up the xPath for an output parameter and you need to fix that
  - Change the integration service configuration in Kony Fabric console and save the changes
  - Re-publish
  - Restart the app? NO! You don't need to, right? Unless something changed in your client code, you can just re-try that service in your app and it should now bring the right data back
  - If you changed an output parameter name or added/removed values then you will have to change the app and re-launch

Services - Testing Considerations (cont'd)

- Common mistakes to look for:
  - Calling the Integration service without initializing the Kony Fabric SDK and calling identity service if required based on service configuration
  - Incorrectly setting up the input parameters for your service call (for example, the key is “newsType" and not “newstype" or “NewsType")
  - Trying to work with the results in code when the results are not there
    - Immediately print out your results to the log or put a breakpoint
    - VALIDATE you're getting the right data back before trying to do any code manipulating the results
    - In your code, always put some IF condition around the results checking if they are there are not
  - TYPOS…ugh!
    - Use code assist whenever possible
    - Check your services in Kony Fabric console to see the service definition input and output parameter names as you type in your code
    - Use logical widget names and variable names
  - "I fixed my service, but my test still fails!" - REPUBLISH your service

Kony Widget API

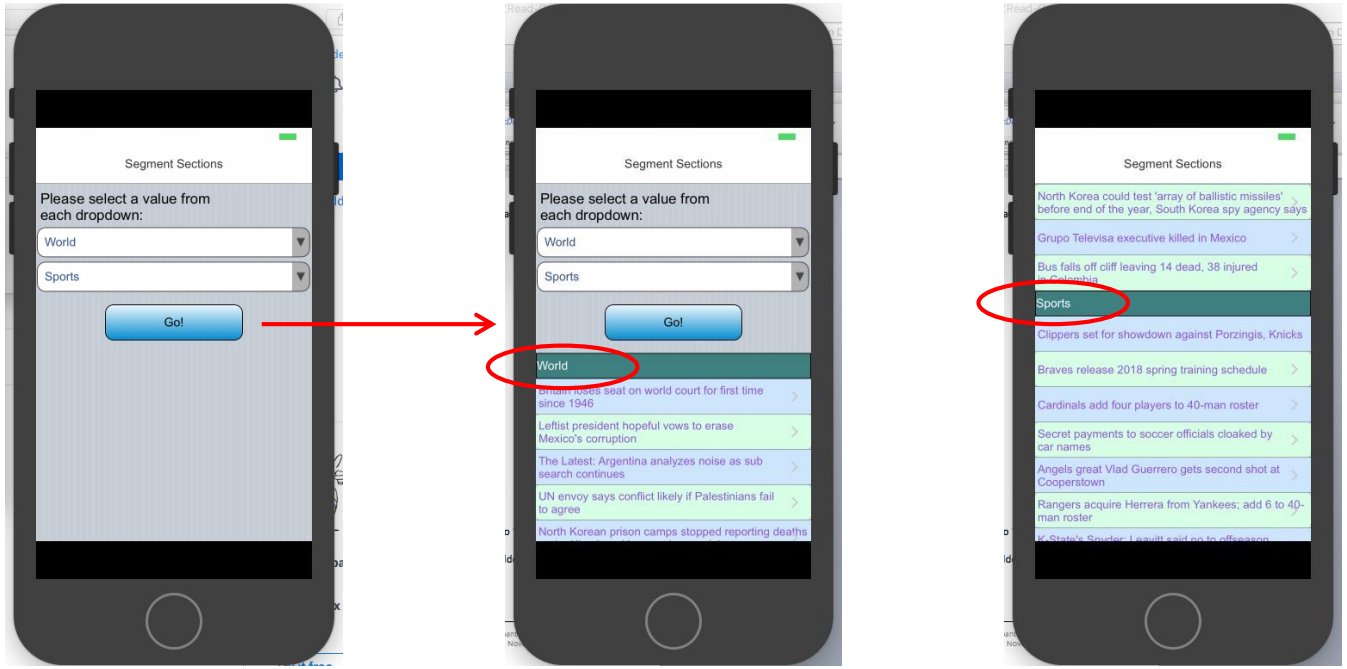
kony Stay Ahead

Segment - Revisited

- The segment widget is part of EVERY app!
- This is a very powerful widget and we've only covered the very basics
- In this module we'll go into other features, including:
  - adding sections to segregate data in groups
  - using segment headers
  - dynamic skinning of rows
  - using row templates
  - dynamically adding/removing rows
  - unique device specific properties

Segment - Sections

- Segments can show a dataset that is divided into sections
  - Each section will have a title
  - Each section will have it's own data set
  - For example, what if we wanted multiple sets of results from FoxNews?



scrolling down to see the next sections…

Segment - Sections (cont'd)

- Segment sections are displayed automatically when the data provides the section data
  - Let's compare how to set data for a "regular" Segment and one with sections:
    - "Regular" data format is:

      [{row data},{row data},{row data}…]

      - The data is an array of row objects
    - Section data format is:

      [["section 1 title", [{row data},{row data},{row data}…]],
      ["section 2 title", [{row data},{row data},{row data}…]],
      ["section 3 title", [{row data},{row data},{row data}…]], …]

      - The data is an array of 2 element arrays.
        - the 2 element arrays are in the format of:

          [<section title>,[array of row objects]]
  - Let's look at a programming example…

Segment - Sections (cont'd)

- In our FoxNews exercise, we took the service results and set them directly into the Segment:
  - The code snippet looked something like:

```
this.view.segArticles.widgetDataMap={lblTitle:"title"};
this.view.segArticles.setData(results.articles);
```

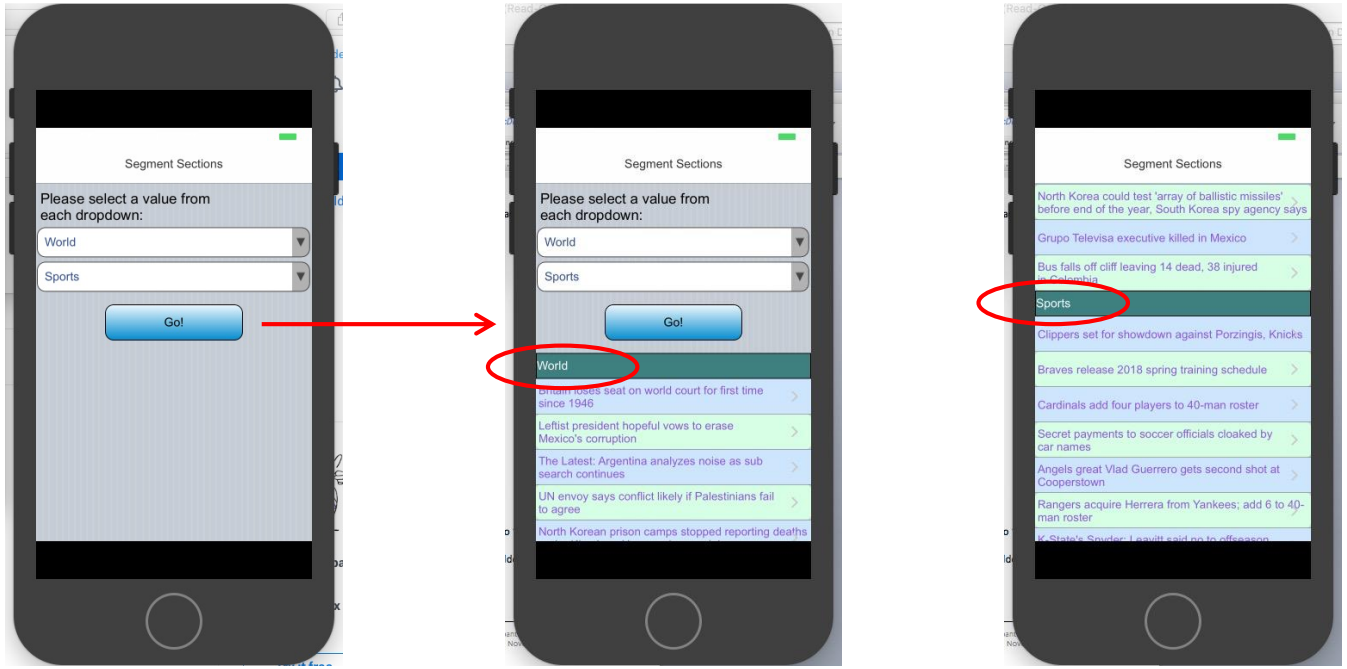
- In our multi-section example, we need to take ALL the results and package them up
  - For simplicity I am hardcoding values, but the code would look like:

```
var articlesForSections=[];
articlesForSections[0]=["World", worldResults.articles];
articlesForSections[1]=["Science & Technology", scitechResults.articles];
articlesForSections[2]=["National", nationalResults.articles];
this.view.segArticles.widgetDataMap = {lblTitle:"title"};
this.view.segArticles.setData(articlesForSections);
```

- Note: The setData method is used for both - depending on the data, the Segment will use the right format

Segment Sections - Exercise

- Let's try setting up some sections
- Modify your Fox News app to show 2 sets of results
  - Change the screen to have 2 sets of dropdowns and add a Go! button
  - On the Go! click, create the data in section as shown below:



Segment Sections - Exercise (cont'd)

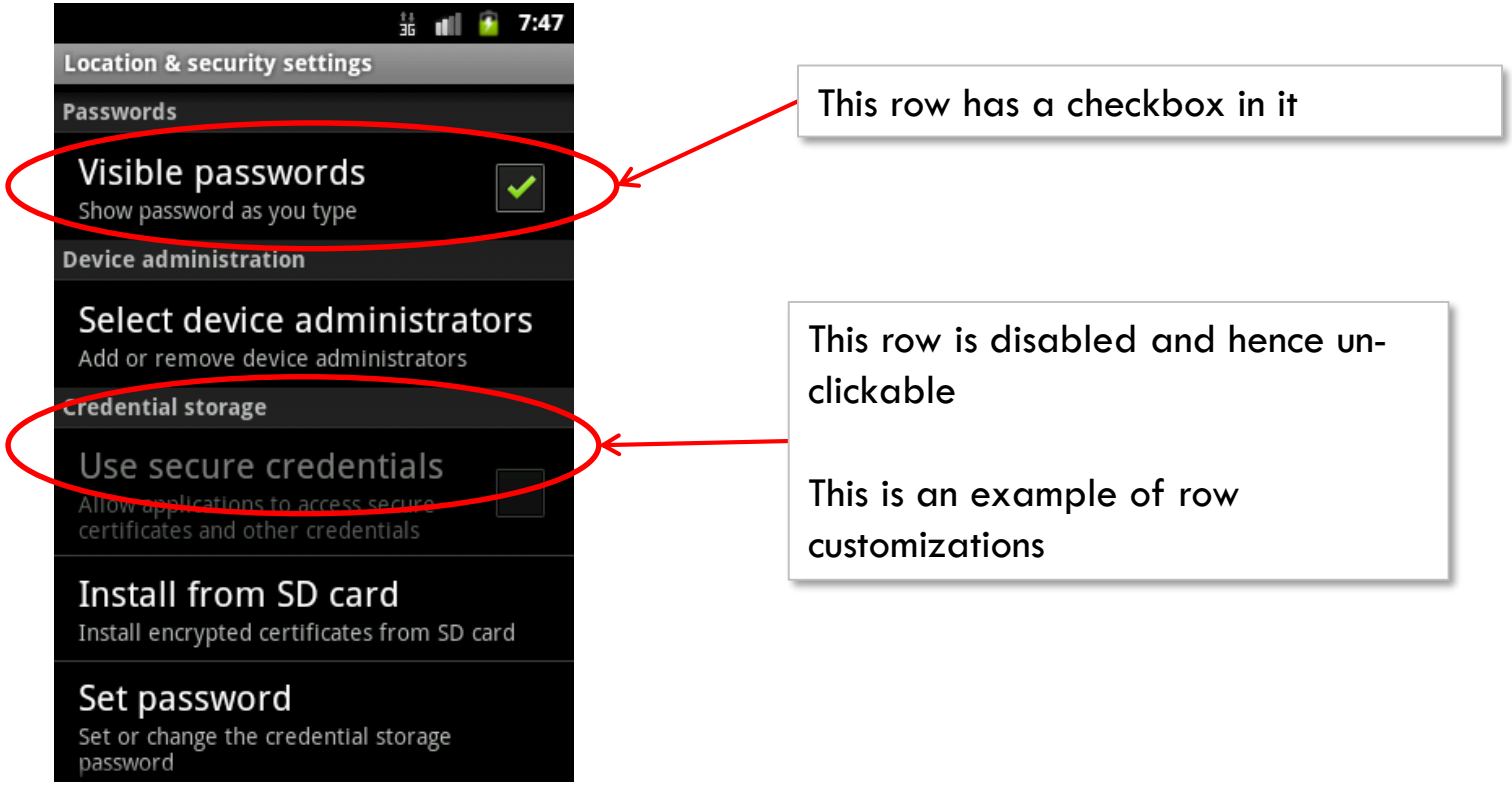
- Let's talk about what you really need to think about for this exercise:
- Building your Segment data set:
  - You do this in the service callback
  - You don't know in what order the data will come back – they are independent calls
  - You must keep track of whether you are processing the first result set or the second
  - Your code will look something like (in pseudo code):

```
//add results to mySegmentData array with a nice section title
if (mySegmentData.length==2)
{
        //we have our results so send data to the segment
}
```

  - Adding the results to the data array will have to know if you are adding data to `mySegmentData[0]` (first data set) or `mySegmentData[1]` (second data set)

Segment - Row Customization

- What if each row in my segment needs to format the data differently?
- Here's an example of an Android settings menu that could be implemented with a segment with sections:



Segment - Row Customization (cont'd)

- There are 2 ways to affect the Segment on a row by row basis:
  - metainfo:
    - This is the data that can be appended to any row
    - Can be used to control: clickability of the Segment skin
  - templates:
    - Create a unique layout that can include/not include widgets to create a unique row look/feel
    - Can selectively apply them to each row
    - Can create templates for rows and section headers

- We'll first take a quick look at metainfo and then we'll talk templates…

Segment - Metainfo

- metainfo is a key we can add to each row of segment data
- metainfo has the following keys we can set:
  - clickable: Specifies if the row is clickable. Set the value to true for the row to be clickable. When false, the onRowClick event is disabled and no focus skin is shown
  - skin: Specifies the skin to be used for a row.

- If metainfo is not specified, the Segment row will act "normally"

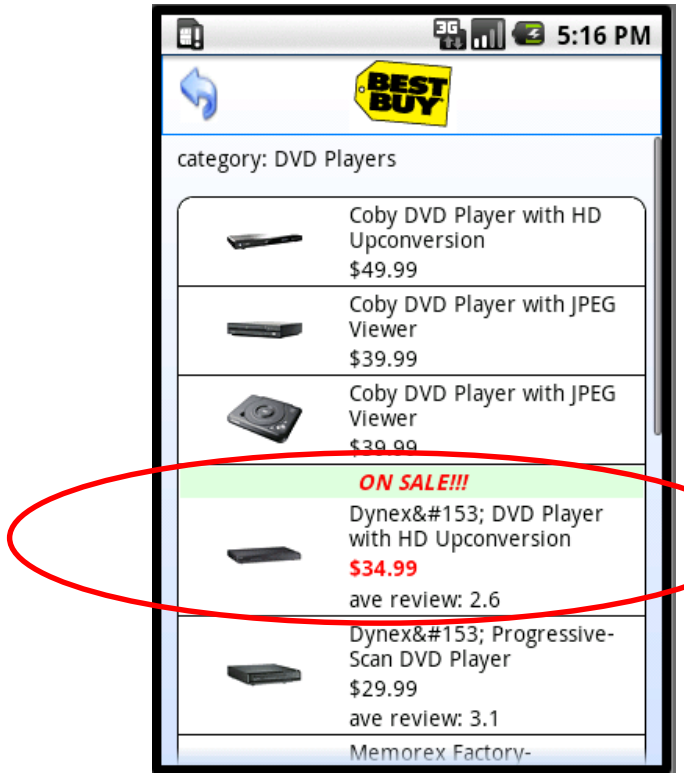
- Remember that the format for a row of data is a series of key/value pairs. For example:

  {fname:"Fred",lname:"Flinstone",age:34}

  - To add metainfo, just append it to the data. For example – to make a row unclickable and with a special "disabled" skin: {fname:"Fred",lname:"Flinstone",age:34,
    metainfo:{clickable:false,skin:skSegDisabled}}

Segment - Dynamic Skinning with Templates

- Here's an example of a list of products
  - A product that is on sale shows a new banner (ON SALE!!!) and changes the skin for the product price:



In this case, using Dynamic Skinning:

- the banner label (ON SALE) is set visible true/false depending if it's on sale
- The skin for the price is set to a bold red if on sale and normal text if not

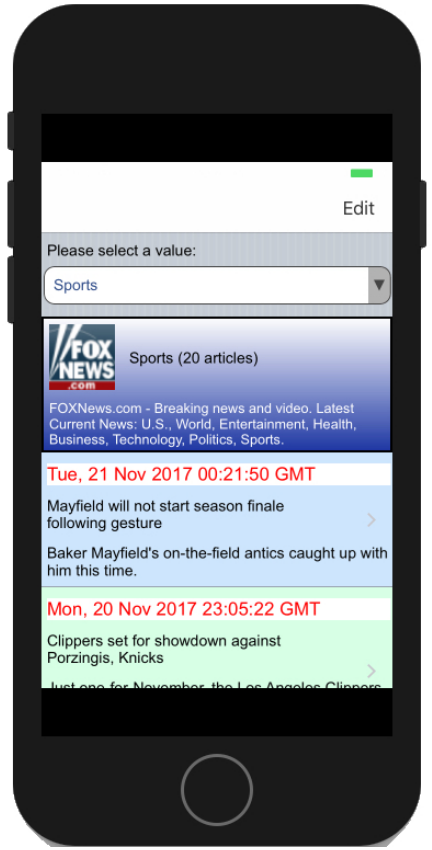
Segment - Templates

- The modifiable properties are very useful if you need to change a few pieces of data
  - Showing an "on sale" banner for those items
  - Bolding some text
  - Generally used for similar pieces of data in the Segment (products, account balances, etc.)
- There are times, however, when you might want to change the complete layout and/or look and feel of a row
  - For example, for the Fox News service:
    - Sports articles have a source field
    - Entertainment articles have a media field showing a picture
    - World articles have a thumbnail
  - You can have a completely different layout showing different widgets for each type of article

Segment Template - Example

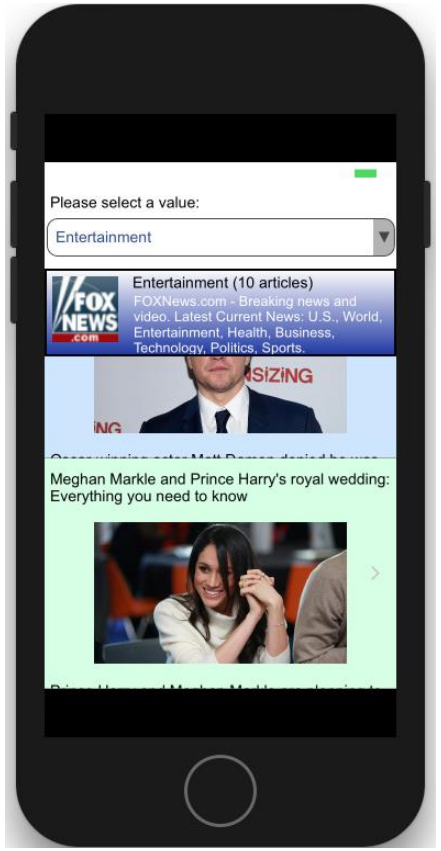
- Here is the Fox News app applying a different template for different news types:

Sports news:



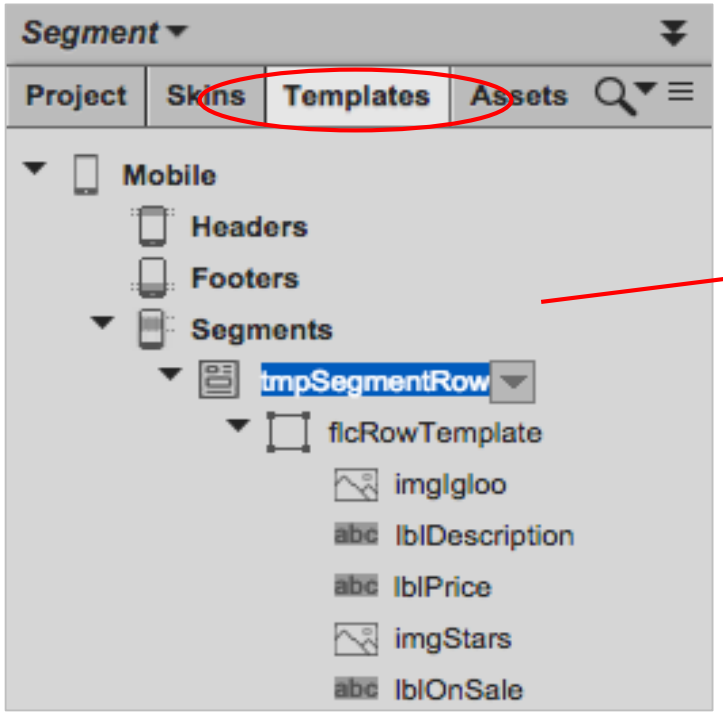
World news:

Entertainment News:

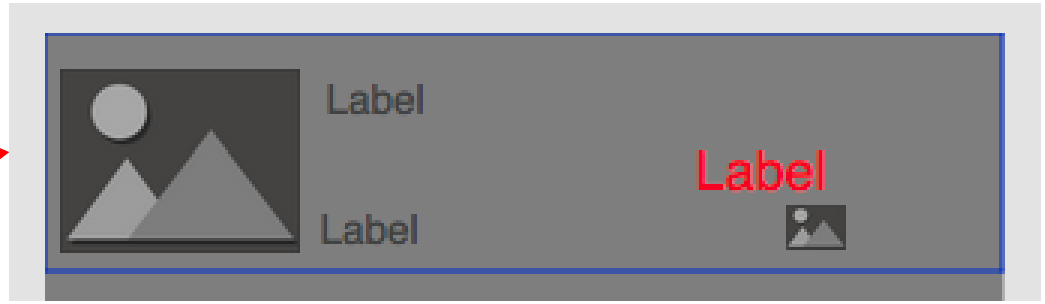


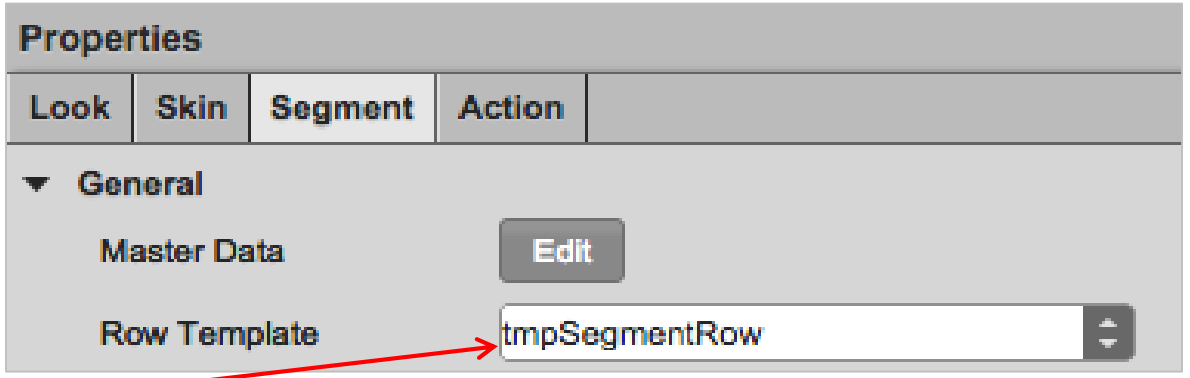
Segment Row Templates

- Remember segment row templates?



Configure your template how you want each row to look:





Now pick your row template and configure the data like before – in canvas or using Master Data

Segment - Using Templates

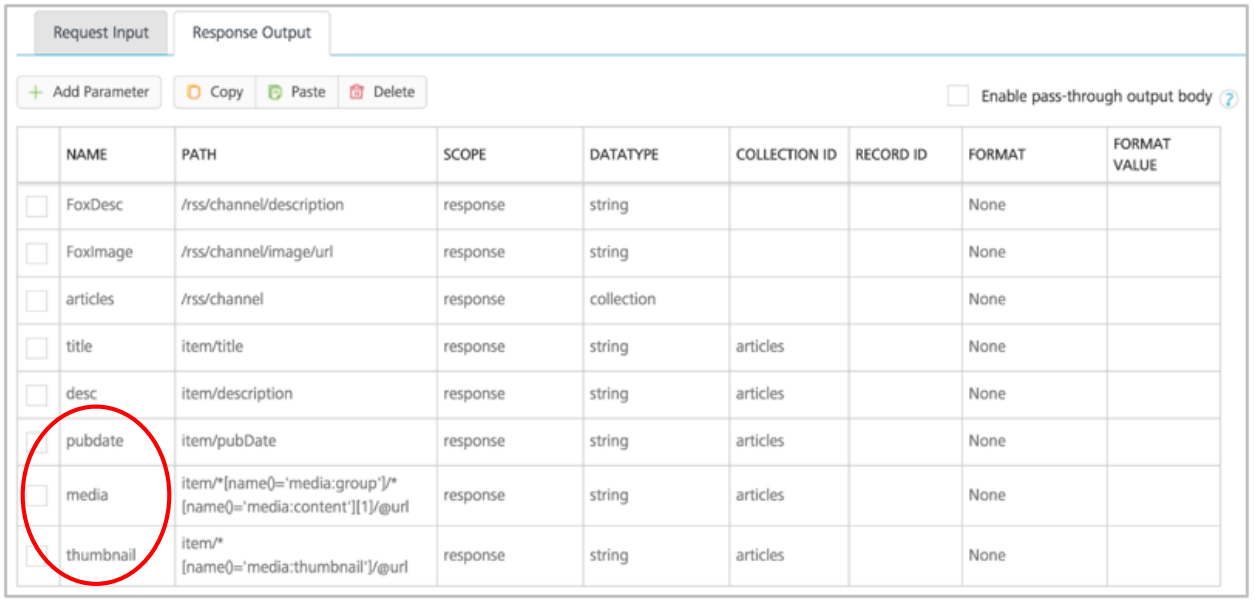
- In the Visualizer, you can specify ONE row template at DESIGN time for all rows:

- In Code, if you want to use a template for a specific row, you can add it to the row data as follows:
  { <row data> … , template:flexSecHeader}
  - Just add the key template and set it to the ID of the top level container in my template

- Rule of thumb – in the IDE you use the template NAME, in code you use the top level container ID

Segment - Creating Templates

- Our Fox News example used 3 templates with different data…how does that work?
- First of all, we want a common data set. We do this by creating ALL our output parameters for all our types:



	NAME	PATH	SCOPE	DATATYPE	COLLECTION ID	RECORD ID	FORMAT	FORMAT VALUE
☐	FoxDesc	/rss/channel/description	response	string			None	
☐	FoxImage	/rss/channel/image/url	response	string			None	
☐	articles	/rss/channel	response	collection			None	
☐	title	item/title	response	string	articles		None	
☐	desc	item/description	response	string	articles		None	
☐	pubdate	item/pubDate	response	string	articles		None	
☐	media	item/*[name()='media:group']/*[name()='media:content'][1]/@url	response	string	articles		None	
☐	thumbnail	item/*[name()='media:thumbnail']/@url	response	string	articles		None	

- pubdate we'll use this for Sports articles
- media is only returned for Entertainment articles
- thumbnail is only returned for World articles

Segment - Creating Templates (cont'd)

- In our code, we need to recognize where to apply each template. For example:

```
var selectedKey = this.view.lstNewsType.selectedKeyValue[0];
for (var i=0;i<results.articles.length;i++)
{
        if (selectedKey === "sports")
        {
                results.articles[i].template="flexSports";
        }
        else if (selectedKey === "world")
        {
                results.articles[i].template="flexWorld";
        }
        else if (selectedKey === "entertainment")
        {
                results.articles[i].template="flexEntertainment";
        }
}
```

- In this example, results.articles[i] is a row in the results – we add the template key and the template's top level flex container ID

Segment - Creating Templates (cont'd)

- The "trick" is the widgetDataMap – you map the appropriate piece of data to the appropriate
- In the previous example, here is what our widgetDataMap looks like:

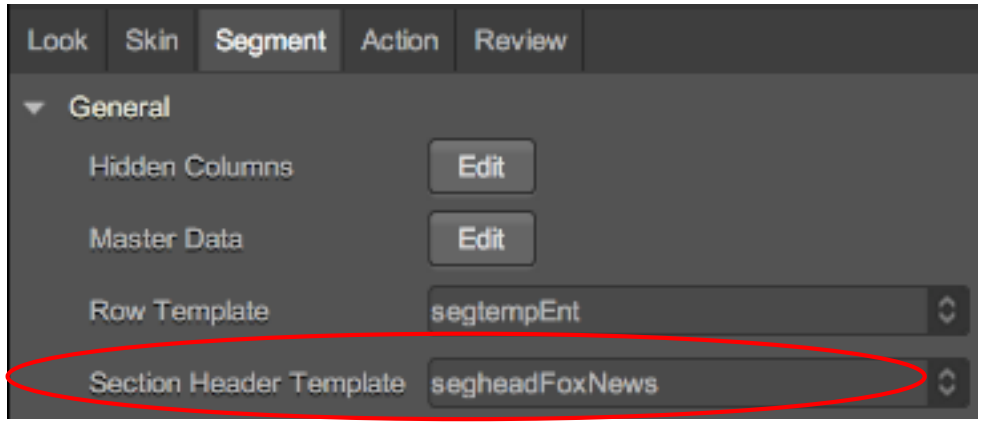
  {lblTitle:"title", lblDesc:"desc", imgPic:"media", imgThumb:"thumbnail", lblPubDate:"pubdate"};
  - lblTitle and lblDesc are labels on all 3 templates
  - imgPic is only on the Entertainment template
  - imgThumb is only on the World template
  - lblPubDate is only on the Sports template

- If my service call returns media data for my Sports articles, it'll not be shown because there is no imgPic widget in my Sports template so that data is ignored
- Just remember – for a segment, there is only ONE widgetDataMap so you must figure out a way to do this common approach if using multiple templates

Segment - Segment Header Template

- You can also create templates for Segment headers
- The same types of rules apply:
  - Create a new Segment template
  - Either:
    - Call it out in the IDE

      …OR…
    - Call it out in code



- In code the section header data is specified as the first element in the 2 element section data array:

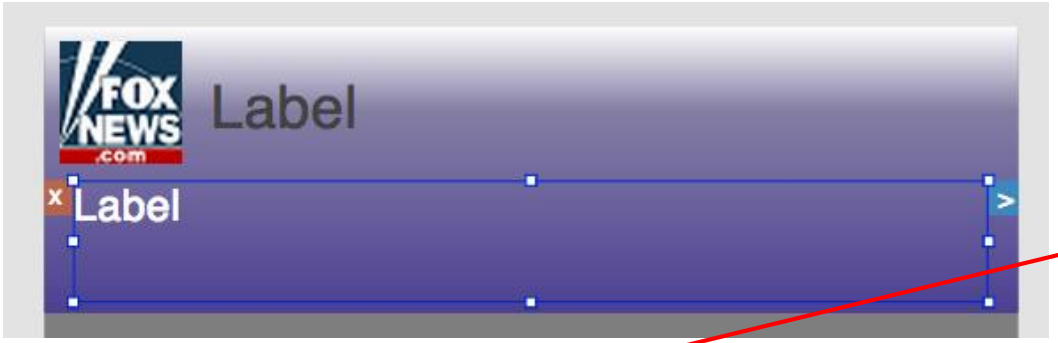
  segData=[<header text>, [row data…]]
- To set data for a section header, you must replace the text with an object specifying the data:

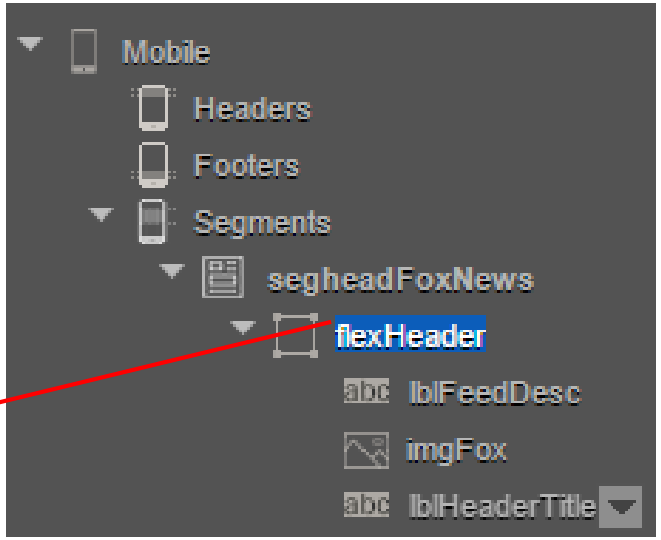
  segData=[{key:value…, template:<template flex ID>}, [row data…]]
  - The key:value pairs in the header allow you to map more than one piece of data if you want

Segment - Segment Header Template (cont'd)

- Note: There is ONE exception to the normal Segment rule: you MUST include the widgets you are assigning data to in the segment's widgetDataMap even if you use the widget name in your mapping.
  - Let's look at an example:

Our segment header template:

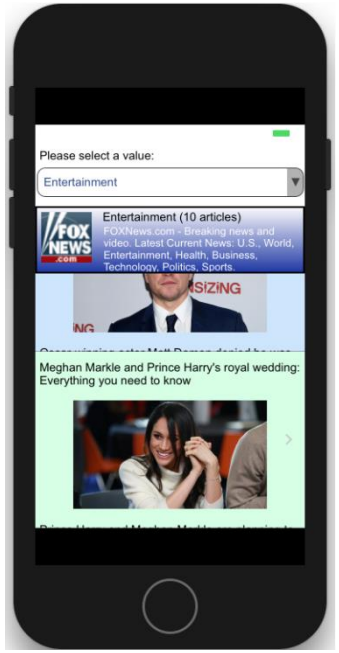




- Here's the code snippet setting the section header data:
  {lblFeedDesc:results.feeddesc, lblHeaderTitle:titleInfo, template:flexHeader}
- Here is what we had to add to the widgetDataMap :
  {lblFeedDesc:"lblFeedDesc", lblHeaderTitle:"lblHeaderTitle",…};
  - Note: we don't have to map the image, since it's static

Segment Templates - Exercise

- Ok, let's try it!
- We'll keep it simple by creating 1 template for the header and 1 template for the actual segment data
- The screens below are just an example. Use your own design!



Segment Templates - Exercise (cont'd)

- If you're not familiar with JavaScript, here is a little code snippet that will loop through a string and replace all the instances of "%20" with a " " (space):
  - In this code snippet we'll assume we're looping through our articles collection and for each article we want to fix the media and thumbnail output parameters

```
for (var i = 0; i < articles.length; i++){
        while(articles[i]["media"].indexOf("%20")>-1)
            articles[i]["media"]=articles[i]["media"].replace("%20", " ");
        while(articles[i]["thumbnail"].indexOf("%20")>-1)
            articles[i]["thumbnail"]=articles[i]["thumbnail"].replace("%20", " ");
}
```

Segment - Memory Considerations

- Segments take a lot of memory - use of Segments should be limited to one or two per page

- For memory optimizations, it is advisable to use a pagination concept to keep the number of rows to a reasonable amount
  - For example:
    - Populate the Segment with the data for the number of rows (fixed per page)
    - Provide the next and previous buttons on the form to navigate through the pagination
    - Populate the next/previous set of data on to the segment depending on the selection

Segment - Other Methods

- So far we've only used the setData method to populate the Segment with data
- That is fine for initially showing the data to the user, but there are times when you need to dynamically manage the data without simply calling setData again to reload all the data
- Here are the Segment methods and situations when you'd use them:
  - addAll(data)– use to append data at the end of the segment's data (even if the segment is empty)
    - Can be used to implement a "more" type function when the user gets to the end of the list and want's more results
    - In a retail app, can be used to show product search results as user checks off more categories to include
    - Example:

```
this.view..segContacts.addAll([{fname:"Alb", lname:"Free"},
                               {fname:"Gail", lname:"Ort"}])
```

      - this would add "Alb Free" and "Gail Ort" to the end of the segment's list of contacts
      - Note the data format: an array of rows (even if only adding 1 row)

Segment - addDataAt & setDataAt

- There are 2 methods available for adding or replacing a single row of data at a certain spot in the data:
  - addDataAt – is used to add 1 row of data at a specific place within the data set
  - setDataAt – is used to replace 1 row of data at a specific place within the data set
    - Can be used to insert a new contact in the list in the correct alphabetical order
    - Can be used to replace an order item in an invoice because the quantity changed
    - Syntax:
      - addDataAt(data, rowIndex, sectionIndex)
      - setDataAt(data, rowIndex, sectionIndex) where:
        - data – the single row of data (ex. {col1:"data", col2:"data"}) to add or set
        - rowIndex – the index (relative to section if sectionIndex is specified) of the row that will be bumped up to insert the row, if addDataAt is used, or the row that will be replaced, if setDataAt is used
        - sectionIndex (optional) – 0 based index specifying what section the data belongs

Segment - addDataAt & setDataAt (cont'd)

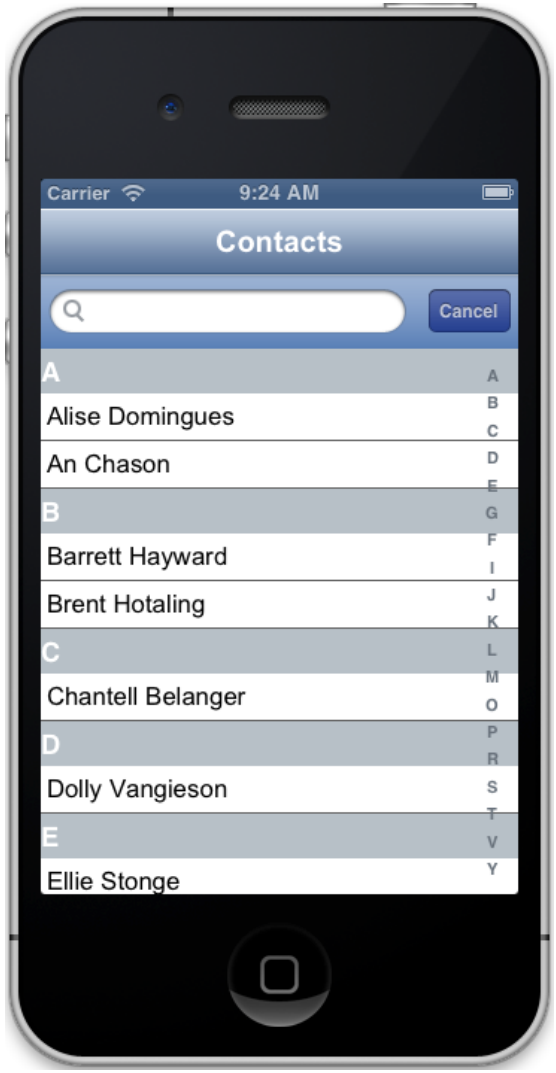
- We will look at a couple of examples to see how these methods work. On the left is a simple contacts list and the sections are shown as grouping the contacts alphabetically:

- Let's say that I misspelled Brent Hotaling's name
  - We'll replace it with Brent Hoteling, his correct name:

    this.view.seg.setDataAt({name:"Brent Hoteling"},1,1)
  - this means replace the data in the second row in the second section with "Brent Hoteling"

- Let's say I have a new contact "Cindy Artos". I'd want that new record right after Chantell Belanger:

    this.view.seg.addDataAt({name:"Cindy Artos"},1,2)
  - Note: if I used rowIndex of 0, it would put it first or if rowIndex is bigger than the collection (as in our example), it puts it last



Segment - addSectionAt & setSectionAt

- There are 2 methods available for adding or replacing a section of data at a certain spot in the data:
  - addSectionAt – is used to add a section of data at a specific place within the data set
  - setSectionAt – is used to replace a section of data at a specific place within the data set
    - In our contacts example, we now need to add a contact for a letter than didn't have a section before – first of that letter
    - Syntax:
      - addSectionAt(data, sectionIndex)
      - setSectionAt(data, sectionIndex) where:
        - data – the complete section(s) of data (ex. [["sec header",[{col1:"data", col2:"data"}]]]) to add or set – can add/set more than one section (note the array of section arrays)
        - sectionIndex (optional) – 0 based index specifying what section the data should replace, if using setSectionAt, or what section should be moved up an index so that the new data can be added, if using addSectionAt.

Segment - addSectionAt & setSectionAt (cont'd)

- Again, let's look at an example (form called "form" and Segment called "seg"):
- I now want to add "Geo Hart" but I don't already have a G section
  - We'll add Geo and the G section:

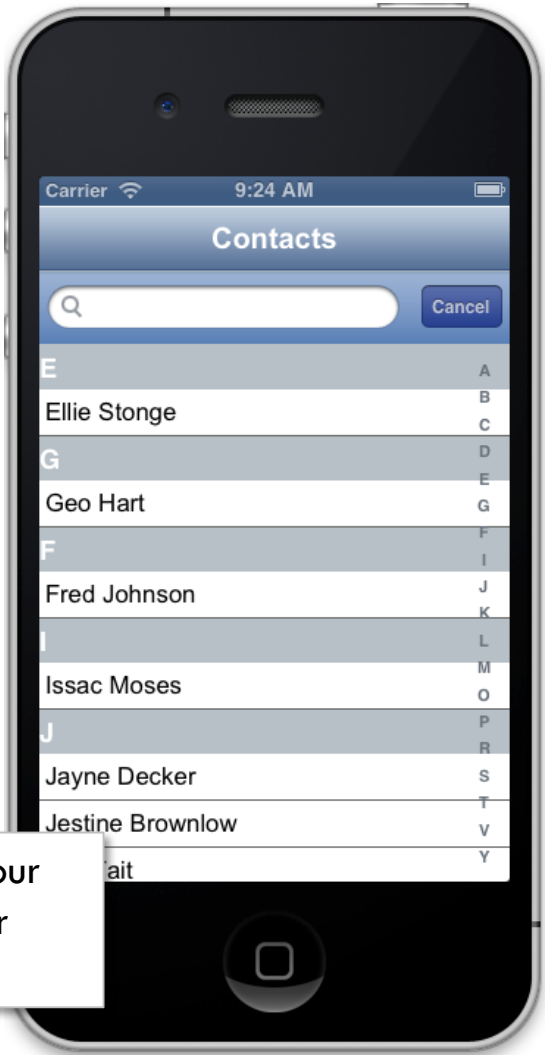
    this.view.seg.addSectionAt([["G",[{name:"Geo Hart"}]]],6)
  - this means we'll add the G section where section index 6 currently is (the I section)
  - It's important to understand the shape of the data we're sending in here: it's an array of sections. A section is an array of rows, with section header.

[["G",[{name:"Geo Hart"}]]]

outer bracket is our collection of sections

next bracket is our section array

inner bracket is our array of rows for that section



Segment - Remove Methods

- There are 3 methods available for removing rows and sections:
  - removeAll() – is used to clear out all the Segment data

  - removeAt(rowIndex, sectionIndex) – is used to delete a row of data
    - rowIndex – the row to delete (0 based)
    - sectionIndex (optional) – if specified, the rowIndex is for this section
    - In our example, to remove "An Chason" (first section, second row) we'd have:

      this.view.seg.removeAt(0,1)
  - removeSectionAt(sectionIndex) – is used to delete a section of data
    - sectionIndex – the index of the section to delete

- Note: When removing or adding Segment rows/sections – be aware that your indexes will all be recalculated
  - For example – to remove rows 5, 6 and 7, call: this.seg.removeAt(4); 3 times

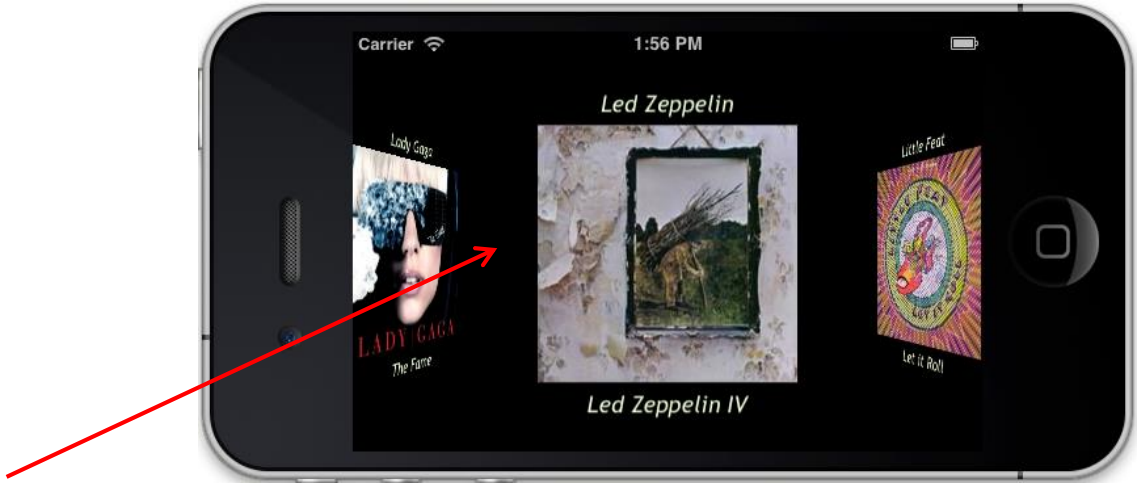
Segment - Data Property

- Use the Segment's data property to see the existing data set for the Segment

- data is structured as an array of data:
  - if you have no sections:
    - data.length returns the number of rows
    - data[i] is the data for row at index i
  - If you have sections:
    - data.length returns the number of sections
    - data[i] is an array where i is a section index
      - data[i][0] is the section information
        - if not using a template, it'll return the heading text
        - if using a template, access template keys for that data
      - data[i][1] is the array of rows
      - data[i][1][n] is the row with index n in section i

Segment - Data Property (cont'd)

- Wow….ok, let's look at each case
- Here is a case where there are no sections:



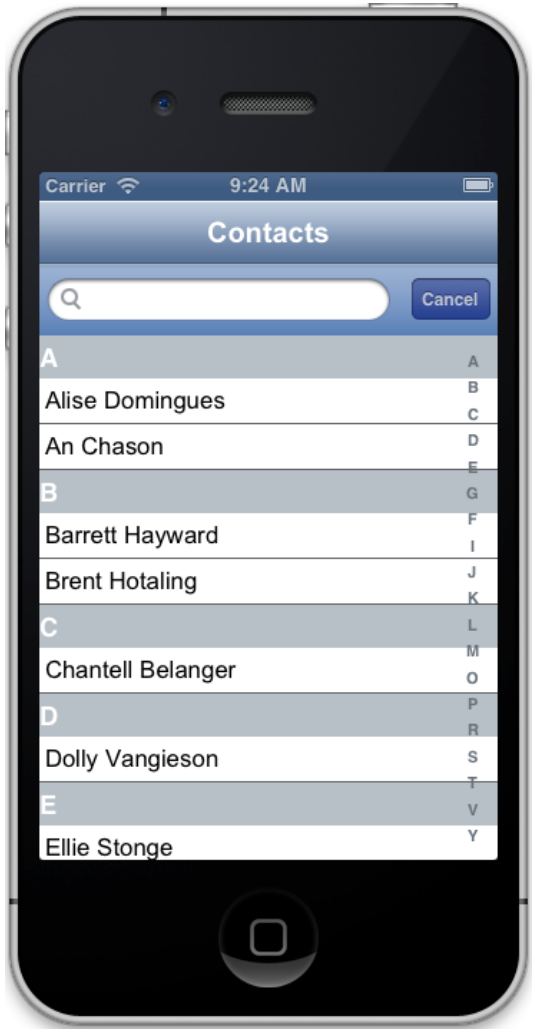
- If this album is at row index 12 out of a total of 25 albums then:
  - this.view.segAlbums.length would return 25
  - this.view.segAlbums.data[12] would return:

    {album:"Led Zeppelin IV",artist:"Led Zeppelin", imgAlbum="ledzepiv.png"}

Segment - Data Property (cont'd)

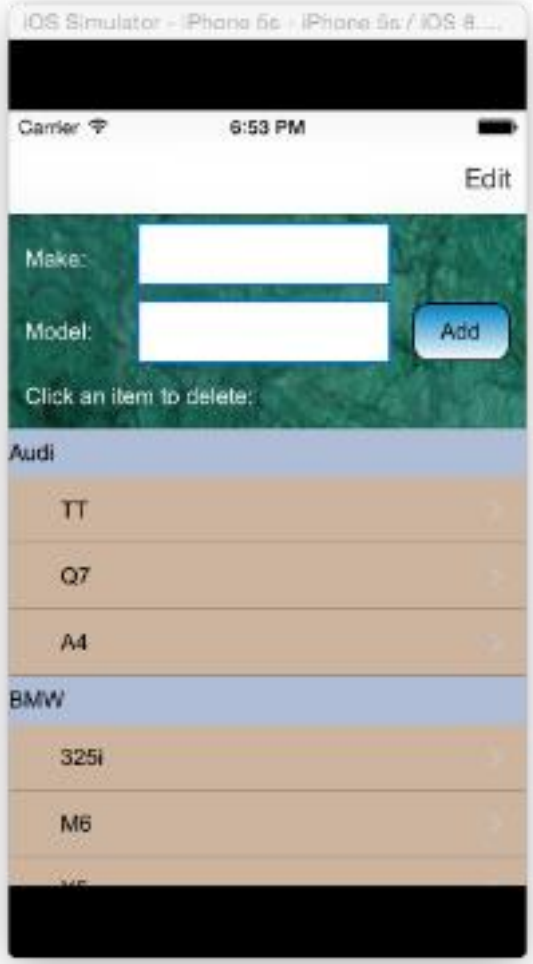
- Here is a case where there are sections. For our example, assume that there are contacts for 17 letters of the alphabet:
- IF the sections do NOT use a template, then:
  - this.view.segContacts.data.length would return 17
  - this.view.segContacts.data[1][0] would return "B"
  - this.view.segContacts.data[1][1][0] would return:

    {name:"Barrett Hayward"}
  - this.view.segContacts.data[1][1].length would return 2 because there are 2 rows in the "B" section
- IF the sections DO use a template (assume the letter is assigned to a widget called lblLetter):
  - this.view.segContacts.data[1][0].lblLetter would return "B"



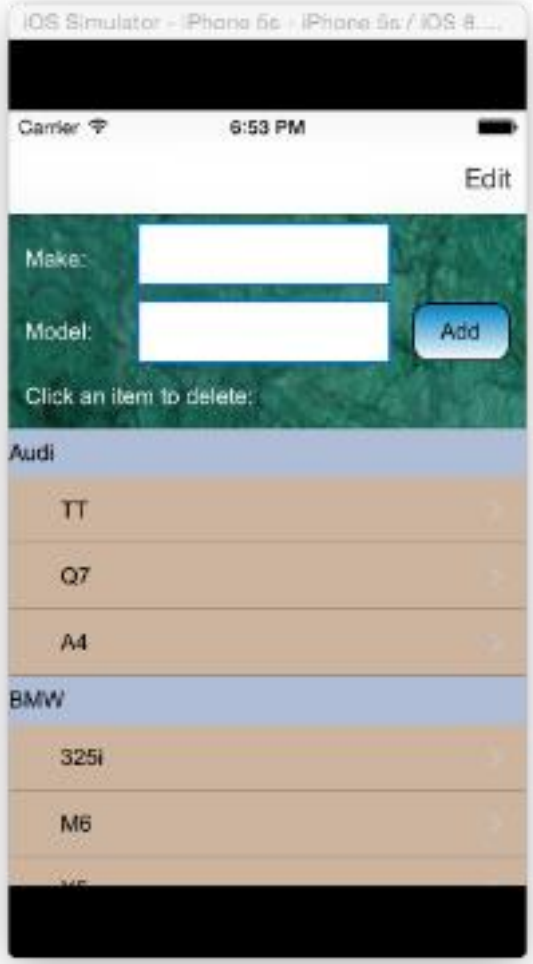
Exercise – Add/Remove Segment Data

- This exercise will bring together all the concepts for the segment data we've just covered
- If you get through this ok, you'll have mastered Segment data
- Create this one page app:
  - Using hardcoded data as a starting point, populate the segment with sections of car makers and rows of car models
  - This example shows simple row and header skinning
  - This app has 2 functions:
    - Adding rows
    - Deleting rows
    - We'll cover each in the next 2 slides…



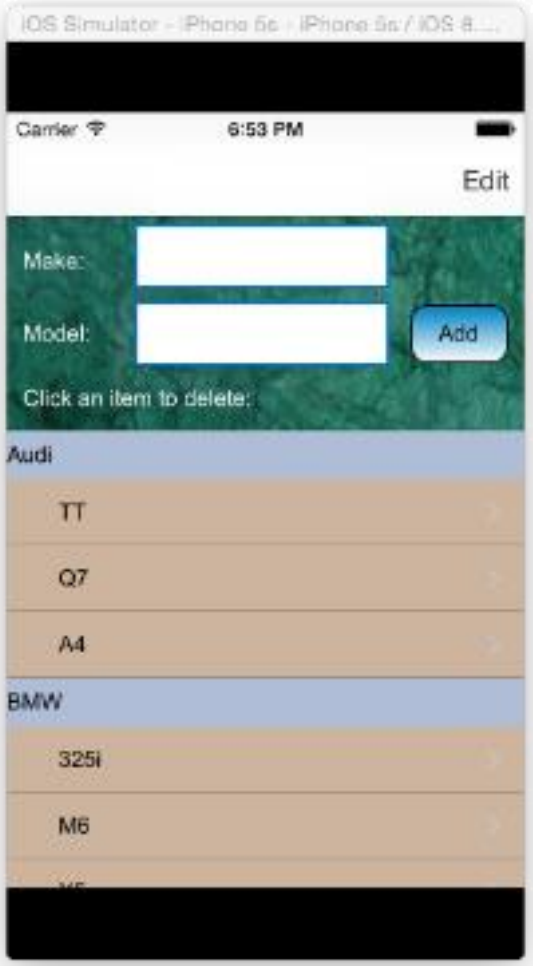
Add/Remove Segment Data - Exercise

- Adding Rows
  - Enter a make and model and click the Add button
  - IF that make already exists as a section:
    - add the model as the first row in the section
  - IF that make doesn't already exist as a section:
    - add the section for the make as the first section
    - add the model as the first row in the section
  - In the example on the right is the result of adding an Audi A4 and Ford Focus
    - note the different row skinning so I can tell the difference



Add/Remove Segment Data - Exercise

- Deleting Rows
  - Click on a row to delete it
  - IF that was the last row in the section – delete the section too
  - In the example on the right is the result of clicking on the Audi TT (and thus removing it) :
- The next slide shows the outline of the code you'll write for this exercise.
- Let's take a look...



Add/Remove Segment Data - Exercise

- You'll need 3 functions: initial population of data, when user clicks to remove a row and an add row function. Here's the outline/sample code you can use (use your OWN form and widget names):

```
setCars: function() {
        var carsData = [
            ["Audi", [{car: "TT",}, {car: "Q7"}, {car: "A4"}]],
            ["BMW", [{car: "325i"}, {car: "M6"}, {car: "X5"}]],
            ["Honda", [{car: "Accord"}, {car: "Civic"}]],
            ["Tesla", [{car: "Model S"}]]
        ];
        this.view.segCars.widgetDataMap = {
            lblCar: "car"
        };
        this.view.segCars.setData(carsData);
    },
    addCar: function() {
        //loop through segment section titles and see if that model exisits

        //if it does exist
        //use the addDataAt API to add the car to that model's section

        //if it doesn't exisit
        //use the addSectionAt API to create the new section with the new car entry

    },

    removeCar: function() {
        //for the selected item, you'll nee both the section index as well as the row index within that section
        //use the removeAt API to remove the row from the section

        //check the NEW length of that section
        //if it's 0
        //remove the section since there are no more cars in it

    }
```

Segment - Animations

- Animations for Segment are supported only in table view
- Animations enables the developers to add animations to row elements
- The application developer can associate the animations to the rows in the following ways
  - At a segment level for all the rows when the rows are coming into visible region
  - At a row level when an operation is performed such as adding or deleting a record etc
  - Animating the rows based on the gestures

Segment Level Animations

- Let's say whenever the user is scrolling the records, we want the records which are coming in to viewport to be animated
- In such cases, we can use the segment level animation
- API for setting animations at the segment level

  setAnimations(visible:animationObject)

  - animationObject:It's the animation object which contains the animation definition
- API for setting animations at the segment level
- Available on all platforms except on Windows

Segment Level Animations (cont'd)

- In the below example whenever the user scrolls the segment, then rows which come into viewport gets translated from right to left
- In the below screenshots, observe that all the cars are translating from right to left when the user is scrolling the segment

Segment Level Animations (cont'd)

- Sample code

```
var transformObject1 = kony.ui.makeAffineTransform();
transformObject1.translate(200, 0);
var transformObject2 = kony.ui.makeAffineTransform();
transformObject2.translate(0, 0);
var animationObject = kony.ui.createAnimation(
  {"0":{"transform":transformObject1,"stepConfig":{"timingFunction":kony.anim.LINEAR}},
  "100":{"transform":transformObject2,"stepConfig":{"timingFunction":kony.anim.LINEAR}}});
 var animationConfig = {duration:1,fillMode: kony.anim.FILL_MODE_FORWARDS};
var animationCallbacks = {"animationEnd":function(){kony.print("animation END")}};
var animationDefObject={definition:animationObject,config:animationConfig,callbacks:an                imationCallbacks};
this.view.segCars.setAnimations({visible:animationDefObject});
```

- In the above code, at 0th step we are configuring the transformobject1 whose x position is 200dp and at 100th step we are configuring transformobject2 whose x position is 0dp
- Hence, when we scroll then the records translates from 200dp to 0dp

Segment level Animations - Exercise

- Build a screen similar to that shown in the screenshot
- Here's how it works
  - Clicking the Translate button and scrolling the segment should translate the rows from right to left
  - Clicking the Scale button and scrolling the segment should scale the rows from 0 to 1
  - Clicking the Rotate button and scrolling the segment should rotate the rows from 90 degrees to 0 degrees
- Create the segment with template as we will extend this exercise for upcoming topics

Animating Rows Based on Gestures

- If we want to animate based on user gestures such as onRowClick, swipe etc., then we can use animateRows API

  animateRows({rows:rowList,widgets:widgetsList,animation:animationObject})

  - **animationObject:** It's the animation object which contains the animation definition
  - **rowList:** List of visible rows on which you want to apply the animations
  - **widgetsList:** List of the widgets on which the animation will be applied, if widget list is empty the animation would be applied on the complete row
  - **animationObject:** It's the animation object which contains the animation definition, configuration and callbacks

Animating Rows Based on Gestures (cont'd)

- Sample code

```
this.view.segCars.animateRows({rows:rowList,widgets:["lblCarBrand"],animation: animationDefObject })
```

rowList is an array of JSON objects and each JSON object contains the row details. For e.g.,:

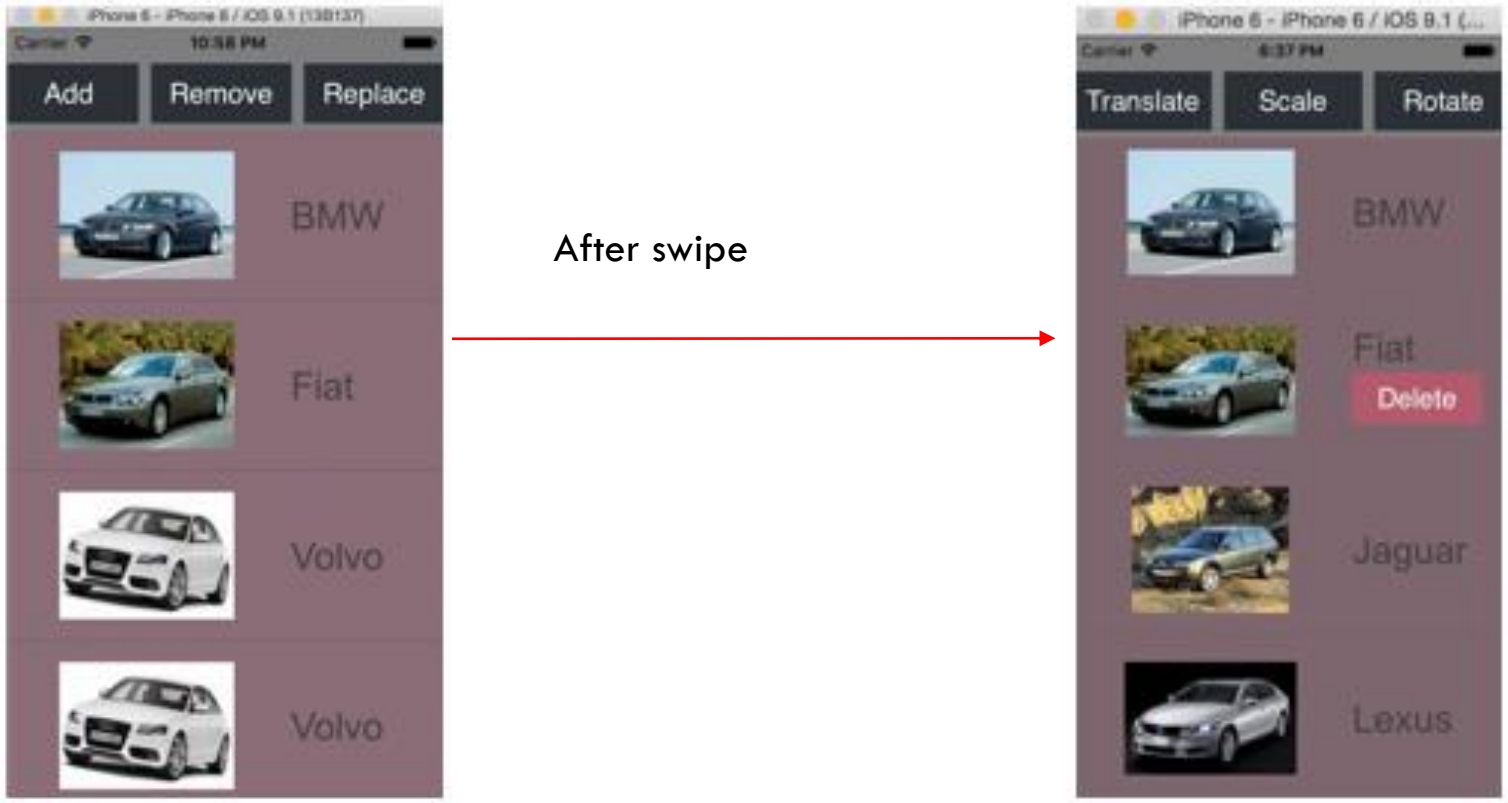
```
var row1 = {sectionIndex:0,rowIndex:1};
var row2 = {sectionIndex:0,rowIndex:2};
rowList=[row1,row2]
```

lblCarBrand is the widgetId

- We can apply this animation only to the visible rows
- Note: In the animation object we can apply only transform properties at row level i.e., we cannot use positional properties (such as left and right etc.) and dimensional properties (such as width and height etc.), but for widgets we can apply any animation

Animation on Gesture - Exercise

- Let's extend the previous exercise
- Whenever the user swipes on any row, then car brand text should move up with an animation and after that a delete button should come from right to left with an animation



Row-Level Animations (During Operations)

- An animation can be associated to a row operation such as adding or removing rows in the segment
- We can animate at row level by passing the animationObject to the below row operation API's

  addDataAt(), setDataAt() and removeAt()

  addSectionAt(), setSectionAt() and removeSectionAt()

  addAll(),setData() and removeAll()
- Sample Code for add, set and remove

```
this.view.segCars.addDataAt(data,rowIndex,sectionIndex,animationObject);
this.view.segCars.setDataAt(data,rowIndex,sectionIndex,animationObject);
this.view.segCars.removeAt(rowIndex,sectionIndex,animationObject);
```

Row-Level Animations - Exercise

- Let's extend the previous exercise
- Whenever the user swipes on any row, then car brand text is moving up with an animation and after that a delete button is displayed
- When the delete button is clicked, then the record should be deleted with scale down animation

Row-Level Animations - Exercise

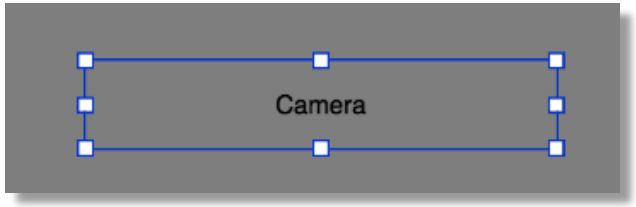
- Let's extend the previous exercise
- Whenever the user swipes on any row, then car brand text is moving up with an animation and after that a delete button is displayed
- When the delete button is clicked, then the record should be deleted with scale down animation

Kony Widget API

Camera Widget - Overview

- The Camera widget is used to invoke the native camera of the device
- The Camera widget is basically a button that is hardwired to invoke the camera
  - The basic Properties like text, text i18n key, skin, alignment, expand etc., that are supported for the Button are also supported for the Camera widget
- The Camera widget has an onCapture event
  - When the camera is invoked, the user can take a picture
  - The user is shown buttons to cancel, use or re-take the image
  - Control is passed to the onCapture event when the user either cancels or accepts the picture they just took

Note: The widget is the only way to invoke the camera



Camera Widget - Image data

- Once the image is captured, you can edit some values in the Service Configurations
- The device's camera will store the image in memory
- The onCapture event handler accepts the camera widget as a parameter
  - The camera widget will have the image in both the rawBytes and base64 properties - you can use whichever you need for your application
  - It's VERY common to show the user the final image on some form. Here's how we'd show the camera image in an image widget in the onCapture callback:

```
function showPicture(){
    this.view.imgPic.rawBytes = this.view.myCamera.rawBytes;
}
```

  - Note: The platform releases the reference to the image handle after the first use
    - If you run that line of code twice, the second time this.view.myCamera.rawBytes would be null
- Images can be VERY large! It is not recommended to save the imageData in global variable
- Kony provides 2 methods for converting between raw bytes and base64:
  - kony.convertToBase64(rawbytes) - converts rawbytes to base64 encoded data
  - kony.convertToRawBytes(base64) - converts base64 data to raw bytes data

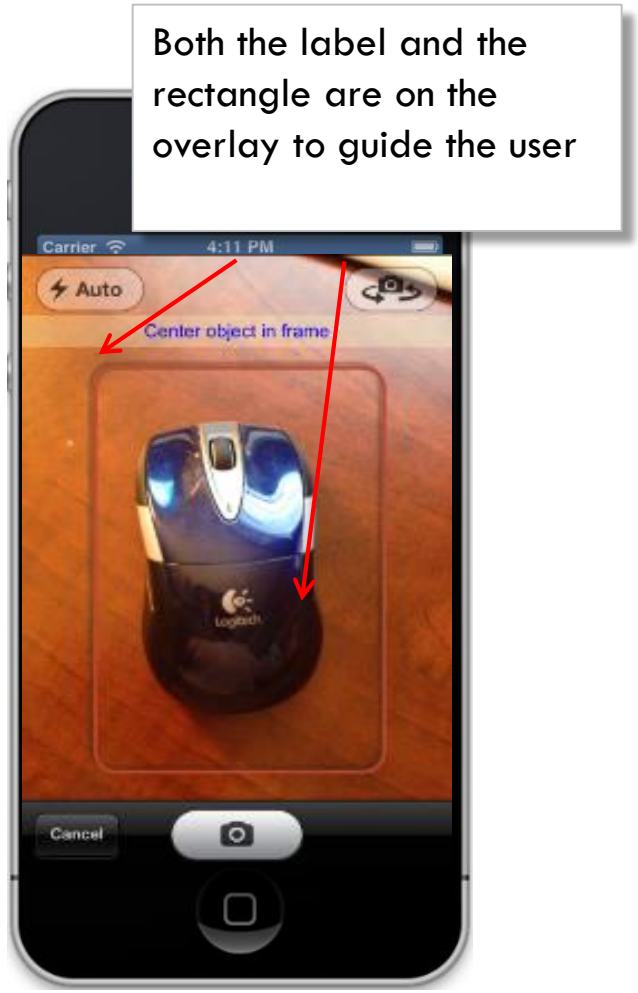
Camera - Properties

- **cameraSource:** Specifies the source of the camera either default, front or rear
  - Below are the available options:
    constants.CAMERA_SOURCE_DEFAULT
    constants.CAMERA_SOURCE_REAR
    constants.CAMERA_SOURCE_FRONT
  - For Android platform when the cameraSource is set to default, the platform checks for the rear camera and if it does not exist then it checks for front camera
  - In Android, this property is not supported when enableOverlay is set to false

- **captureMode:** Specifies the capture mode of the camera either photo or video mode
  - Below are the available options:
    constants.CAMERA_CAPTURE_MODE_PHOTO
    constants.CAMERA_CAPTURE_MODE_VIDEO

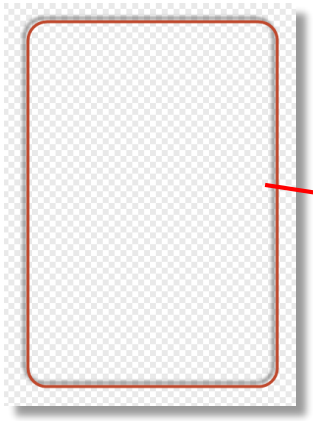
Camera Widget - Overlay

- An overlay is an image that you superimpose on top of the camera viewfinder to aid the user in taking the picture
  - In this example, the overlay is used to help the user to center the focus for capturing the image
  - Note: The final image will not have any of the overlay components on it
- The overlay is defined using another form that you've configured with all the visual components to be shown on the camera screen

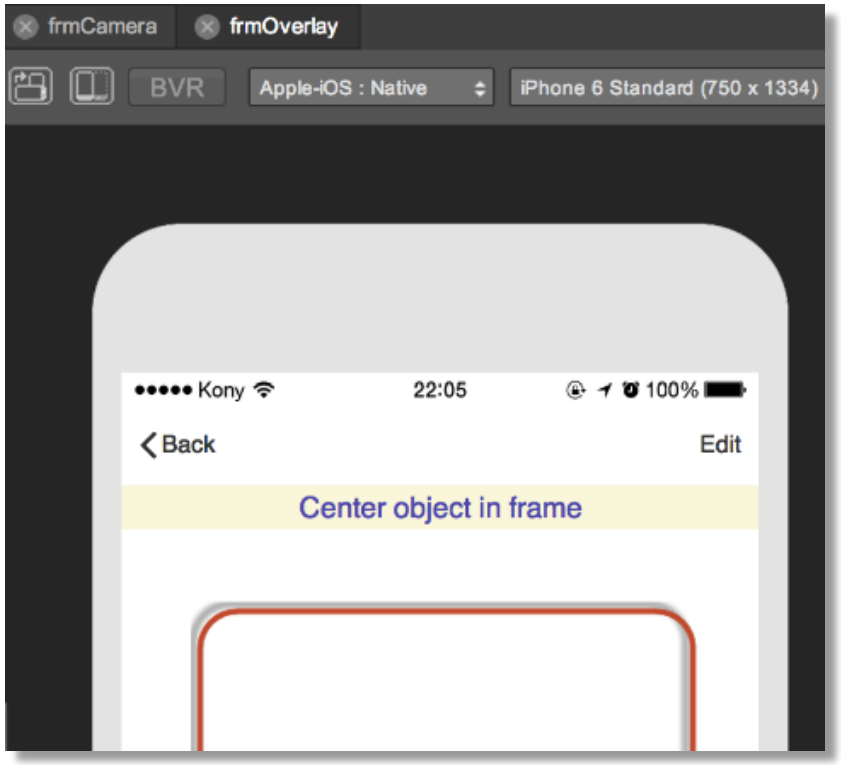


Camera Widget - Overlay (cont'd)

- The overlay is defined using another form that you've configured with all the visual components to be shown on the camera screen
- Here's the overlay screen used in our example:
  - A label with a 70% transparent skin
    - In our example, the image is:



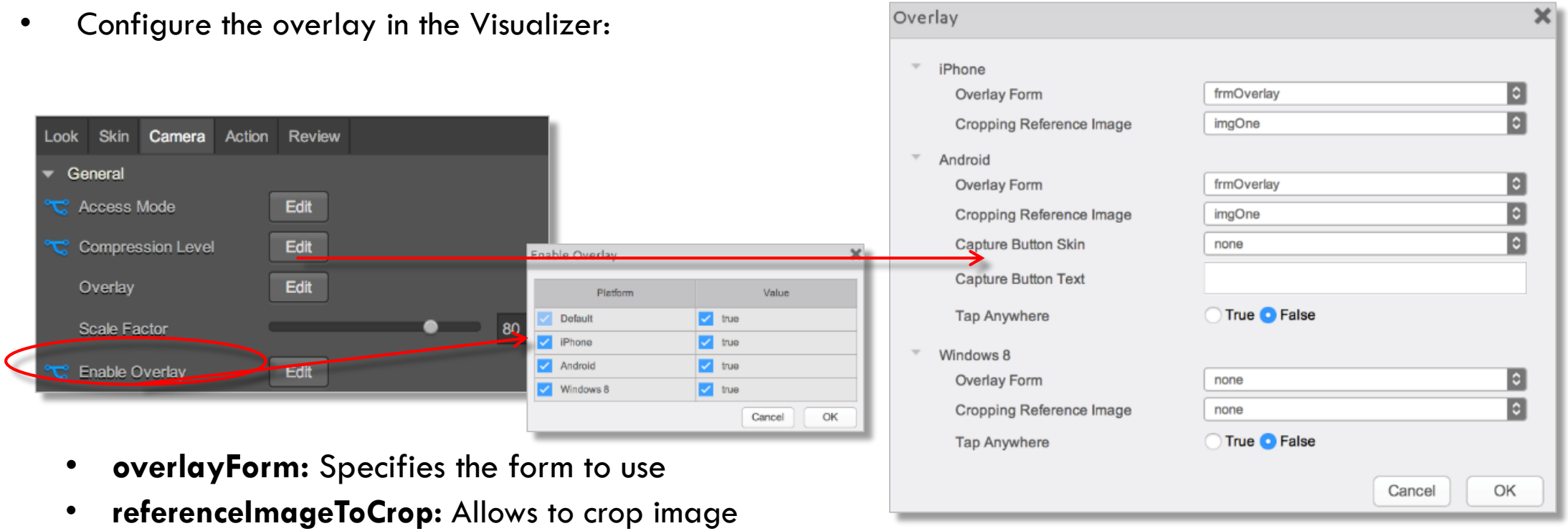
The image is transparent inside and out - anything solid will block the camera image behind it



  - A form skin that has a transparent background
  - Layout tweaking to get it positioned just right

Camera Widget - Overlay (cont'd)

- Configure the overlay in the Visualizer:



- **overlayForm:** Specifies the form to use
- **referenceImageToCrop:** Allows to crop image dimensions
- **tapAnyWhere:** Allows to tap the screen to take the picture instead of hitting the capture button

Camera - Properties (cont'd)

- **cameraOptions:** Specifies the camera options that can be used on an overlay form
- This property should be set through the code and it expects JSON object

- Use the following keys and values to configure the properties in the object:

flashMode

- constants.FLASH_MODE_AUTO: Specifies the flash must be turned on when required
- constants.FLASH_MODE_ON: Specifies the flash must be turned on when you take a picture
- constants.FLASH_MODE_OFF: Specifies the flash must not be turned on even when you take a picture
- constants.FLASH_MODE_ALWAYS_ON: Specifies the flash must not be turned on constantly when the camera is open

hideControlBar

- True: Shows the control bar of the respective platforms
- False: Hides the control bar
- Note: This is not supported in iPad

Camera - Properties (cont'd)

- Use the following keys and values to configure the properties in the object (cont'd):

  captureButtonSkin
  - Specifies the skin for a captured button
  - This option is applicable on Android platform only and is considered only when hideControlBar is set to true and captureMode is photo

  captureButtonText
  - Specifies the text for a captured button
  - This option is applicable on Android platform only and is considered only when hideControlBar is set to true and captureMode is photo

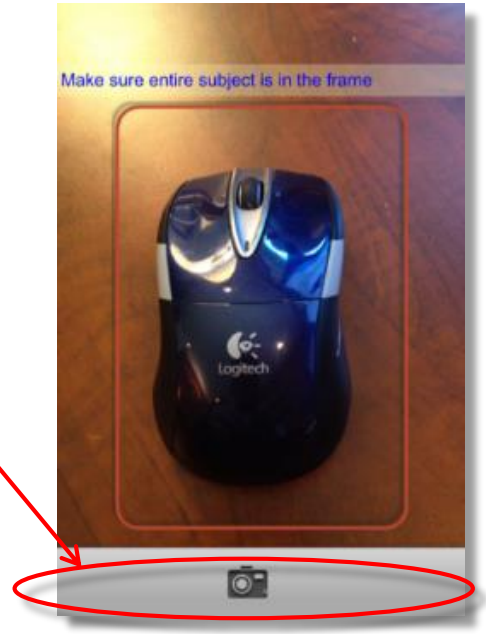
  captureMode
  - Photo
  - Video

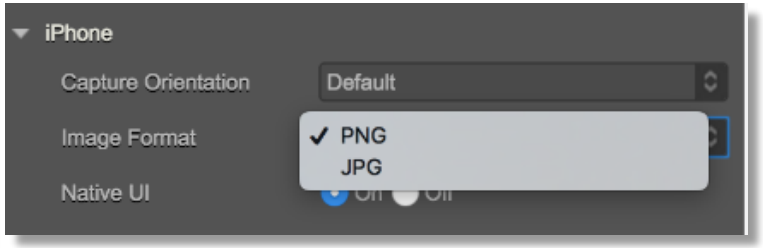
Camera - Properties (cont'd)

- The iPhone has a few unique properties:
  - **Image Format -** Specify if you want a JPEG or PNG format for the image
  - **Capture Orientation** - Specify default, portrait or landscape for the final image
  - **Native UI** - When set true utilizes all the normal camera options vs. when set false it only shows a capture button

The native interface includes some camera controls including the cancel button

Turning off the native interface leaves you only with a capture button





Camera - Properties (cont'd)

- The following are the few more properties that affect the final image:
  - **accessMode** - specifies how the captured image must be stored
    - **CAMERA_IMAGE_ACCESS_MODE_PUBLIC** - Stores the image for everyone typically in the phone's photo gallery
    - **CAMERA_IMAGE_ACCESS_MODE_PRIVATE** - Stores the image on the device but it's only accessible via the app
    - **CAMERA_IMAGE_ACCESS_MODE_INMEMORY** - Never writes the image to the device - best if photo is confidential (like a check)
  - **compressionLevel** - Value between 0 and 100 indicating compression for the image where 0 is no compression (best quality)
  - **scaleFactor** - Value between 10 and 100 indicating the % reduction in size - a value of 10 returns an image that is 10% of the original size

Camera - Properties (cont'd)

- videoQualityLevel: Specify the quality of the video that has to be captured and we have to set it through the code
  - Following are the options for iOS
    CAMERA_VIDEO_QUALITY_MEDIUM (Default), CAMERA_VIDEO_QUALITY_HIGH
    CAMERA_VIDEO_QUALITY_LOW, CAMERA_VIDEO_QUALITY_640x480
    CAMERA_VIDEO_QUALITY_1280x720, CAMERA_VIDEO_QUALITY_960x540
  - In case of Android, we have to get the supported quality levels on the device using supportedVideoQualityLevels API and then set the videoQualityLevel
  - supportedVideoQualityLevels API returns an array of possible quality levels and it is available only for Android
  - Note: When the enableOverlay property is set to false, only one option is supported that is constants.CAMERA_VIDEO_QUALITY_HIGH
- videoDuration: Specify the video duration of the captured video in seconds

Camera - Properties (cont'd)

- iPhone has the following unique property in video mode:
- videoFormat: Specify the video format of the captured video
  - Formats are:

    CAMERA_VIDEO_FORMAT_MP4

    CAMERA_VIDEO_FORMAT_MOV

Camera - Properties (cont'd)

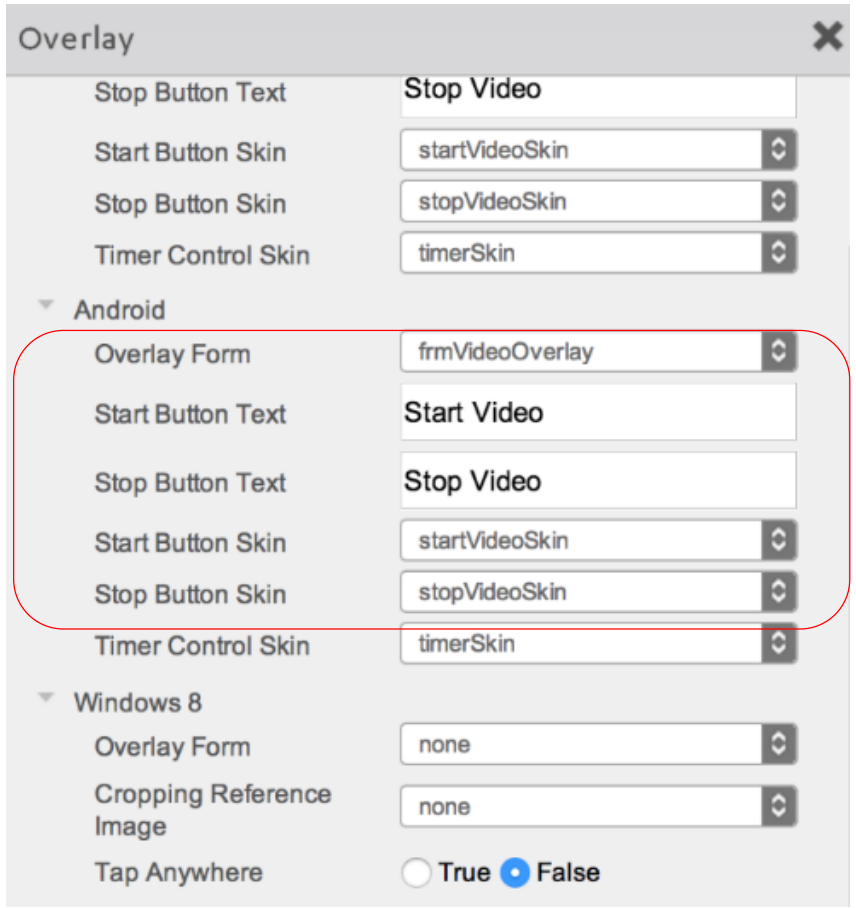
- Android has the following unique properties:
- **focusMode:** Specifies the focus mode for the camera and is considered when the **enableOverlay** is set to true
- Its available only for Android and we have to set it through code
  - Below are the available options

    constants.CAMERA_FOCUS_MODE_AUTO

    constants.CAMERA_FOCUS_MODE_CONTINUOUS
- **videoStabilization:** This property enables you to reduce the shaking of the camera while shooting a video
  - It expects a Boolean value and setting to true increases the stabilization
  - Its available only for Android

Camera - Properties (cont'd)

- If camera captured mode is video and overlay is enabled, then you can configure the following in the overlay configuration
  - Start Button Text
  - Stop Button Text
  - Start Button skin
  - Stop Button skin
  - Timer Control skin
- Once we launch the camera, it shows start button and timer control
- When the user clicks on start button then it starts recording the video and displays stop button and also the time in the timer control
- Stop button to stop the video



Camera - Methods

- **stopVideoCapture:**
  - This method allows you to stop a video that is being captured using startVideoCapture method
  - An onFailure event, callback is invoked when an error occurs using a camera widget. For example, you set a camera source, but it is not available on the device

- **Method signature**
  - onFailure (source,errorcode)
  - source: Its the handle to the widget reference
  - errorcode: Specifies the error code

Camera - Methods (cont'd)

- **getSupportedCameraSources:**
  - This method returns an array that contains either constants.CAMERA_SOURCE_REAR or constants.CAMERA_SOURCE_FRONT
  - If the device has both the cameras, the returned array will contain both these constants
- **startVideoCapture:**
  - This method allows you to capture a video programmatically that will end as configured in the videoDuration property
  - If the videoDuration is not configured, video capture must be stopped using stopVideoCapture method

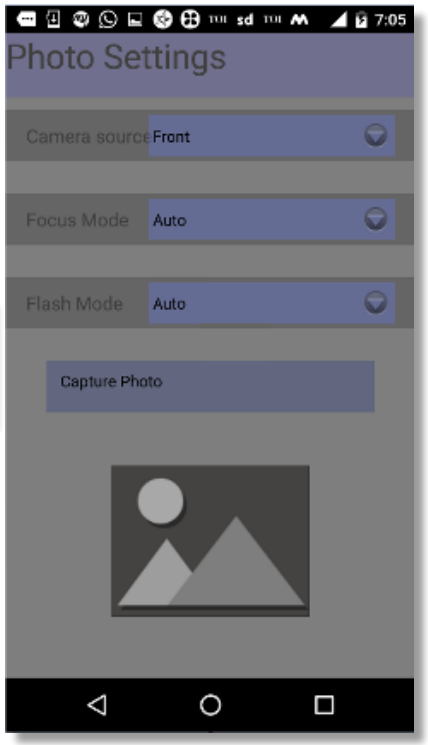
Camera Widget - Memory Management

- The rawbytes of the captured image are available to an application until the application closes or until the rawbytes are manually deleted
- It is always advisable to release the captured data after its usage
- The captured raw data can be released from memory using the releaseRawBytes method
- Syntax: **this.view.myCamera.releaseRawBytes(rawBytes)** / **this.view myCamera.releaseRawBytes(rawBytes)**
- Where:
  - **rawBytes:** Specifies the rawbytes of the captured image from the camera which are to be released.
    - **Note:** this.view.myCamera.releaseRawBytes(rawBytes) is implemented for iOS and Android
    - **Note:** Always do a "null" check on the raw bytes being passed to this method as releasing a raw bytes which is already released results in exception
  - If multiple handles (variables pointing) to the same rawbytes exist, and if you release the rawbytes using any one of those handles, the other handles become obsolete
  - If you store the rawbytes of the captured image in some location on the device by using the kony.store.setItem API, and you call this method, the rawbytes are deleted from the disk or in-memory, but the image stored in the specific location remains intact (you must delete the stored image manually)

Camera Widget Photo - Exercise

- In this exercise, we will launch the camera in the photo mode with an overlay
- We will also provide the options to the user to select the camera source, focus mode and flash mode so that camera should be launched with the user selected option
- Captured image should be displayed on the form

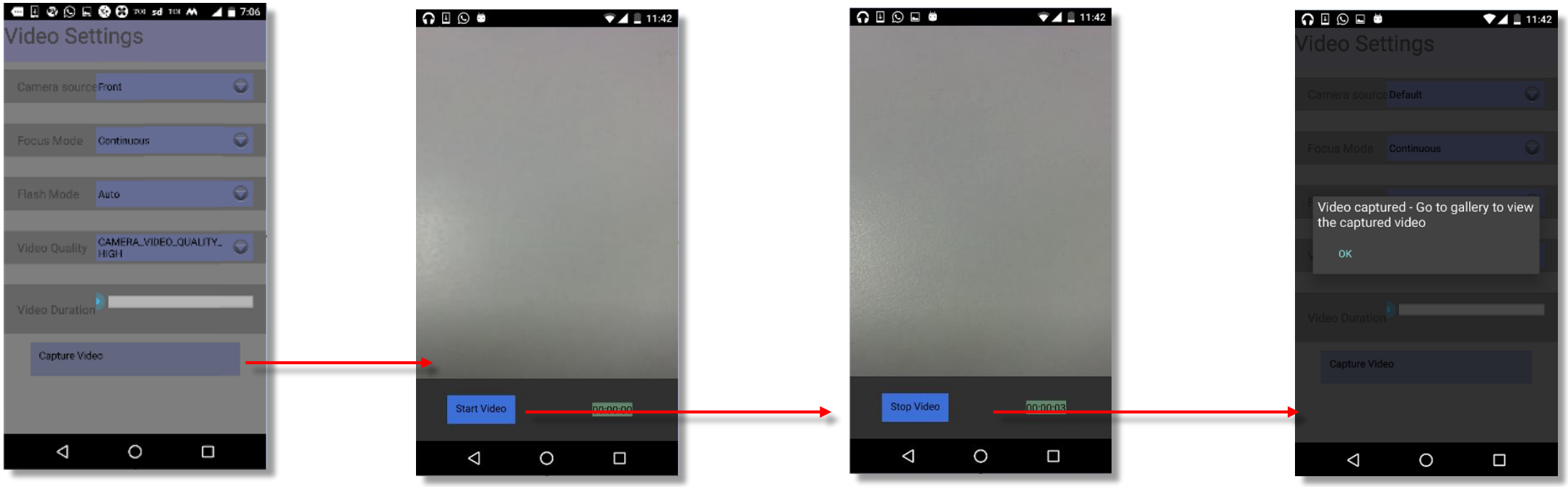
Camera with overlay



When we click on Capture button then the captured image in the overlay should be displayed on the form

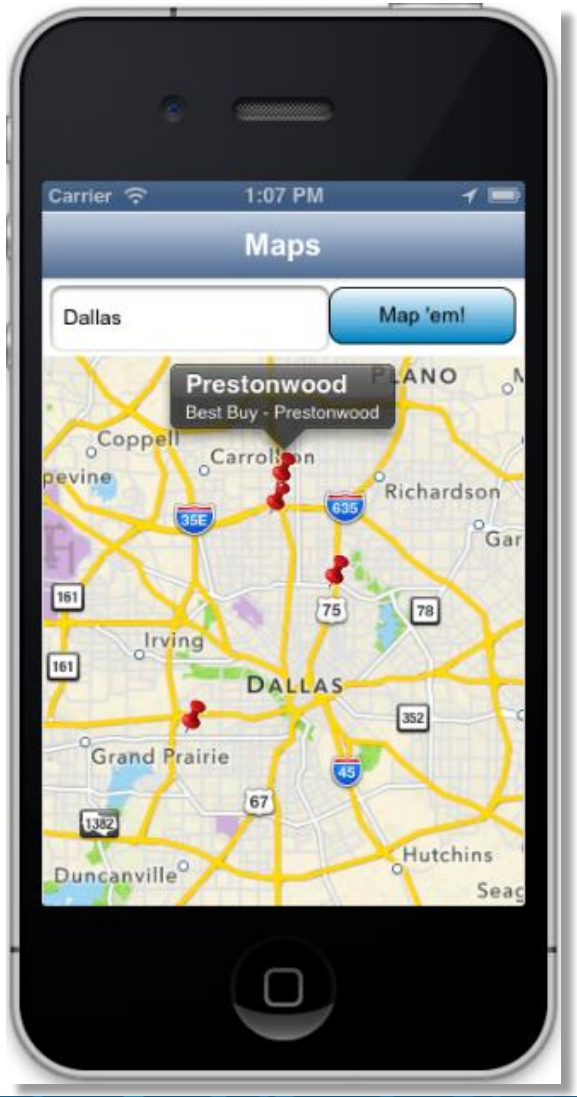
Camera Widget Video - Exercise

- We will provide the options to the user to select the camera source, focus mode, video quality (In android, video quality options should be populated using supportedVideoQualityLevels API) and video duration
- We will also configure start & stop buttons and timer control in overlay configuration
- Once the video is captured, an alert should be displayed on the form to view gallery



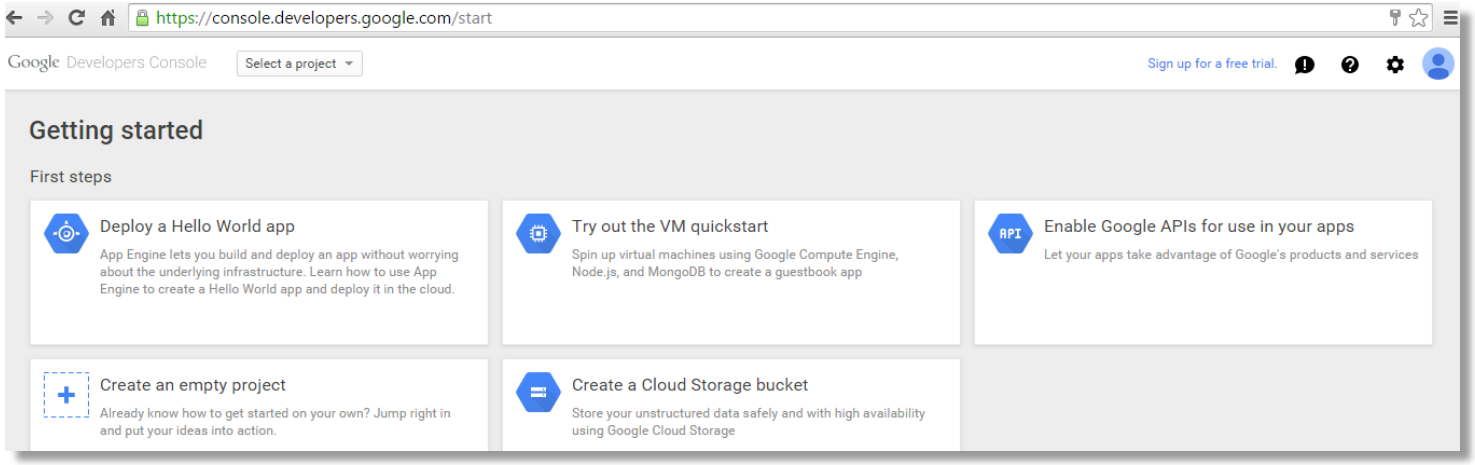
Mobile Maps - Overview

- The Map widget is used to display a map on the device
- Maps are a very common feature in most applications
- The Map widget renders a map using the service(s) provided by the platform
- The generated maps use the native interfaces for panning and zooming in/out
  - iPhone is all touch and gestures
  - Android has built in + - buttons to zoom in/out
- Maps use pins to show the points
  - When the user taps a pin, a callout with info is displayed
- In general, maps take up a lot of memory - best to use one map in the app or minimize maps if possible



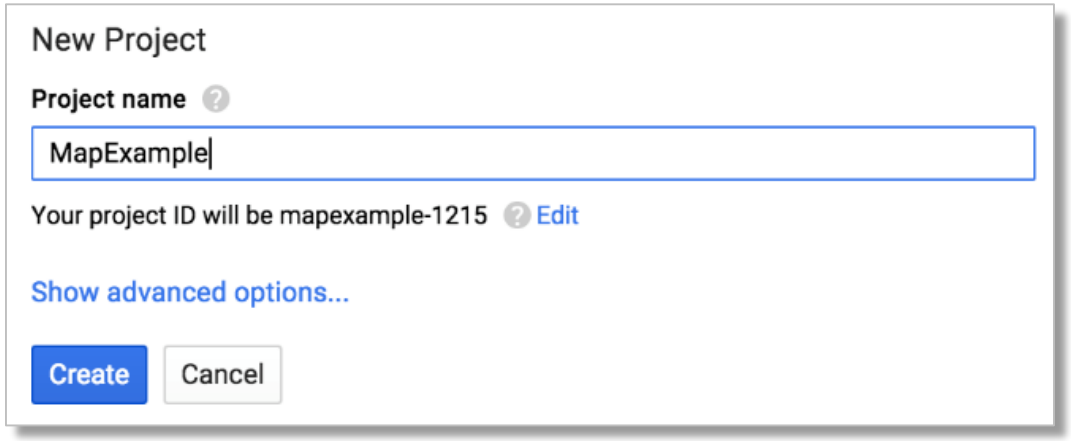
Mobile Maps - Keys

- You have a map key embedded in your app to access mapping services such as Google and Bing maps
  - Functional Preview has built-in own so you don't need when testing using that app
- Getting your Android requires a Google account
- Start by going to: https://console.developers.google.com/
- Here you need to make sure you can access this area (logging in with your Google account):

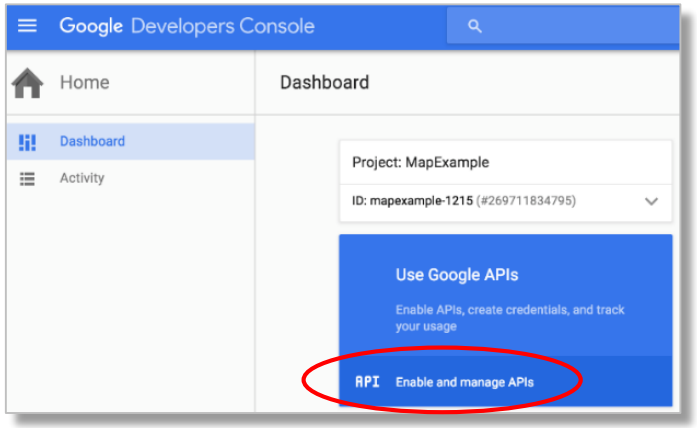


Mobile Maps - Google key

- Your map keys are associated with a project so we'll have to create a Project:

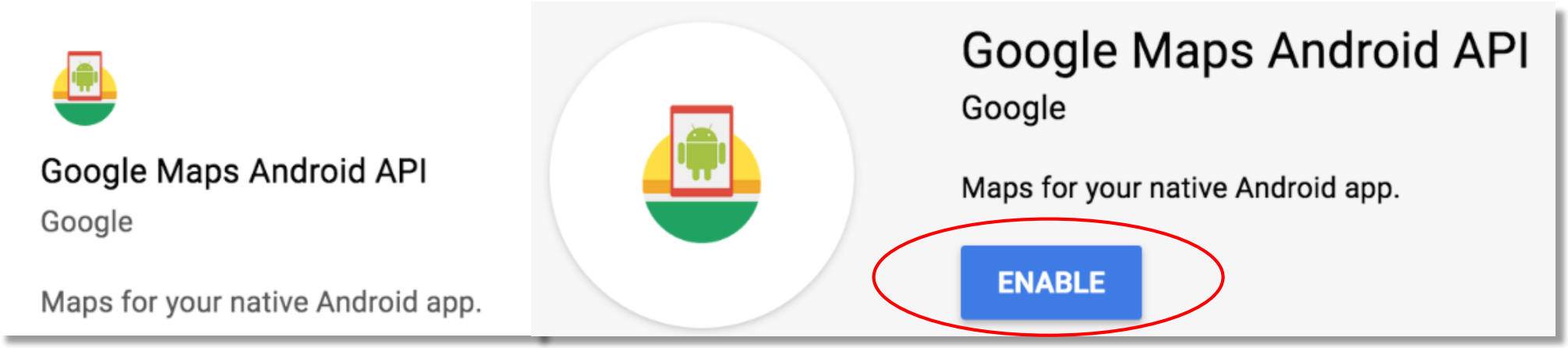


- Once that is done, you then need to go into the API's...



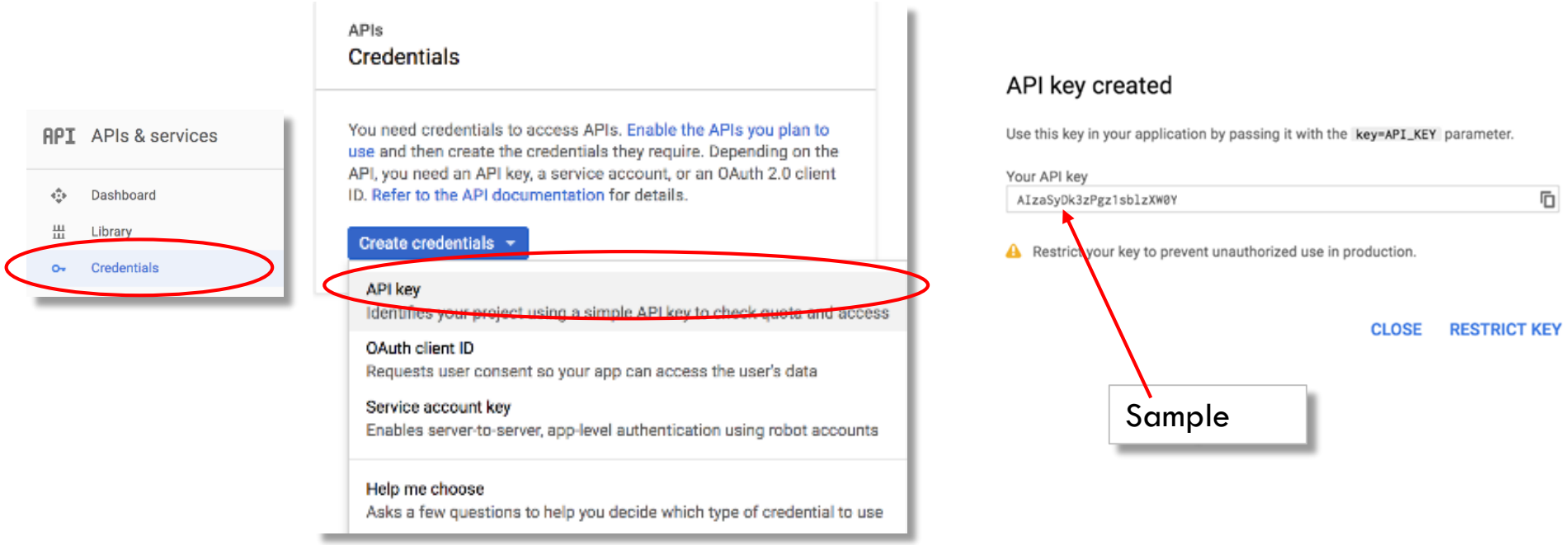
Mobile Maps - Google key

- Click on the Library option and search for Google maps
- From the API list, find the Google Maps Android API and turn it on by clicking on the Enable API button



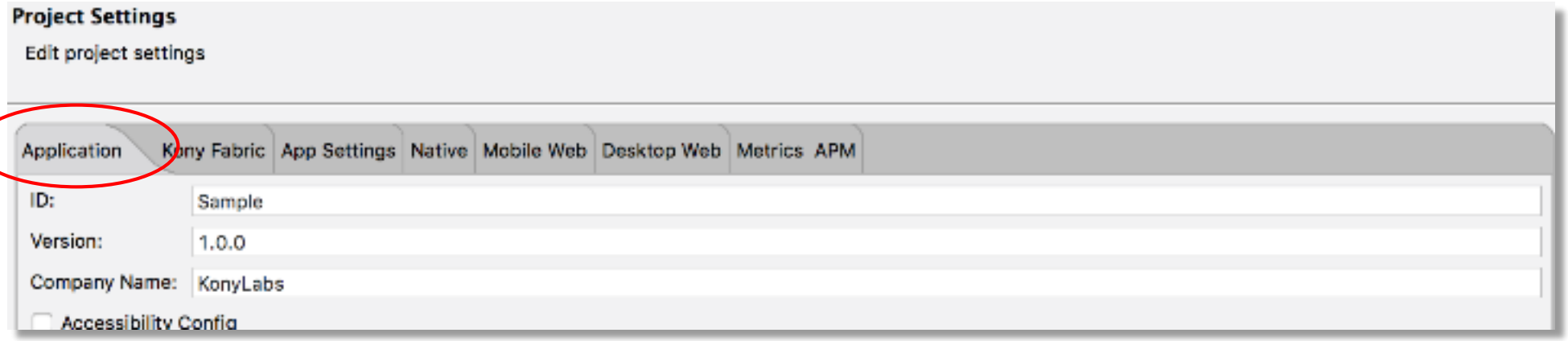
Mobile Maps - Google Key (cont'd)

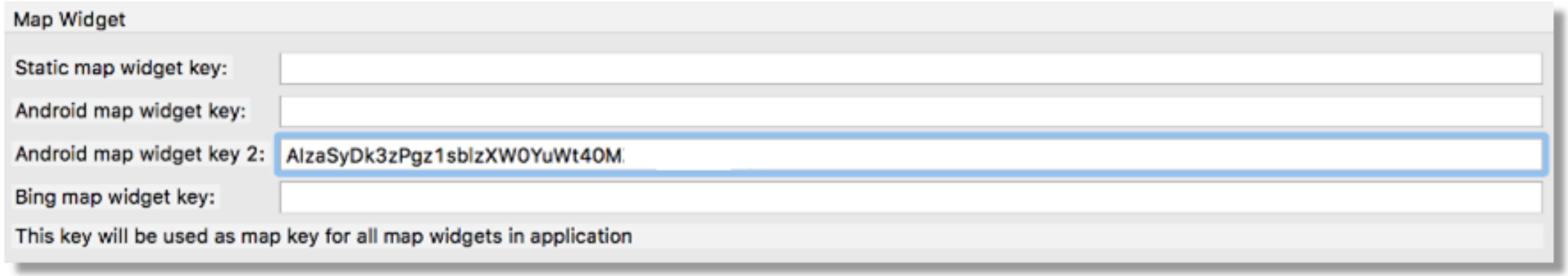
- Now, we can go to Credentials to get our key:



Mobile Maps - Google key (cont'd)

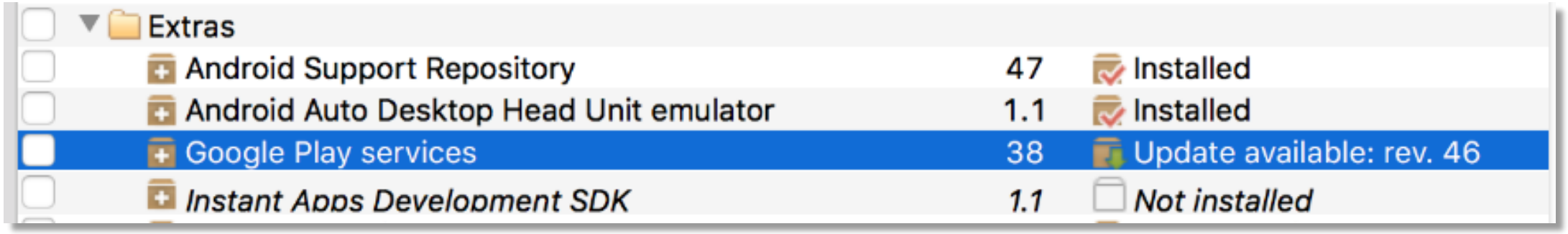
- The last step is actually USING your keys in your application properties:





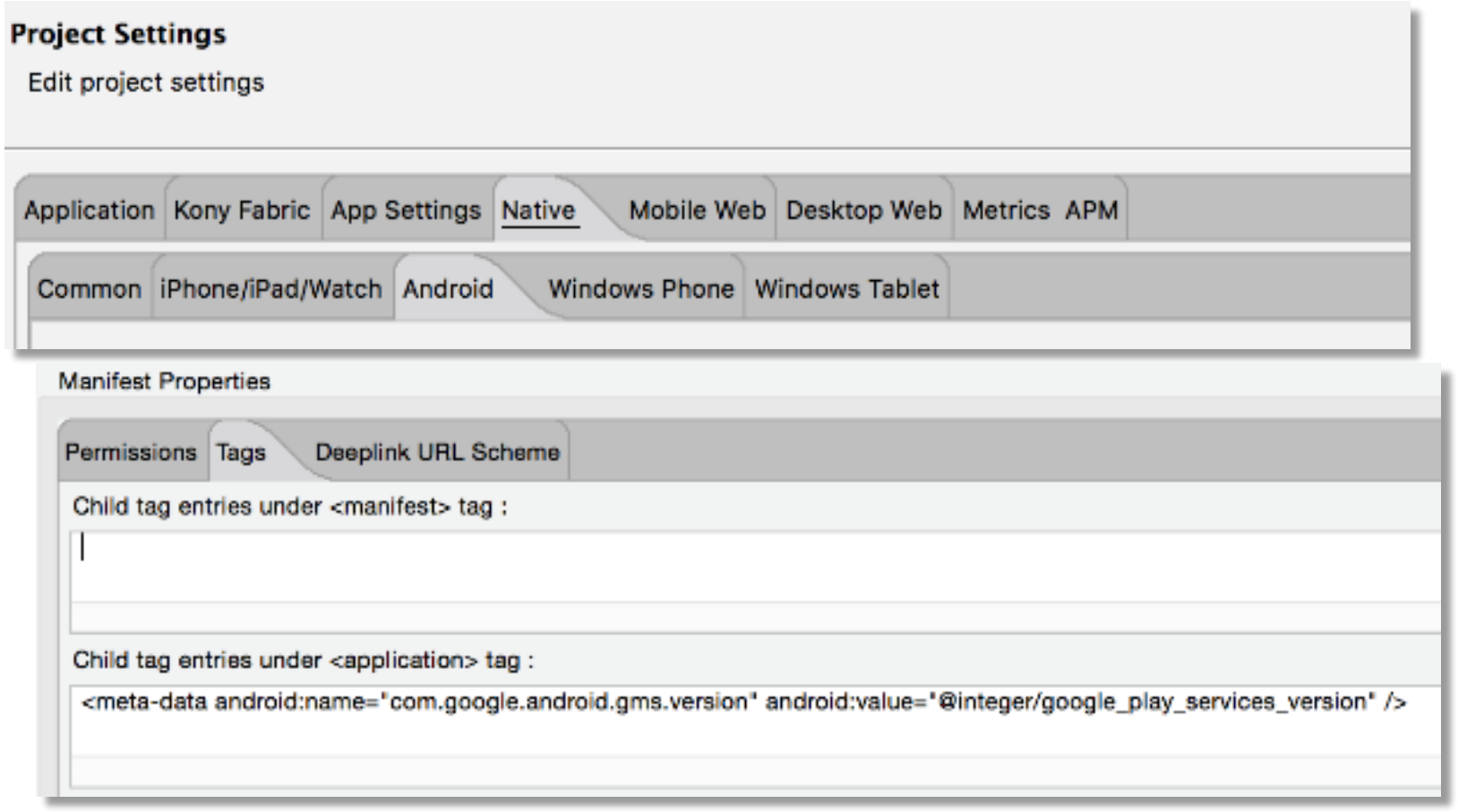
Google Play Services

- For the Android key to work, you'll also need to make sure you have Google Play services installed in your Android SDK
- Here is what you should see when you run the SDK manager:



Android Native Properties

- The last step is to configure some Tags in the Android Native properties:



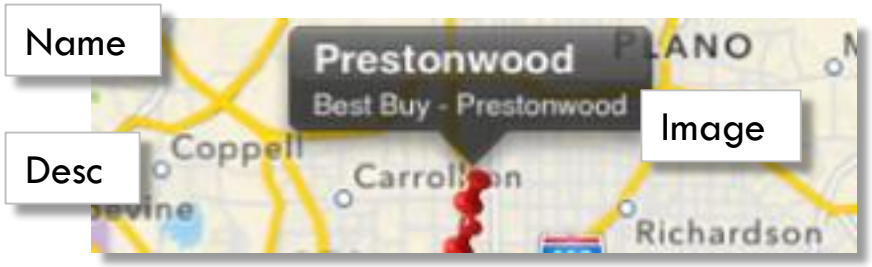
- You must enter this manifest just as you see here

Mobile Maps - Map Data

- All the data for the map is specified in the property: **locationData**
  - **locationData** can only be set in code (Not using Visualizer master data at design time)
- **locationData** is an array of objects

- Each object represents one map point and has the following keys:

  - **Id -** Set the id for each pin data; mandatory, if you are going to update them later
  - **lot** - Specify the latitude
  - **lon** - Specify the longitude
  - **name** - Set the value shown in the pin
  - **desc** - Set the location description in the pin
  - **image** - Specify a pin image



  - **meta** - Specifying how the pin will be displayed on the mobile web versions and has the following keys:
  - color and label - Used for drawing the pin. Typical label values are "A", "B", and so on…
  - **Note:** Even if providing only 1 pin, you still need to create an array of one pin

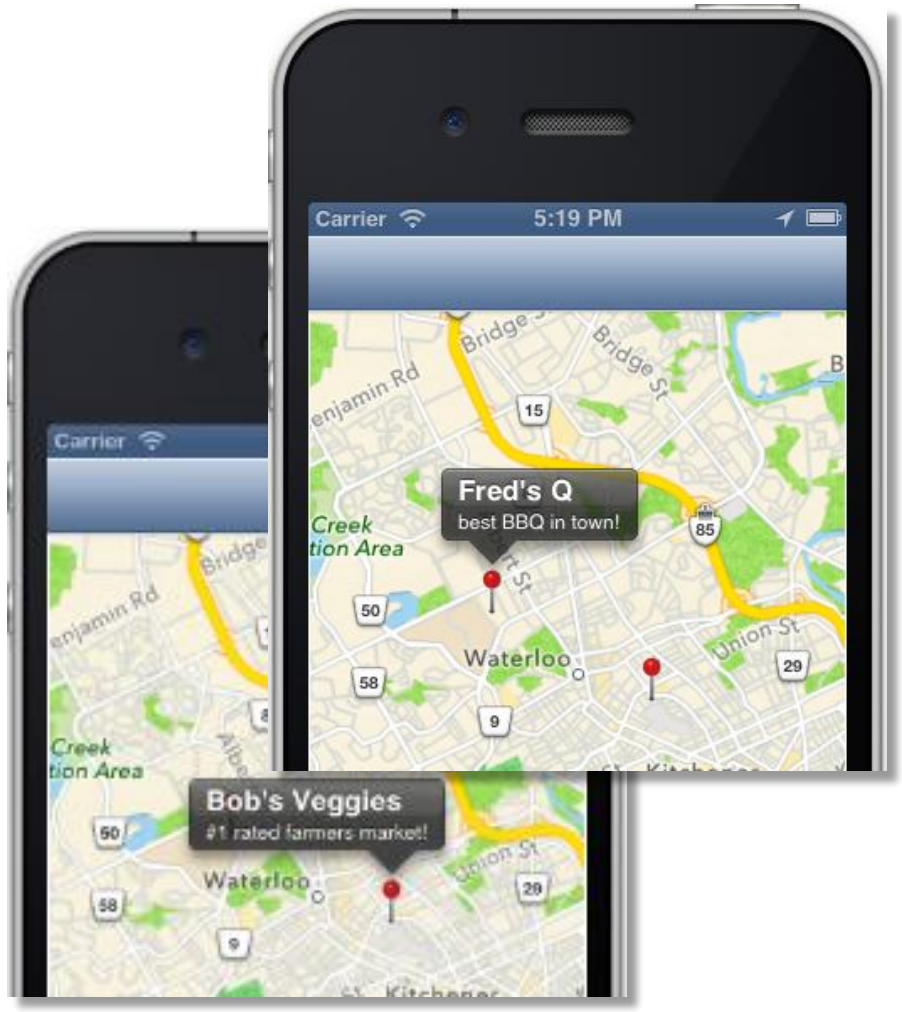
- Updating locationData refreshes the map automatically

Mobile Maps - Map Data (cont'd)

- Here is a sample code snippet to set two points on the map:

```
locData=[{lat:"43.47591", lon:"-80.53964",
        name:"Fred's Q", desc:"best BBQ in town!",
        image:"redpin.png", meta:{color:"green",
                label :"A" }},
        {lat:"43.4643", lon:"-80.51009",
        name:"Bob's Veggies",
        desc:"#1 rated farmers market!",
        image:"redpin.png", meta:{color:"green",
                                label :"B" }}];
this.view.mapStores.locationData=locData;
```

- Note: It's an array of point objects



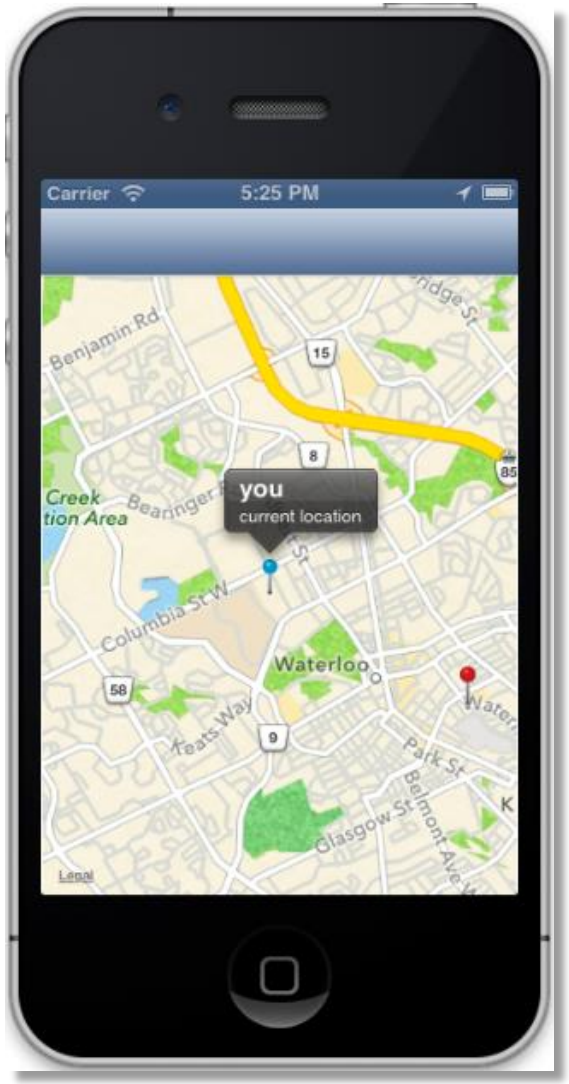
Customization of Map Pins

- From version 7.0, apart from an image that is bundled as part of the application, we can also have images from the following sources:
  - **File Path:** Locally stored images can be used as map pin images
  - **Base64 String:** Base64 encoded string format of an image can be used as a map pin image. This can be downloaded images or local images
  - **Image Object:** Image objects created using the new image manipulation APIs can be used as a map pin image
- When we use any of these option, we have to pass an JSON object to the **image** property of **locationData**
- The JOSN object takes source, sourceType and anchor properties

```
image:{
    source:"mappin.png",
    sourceType:kony.map.PIN_IMG_SRC_TYPE_RESOURCES,
    anchor:kony.map.PIN_IMG_ANCHOR_BOTTOM_CENTER
     }
```

Mobile Maps - Coding for the Map Widget

- Here are some tips and best practices (so far):
  - The map will be centered around the first pin in **locationData** so consider what you want that point to be
  - It's very typical to plot the user's current position on the map:
    - Best to use a different colored pin for this point
    - Typical to put this point first to center the map around the user
    - Don't forget to put useful info in the callout
  - If you don't want your pin clickable, then you can disable the callout from showing by setting **showcallout:false** in the pin's **locationData**



Mobile Maps - Other Coding Methods

- **locationData** can contain more information other than the required values
  - Just like the segment, the map pin can contain other information
  - For example, address information or phone number of store locations
- Here is an example with extra data (address, city and phone):

```
locData = [{lat:"43.47591", lon:"-80.53964", name:"Fred's Q",desc:"best BBQ in town!", image:"redpin.png",
meta:{color:"green", label :"A" }, address:"123 Sesame St.", city:"New York", phone:"212-555-1212"},

{lat:"43.4643", lon:"-80.51009", name:"Bob's Veggies",desc:"#1 rated farmers market!", image:"redpin.png",
meta:{color:"green", label :"B" }, address:"100 Wall St.", city:"New York", phone:"212-555-1313" }];

this.view.mapStores.locationData=locData;
```

- When a map point is clicked, ALL the point data is available

Mobile Maps - Events

- The Map widget supports 3 events:
  - **onClick** - Fired when the map is touched away from a pin or pin callout
    - Returns the point latitude and longitude as an object passed to the handling function
    - For example:

```
function onClickHandler(locData){
        kony.print("lat: "+locData.lat +" & lon: " + locData.lon"); }
```

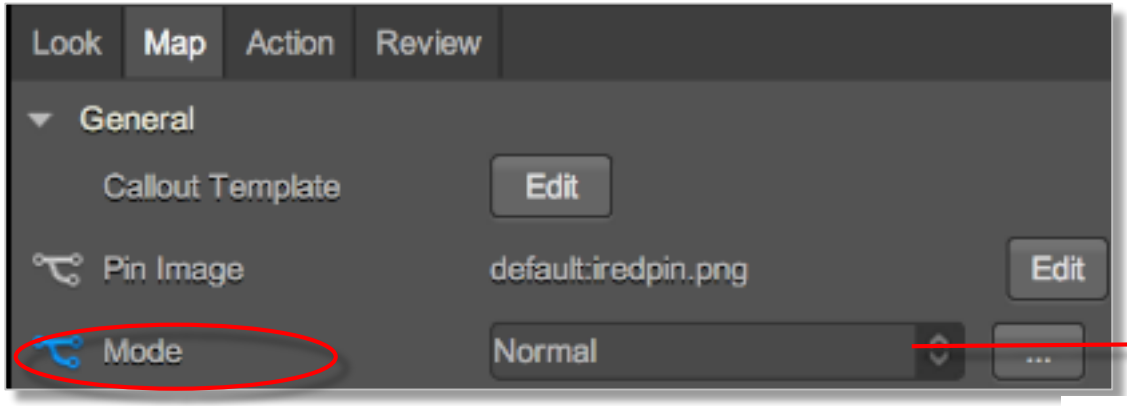
  - **onPinClick** - Fired when the pin is tapped (thus showing the callout)
    - Returns the locationData object for the pin including any other data you added
    - For example:

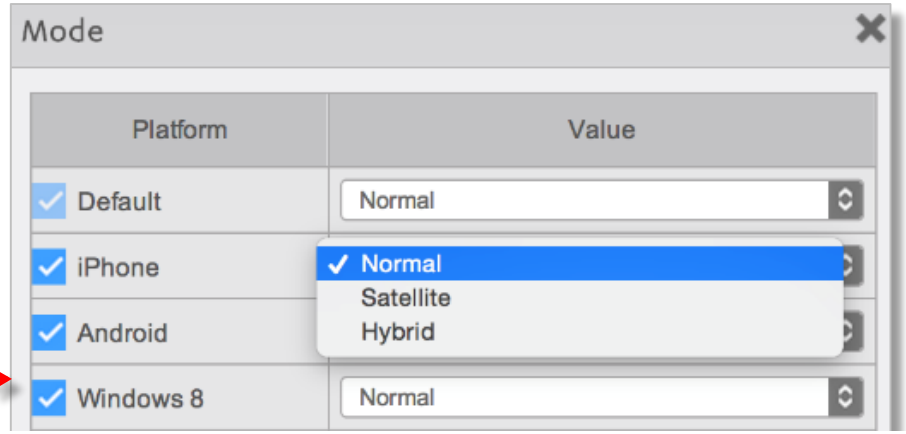
```
function onPinClickHandler(locData){
        kony.print("pin description: "+locData.desc); }
```

  - **onSelection** - Fired when the pin callout is tapped
    - returns the locationData object for the pin including any other data you added

Mobile Maps - Map view mode

- The **mode** property dictates which view to use for the map:





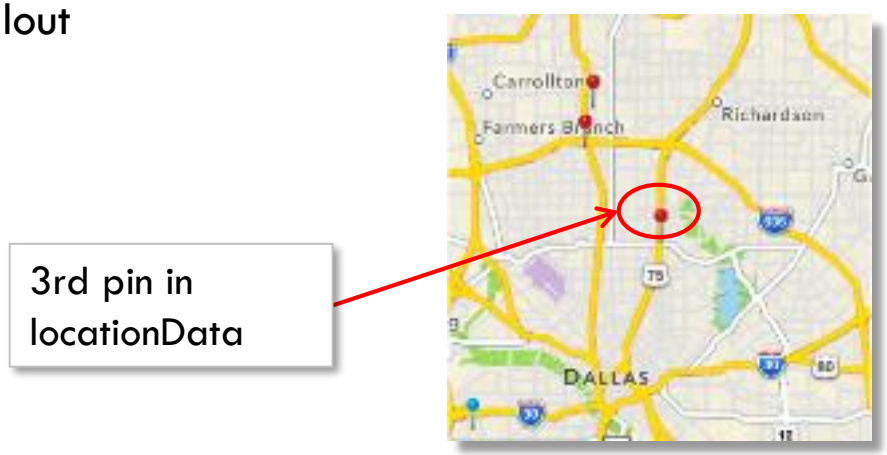
- Each phone supports different views:

Modes	Normal	Satellite	Hybrid	Street	Polygon	Traffic	Terrain
IOS	Yes	Yes	Yes	No	No	No	No
Windows Phone/Kiosk	Yes	Yes	Yes	No	No	No	No
Android	Yes	Yes	No	No	No	Yes	No
SPA	Yes	Yes	Yes	No	Yes	No	Yes
Mobile Web (basic)	Yes	Yes	Yes	No	No	No	Yes
Mobile Web (BJS)	Yes	Yes	Yes	No	No	No	Yes
Mobile Web (Advanced)	Yes	Yes	Yes	No	Yes	No	Yes
DesktopWeb	Yes	Yes	Yes	No	Yes	No	Yes

- **Note:** You can set this value in code to change it programmatically

Mobile Maps - APIs

- There are times when you'll want to programmatically move the map focus to a specific place
  - This place can be either an existing pin OR a set of coordinates
- **navigateTo(index, showcallout)** - allows you to navigate programmatically to an existing pin - placing it in the center of the map - where:
  - **Index** - is the 0 based index of the pin in the map's locationData
  - **showcallout** - is a Boolean value when set to true will cause the pin's callout to be displayed.
  - **For example: this.view.myMap.navigateTo(2,false)** would place the 3rd pin in locationData in the middle of the map and not display the callout



Mobile Maps - APIs

- **navigateToLocation(locData,showcallout,dropPin)** - Allows you to navigate programmatically to the specified location on the map and optionally display a pin and/or the callout
  - **locData** - a complete pin's object (a row for the map's locationData)
  - **showcallout** - is a Boolean value when set to true will cause the pin's callout to be displayed
  - **dropPin** - is a Boolean value when set to true will put a pin on the map as configured by the locData (image key for pin image)
  - For example:

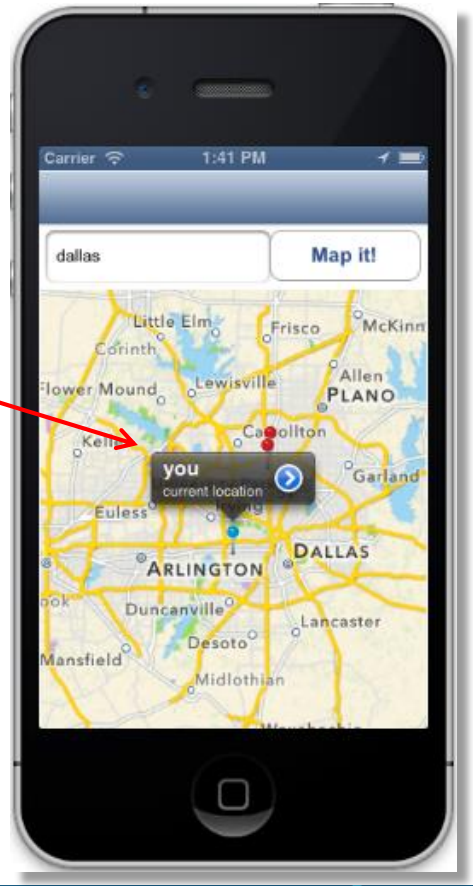
```
this.view.myMap.navigateToLocation({lat:32.876068, lon:-96.898529, name:"New Point", desc:"user clicked", showcallout:true, image:"iredpin.png", meta:{color:"red",
label :"A"}, calloutData: {name:"New Point", address:"no address specified", phone:"NA"}}}, true, true);
```

  - This would navigate to the pin specified in the pin data, show the callout (using the template) and show the pin iredpin.png

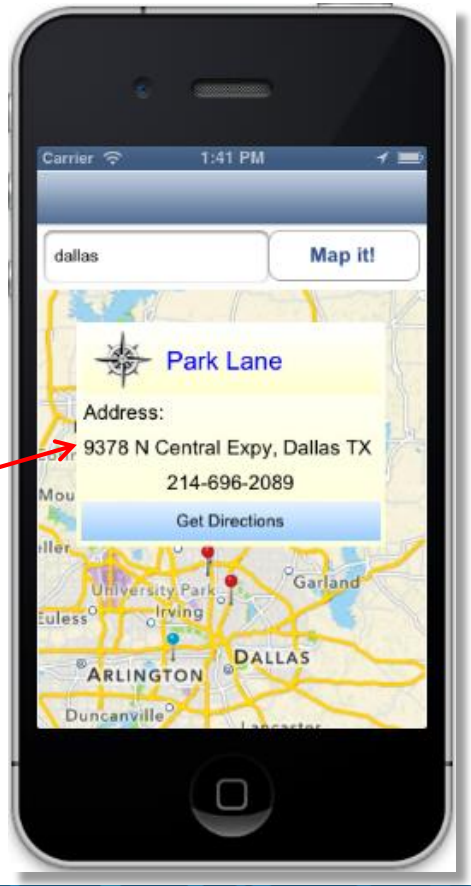
Mobile Maps - Custom Callouts

- The default callout is fairly restricting only showing the pin's "name" and "desc" properties
- Just like with Segments, you can create a callout template and use it with your map

Here is the default callout:
Note: that blue arrow "shows" up when you have attached code to the onSelection event

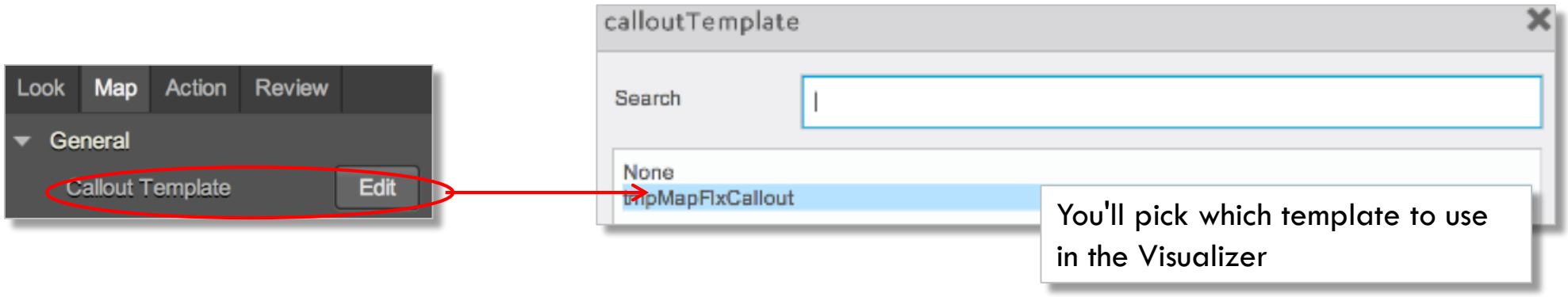


Here is an example of a callout using a template:



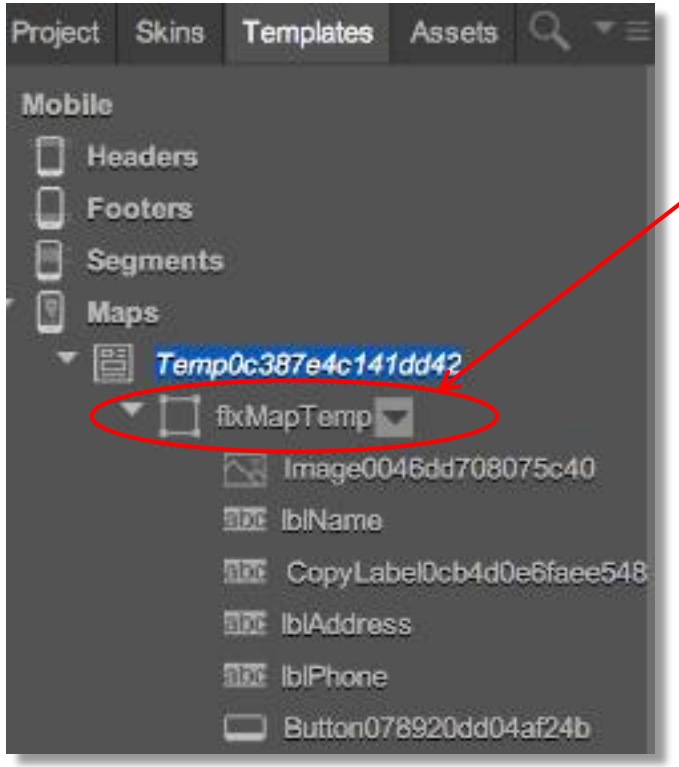
Mobile Maps - Callout Templates

- To use custom callouts, there are a few things that have to set up:
  - Create the template you're going to use
  - Specify the data to be used for the template (for each pin)
  - Specify a **widgetDataMapForCallout** object to map the pin data key (in locationData) with the widget names on the template
  - You tell your map to use that template by setting the calloutTemplate property to the name of your template
  - **Note:** You are limited to 1 callout template for the map (not like segments where each row can have a unique template)

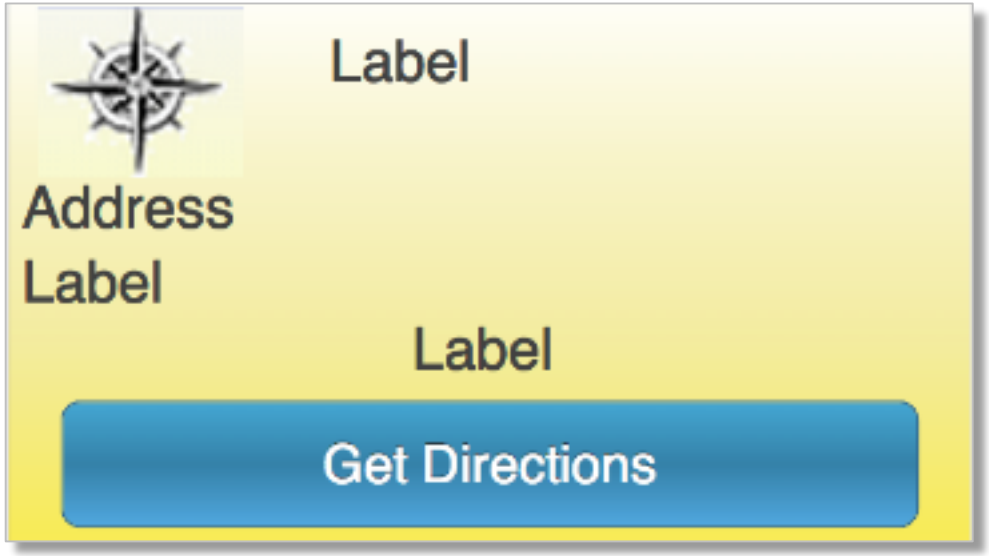


Mobile Maps - Creating the Template

- **NOTE:** We have to create our template AND our form that uses this template in Visualizer using the old box model
  - The first widget must be an FlexContainer
    - Within that FlexContainer put whatever you want:



Our Template starts with a FlexContainer



Mobile Maps - Callout data

- When specifying the locationData for each pin, specify a new key, calloutData, and set all the data you want to show in that object
- Let's look at an example:
  - In my callout template, I've provided a place to put the store name, the store's address and phone number.
  - Here is what the data for a pin might look like:

    {lat:32.876068, lon:-96.898529, name:"Park Lane", desc:"Best Buy - Park Lane", showcallout:true, image:"iredpin.png", meta:{color:"red", label :"A"}, calloutData: {co_name:"Park Lane", address:"9378 N Central Expy, Dallas TX", phone:"214-696-2089"}}

  - Note the structure of the **calloutData** key

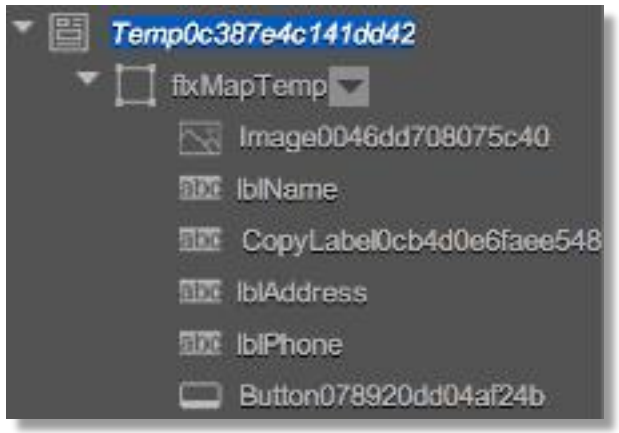
Mobile Maps - widgetDataMapForCallout

- The last piece of hooking up all the data is to map the data from calloutData to the widgets in the callout template
- For our example, here is the **calloutData** and our template outline:

```
{co_name:"Park Lane",
 address:"9378 N Central Expy, Dallas  TX",
 phone:"214-696-2089}}
```

- Here is how we'd set up the **widgetDataMapForCallout:**

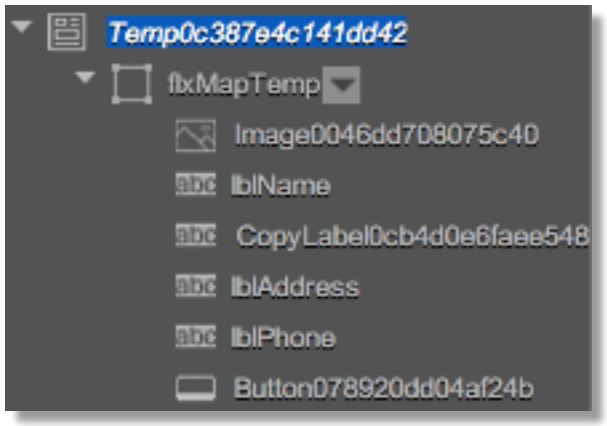
```
This.view.myMap.widgetDataMapForCallout = {lblName:"co_name",
lblPhone:"phone"};
```



- This example doesn't map data to **image0046dd708075c40**, **CopyLabel0cb4d0e6faee548** and **Button078920dd04af24b** because those widgets have static text on them

Mobile Maps - Template notes

- You'll note that our template does have static values in it:
  - The image is fixed to add visual interest to the callout
  - The other label is used to say "Address:" before we show the address data



- You'll also note that the template has a button on it:
  - The button click event needs to be handled in the template code
  - The button will NOT know which pin it's on since it's on the template used for all pins
  - Best practice is to use the **onPinClick** event to set a **currentPin** variable to that pin's data and the button code can use that

Mobile Maps - Other APIs

- kony.map.containsLocation(location, shapeData)
  - This API can be used to identify whether the given location is present in the shape described by shareData or not
  - “location” is a JSON object containing a latitude and longitude
  - “shapeData” is an array of JSON objects containing multiple locations of a shape
  - This API returns boolean
- kony.map.distanceBetween(location1, location2)
  - kony.map.deco: This API can be used to identify the distance between two locations
  - “location1”, “location2” is a JSON object containing a latitude and a longitude
- kony.map.decode(encodedPolylineString)
  - This API should be used to decode the encodedPolylineString which is part of each “step” in the kony.map.searchRoutes API

Mobile Maps - Navigation

- We can also draw navigation routes using the Google web services
- Google provides web services to return directions data between 2 locations:

  http://maps.googleapis.com/maps/api/directions/xml?origin=<lat,long>&destination=<lat,long>&sensor=false

  - Create a service and you can get the list of intermediate points and turn-by-turn directions
- But you're still stuck with straight lines
- What can you do?
- Remember the kony.application.openURL() method? That's the answer!
  - This will open up the device browser to Google's browser map application with directions between the points
  - Let's see how this would work

Mobile Maps - Navigation (cont'd)

- In the template examples we used, our template had a "Get Directions" button as shown on the right:
  - The first question to ask is: "get directions FROM where TO where?"
    - In this case, the TO is the store in the pin
    - The FROM can be our blue pin showing our current location
  - Get the coordinates for both those pins and in your button click event you only need:

    kony.application.openURL("https://maps.google.com/?saddr="+homePin.lat+","+homePin.lon+"&daddr="+toPin.lat+","+toPin.lon);

    - where homPin and toPin represent the data for those 2 pins
- Note: whenever doing something programmatically you probably want to close the callout:

  this.view.myMap.dismissCallout(pinData);

Mobile Maps - Navigation (cont'd)

- Here is what we get:

Clicking the button in the app…

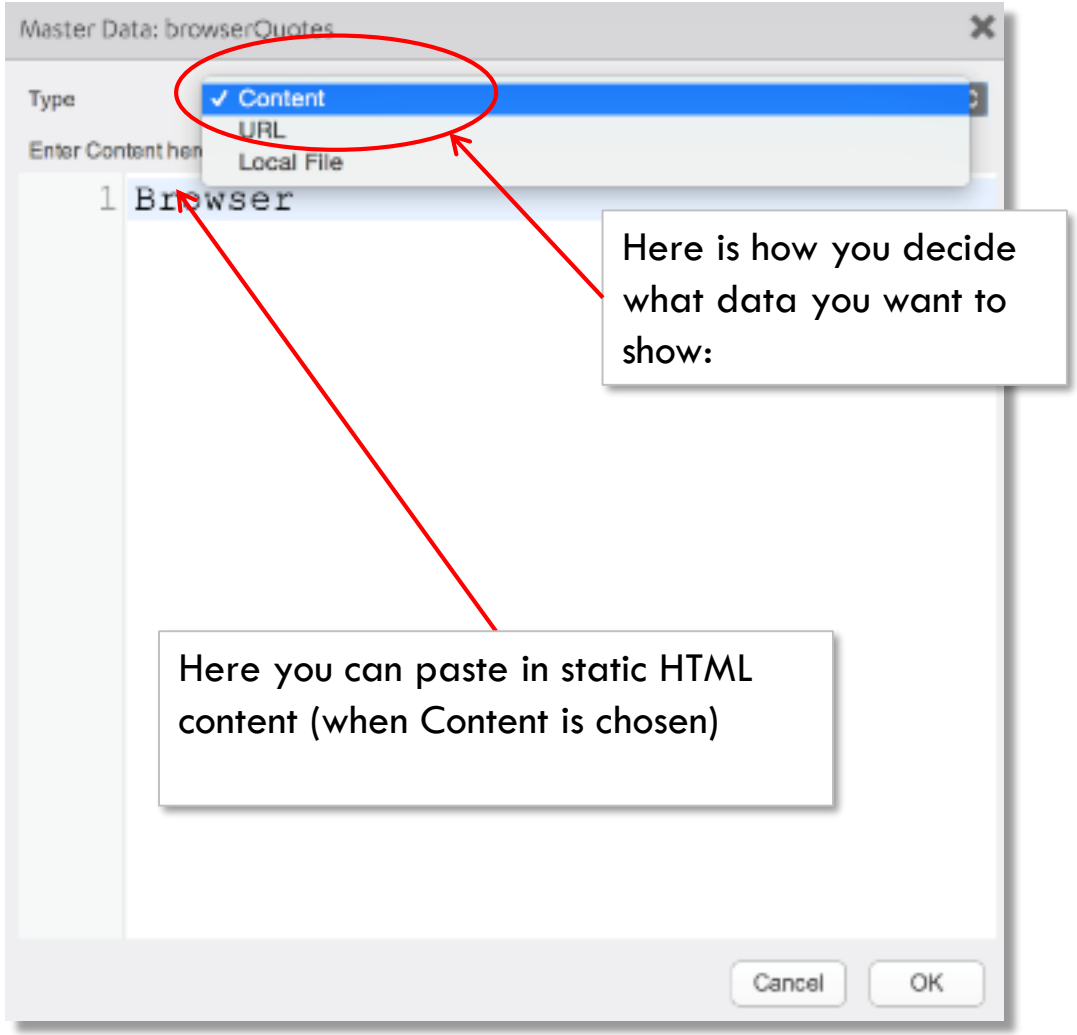
…opens the device browser to the Google map app

Browser Widget - Overview

- Using **kony.application.open URL** API, you will be able to open native browser
- The problem with this is that the user is taken OUT of the app into the device's browser - requires the user to go back to the app when done
- The browser widget is used to display an in-app browser
  - By using browser widget, the flow control still remains within the application
  - The rendering capability of the browser widget is still the same as that of native device’s browser
  - With the browser inside the app, you can now control that user's experience including the navigation within the browser
  - The browser controls are not visible, so you can add the browser widget inline to any form for a seamless look with your native components

Browser Widget - Overview

- The browser has a **masterData** property in the Visualizer that can be used to set the info for the browser
- You have 3 choices:
  - Use static content - Lets the browser display HTML content you have
  - Use a URL to browse content on the web
  - Use locally packaged web content

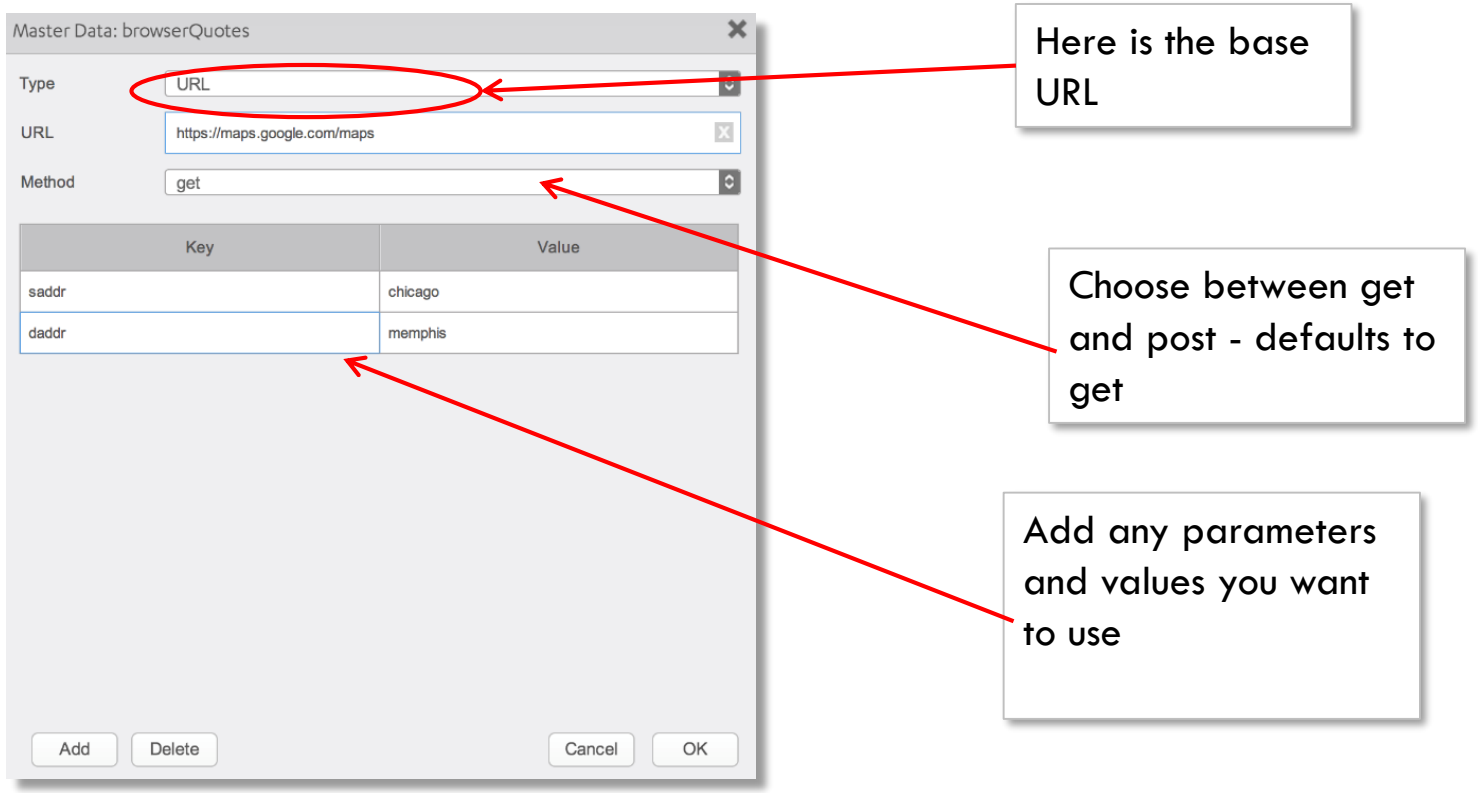


Browser Widget - Static content

- Static content can be useful:
  - If you want individualized styling for pages
  - If you have a way to get or generate the HTML
  - Can support any data - It is all in the HTML and not built out from widgets on a form
  - Doesn't look like a browser
- To set the data programmatically, the use the browser widget's **htmlString** property and set it to the HTML content as a string
  - The screenshot shows an example of how this looks like:

Browser Widget - URL

- The browser widget can also display external content by specifying URL information:



- In this example, the resultant final URL that will be called is:
  https://maps.google.com/maps?saddr=chicago&daddr=memphis

Browser Widget - URL (cont'd)

- Programmatically, to set the URL value and parameters, use the browser widget's **requestURLConfig** property
- **requestURLConfig** is an object with the following keys:
  - **URL** - specifying the base URL starting with http://
  - **requestMethod** - is an optional key set to BROWSER_REQUEST_METHOD_GET or BROWSER_REQUEST_METHOD_POST
  - If you don't set this value, it'll be set to _GET by default
  - **requestData** - is an optional key that is an array of parameter arrays in the format:
  - [[param1,value1], [param2,value2], …] - NOTE: The parameter is set in an array where the first element is the parameter name and the second element it's value
    - Note, if only 1 parameter is needed, you must still send an array containing that one parameter array

- Here is the same example we just saw set up in the Visualizer but done in code:
  - this.view.myBrowser.requestURLConfig = { URL:"https://maps.google.com/map",
    requestMethod: constants.BROWSER_REQUEST_METHOD_GET,
    requestData: [["saddr","chicago"],["daddr","memphis"]] };

Browser - Events

- The Browser exposes 2 events: **onFailure** and **onSuccess**
  - These events are raised every time a URL (not static content) is loaded in the browser
  - Both accept an **eventobject** that is the browser widget itself
    - This is of little use - it just reflects back the ORIGINAL configuration of the browser
  - Note: currently there is no way to get the current URL from the browser widget if the user is allowed to navigate within the browser
  - These events are typically used to know if the browser should be shown or not:
    - You want to FIRST try to load the data (URL or static content)
    - IF it was successful, then display that browser widget
    - IF it was unsuccessful, go to plan B - whatever your backup plan is
  - Since it's not displayed yet, the user is not bothered with a widget showing the wrong thing (nothing, typically if the page doesn't load)

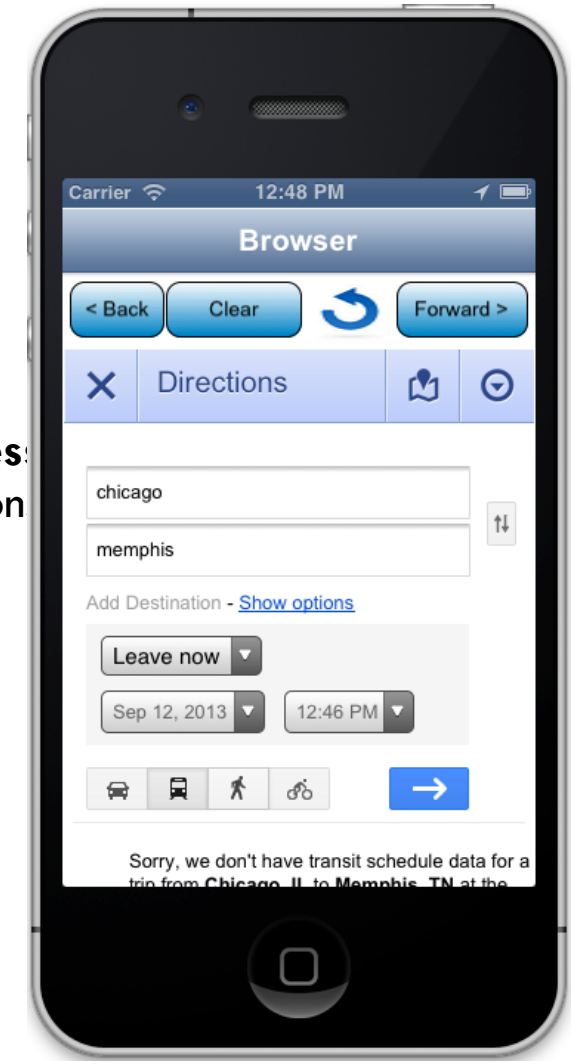
Browser - APIs (Rich Client only)

- The browser widget does provide you the ability to control some of the browser features: with the following methods:
  - **canGoBack()** - returns a Boolean value indicating if there is a page to go back to - will always return false when the browser first loads
  - **canGoForward()** - returns a Boolean value indicating if there is a page to go forward to - will always return false when the browser first loads
  - **goBack() -** is used to navigate one step back in the browser history
  - **goForward() -** is used to navigate one step forward in the browser history
  - **reload() -** is used to reload the current web page
  - **clearHistory() -** clears the page history

- Let's look at an example of how all these can work…

Browser - API Example

- Let's look at this simple example where we expose these functions through buttons:
  - the < Back button will call the **goBack()** method
  - the Forward > button will call the **goForward()** method
  - the Clear button will call the **clearHistory()** method
  - the button will call the **reload()** method
  - When the page loads or ANY time we navigate on the page (i.e., in the **onSucces** event), we "recalculate" if we should enable/disable the back and forward button
    - use the **canGoBack()** and **canGoForward()** methods to check
  - Here's what it looks like with both disabled (when the browser first loads):



Browser - API Example (cont'd)

- The functionality for all the buttons is straight forward - just call the method
  - For example, the Back button's code is simply:

    this.view.myBrowser.goBack();
- The only tricky part is knowing when to enable/disable the navigation buttons
  - Here is the code to set them in the browser's onSuccess callback:

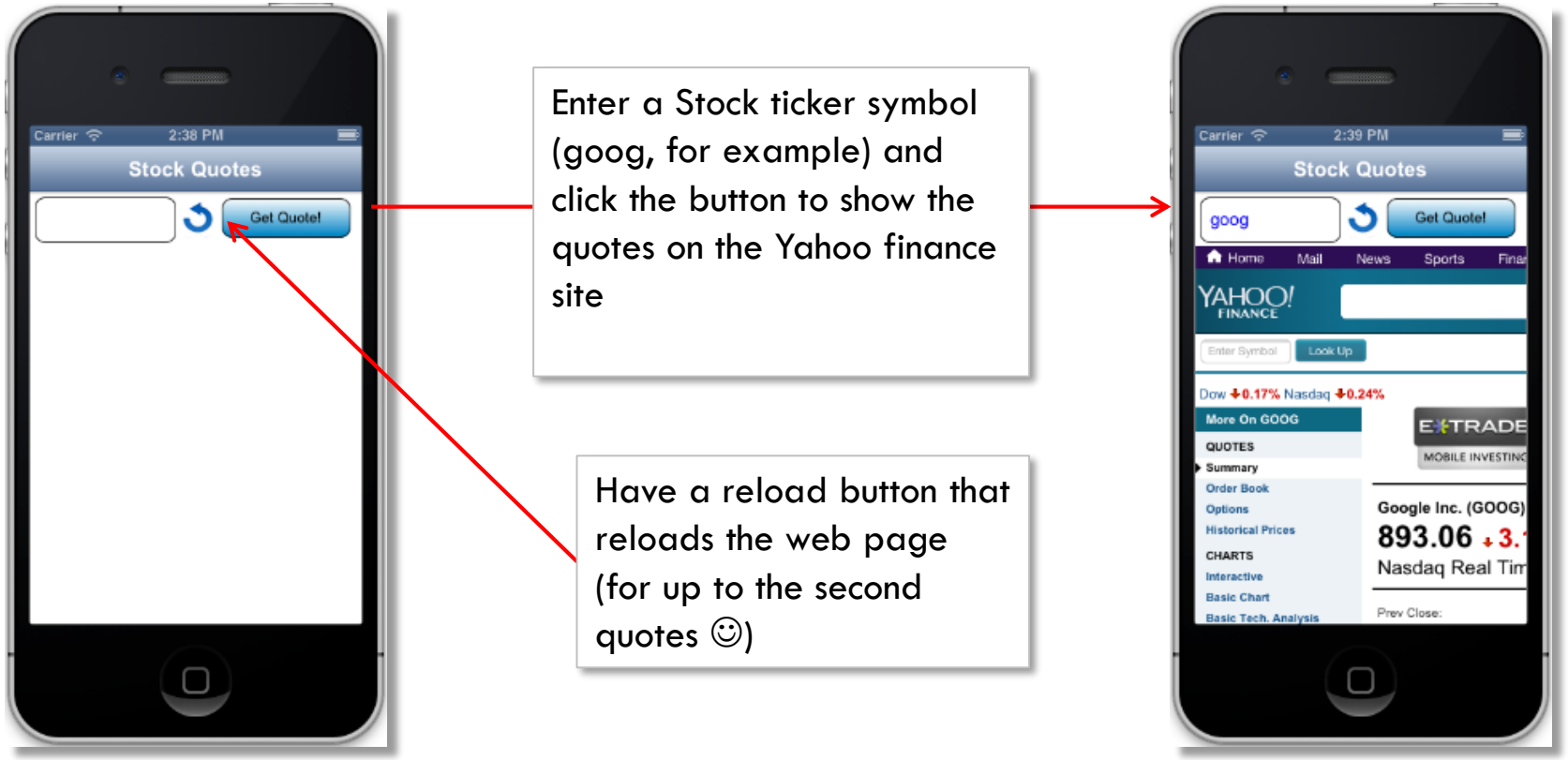
```
browserSuccess:function(browser){
                    if (browser.canGoBack()) {this.view.btnBack.setEnabled(true);
                    this.view.btnBack.skin="skButton";}
                      else {this.view.btnBack.setEnabled(false);
                      this.view.btnBack.skin="skBtnDisabled";}
                      ...if (browser.canGoForward()){
                      this.view.btnFwd.setEnabled(true); this.view.btnFwd.skin="skButton";}
                      else { this.view.btnFwd.setEnabled(false);
                      this.view.btnFwd.skin="skBtnDisabled"}
            }
```

Browser - Considerations

- The Browser widget has the following important considerations:
  - The Browser widget, unlike other widgets, can be slow to load - ESPECIALLY if loading a full web page
  - The Browser widget uses a lot of initial device memory
  - The memory usage increases in proportion to the number of images and static text rendered
  - If there are multiple instances of the Browser widget in the same application, there may be issues related to sharing of information (cookies, for example)
  - You cannot have multiple Browser widgets in a screen. As a guideline, its best not to have more than two Browser widgets in an application
- Best practice is to:
  - Control where the user can "go" in a browser
  - If you want the user to just free-navigate the web, consider the **openURL** method to kick out to the device native browser - outside of the app

Browser widget - Exercise

- Let's build the app show on the screens below:



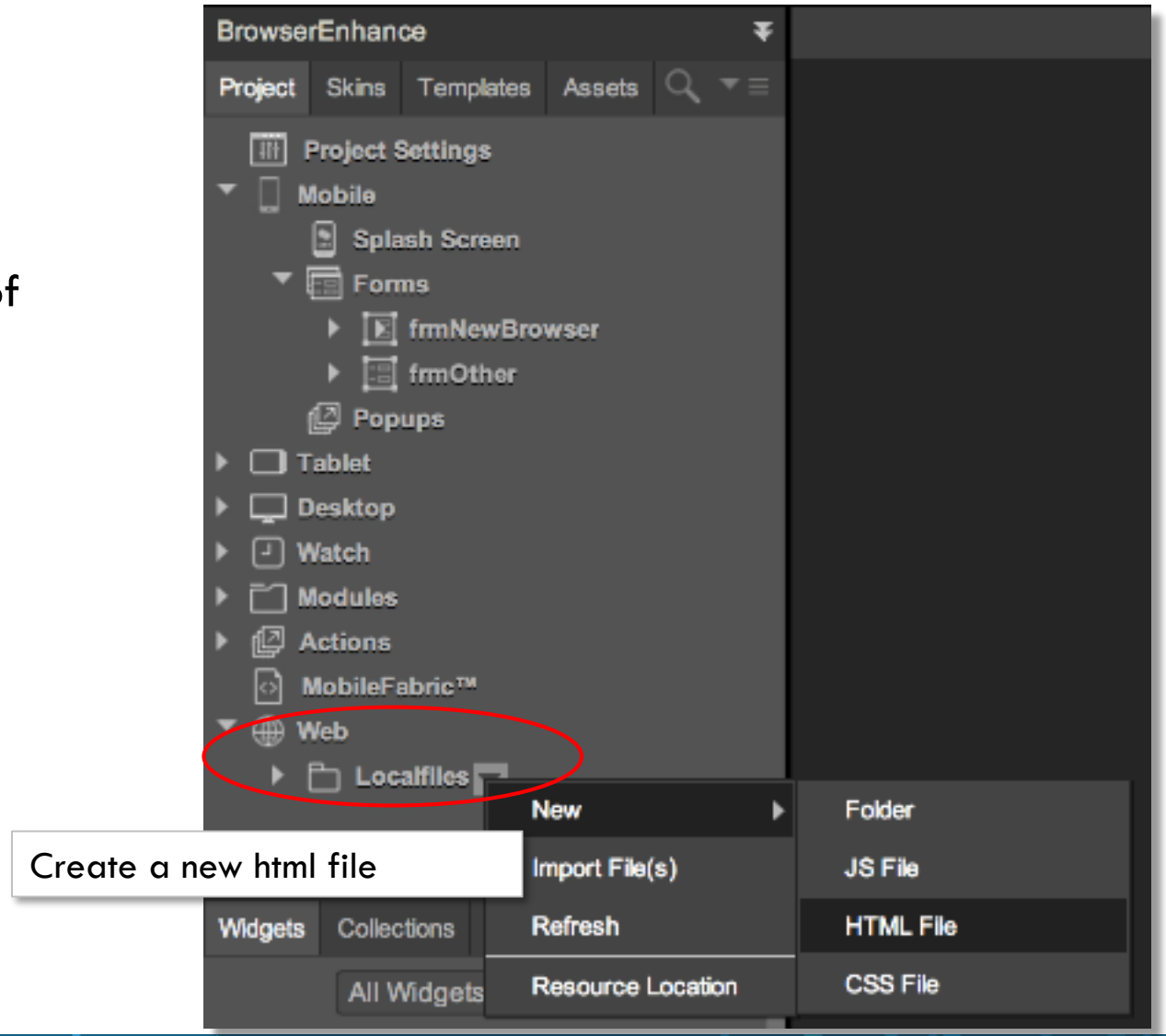
- The format for the URL is: http://finance.yahoo.com/q?s=symbol so the page shown above is showing this URL: http://finance.yahoo.com/q?s=goog

Browser - Locally Packaged Web Content

- Browser widget supports locally packaged web content also
- There is a new API to enable developers to communicate with Kony application context from browser context
- Local packaged web content
  - Browser Widget supports rendering html pages packaged locally under web/local files in Kony Visualizer
- Native communication API
  - kony.evaluateJavaScriptInNativeContext(JavaScript);
  - enableNativeCommunication property controls access to native JS context

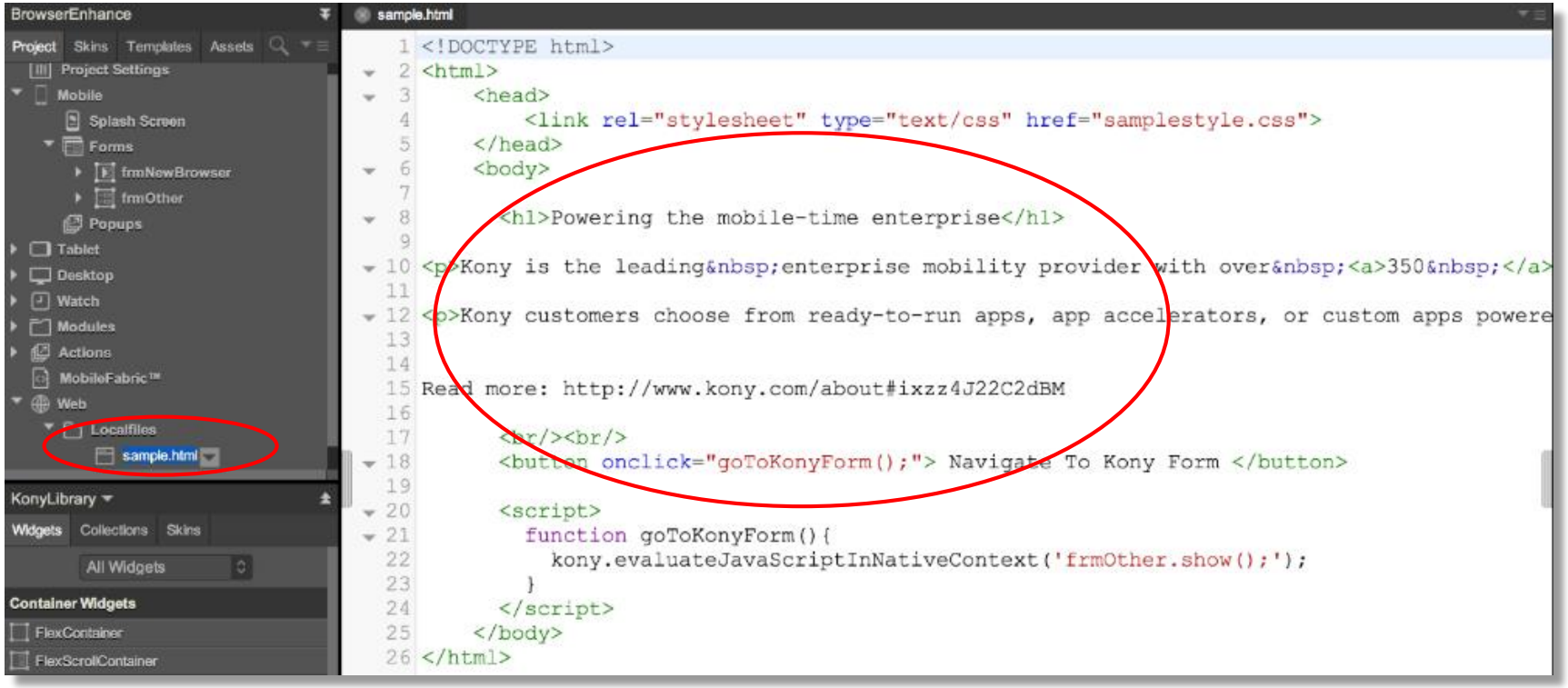
Locally Packaged Content - Use Cases

- In case, where the developer has static content not hosted on an external server, the browser needs to support loading the local web assets
- If user needs to invoke Kony API on click of a button in HTML page, a mechanism is needed to execute this JavaScript in Kony context
- We can create/add local web assets (HTML, JS, CSS files) into **Web** folder as shown in the next slide



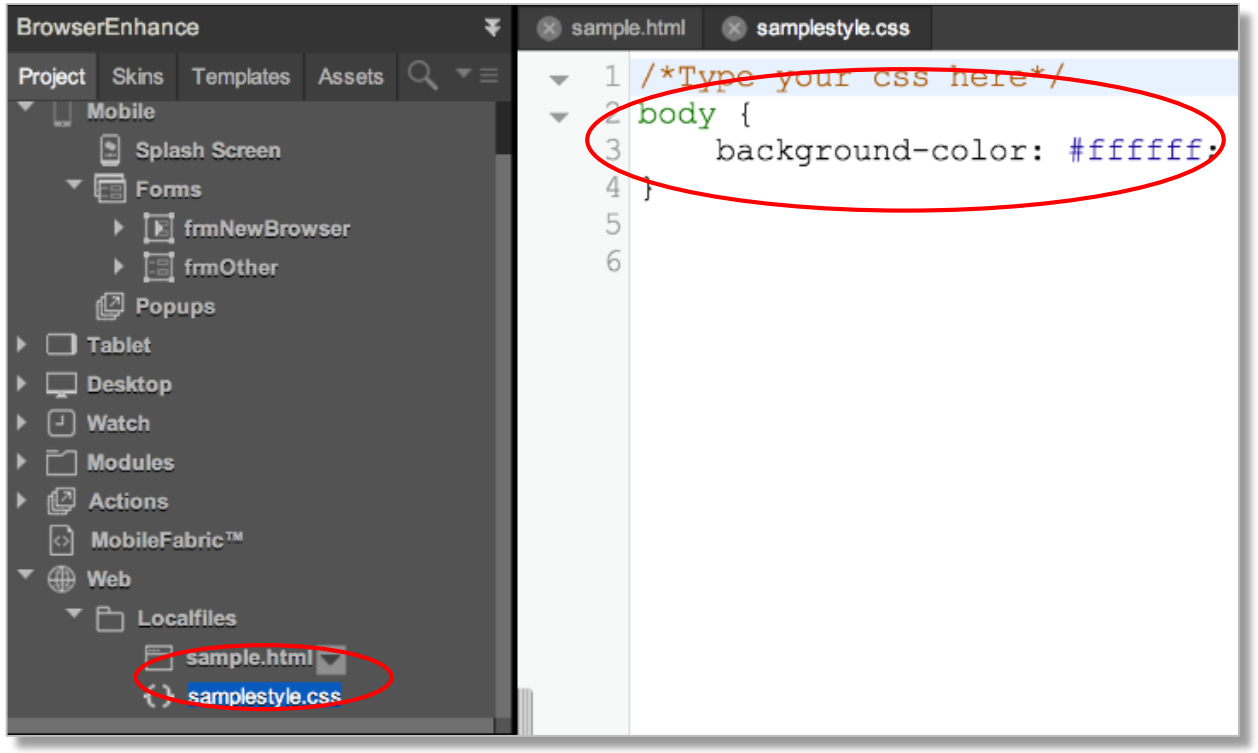
Local Web Content - HTML

- Create a html file and add the html content



Local Web Content - CSS

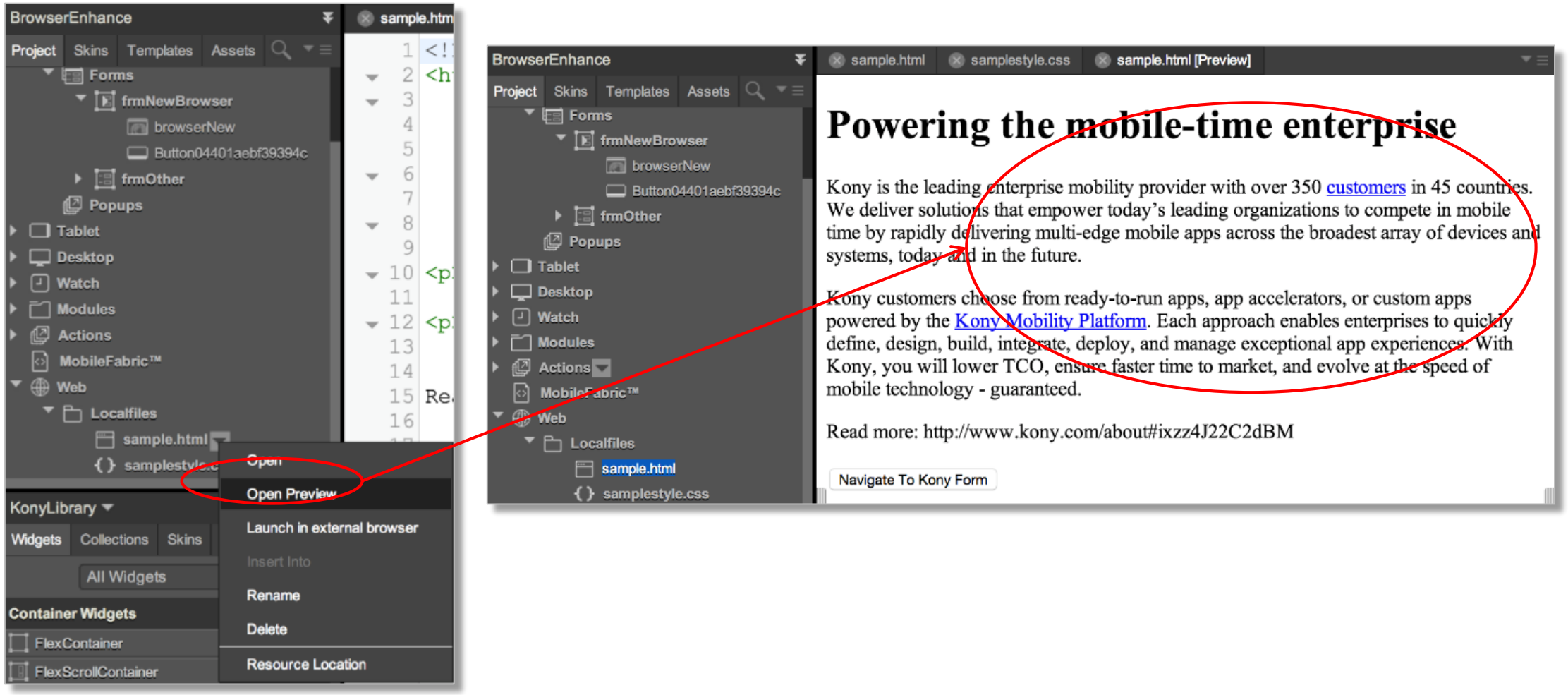
- Create a CSS file and add the CSS styles



- Similarly, you can also create a JS file and add the JavaScript code for your web page

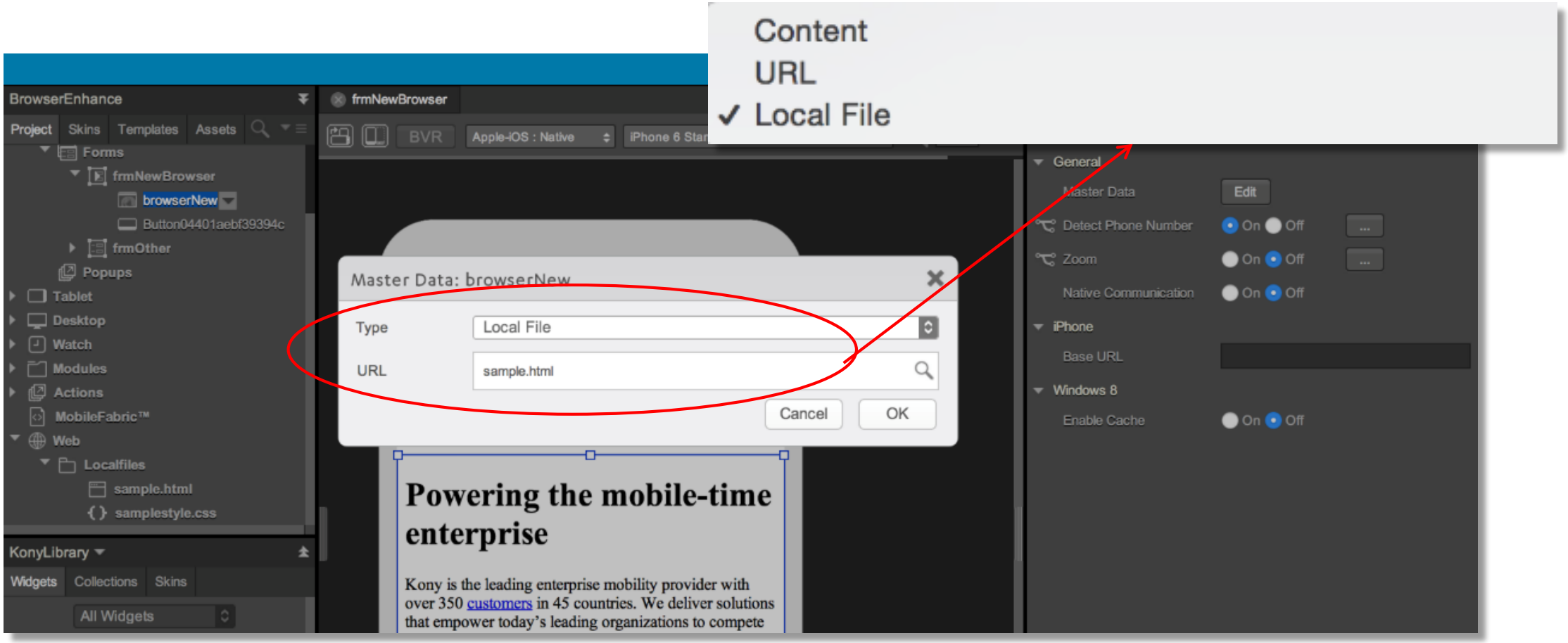
Local Web Content - HTML Preview

- We can see the preview of the html page



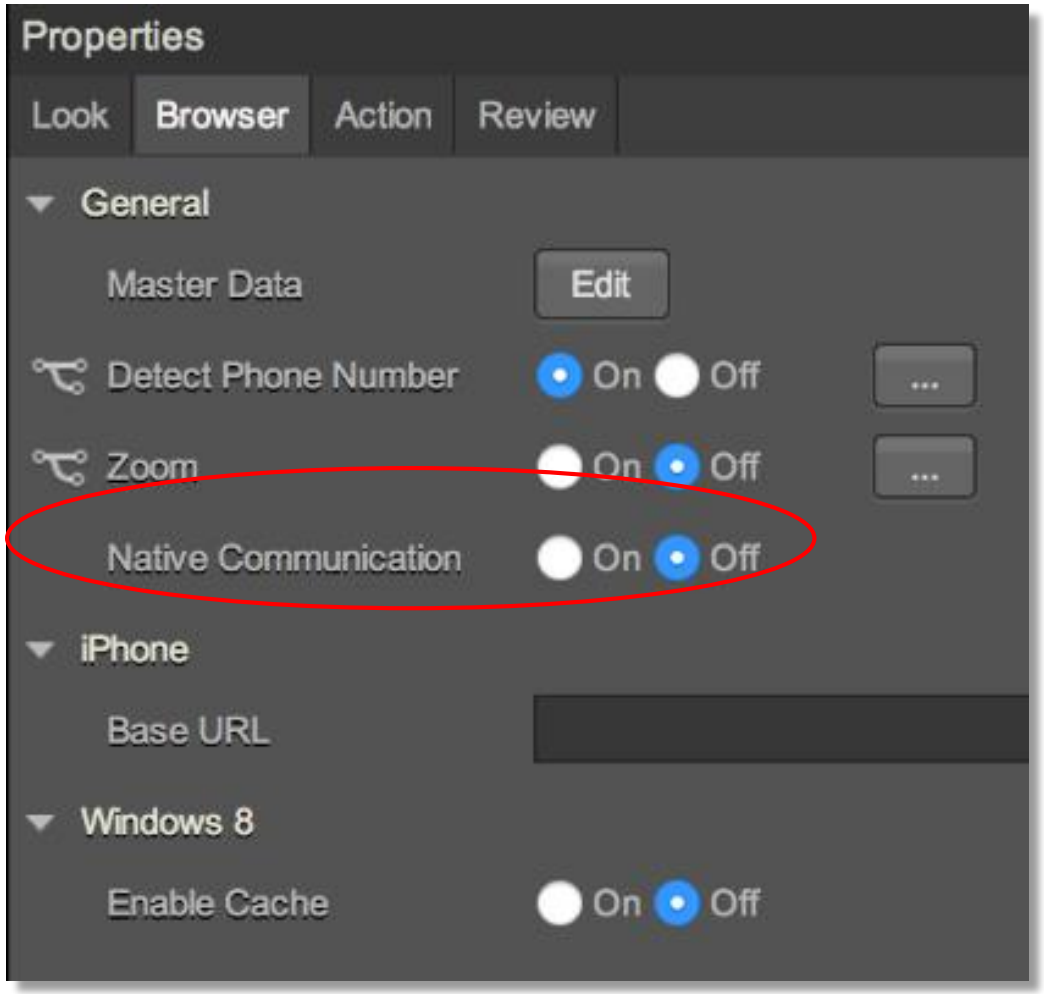
Browser Widget - Using Local Web Content

- Now, go to **Master Data** property of the Browser Widget and configure the local html file that we created



Native Communication Property

- ***enableNativeCommunication*** - This property enables the access to Kony native capabilities from within web app's (HTML page) JavaScript code
- ***kony.evaluateJavaScriptInNativeContext(String javascript)*** - This function is used to execute the JavaScript code in the ***javaScript*** parameter



Native Communication API - Example Code

- Example Usage:
  - kony.evaluateJavaScriptInNativeContext (“form1.show()”);
  - kony.evaluateJavaScriptInNativeContext(“displayAccountListForm()”);

```
<html>
 <head>
 <link rel="stylesheet" type="text/css" href="samplestyle.css">
 </head>
 <body>
  <h1>Powering the mobile-time enterprise</h1>
  <p>Kony customers choose from ready-to-run apps, app accelerators, or custom apps powered by the <a
href="/products#experience-platform">Kony Mobility Platform</a>. Each approach enables enterprises to quickly define, design, build,
integrate</p>
 Read more: http://www.kony.com/about#ixzz4J22C2dBM
  <br/><br/>
  <button onclick="goToKonyForm();"> Navigate To Kony Form </button>
  <script> function goToKonyForm(){
     kony.evaluateJavaScriptInNativeContext('frmOther.show();');
     }
  </script>
 </body>
</html>
```

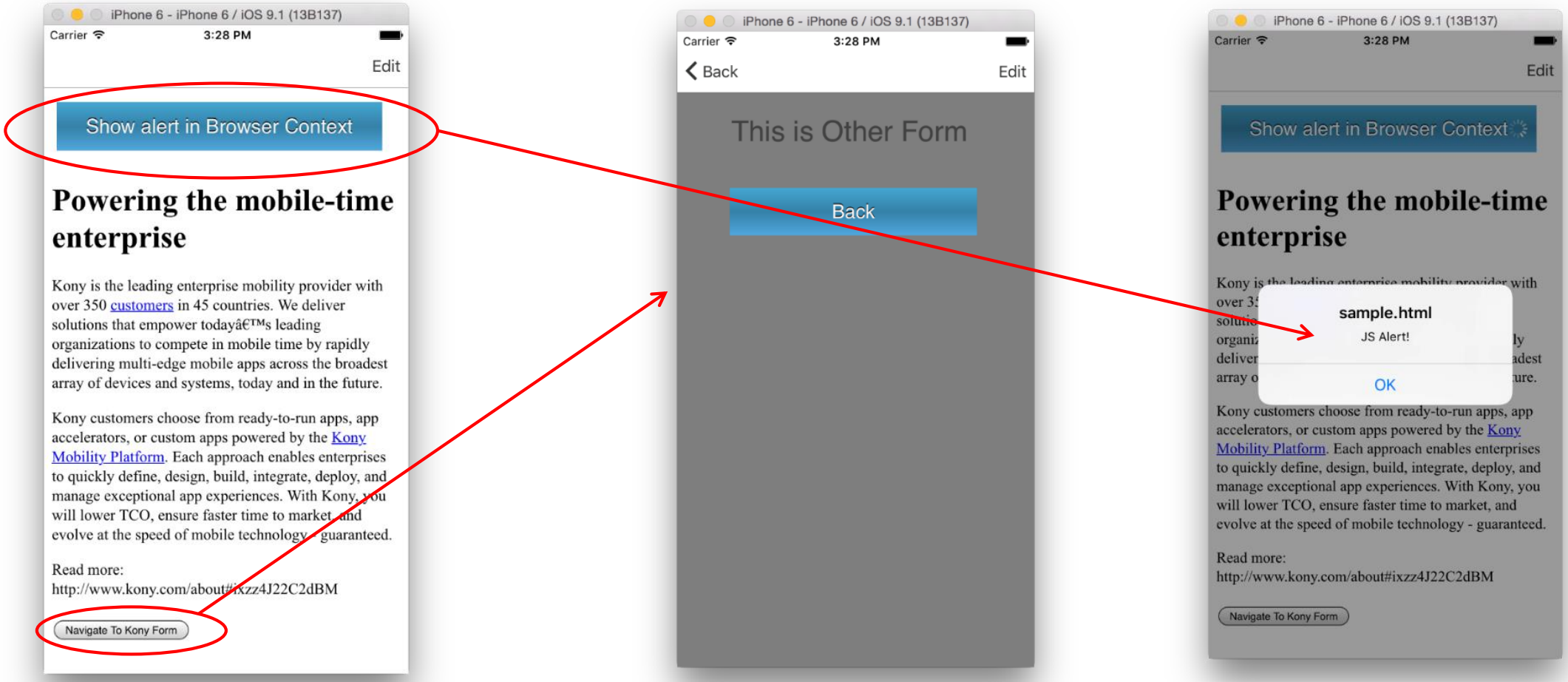
Browser Context API

- ***evaluateJavaScript(String javascript):***
  - The above function is used to execute the JavaScript code in the ***javaScript*** parameter in Browser context
- **Example Usage:**
  - *frmNewBrowser.browserNew.evaluateJavaScript('alert("JS Alert!")');*

Exercise

- Lets create a sample exercise to load Browser widget using the local html file
- Lets make use of native communication API to navigate to Kony form from the html file loaded in Browser Widget
- Lets also execute the JavaScript code in the Browser context on clicking on a Button

Exercise (cont'd)

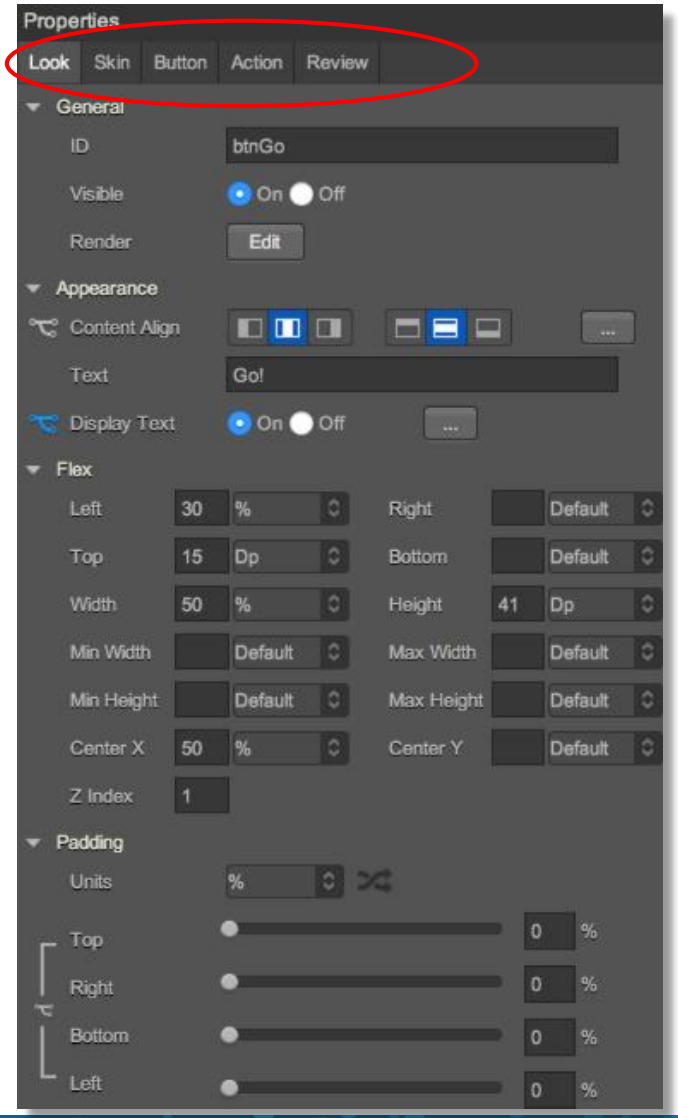


Dynamic Widgets - Overview

- We can show/hide widgets using setVisibility, but there are times when you'll want to programmatically add widgets to your form
- This is called Dynamic Widgets
- Use Cases:
  - You want to create a dynamic input form
    - Based on the type of user, you have different pieces of data to collect
  - In your booking app, you specify the number of guests
    - For each guest there needs to be a section to enter name, age, etc.,
  - You have a screen that is ALWAYS changing, so you create a web service that returns the data AND the info needed to dynamically create the whole screen, form and widgets to present the data to the user
    - The user never needs to re-download the app just because you have to change that screen AGAIN

Dynamic Widgets - Overview

- The API provides for creating forms, containers and widgets as well as ADDING widgets to an existing form
- When creating a widget (form included), you need the following information (button example):
  - Basic Configuration - Properties under the General in all the tabs (Look, Skin, Action etc.,) and also flex properties (top, left etc.,)
  - Layout Configuration - Properties under the Appearance and Padding sections in Look tab
  - Platform Specific Configuration - properties for the different devices/platforms/channels in all the tabs
- Anything not specified gets the default value(s)



Dynamic Widgets - API

- To create an instance of a widget (forms and containers included) here is the general syntax:

  var sampleWidget = new kony.ui.widgetName(basicConf, layoutConf, pspConf)

  - where:
    - **widgetName** - Is the name of the widget you want
    - **basicConf** - Is the object containing all the basic configuration information
    - **layoutConf** - Is the object containing all the layout configuration information
    - **pspConf** - Is the object containing all the platform specific configuration information

Dynamic Widgets - API (cont'd)

- After creating a widget, how do you add it to your form?
- For ANY container widget (**flexcontainer, flexscrollcontainer, flex form, normal form** etc.), there are 2 methods:
  - **add(widgets) -** Is used to add the widgets to the container (at the end of the container) where:
    - widgets - is a comma separated list of widgets created using the API we just saw
    - For example - to add 2 buttons (assume newBtn1 and newBtn2 already defined) to the end of the form: For free Form project this.view.add(newBtn1, newBtn2) / In case of MVC project this.view.add(newBtn1, newBtn2)
  - **addAt(widget,index)** - Is used to add the widget to the container at the specified index where:
    - widget - is the widget created using the API
    - index - a 0 based number indicating, within, container, the widget should be added
    - For example, to add a button (assume newButton already defined) as the first widget on the form: For Free form project this.view.addAt(newButton,0) / In case of MVC project this.view.addAt(newBtn1, newBtn2)

Creating Forms Dynamically

- Creating forms is no different than creating widgets - all the same rules apply
- There is one additional event, however that you can take advantage of if needed - the **addWidgets** event
  - **addWidgets** is fired when a form is first instantiated (i.e., after we create it with the new keyword)
  - You can reference another function that can be used to populate the form with widgets if so desired
    - **Note:** you can always do it all inline in a module too - choice is yours
  - Here's an example of how we'd specify that event function call:

```
var frmBasic = {id:"myForm", "layoutType":kony.flex.FREE_FORM, addWidgets:createFormWidgets};
var frmLayout ={};
var frmPSP ={};
var frm = new kony.ui.Form2(frmBasic, frmLayout, frmPSP);
```

  - You would then create the function **createFormWidgets** that would create the widgets and add them to the form

Simplified API

- There is a simpler new way to dynamically create widgets using a simplified API
- Using the simplified API you can just create a new instance of a widget and pass 0, 1 or more properties into the constructor to set those initial values
  - What will all the other values be? they'll be the default values for each widget
- For example, to create a new button:

  var myBtn = new kony.ui.Button();
  - This creates a button with the default skin and other values
  - The ID is generated automatically for you
- You can set any initial values you want by passing them into the constructor as a JSON object:

  var myBtn = new kony.ui.Button({id:"myBtn1",text:"click me!"});
  - This creates a new button, sets it's text and widget ID and leaves the rest as default
- In either example, you'd then add that button to the form/container by using that widget's add method

  For example: this.view.add(myBtn);

Removing Widgets Dynamically

- Just like you can dynamically add widgets, you can remove them too!
  - **remove(widget)** - is used to remove the widget from container where
    - widget - is the widget to be removed - note: this is the widget object not the widget ID string
    - For example: myForm.remove(frmDynamicWidget.btn1);
    - For Free form project myForm.remove(myForm. btn1) / In case of MVC project this.view.remove(this.view.btn1) s
  - **removeAt(index)**- is used to remove the widget from the container at the given index where:
    - index - a 0 based number indicating where, within, the container the widget should be removed
    - For free Form project myForm.removeAt(0) / In case of MVC project this.view.removeAt(0)

Replacing Widgets Dynamically

- You can also replace a widget with another widget
  - replaceAt(widget,index) - is used to remove the widget at the specified index and replace it with the widget where
    - widget - is the widget to be added once the old one is removed - note: this is the widget object not the widget ID string
    - index - is the 0 based index of the widget to remove - the new widget will be inserted in this same index
    - For example: this.view.**replaceAt**(newWidgetDef,4);
      - In this case, whatever widget was the 5th widget on the form will be replaced with the widget specified as newWidgetDef

        Note: this method is not applicable to FlexContainer and FlexScrollContainer
- As an example, maybe you have a form that is read-only summary information but you don't have all the data
  - You can dynamically go to the labels that have no data and replace them with an FlexContainer, label and textbox asking the user to supply that data

Animations

- The **addAt, removeAt,** and **replaceAt** methods also allow for an optional parameter that configures the animation to be used when adding or removing a widget.
- For example, to animate a widget appearing on the screen, the method signature is now:

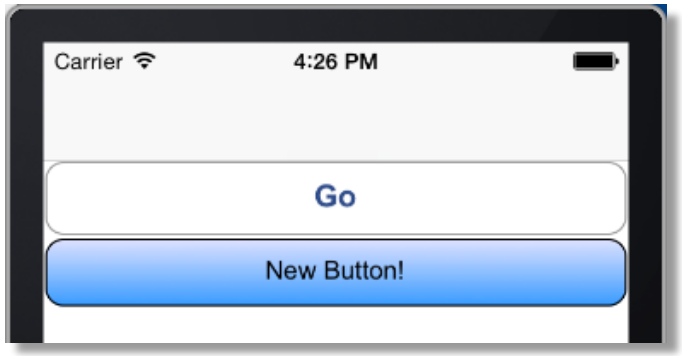
addAt(widget,index,animationConfig) where
animationConfig is an object that specifies the following data:
animEffect - can be set to one of the following constants:
ANIMATION_EFFECT_EXPAND - to increase the height from 0 to the final height
ANIMATION_EFFECT_REVEAL - to increase transparency till it's fully shown
ANIMATION_EFFECT_NONE - means it will just appear or disappear with no special effects
animDuration - a number in seconds for how long the animation should last
animDelay - a number in seconds to delay the start of the animation

animCurve - can be set to one of the following constants:
ANIMATION_CURVE_EASEIN - start slow and then speed up animation
ANIMATION_CURVE_EASEOUT - start fast and then slow down animation
ANIMATION_CURVE_EASEINOUT - start slow, speed up and then finish slow
ANIMATION_CURVE_LINEAR - steady animation speed
animCallBacks - an object where you can specify callbacks for the following 2 events:
animStarted - specifies the function to be called once the animation starts
animEnded - specifies the function to be called when the animation is complete

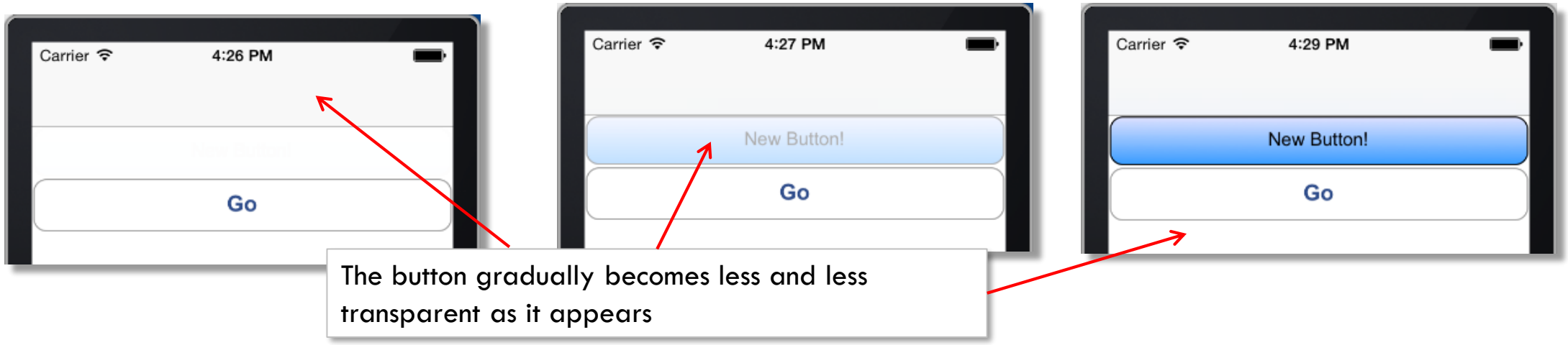
All of these animation configuration settings are optional - set only what you need; Let's look at an animation example on the next slide...

Animations (cont'd)

- Here's an example app I'll use to show the animation in action:



- When I click the Go button, it will remove the blue button and re-add it as the first button on the form using a Reveal animation:



Animations (cont'd)

- Here is the code in the button click event to trigger the animation:

```
//let's get our button saved in memory first
var tempWid = this.view.btnBlue;
//now we can remove the button from the form
this.view.removeAt(1);
//configure the animation configuration object
var animationConfig = {animEffect:constants.ANIMATION_EFFECT_REVEAL, animDuration:10, animDelay:1,
animCurve:constants.ANIMATION_CURVE_EASEOUT};
//and use the animation to re-add the button back to our form
this.view.addAt(tempWid, 0,animationConfig);
```

Considerations

- It is required that no 2 widgets have the same ID
  - You'll need to programmatically assign unique IDs - typical to use a counter like we did in our example to generate unique IDs
- If a widget already exists (via the Visualizer or already created/added widgets), you can't just replace the definition and expect it to work.
  - Here are 2 examples of things that will NOT work to change a widget:
    - Example trying to illegally redefine a widget:

      ```
      this.view.label1 = new kony.ui.label(...);
      ```

    - Example trying to illegally remove a widget:

      ```
      this.view.label1 = null;
      ```

  - The proper way to redefine a widget is to remove it and THEN re-create it and add it
    - There are some properties that are read-only for existing widgets, this is the only way to change those properties programmatically

Considerations (cont'd)

- Some of your created widgets may have events that need to be handled - how do you configure that?
- As one of the basic configuration properties you specify the event and the event handler function - here's an example:

```
basicProp = {id:"newBtn, text:"button1", onClick:btnClickHandler};

newBtn = new kony.ui.Button(basicProp, {}, {});
```

- Ok, that's great, but what if I created 7 buttons dynamically?
  - Whenever you specify a callback in code like this, you will automatically get passed the **eventobject** - the widget that raised the event
  - Make sure your handler accepts one parameter - for example:

```
function btnClickHandler(button){
        kony.print(button.text + " was just clicked!");
        }
```

  - You now know which button trigged the callback

Considerations (cont'd)

- With Dynamic widgets, the screen will have a number of widgets that are added programmatically
- We saw how to know what widget was used by the event raised, but many times you'll want to access these widgets programmatically
  - The form object contains objects for each widget that is accessible with:

    <form>["<widget ID>"] that returns the widget object
  - You can now retrieve the text property, selectedIndex, etc - whatever you need from that widget

Considerations (cont'd)

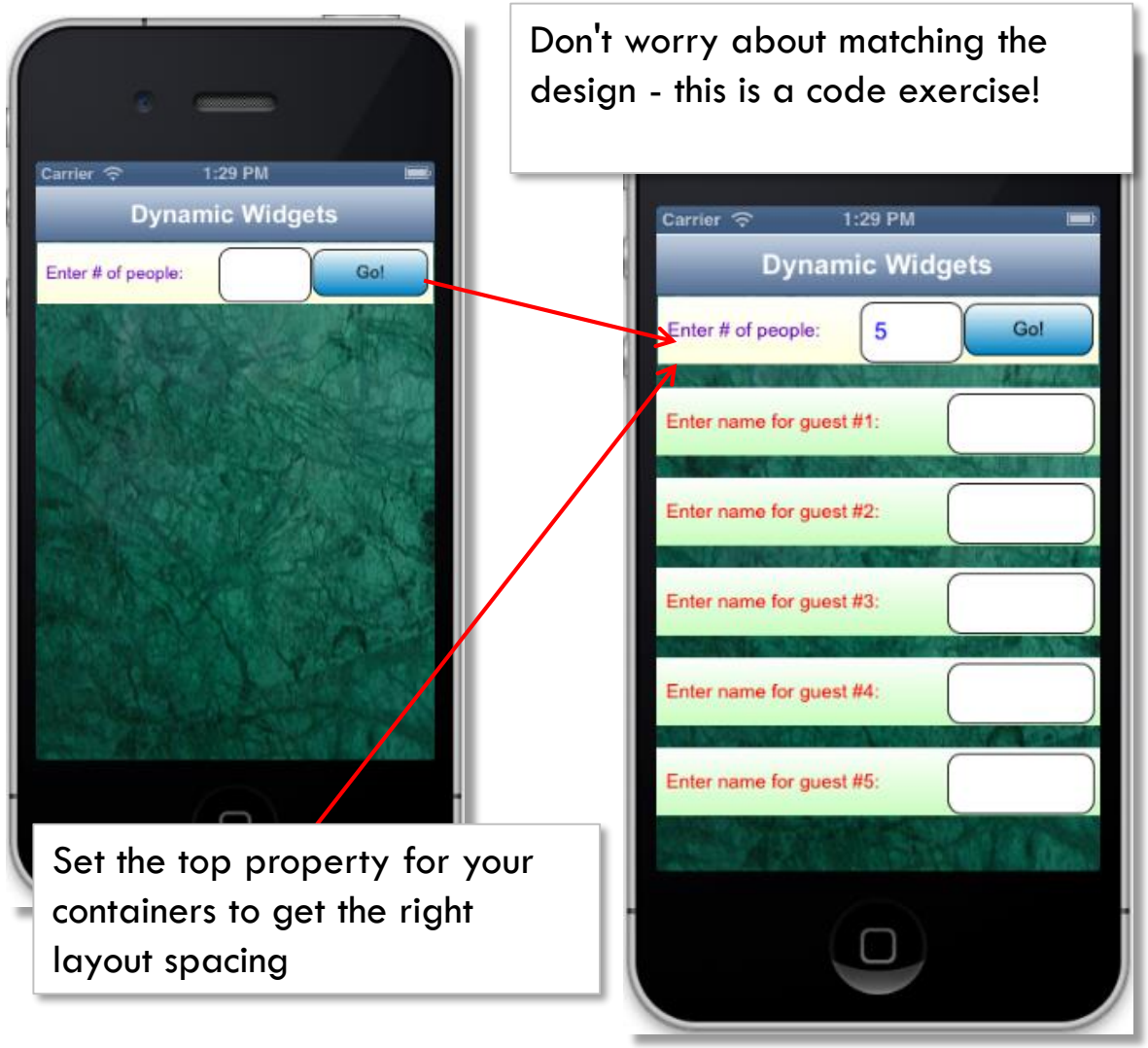
- For example, in my form, I dynamically create a bunch of textboxes ("txt0", "txt1"…) and I want to read all the values:

```
For (var i=0;i<numWidgets;i++){
        kony.print(this.view.["txt" + i].text);
}
```

- Note: EVERY widget also has an “info” property (part of basic configuration)
  - “info” can be set to anything you want
  - Using this may help with programmatic access to your dynamic widgets

Dynamic Widgets - Exercise

- Let's try it:
- Build the app shown here:
  - Type in a number
  - Click Go! to generate the entry fields for that many guests
- What happens when you click the Go! button *again*?
  - To properly handle this, you'll need to:
    - keep track of the containers you added the first time
    - When Go! is clicked, remove all those containers (removes children widgets too) then add your new containers w/widgets



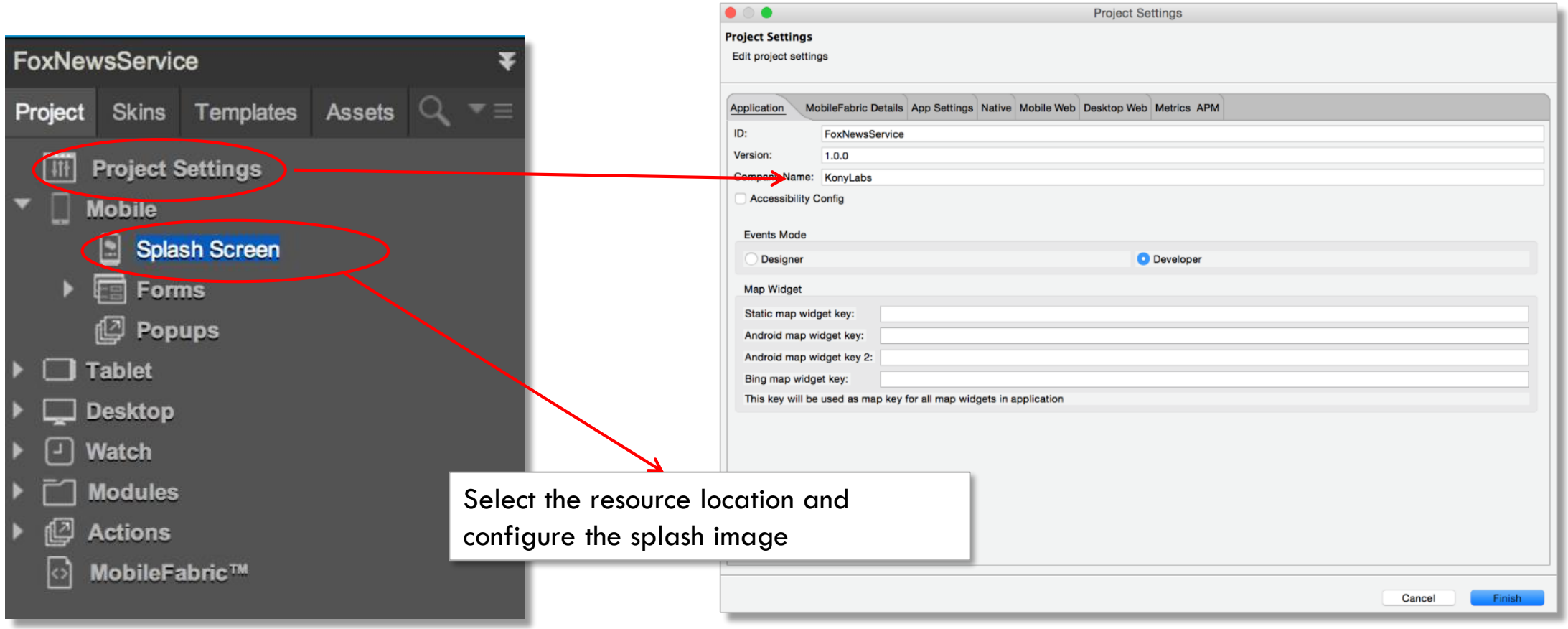
Other Application APIs

Application Properties

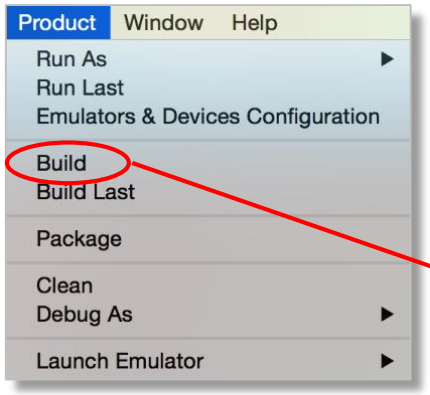
- For every application, there are a lot of project level properties that determine how the app will function
- We have already seen a few examples of these:
  - Setting up the Kony Fabric information for our services
  - Choosing the build type - Debug or Release
  - Setting device-level permissions for Android apps
- We will also use a few more properties throughout this training
- We will go through a few more application level properties but we will NOT cover them all

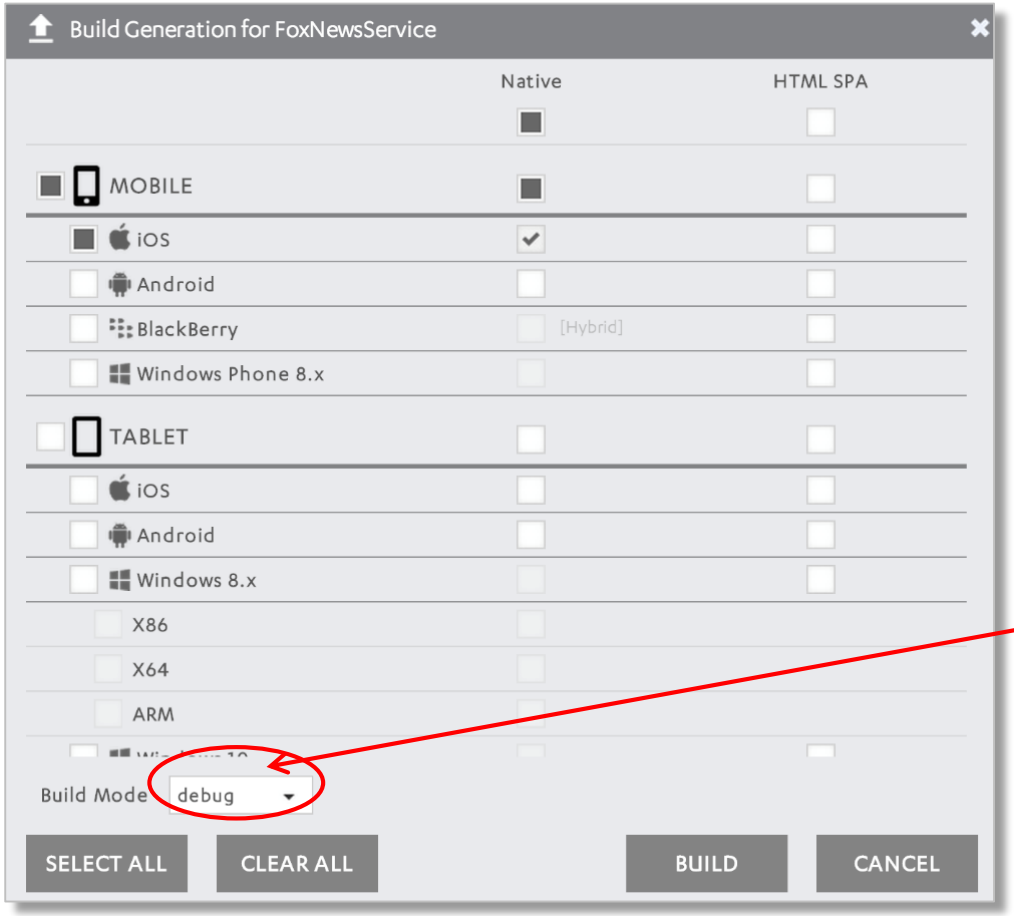
Application Properties (cont'd)

- To bring up the property information, right-click the application for which you want to set properties and select properties



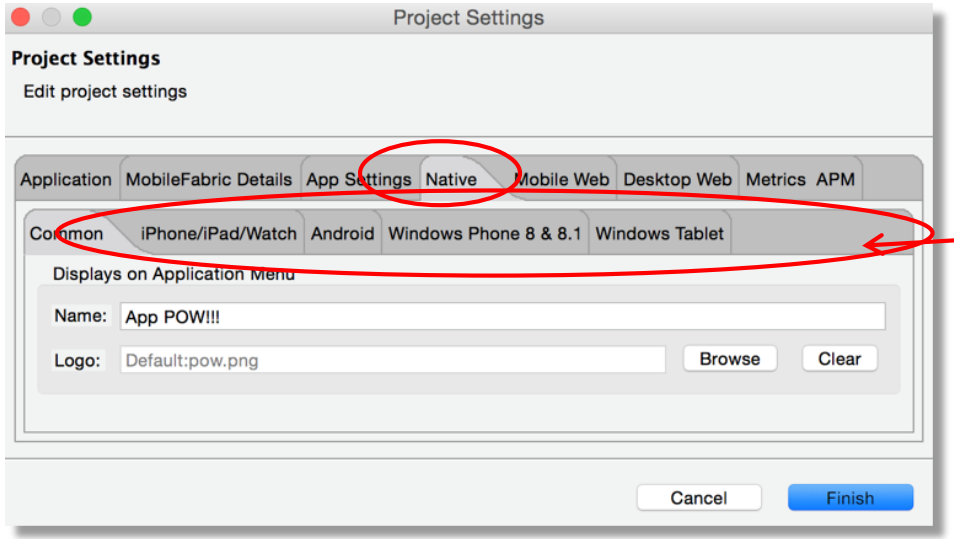
Build Mode





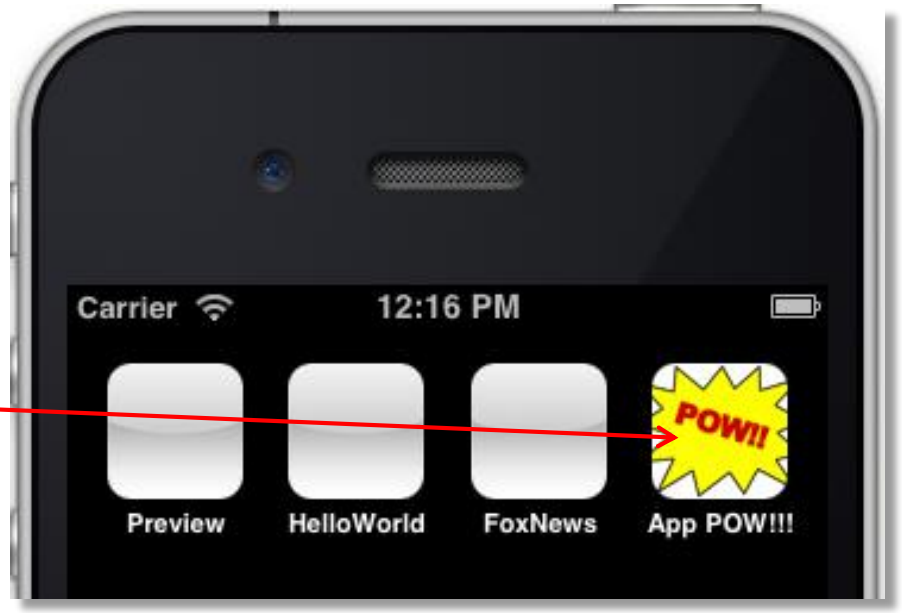
Change between debug and release

Native App - Common settings



The device specific tabs have a LOT of settings in them
These settings are generally about how the application should behave or look that is 100% unique to that device
If developing for any of these, be sure to check out each option thoroughly

Setting the Name and Logo will determine how your app looks on the user's device



Application life cycle event hooks

Application Life Cycle Events - Overview

- There are 4 hooks for the application life cycle events:
  - pre-appinit
    - This event is fired after the splash screen, if any, is launched
  - post- appinit
    - This event is fired once the application has completed initializing
  - App Service
    - This event fires when a 3rd party app or website calls your application
    - Your job is to write code to return the proper form to show at startup (if you want to override the default)
  - Deeplink
    - This event fires when a Mobile Web app is called for the first time
    - Your job is to write code to return the proper form to show at startup (if you want to override the default)
      - Note: App Service is the new Deeplink and works for all channels. If you ONLY want to link into your app from the web, configure ONLY Deeplink

Application Life Cycle Events - pre-appinit

- In pre-appinit you have no access to things like: global variables, data store keys, skins, forms or widgets.
- pre-appinit is a good time to:
  - Build dynamic app menus (the only place it can be done)
  - Call services to retrieve necessary localization datasets
  - Setting callbacks for background and foreground events
    - kony.application.setApplicationCallbacks(eventCallBacks) where eventCallBacks is an object used for identifying the specific event and the callback for that event
      - For example, to print a message when the iPhone is put in the background:
      - kony.application.setApplicationCallbacks({oninactive:doSleep}) where doSleep is a function that prints the message
    - Please refer to the Kony API documentation for details about all the keys you can use to register events

Application Life Cycle Events - post-appinit

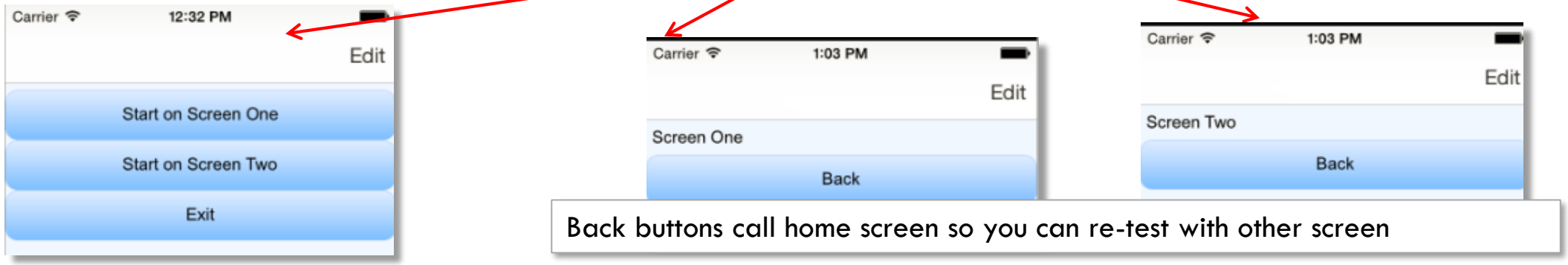
- In post-appinit you now have access to: global variables, data store keys, skins, forms and widgets on those forms.
- post-appinit is a good time to:
  - To invoke service calls to retrieve data and display it on the startup screen
  - Display an interstitial screen - note that the startup form will show over it when the startup form is initialized (destroy the interstitial screen when done with it)
  - Dynamically set the startup form
    - If the function returns a form, that form will be displayed instead of the default startup form

For example:

```
var userPref = kony.store.getItem("userPreferenceStartForm")
if(userPref == "payments") {
      return frmPayments;
} else { return frmServices; } }
```

Exercise - Application Life Cycle Events

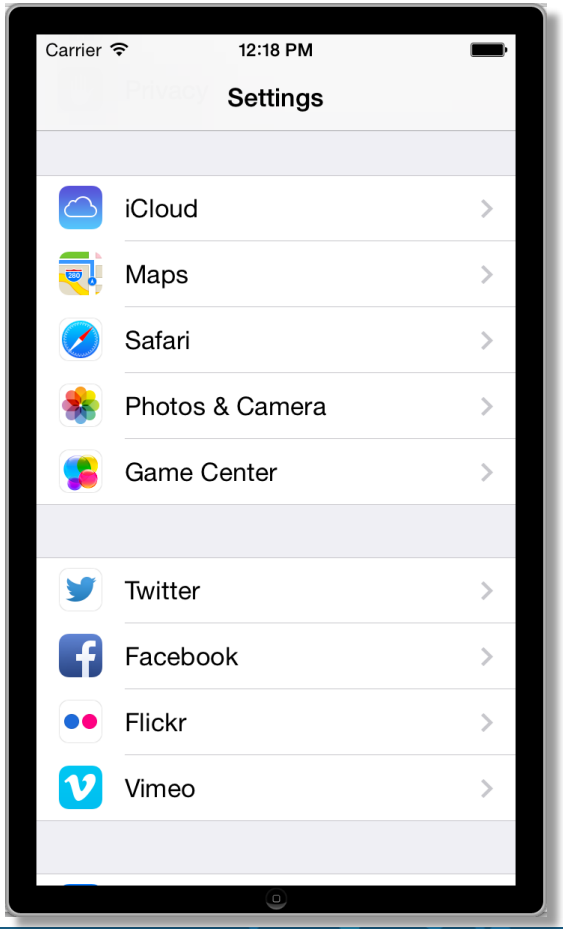
- Create a 3 screen app with forms: "home", "screen1" and "screen2“



- On the "home" screen, have 3 buttons that are labeled "Start on Screen One“, "Start on Screen Two“ & “Exit”.
- Clicking either button will save, to the device, the name of the corresponding form
- Upon application startup, read the value and do the following:
  - If the value is null (i.e., the first pass), start with the home screen
  - If the value is "screen1" show screen One on startup
  - If the value is "screen 2" show screen Two on startup
- Use **kony.application.exit()** for the Exit button to kill the app
- Register for the inactive event and print something - test by putting app in background

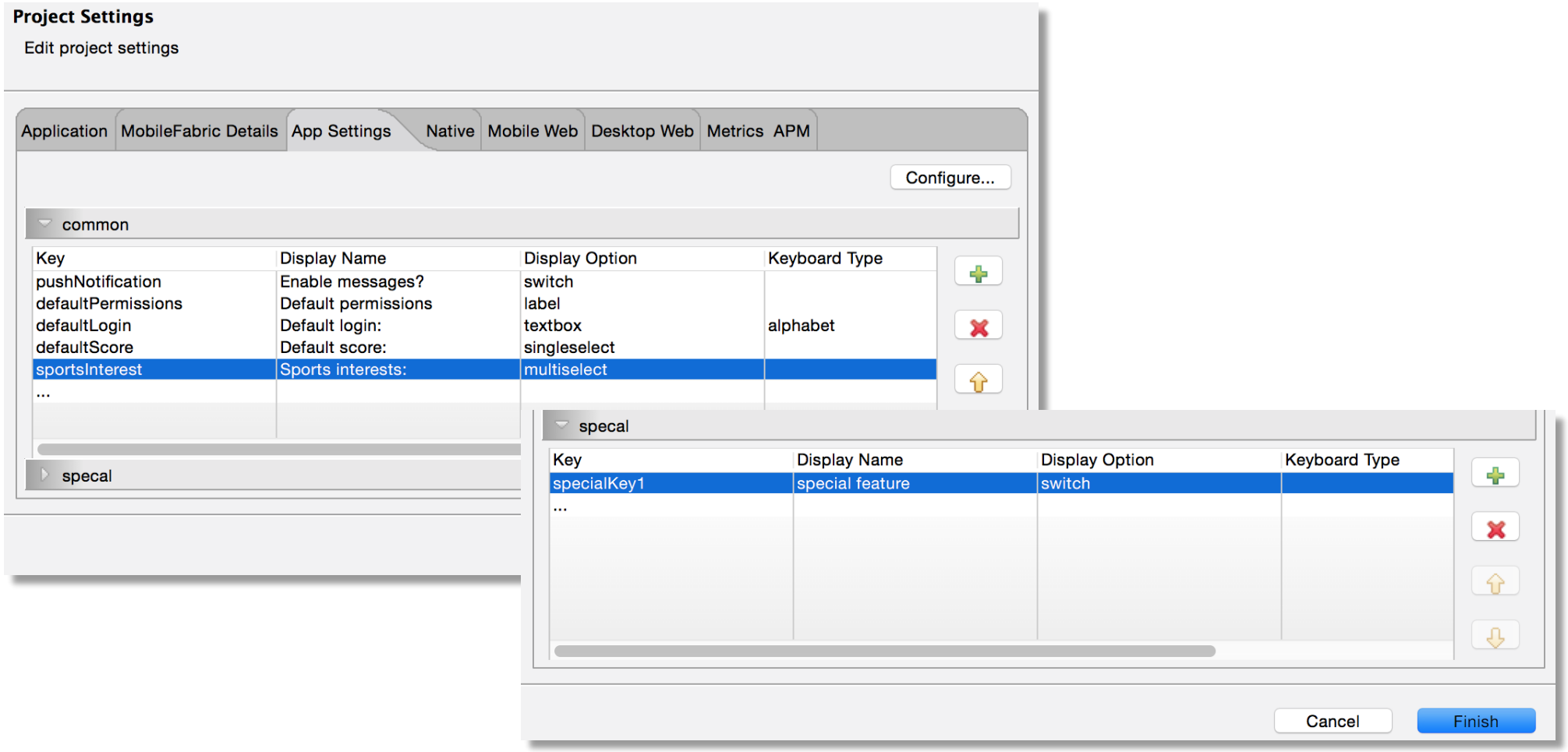
App Settings - iOS

- This feature allows the developer to enable application level settings so that the end users of an application can modify configurations and change the application behaviour
  - Only available on iOS
- Some example use cases are:
  - Enable/Disable push notification of the application
  - Define default first page in the app
  - Set other app options on/off
- The user sets these values in the application settings area:
  - The user will have to leave the app to set these values
  - You'll have to read the settings upon return to the app



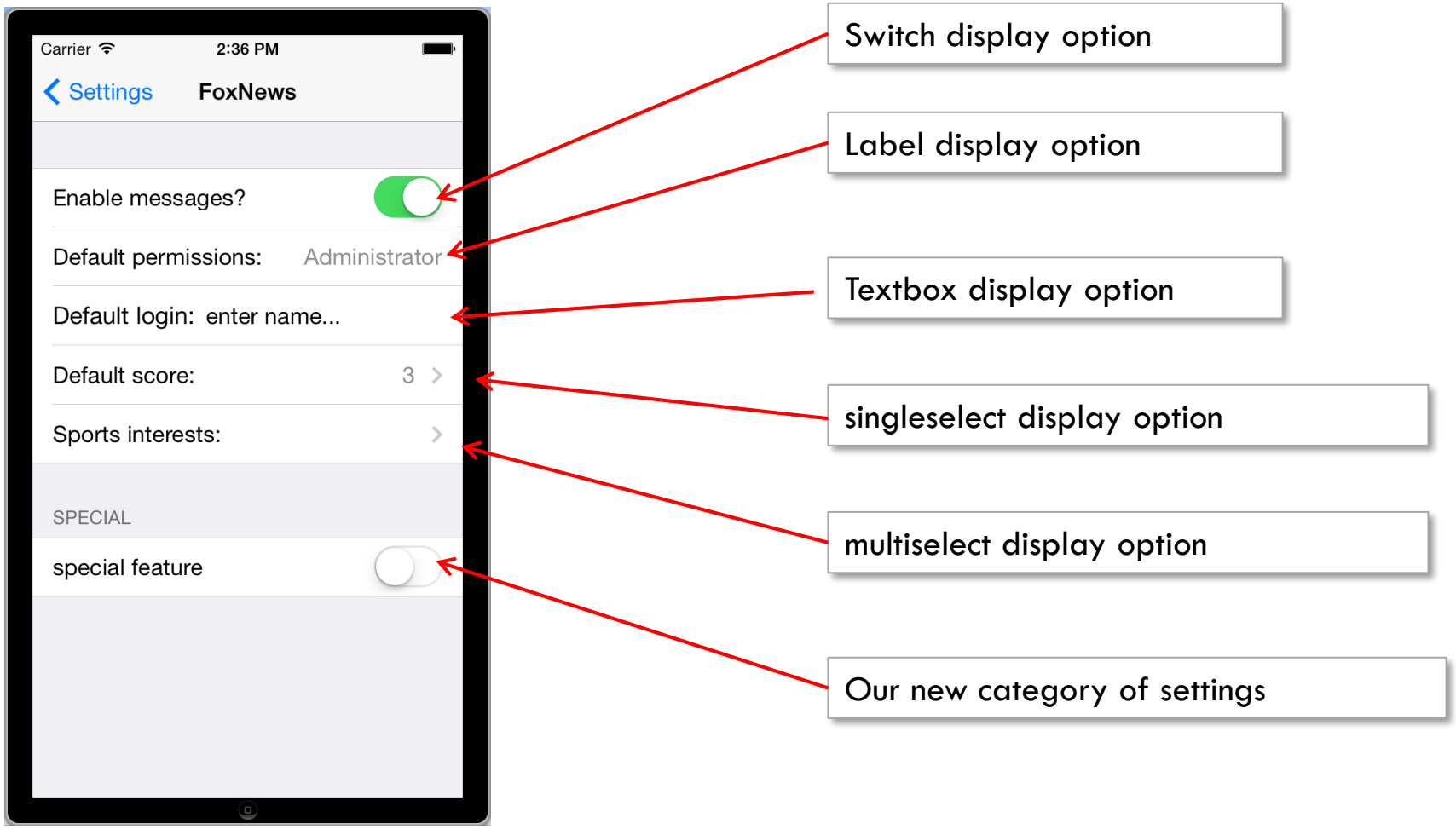
App Settings in Properties

- App settings are configured in the App Settings tab:



App Settings Example

- Let's take a look at an example (from the previous slide):



App Settings - Reading Settings

- kony.application.settings.read(key, successcallback, failcallback) - is used to read one of the applications settings where:
  - key is the key of the setting you want the value for
  - successcallback is a function called when the read is successful - it has the following signature:
  - successcallback(key,value) where:
    - key is the key you specified in the read call
    - value is the value of the setting
  - failcallback is a function called when the read is unsuccessful - it has the following signature:
  - failcallback(code,message) where:
    - code is the returned error code
    - message is the returned error message

App Settings - Writing Settings

- **kony.application.settings.write(key, value, successcallback, failcallback)** - is used to set one of the applications settings where:
  - **key** is the key name you're setting a value for
  - **value** is the value - needs to be of the right type for the display option
  - **successcallback** is a function called when the read is successful - it has the following signature:
    - **successcallback(key,value)** where:
      - **key** is the key you specified - it's just repeating it back to you
      - **value** is the value of the setting - it's just repeating it back to you
  - **failcallback** is a function called when the read is unsuccessful - it has the following signature:
    - **failcallback(code,message)** where:
      - **code** is the returned error code
      - **message** is the returned error message

App Settings - Read & Write Example

- Consider this example:

```
function testSettings(){
    kony.application.settings.read("sportsInterest", successCallback, failureCallback);
    kony.application.settings.write("defaultPermissions", "user", successCallback, failureCallback);
}
function successCallback(key,value){
    kony.print("key: " + key + " &&& value: " + JSON.stringify(value));
}
function failureCallback(code,message){
    kony.print("error code: " + code + " &&& message: " + message);
}
```

sportsInterest is configured as a multi-select - we'll expect an array of values back

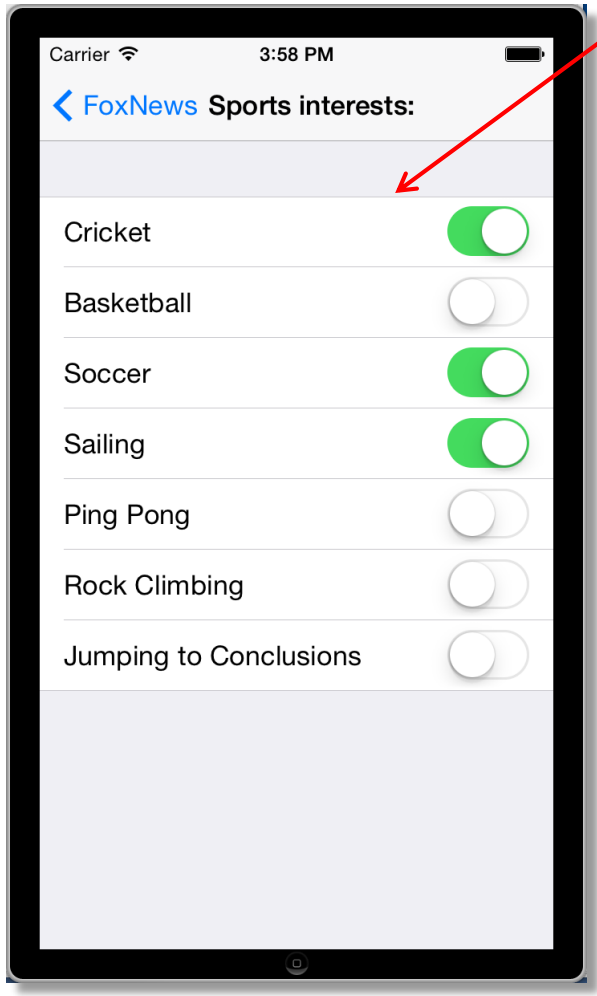
defaultPermissions is configured as label so we'll pass a string value

- Let's see what happens when we run this...

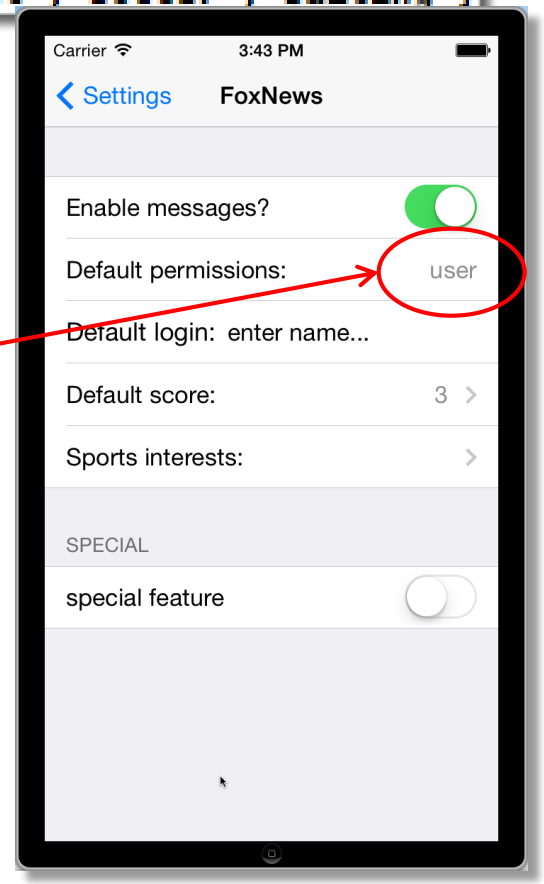
App Settings - Read Example

With these values set, here's what we see in the logs:

```
key: sportsInterest &&& value: ["Cricket","Soccer","Sailing"]
```



When the write is performed here is our new settings value shown:



App Settings - Considerations

- To change settings, the user will send the app to the background - best to re-read values every time the app gets focus
- For example:

```
var callbackFunctions = {onactive:activeEvent, oninactive:inactiveEvent,
onbackground:backGroundEvent, onforeground:readSettingsChanges,
onappterminate:appTerminateevent};
kony.application.setApplicationCallbacks(callbackFunctions);
function readSettingsChanges(){
        kony.application.settings.read("Notifications", successcallback, failurecallback)
}
```

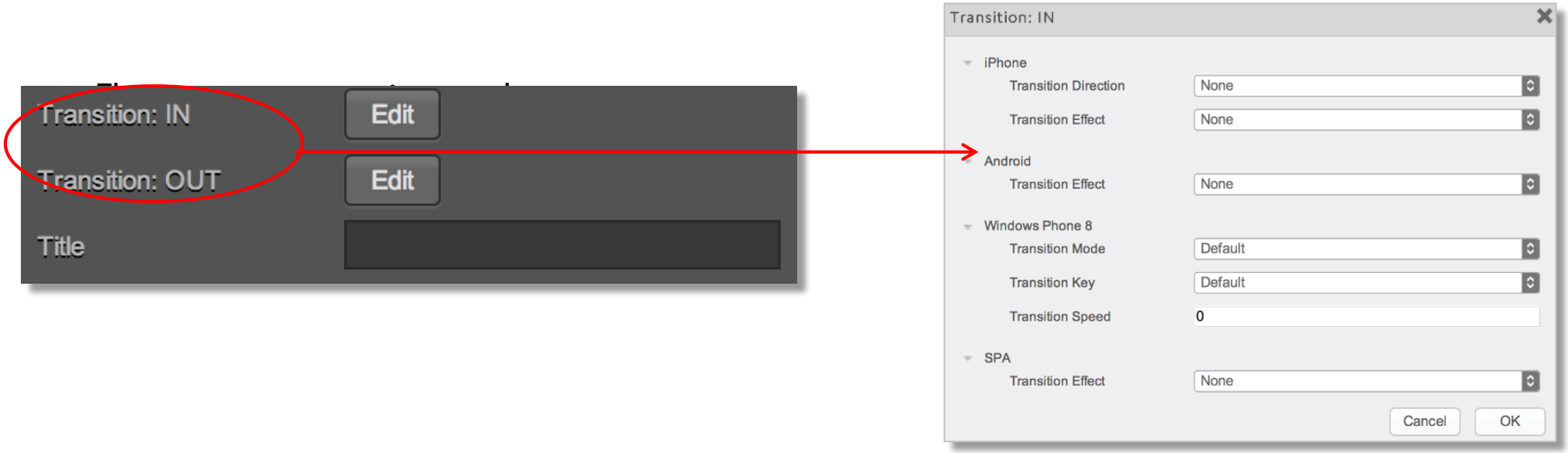
- In the **successcallback** function, you'd implement your settings changes in your app

Form - Advanced

- For the most part, we've used the form as just a container for our widgets
- We'll now look at the various things we can configure for the form, the different features of the form and the different way they are used
  - There are a lot of platform specific properties on the form
  - We'll only scratch the surface so please refer the Widget User Guide for all the details on all the properties

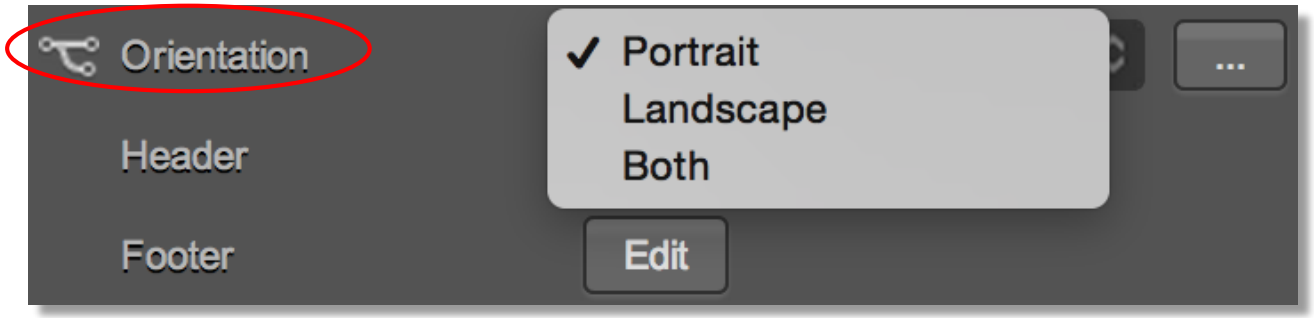
Form - Transition Property Security Level

- As you navigate in an app, you can specify how the various forms are drawn - we call these transitions
- There are 2 properties:
  - Transition: IN - specified how the form will transition IN to view
  - Transition: OUT - specifies how the form will transition OUT of view
  - Note: both have the same configuration options. One is shown below



Form - Orientation

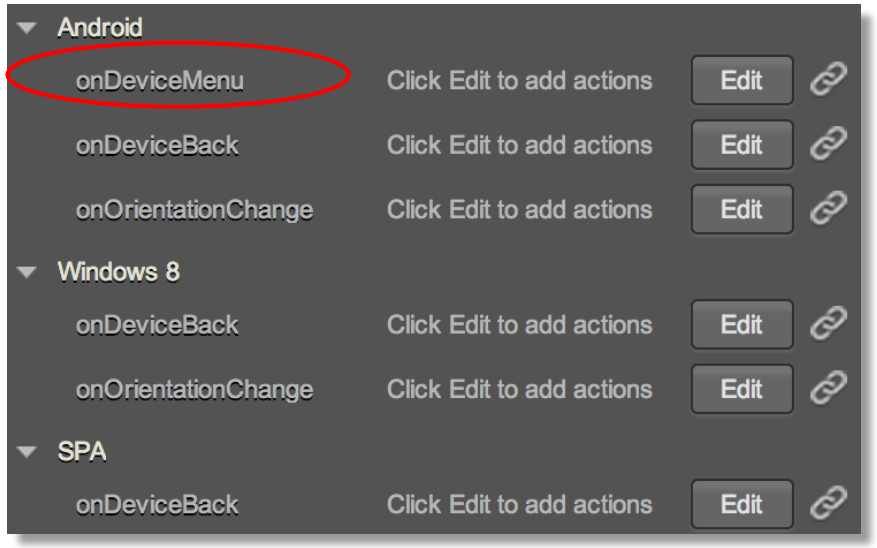
- **Orientation** is used to specify what orientation the form should be displayed in
- You can specify Portrait, Landscape or Both on a device by device basis:



- Setting to **Both** means the user can rotate the display and the form will change orientation accordingly. The other values fix the display to a certain orientation
- **onOrientationChange** is an event triggered when there is a change from portrait to landscape or vice versa. The JS function used for handling this event will be passed in the new orientation value: 0 for Portrait and 1 for Landscape.
- The Kony app will adjust the display for you and typically there is nothing to be done
- IF you want, you can display an entirely different form if you want the user experience to differ in different orientations

Form - Device Back Event

- On many devices, there is a Back button on the device itself. You can control if/what happens when that button is used for each form with the onDeviceBack property:



- If you don't assign an action, the back button retains it's native functionality
  - Note: attaching a function that does nothing, will override the normal functionality and now the back button is effectively disabled on this form
- Note: you can set the event handler in code as follows:

  this.view.onDeviceBack=<function>

Form - get form methods

- There are methods that allow you to access the form "stack"

- There are 2 methods for accessing forms:
  - In free form project, kony.application.getCurrentForm() will return the current form as a form object (i.e., not the form's ID). But in MVC, it returns current form ID.

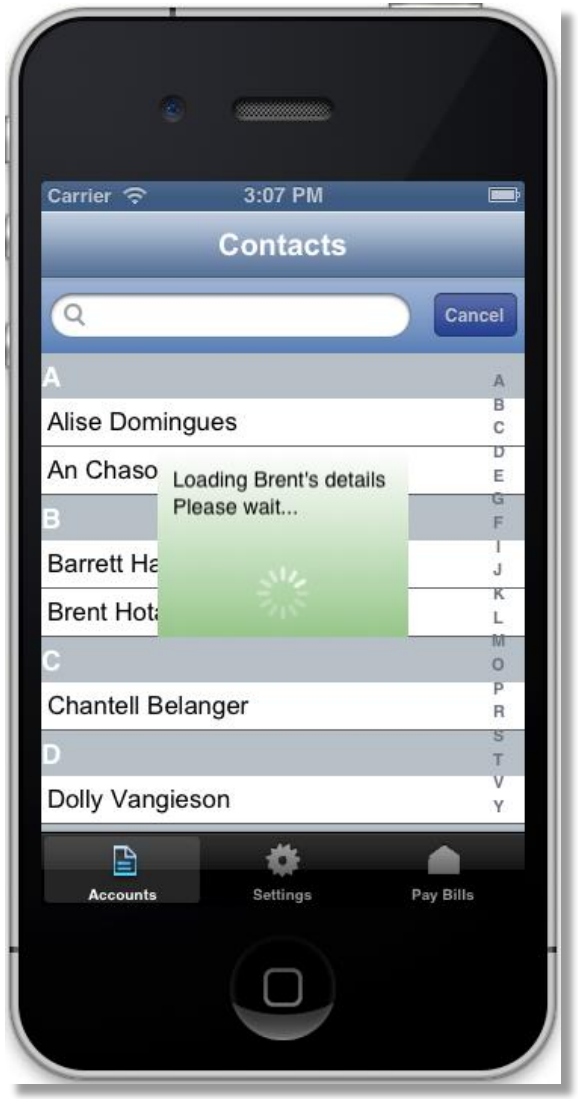
  - In free form project, kony.application.getPreviousForm() will return the previous form as an object (i.e., the form you would go back to). But in MVC, it returns previous form ID.

Interstitial Screens - Overview

- An Interstitial screen is just a form that is displayed to the user while something else is going on in the background
  - For example:
    - While the user is logging in (the service call is executing) display an interstitial screen with a big ad for some product
    - Once the login process completes, dismiss/destroy the screen with the ad
    - If the user clicks on the ad, take the user to that product upon login
- If you know you'll have a long wait for a process, best to display something to the user that is more interesting than the default spinner
  - Even distracting the user's attention for 2 seconds with an interesting Interstitial screen makes the wait SEEM much less
  - If displayed while a service is running, consider a "cancel" button to cancel the service (for example, "whoops, didn't mean to kick off THAT search…oh no!")
- If the Interstitial screen may be "too much", consider a Progress screen…

Progress Screens - Overview

- A Progress screen is a much less customizable, simple "please wait" type of display
  - You cannot put widgets on the Progress screen
- Progress screens should be the minimum UI requirement for any transaction that makes the user wait
  - They allow you to at least tell the user what is going on with a simple message
  - Can optionally show the progress indicator
  - Can be skinned
  - The layout is very simplistic - the example to the right has an "\n" to create a line break
    - This is about as fancy as you can get



Progress Screen - Showing

- You can only show progress screens in code:
- **kony.application.showLoadingScreen(skin, text, position, isBlocked, showProgressIndicator, properties)** is used to display the progress screen where:
  - **skin** is the skin to use. Any skin that specifies background and font will work
    - Note: use an image (it'll display actual size) on form skin if you want an image
  - **text** is the displayed text. Note: use "\n" to create a line break
  - **position** indicates the position of the screen. Supported values are **LOADING_SCREEN_POSITION_FULL_SCREEN** or **_POSITION_ONLY_CENTER**
  - **isBlocked** indicates if the UI should be blocked. This is typical
  - **showProgressIndicator** is a Boolean value to indicate if the progress indicator should be displayed
  - properties - is an object with the following keys:
    - **enableMenuKey** - a Boolean to indicate if the device menu key should be enabled
    - **enableBackKey** - a Boolean to indicate if the device back key should be enabled

Progress Screen - Dismissing

- **kony.application.dismissLoadingScreen** is used to dismiss the Progress screen
  - This method is the only way to dismiss the Progress screen displayed using **kony.application.showloadingscreen** method
  - If there is no Progress screen displayed, calling this method does nothing - namely there is no harm in calling this
- Note: You MUST dismiss the Progress screen before calling any other form/popup
- Let's take a look at an example of how we'd use these 2 methods in an app…

Progress Screen - Example

- Here is an example of showing a Progress screen while loading contact details from a web service:
- Here is what the code would look like on the segment's onRowClick event:

```
function getContactDetails(){
        var name = //contact's first name from segment
        kony.application.showLoadingScreen(
       "skProgress",
       "Loading" + name + " details \nPlease wait...",
       constants.LOADING_SCREEN_POSITION_ONLY_CENTER,
       true, true,
       {enableMenuKey:false, enableBackKey:false})
        // make the service call
}
function contactDetailCallback(){// process the results
        kony.application.dismissLoadingScreen(); }
```

Idle Timeout - Overview

- It's possible that the user opens an application, logs in, navigates around a bit and then leaves the app unattended for some time
- In some cases, we might want to DO something if the user has been inactive too long
- For example:
  - Logout of an application - don't want sensitive info displayed forever
- Kony provides some methods which allow us to define the idle timeout
- When the timeout period expires, you can configure the code that gets executed
  - To enable these features, the form's Enable Idle Timeout property must be set to true
  - Kony provides two methods to handle the timeout events:
    - **kony.application.registerForIdleTimeout**
    - **kony.application.unregisterForIdleTimeout**

Idle Timeout - registerForIdleTimeout

- **kony.application.registerForIdleTimeout(timeoutValue, callback)** is used to enable the form's idle timeout after the specified period of inactivity where:
  - **timeoutValue** is the timeout value in minutes
  - **callback** is the function that must be executed after the timeout has occurred

- The call to **registerForIdleTimeout** can be done anywhere in your code
- ANY form that is enabled can trigger the timeout - your callback may or may not care which form called it
- Here is an example to show a login screen after logging out on a timeout in an application:

```
function callBack() {
        //business logic to display the login form.
}
```

Idle Timeout - unregisterForIdleTimeout

- **kony.application.unregisterForIdleTimeout()** is used to disable the idle timeout function
  - The call to **unregisterForIdleTimeout** can be done anywhere in your code
- Note: When a timeout occurs, the idle timeout is automatically unregistered
- Typical to see the re-registration for idle timeout in the timeout callback function to make sure it's always enabled
  - Make sure idle timeout is turned OFF for the default screen (usually the login) - since you expect the user to be sitting on this screen
- Please look at the Kony API guide for a good write up of using this feature

Gesture Overview

- Gestures are part of life in the mobile world
- The Kony API lets you create your own gesture events
  - Gestures are applicable only on mobile or tablet devices that have touch support
  - You can then do whatever you want in the event handler
- Gestures are not done at the widget level but rather at the container level
  - Form, Flex Container and Flex Scroll Container widgets can have gestures
- Here is the list of gestures that you can assign:
  - Tap - single or double
  - Swipe - you can specify how far they must swipe to trigger the event
  - Longpress - you can specify how long they must press to trigger the event
  - Pan - you can specify minimum number of fingers to trigger the event
  - Rotation - you can trigger continuous events when trying to rotate a widget
  - Pinch - to make something smaller or larger

addGestureRecognizer

- One method allows you to set a recognizer for all supported gestures - **addGestureRecognizer**
  - The **addGestureRecognizer** method is common to all widgets
  - Signature: **<widget>.addGestureRecognizer(gestureType, gestureConfigParams, onGestureClosure)**
    - **gestureType [number]** - specifies the type of gesture to be detected on the widget
    - **gestureConfigParams [object]** - specifies a JSON table with the required configuration parameters to setup a gesture recognizer, which vary based on the type of the gesture
    - **onGestureClosure [function]** - specifies the function (handler) to be executed when a gesture is recognized
- **Note: setGestureRecognizer** was used in previous versions of the Kony platform but has been deprecated and should not be used with any new apps

Gesture Handler Function

- The final parameter for **addGestureRecognizer** is **onGestureClosure [function]**, which specifies the function (handler) to be executed when a gesture is recognized
  - Create a the gesture handler function with the required signature and reference it from the **addGestureRecognizer** method
  - Sample code:

```
var tapID = myForm.myFlex.addGestureRecognizer(constants.GESTURE_TYPE_TAP, {fingers:1,taps:2}, tapHandler);
function tapHandler(mywidget, gestureInfo){ code to execute when gesture is recognized}
```

Gesture Handler Function Signature

- function onGestureClosure(widgetRef,gestureInfo,context)
  - widgetRef - specifies the widget on which the gesture was recognized
  - gestureInfo - a JSON table with information about the gesture
  - context (optional) - a table with segmentedUI row details
- The gestureInfo parameter provides comprehensive information about the gesture, which the developer can use in the function
  - gestureInfo provides the information about the gesture as an array of key-value pairs.
  - Sample code:

```
var tapID = myForm.myFlex.addGestureRecognizer(constants.GESTURE_TYPE_TAP, {fingers:1,taps:2}, tapHandler);
function tapHandler(mywidget, gestureInfo){
        If (gestureInfo.gestureType==1){
        //code to execute if gestureType is tap  }
```

gestureInfo Keys

- gestureInfo includes the following key-value pairs:
  - **gestureType [number]** - indicates the gesture type
    - 1 - tap, 2 - swipe, 3 - longpress,4 - pan, 5 - rotation, 6 - pinch, and 7 - righttap
  - **gesturePosition [number]** - indicates the location on the widget where the gesture was recognized
    - 1 - top left, 2 - top center, 3 - top right, 4 - middle left, 5 - middle center, 6 - middle right, 7 - bottom left, 8 - bottom center, 9 - bottom right, 10 - center
  - **swipeDirection [number]** -for swipe gestures only, it indicates the direction of the swipe
    - 1 - swipe left, 2 - swipe right, 3 - swipe up, 4 - swipe down.
    - Swipe direction is with regards to the view; not the device orientation

- **Note:** for a list of additional gestureInfo keys, please refer to Kony documentation

Sample Code

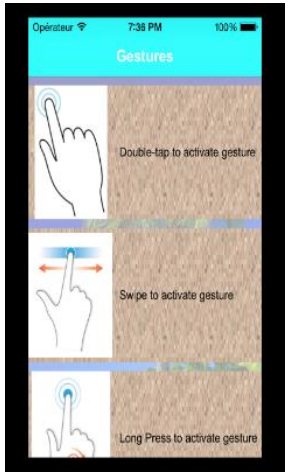
```
// gesture recognizer for the swipe gesture
        var tapID = myForm.myFlex.addGestureRecognizer(
        constants.GESTURE_TYPE_SWIPE, {fingers:1},swipeHandler);

// swipe handler function
function swipeHandler(mywidget, gestureInfo){
    If (gestureInfo.swipeDirection == 2){
        alert("Swipe to the right recognized");
        mywidget.setVisibility(false);
  }
}
```

Exercise - Gestures

Ok, let's try it!

- Create a form with 3 flex containers and whatever widgets you want inside (in the screenshot on the left added nice images of the gesture that works for that container)
- Set up gestures as follows:
  - First container - double tap (1 finger, 2 taps)
  - Second container - swipe (at least 100 pixels)
  - Third container - long press (at least 2 seconds)



- For each gesture, show an alert indicating that the gesture was handled
- For each gesture handler, print out the returned information to see what data you get back
- Note: need to run on Android 4 or higher…

Internationalization - Overview

- Internationalization is the process of designing or developing an application in such a way that it supports various languages
- For an application to support Internationalization, you do not need to make any major changes in the application code or logic
  - Note: in most of the IDE, you'll see Internationalization is abbreviated as I18n
- Benefits of an internationalized application:
  - The same application can run on multiple locales
  - Widgets' text is not hard-coded in the application. Instead localized keys are retrieved dynamically.
  - Support for new locales does not require recompilation
  - Region-dependent data such as dates and currencies, appear in formats that confirm to the end user's region and language

Internationalization - Resource bundle

- A resource bundle is a set (think JSON object) of key-value pairs where:
  - The key is a unique identifier given to a piece of text in the application
  - The value is the localized text
- Resource bundles need to be "available" for any language that you want your app to use
- If you want your app to run in English, Turkish, Japanese and Dutch…
  - You'll need resource bundles for each language where:
    - All the keys are the same
    - The values are unique to each language
- Typically resource bundles are downloaded from an external source - some content management system
  - Note: we can also set these up in the IDE for development - not recommended for production

Internationalization - How it works

- During development, each widget's text property's Text I18N key in the app is assigned an I18N key
- When a Kony app launches, one of the first things to happen is the device's locale settings are checked.
- The app then checks to see if it has a resource bundle for that locale
  - If there is a resource bundle then it uses that to display the localized text
  - If not, then it'll display the default specified
    - Note: the text values you hardcode for the widgets will never be used
- Languages have a core language and optional locale
  - For example there is "fr" for French, but there is also "fr_ca" for French Canadian
  - The system will always try to match the EXACT language and locale (ex: "fr_ca")
    - If that resource bundle is not available, it will try the language only bundle (ex: "fr")
      - Also, you get the default - whatever you chose in the IDE

Internationalization - Testing

- Testing I18N functionality means that you need to be able to switch the language settings on the device
  - You have to be careful! changing the device language also sets ALL the system text to that language
    - Try setting it to Japanese and then find your way back to the language settings…it can be challenging if you don't know the language
- Changing device locale settings typically will require your application to be restarted if you only check locale settings on startup
  - iOS emulators are easy to test since it's quick to deploy and restart the app
  - Android emulators take a bit more to kill the app and restart
    - Android device used for testing is quick to deploy but slow restart the app
  - Note: for testing, consider putting a button somewhere with kony.application.exit() in the onClick event to kill the running app

Internationalization - API

- While you can set up the localized strings in the IDE, this is not practical for production use
  - To change a single character, you'd need to redeploy the entire application
  - To add a new language, you'd need to redeploy the entire application
  - The app would be bloated with ALL the language resource bundles even if they were never used
- The solution is to manage Internationalization through the API in code
- This ASSUMES that there is some remote system that has all the resource bundles
  - Will need services connected to that system to retrieve the bundles as necessary
- The API lets you manage all aspects of Internationalization in your applications
  - Note: the one thing you CAN'T do with the API is change the device language settings - the user must do that
- Let's look at the API…

Internationalization - API (cont'd)

- The first few methods we'll examine allow us to know what is going on the device:
  - **kony.i18n.getCurrentDeviceLocale()** returns the locale that the device is set to as a JSON object. Example: {language:"en", country:"US", name:"English US"}
    - In our exercise this would have returned a language value "en", "es" or "fr" depending on which is loaded based on the device setting
  - **kony.i18n.getSupportedLocales()** returns a list of all the supported locales of the device as an array of those JSON objects:
    - Example when printing this out on the Android simulator (a LOT of data):

Internationalization - API (cont'd)

- **kony.i18n.isLocaleSupportedByDevice(locale)** - returns true if locale is in the list of the device's supported locales
  - Typically used to check before trying to programmatically apply a resource bundle for that locale
    - **kony.i18n.getCurrentLocale() -** returns the locale of the current *resource bundle* as a JSON object. For example:{language:"en", country:"US", name:"English US"}
- Here is a use case example for these methods:
  - In our app, we present the user with a list of languages to use in the app
  - Since every device may support different locales, we check with the **isLocaleSupportedByDevice()** method to make sure it's supported
  - If it is, check to see what locale is currently being used with the **getCurrentLocale()** method - no need to do anything if that locale is already being used
  - Finally, we can check what the device is set to with the **getCurrentDeviceLocale()** method to alert the user that the device is set to <whichever> locale and that switching to the new locale will make this app run in a different locale

Setting Locale

- The next 2 methods are used to set locale information for the application (not device):
  - **kony.i18n.setDefaultLocaleAsync(locale, onsuccess, onfail, info) -** is used to set the default locale to the specified locale
    - This is what we did in the IDE by choosing a default locale
  - **kony.i18n.setCurrentLocaleAsync(locale, onsuccess, onfail, info) -** is used to set the current locale
    - This causes the resource bundle associated with locale to be loaded
  - Both have the same signature where:
    - **locale -** is the locale string (ex: "en", "en_US", etc…)
    - **onsuccess -** is the function to call if the locale was successfully set
    - **Onfail -** is the function to call if the locale couldn't be set
    - **info -** is an object containing anything you want - it'll be passed to the callback functions above to help identify which call created the callback (for a common callback)

Setting Locale example

- Here is an example showing the set methods in sample code:

```
kony.i18n.setDefaultLocaleAsync("es",changeLocaleWorked,changeLocaleDidntWork,{”
type":"default"})   //to change the default locale to Spanish
kony.i18n.setCurrentLocaleAsync("fr",changeLocaleWorked,changeLocaleDidntWork,{”
type":"current"})   //to change the current locale to French
//callback if the change worked
//info is used to know if were setting the default or current locale
function changeLocaleWorked(oldLocale,newLocale,info){
        kony.print(info["type"] + " locale changed from" + oldLocale + " to " + newLocale);
}
//callback if the change didn't work
//info is used to know which failed - setting the default or current locale
function changeLocaleDidntWork(errCode,errMsg,info){
          kony.print("problem changing " + info["type"] + " locale - error: " + errMsg); }
```

Creating a resource bundle

- **kony.i18n.setResourceBundle(bundle,locale)** is used to create the resource bundle for the specified locale where:
  - bundle is an object where the key-value pairs are the I18N keys and the corresponding localized text
  - locale is the locale for which the resource bundle will be created
  - For example, to recreate the resource bundles we created in the IDE (for English, Spanish and French):

```
//creates our English resource bundle
kony.i18n.setResourceBundle({btnA:"English", btnB:"Spanish", btnC:"French", frmTitle:"I18N testing"},"en");
//creates our Spanish resource bundle
kony.i18n.setResourceBundle({btnA:"Inglés", btnB:"Español", btnC:"Francés", frmTitle:"I18N pruebas"},"es");
//creates our French resource bundle
kony.i18n.setResourceBundle({btnA:"Anglais", btnB:"Espagnol", btnC:"Francais", frmTitle:"I18N test"},"fr");
```

Managing resource bundles

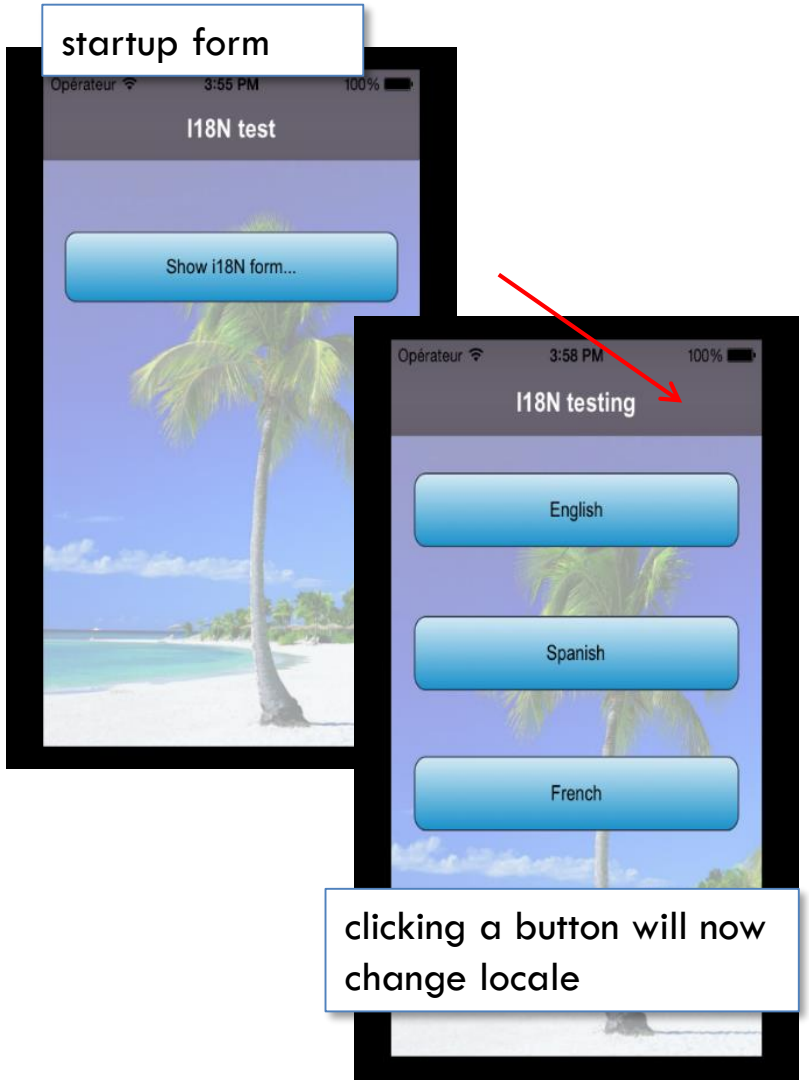
- **kony.i18n.isResourceBundlePresent(locale)** returns a Boolean value indicating if a resource bundle exists for a given locale and is used to check before using a resource bundle
- **kony.i18n.deleteResourceBundle(locale)** deletes the resource bundle for the specified locale
  - Typically used to remove bundles that you know you won't need any more - conserve memory
- **kony.i18n.updateResourceBundle(bundleData,locale)** adds the key-value pairs to the resource bundle for the specified locale where:
  - **bundleData-** is an object where the key-value pairs are appended to the existing resource bundle
    - **Note:** if the bundle doesn't exist, this will create that resource bundle using bundleData
  - **Locale -** is the locale indicating the resource bundle to update
- **Note:** to update existing strings, use **setResourceBundle()** to overwrite existing values

Accessing a Resource Bundle

- There are times when you want to get a value out of the resource bundle
  - For example, you want to program an error message that is properly localized
  - **kony.i18n.getLocalizedString(key)** will return the text for the specified key in the CURRENT resource bundle
- Typically a resource bundle can contain a LOT of data
- If the data is retrieved from an external service, best to cache it on the device
  - When the app starts again, it'll be available
- It's typical to perform all the resource bundle "management" before any form is shown
  - Once a form is shown, changing locales will not refresh the current screen - you need to manage that
  - Typical to do all this activity in the application's pre-appinit or post-appint events

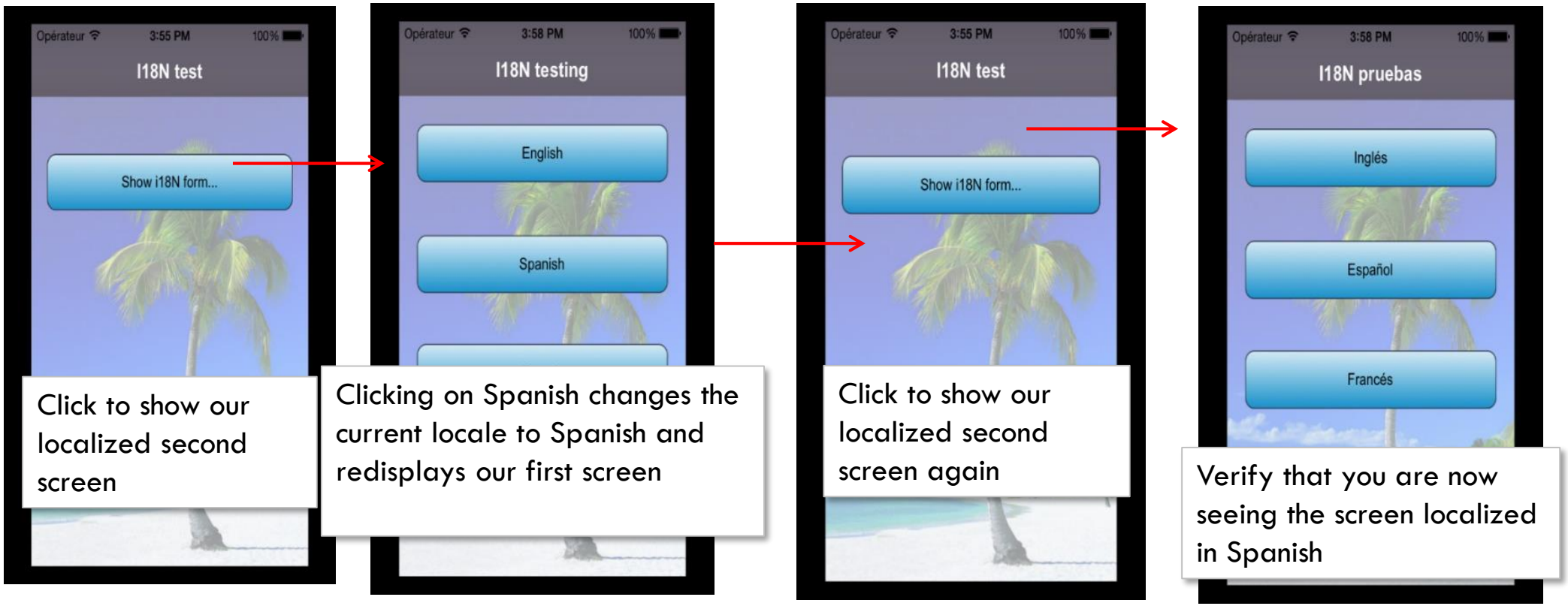
Exercise - I18N in code

- Let's try this out..
  - Since you can't "refresh" the form you are on, we'll add a startup form with a button to launch our I18N test screen
  - Program the buttons on the second screen to actually change the current locale to the language specified on the button
    - After changing locale:
      - Navigate back to the new startup form
      - Destroy the second form - causes it to reload upon next show to use the new language strings
  - Test by navigating back to the second form to see the buttons localized according to which button you pressed last time
  - Let's look at an example flow…



Exercise - I18N in code (cont'd)

- Let's look at this workflow:



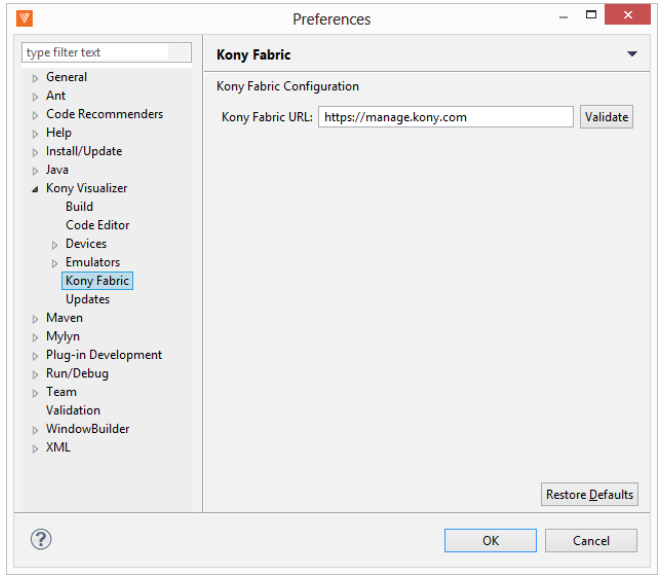
Other Channels

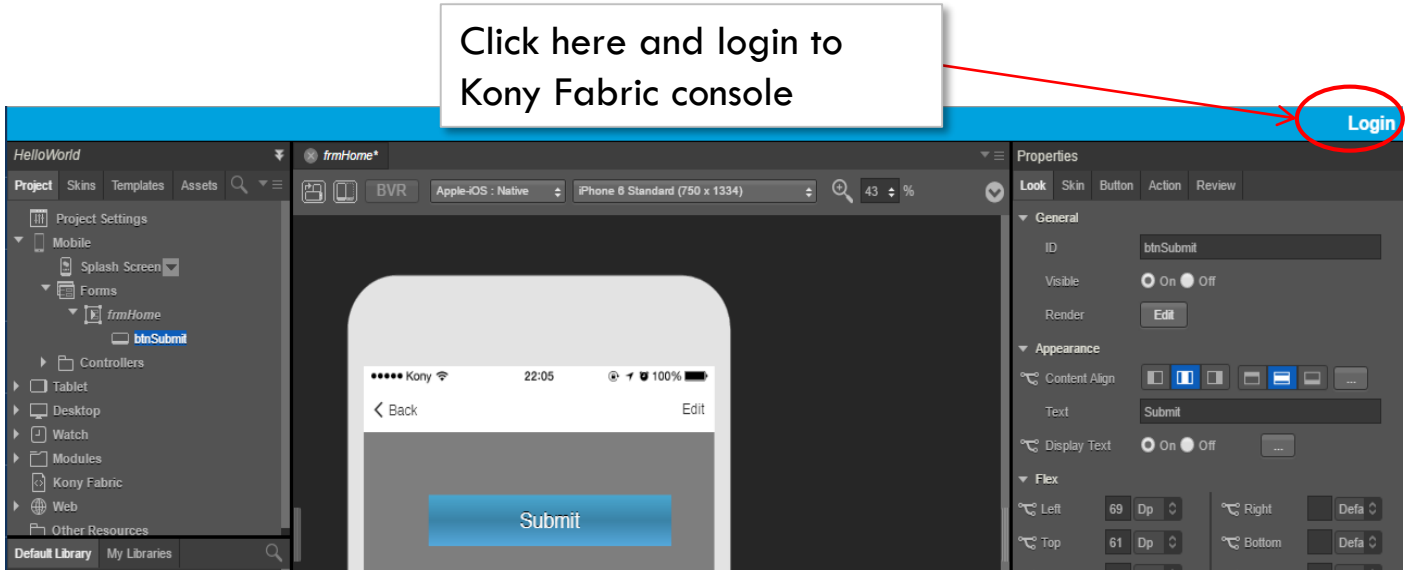
Overview of SPA

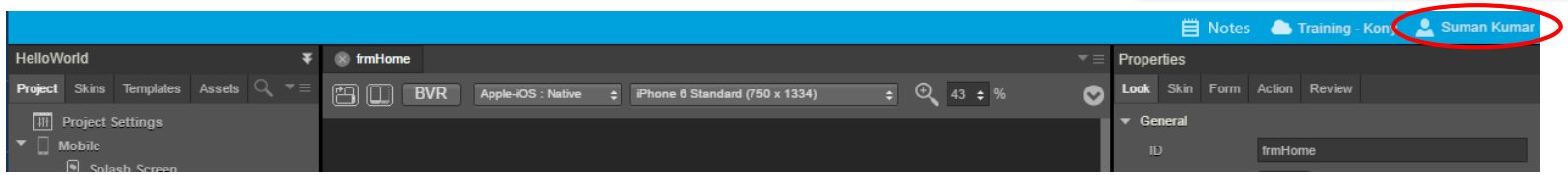
- A Single Page Application (SPA) represents a special type of the web application:
  - All the application screens are compiled down to JavaScript and CSS
  - HTML5 manifest is created indicating all the files that are used
  - These files are hosted on a web server somewhere (Kony server, for example)
- When the user first browse the URL, here is what happens:
  - The device is checked to see if it supports SPA
  - The manifest is read and all files are packaged up and downloaded onto user's device
  - There it is run locally using the native web browser (must be HTML5 compliant)
- If/when there are changes, next time the user launches the app, the manifest on the server is checked and the new app is downloaded to the device
- With the app running on the device, connectivity to that initial web server is no longer needed, so the app essentially runs as an on-device application

Building and Publishing SPA

- We need to use Kony Fabric server for deploying the SPA applications. So, first thing that we need to do is configure Kony Fabric details for an app
- Go to Visualizer Preferences and configure Kony Fabric URL and then login to Kony Fabric console in Visualizer

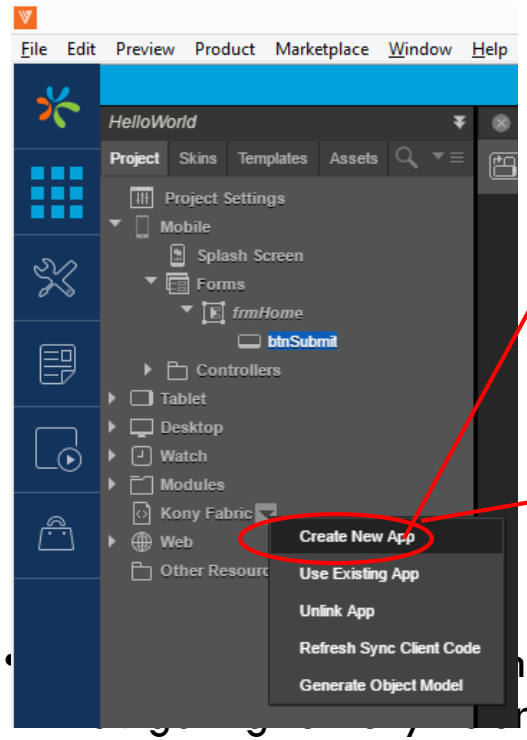




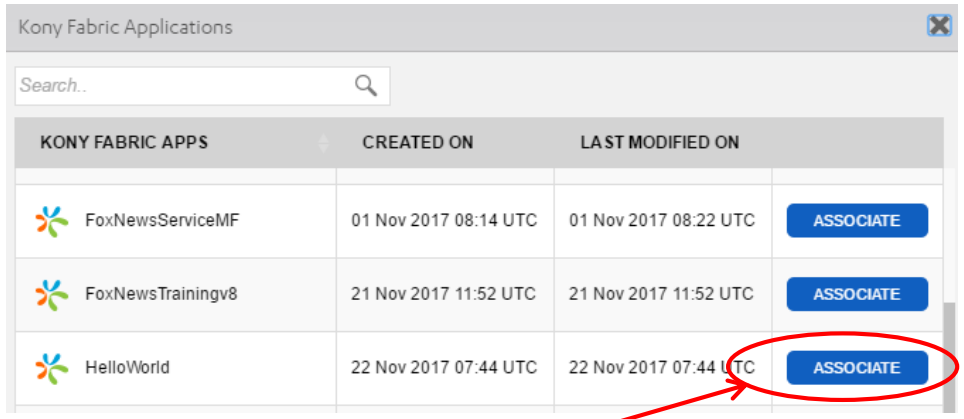


Building and Publishing SPA (cont'd)

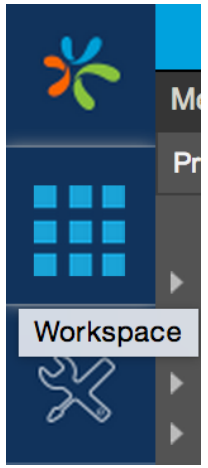
- Now, we need to create a new app in Kony Fabric using the below option



Create a new app in Kony Fabric



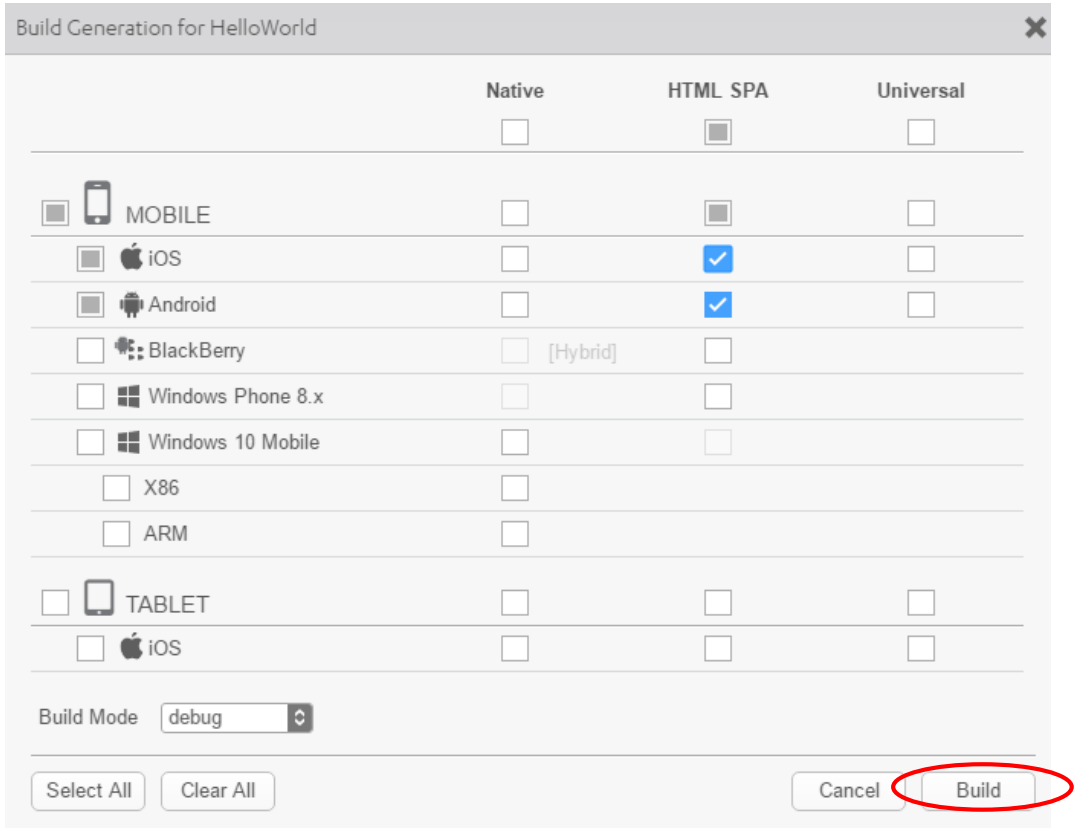
If an app is already created in Kony Fabric and you want to use it

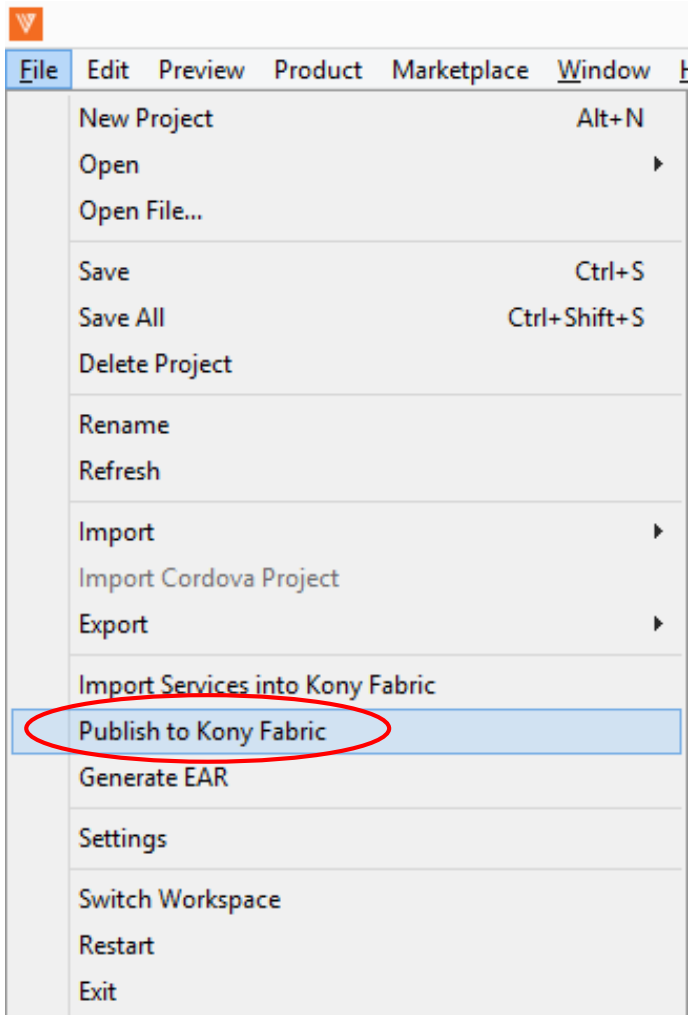


- ew app or using an existing app, you will be ric console in the Visualizer itself. You can click on **Workspace** to come back to Visualizer project perspective

Building and Publishing SPA (cont'd)

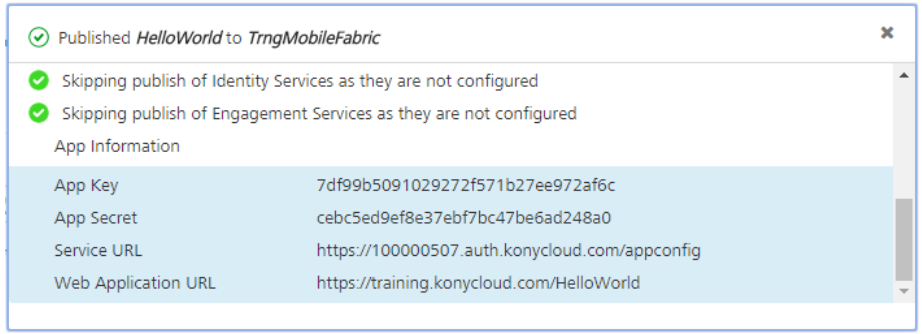
- Now, build for SPA platform and publish to Kony Fabric

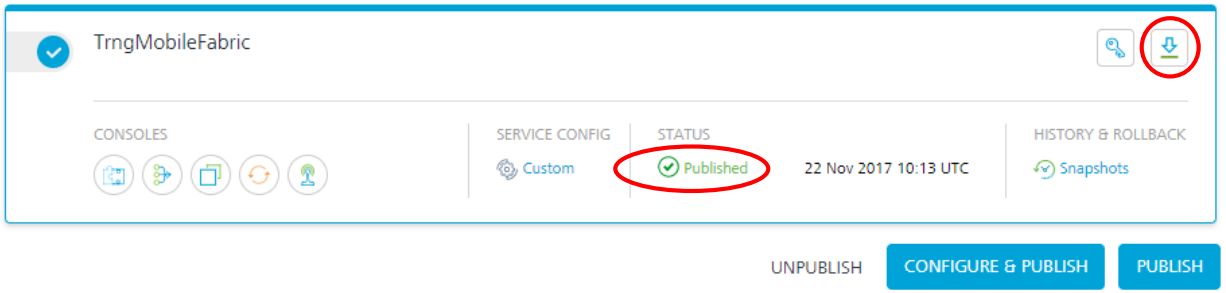


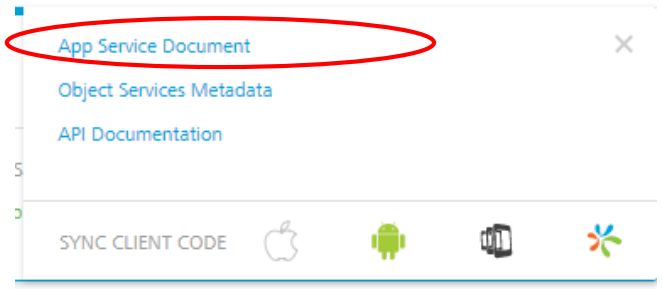


Building and Publishing SPA (cont'd)

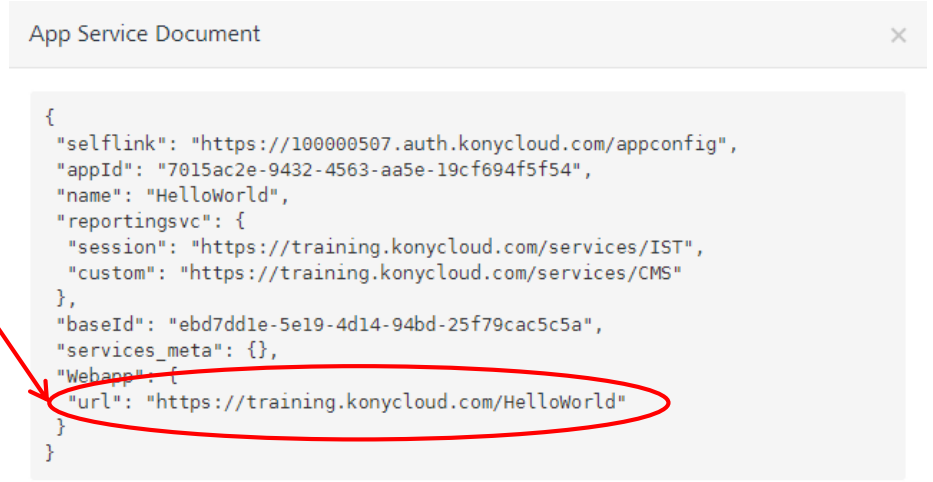
- Once the publish is done, click on the cloud icon and then select App service Document as shown below…





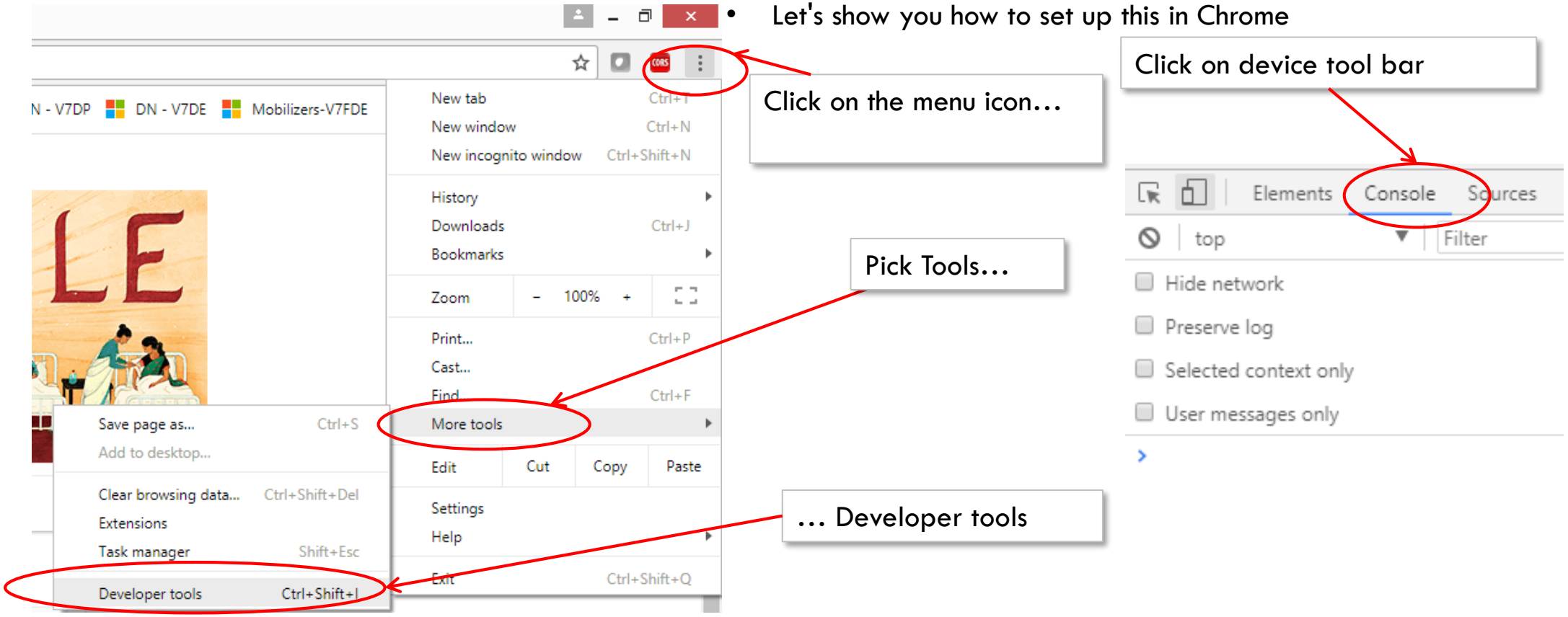


This is the URL to launch our SPA app

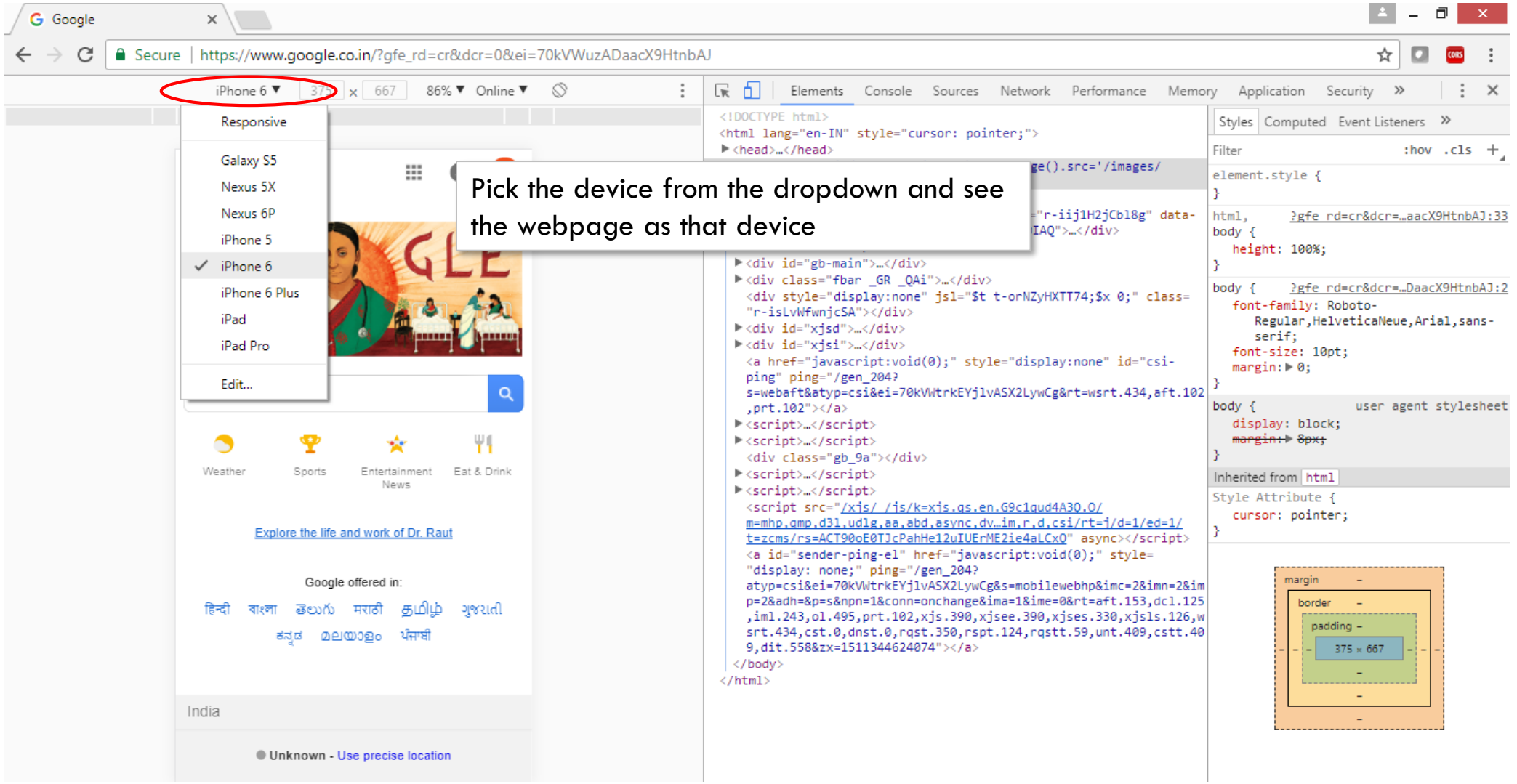


Launching SPA App

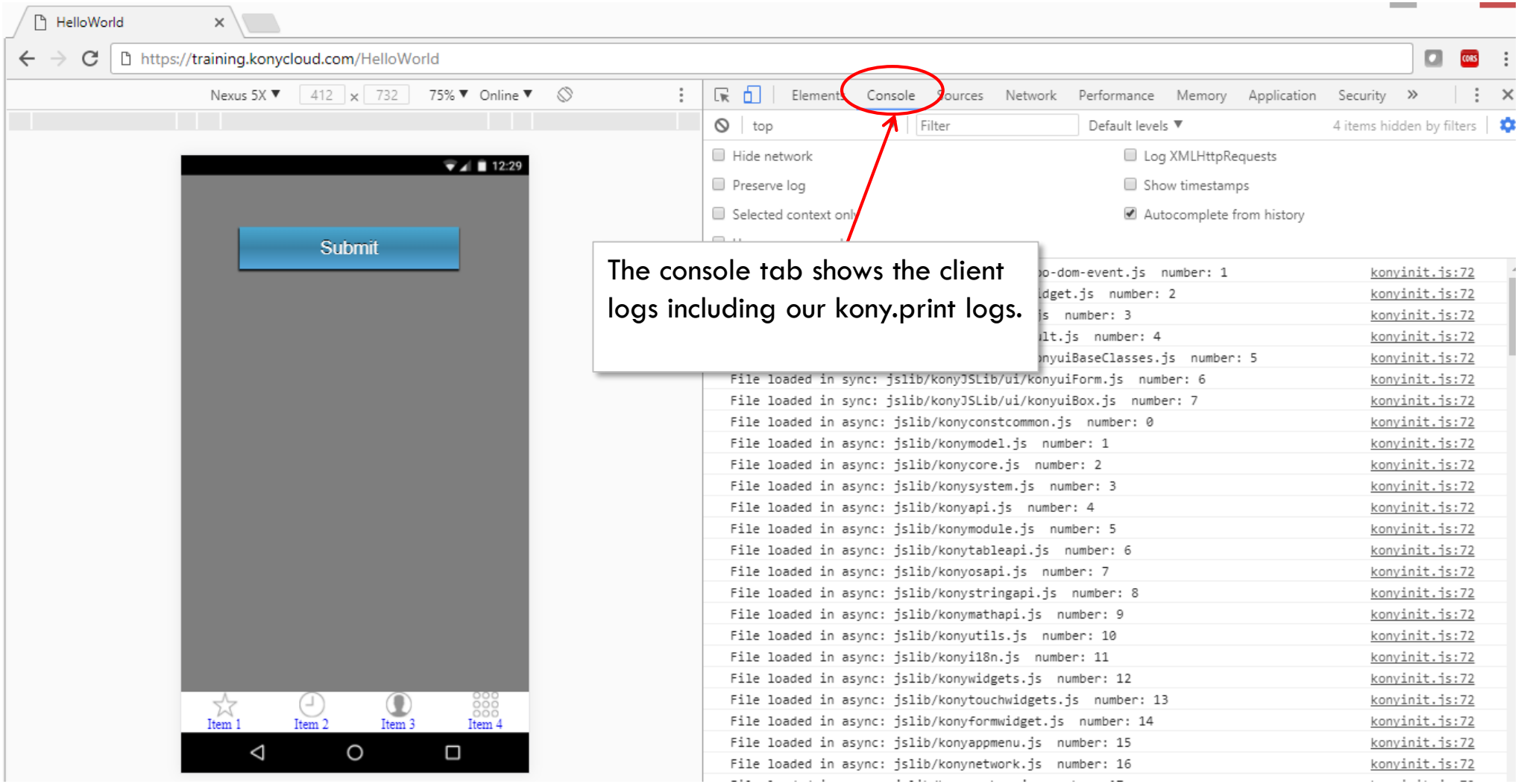
- You can launch and test the SPA app in the device's browser or Simulator/Emulator's browser
  - Using Chrome, you can mimic the device and see your print statements
  - Let's show you how to set up this in Chrome



Launching SPA App (cont'd)

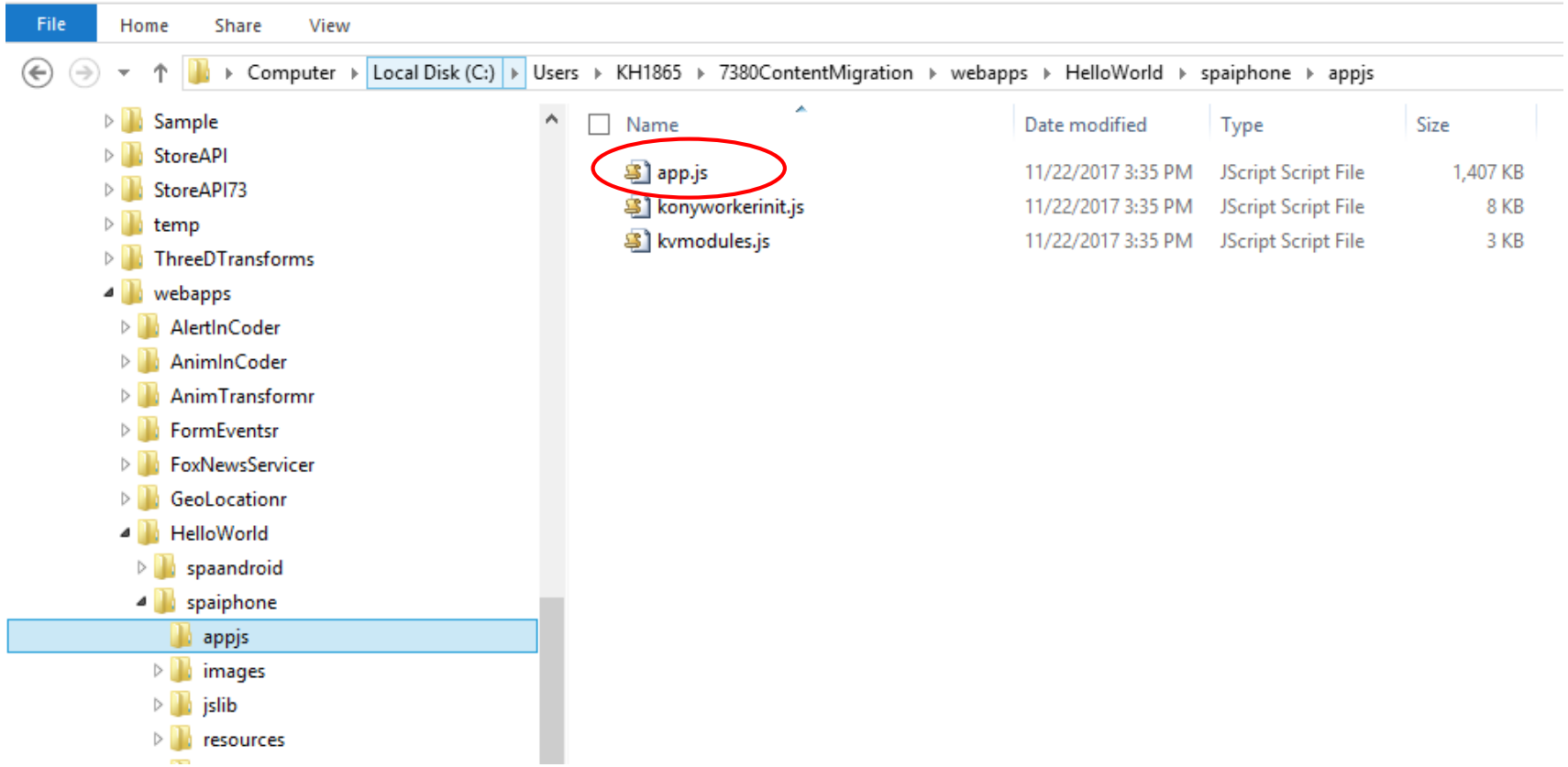


Launching SPA App (cont'd)

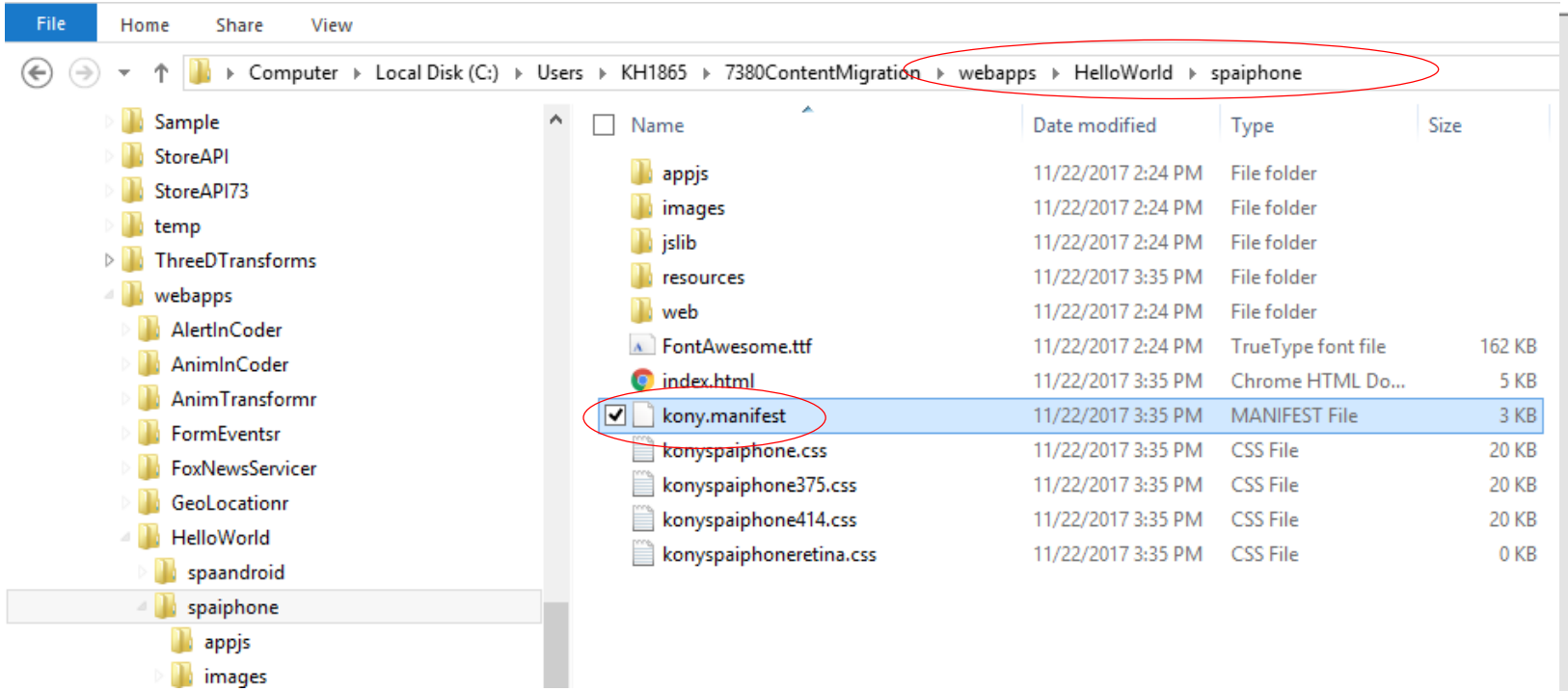


SPA Build

- All the forms, skins and code is copied to one file: app.js - and can be found at:
  /<workspace>/webapps/<app name>/<build type>/appjs/



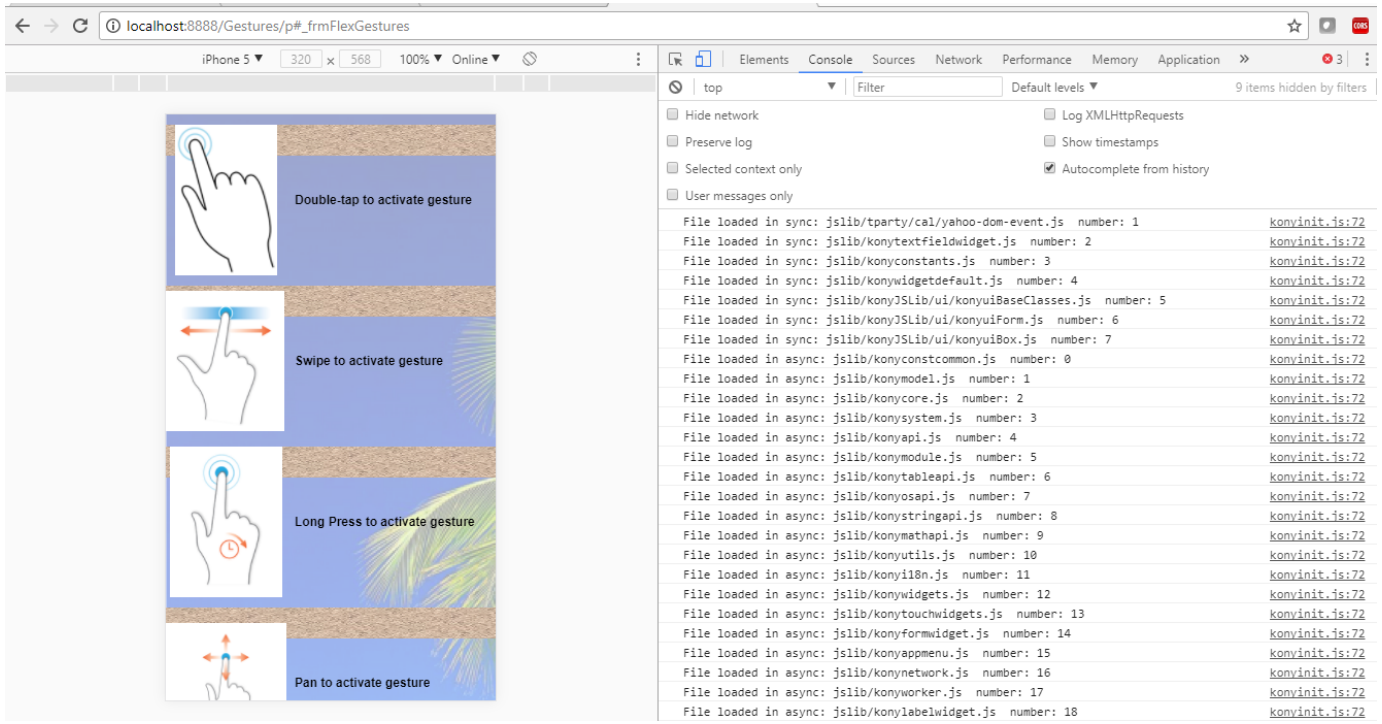
SPA - Manifest




```
1  CACHE MANIFEST
2  #Time stamp Wed Nov 22 15:35:31
   IST 2017
3  #Version 1.0.0
4  CACHE:
5  jslib/konyinit.js
6  jslib/konycore.js
7  jslib/konysystem.js
8  jslib/konyapi.js
9  jslib/konytableapi.js
10 jslib/konyosapi.js
11 jslib/konytimerapi.js
12 jslib/konyphoneapi.js
13 jslib/konygeolocationapi.js
14 jslib/konyhybridapi.js
15 jslib/konystringapi.js
16 jslib/konymathapi.js
17 jslib/konyutils.js
18 jslib/konywidgets.js
19 jslib/konytouchwidgets.js
20 jslib/konyformwidget.js
21 jslib/konyappmenu.js
22 jslib/konynetwork.js
23 jslib/konylabelwidget.js
```

SPA - Exercise

- Let's run our Gesture exercise as SPA
- Do an SPA build for iPhone
- Launch Chrome
- Configure it to use the user agent for iPhone
- Run & look at the console for your printouts



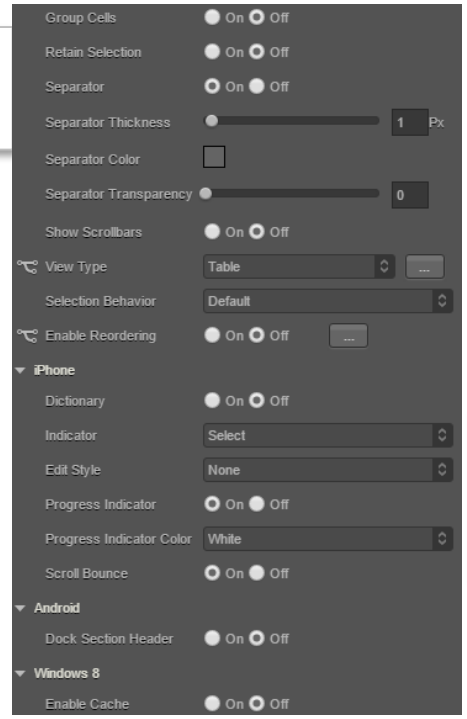
Project Folders

- So far, we've only worked in the mobile folders for building smartphone apps/websites
- When it comes to tablets and desktop, there is way more real estate for the UI
  - Re-using the mobile forms doesn't make sense
  - Re-using the skinning, code, services, etc., is necessary!
- For the UI, there are separate folders for tablets and desktop
- If you are creating an app for both mobile and tablet and/or desktop:
  - Re-use form names and widget names for maximum code re-use
  - You'll probably need new images for tablet and desktop - higher resolution/bigger
  - Re-think the usage flow - typical to reduce the screen count by combining features on single screen

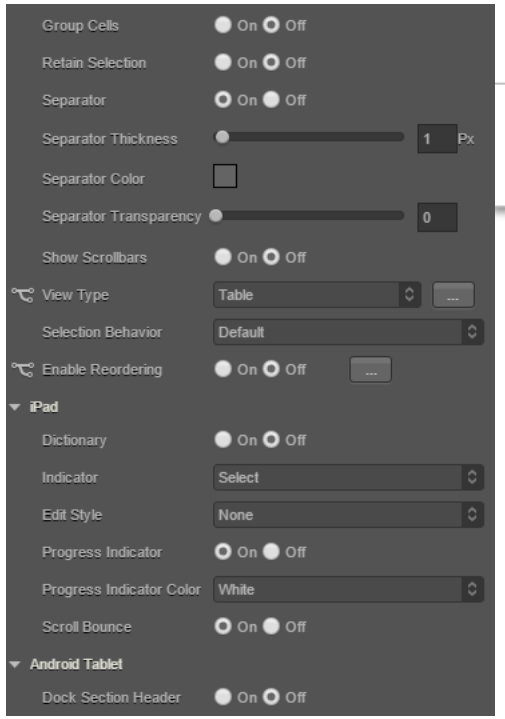
Tablet Widgets and Properties

- For tablet, you have access to the same widgets and layout principles as you do for mobile apps
  - There are additional Platform Specific Properties that are added for the tablet devices
  - For example, let's compare the PSP for mobile vs. tablet for a Segment widget

Mobile properties:



They are the same except Windows Phone 8 for mobile



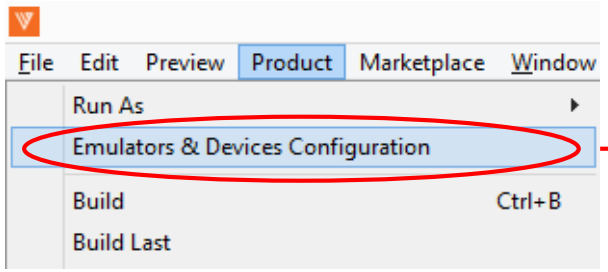
Tablet properties:

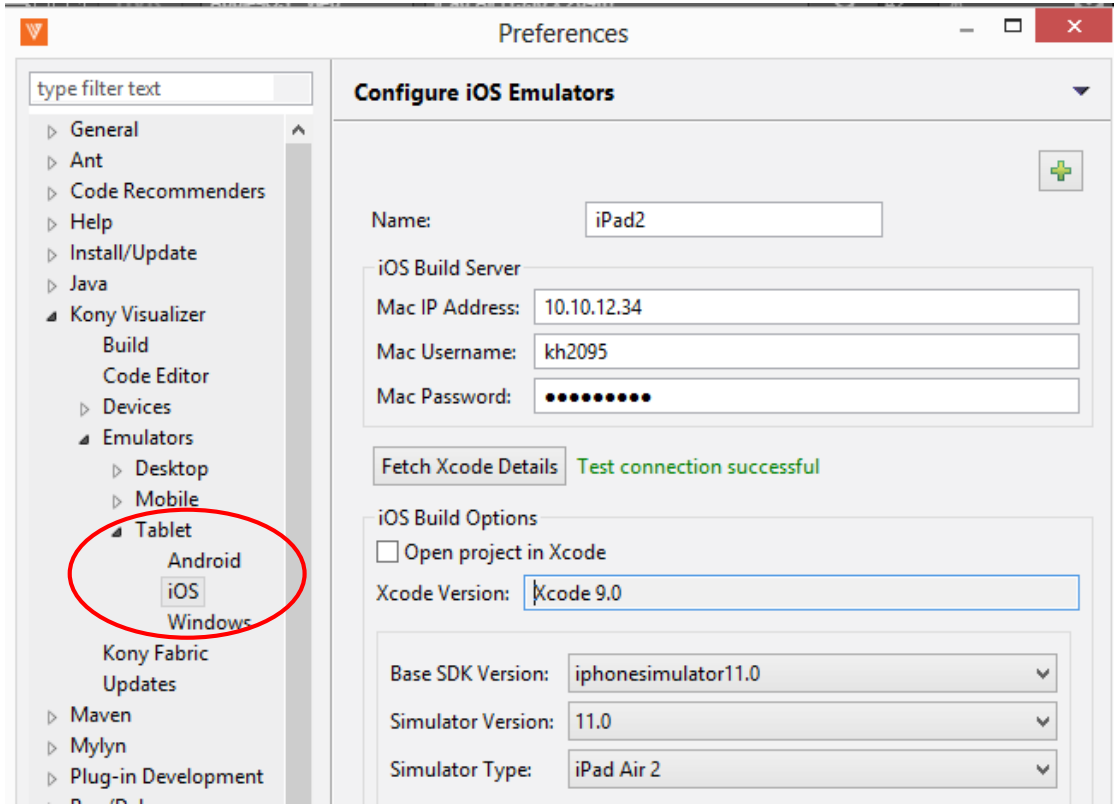
Tablet Design

- The biggest difference in the UI, therefore, is HOW you layout your widgets
  - Very common to have "columns" of data
    - For example: an email app with a list of emails on the left and reading pane on the right for the selected email
  - Different screen scroll paradigm:
    - Mobile apps typically have screens taller than the device so the user scrolls down to see it
    - Tablet apps may have scrollable sections but typically have a static screen that doesn't scroll
      - Use the FlexScrollContainer widget to create areas of the screen that scroll without scrolling the whole screen

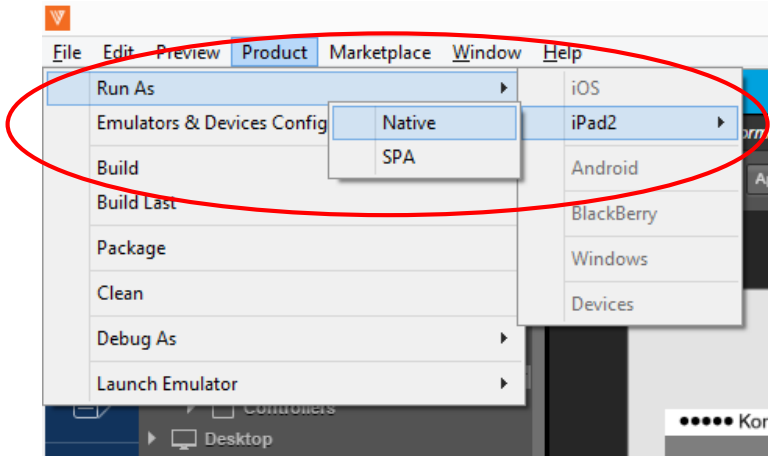
Emulator Notes - iPad

- The iPad simulator is configured just like the iPhone simulator



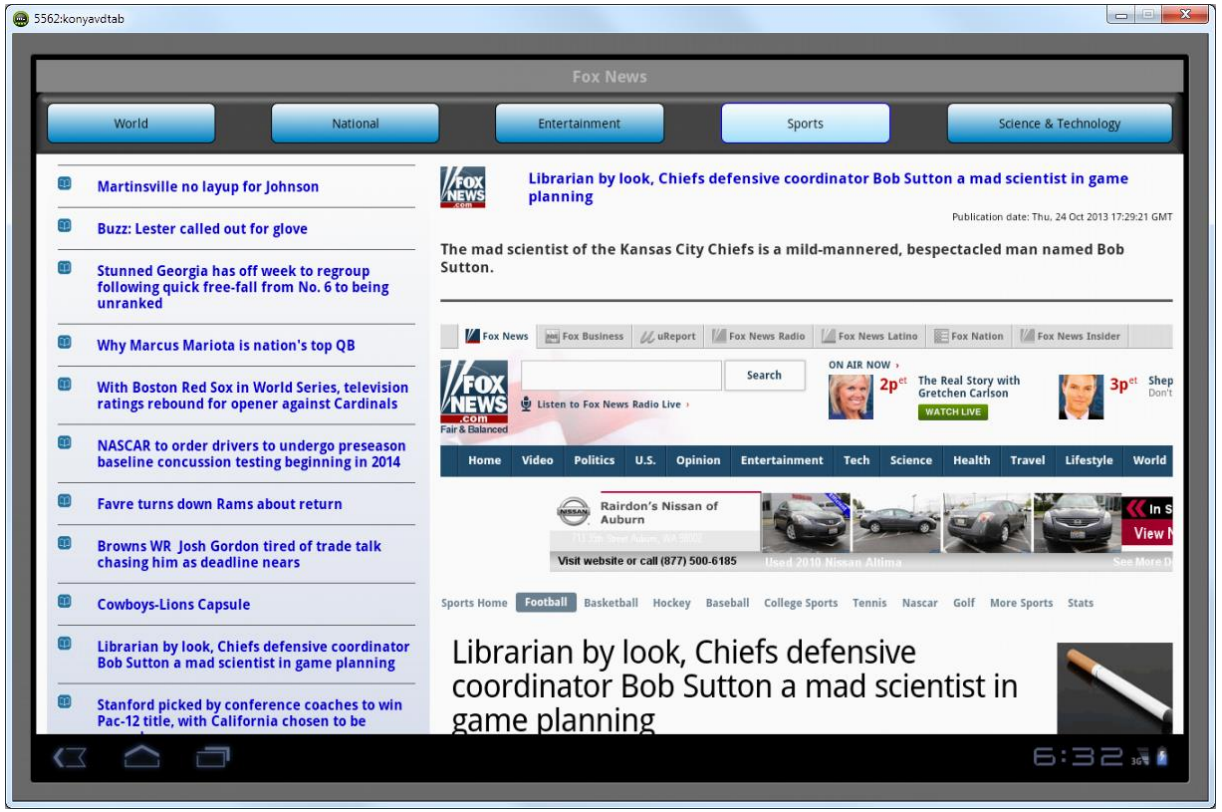


- Now, you can run for iPad

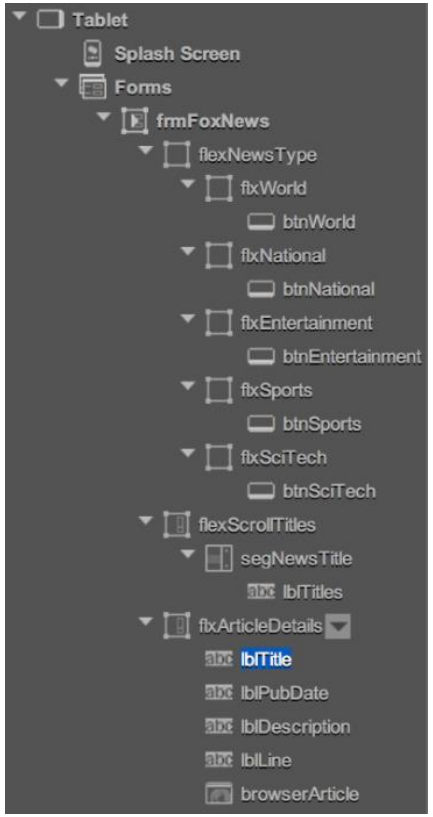


Tablet Layouts

- Layouts will grow in complexity because you'll tend to have lot more side-by-side data and just MORE stuff on the form
- For example:

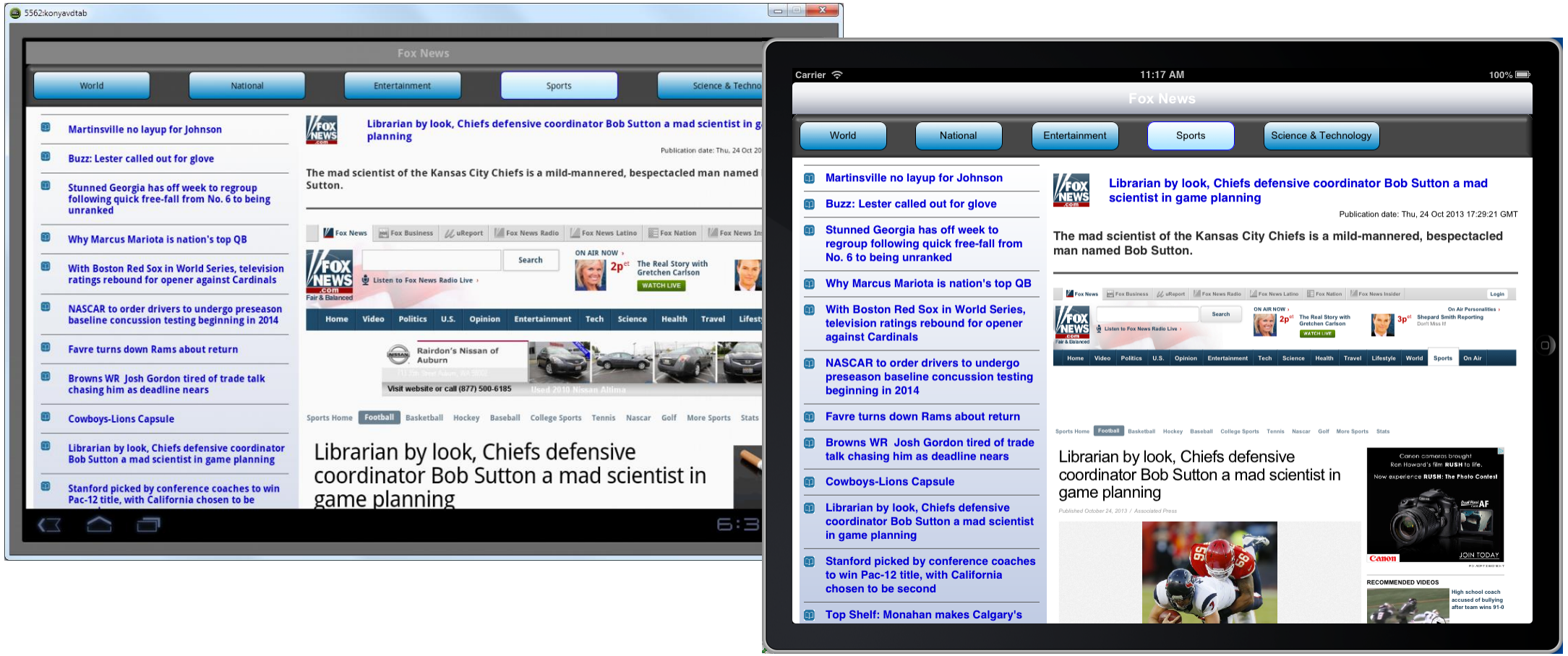


With all the room, even this simple looking layout, we can use a lot of widgets:



Tablet Layouts - rendered

- Here's an example of that previous form running on both tablets: Android Galaxy and iPad



Copy and Paste

- You can copy and paste widget(s) from mobile forms to tablet forms.
  - All properties are copied over, if they are not mobile type specific
    - Example: Templates are defined for channel (mobile, tablet or desktop) - Copying a widget that uses a template will NOT copy over the template info
  - Event definitions (like `onClick`) are copied
  - All skins and resources are copied with the same caveat as above
    - Example: An image pointing to a mobile folder will not copy over the `src` property

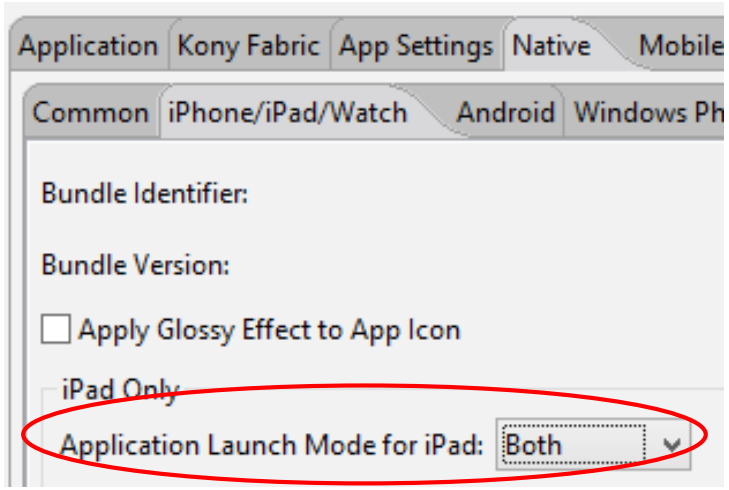
Tablet - Write Once Run Everywhere

- The application code is divided into - data model, service, business logic, UI navigation and pure UI logic
- The goal is to maximize the re-use of the first 3 and only have uniqueness when it comes to the UI code
- If you do have uniqueness to the UI, you'll do one of the following:
  - Use an ifdef (no code bloat for other platforms) and write separate code
  - Note: If you are using free form way of creating a project, Use function that get the UI element passed into it, so you can reference it generically
    - For example:
      - Both tablet and phone have a form with a segment - `segProduct` - that needs to get populated with data, but the forms are named differently
      - Write your function as: `myFunction(myform)`
      - Then, access `myform.segProduct` in your function
      - In each call, pass the correct form name

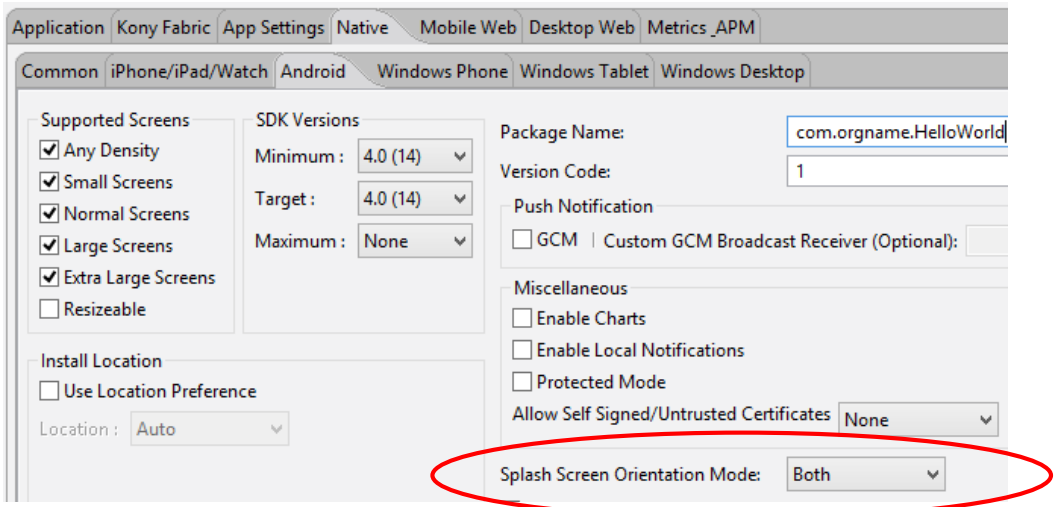
Orientation Support

- There are no differences in a form's orientation properties between the iPhone and the iPad
- There are, however, differences on how you enable some orientation features
- For tablets, go to the application properties under the native tab and here is how you configure for Android and iPad

For Android Tablet – the forms control the orientation BUT here is where you configure the splash screen - typical to enable both here



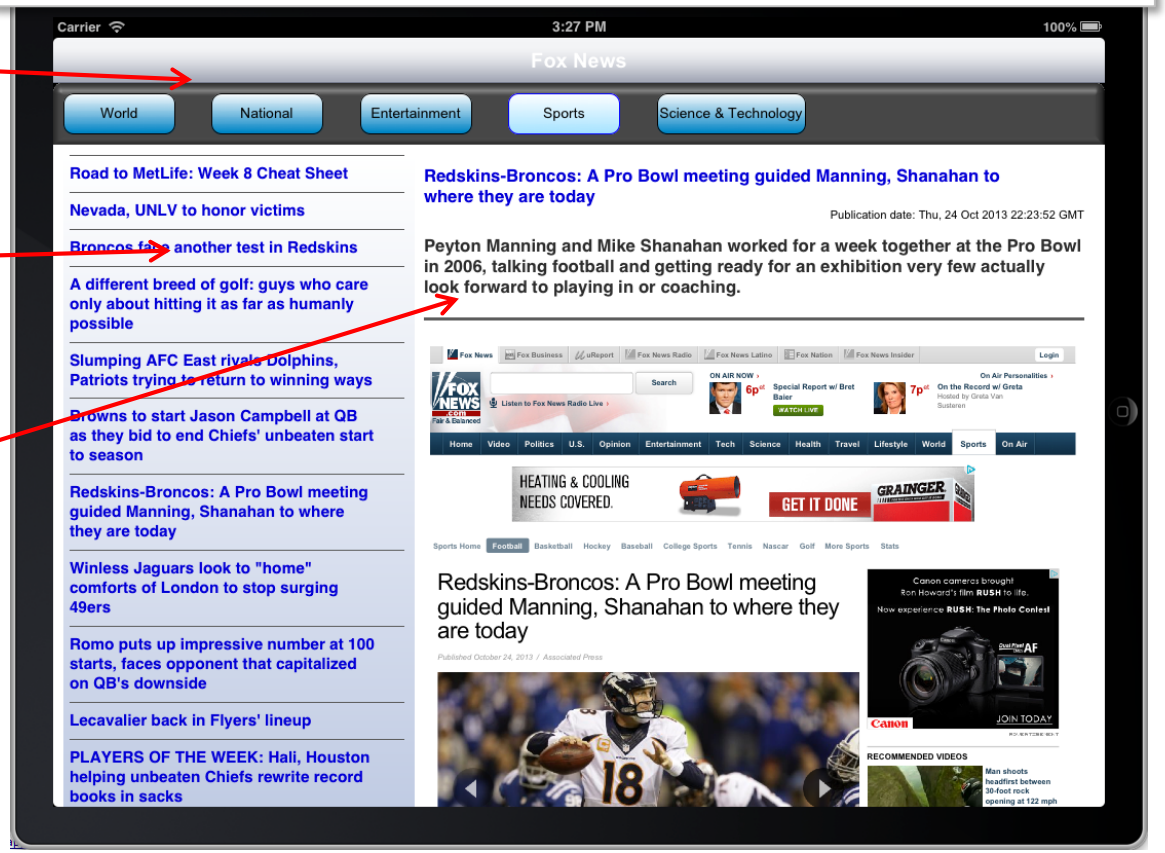
For iPad - if an orientation is not turned on here, the form's setting will not work – typical to enable both here



Tablet App - Example

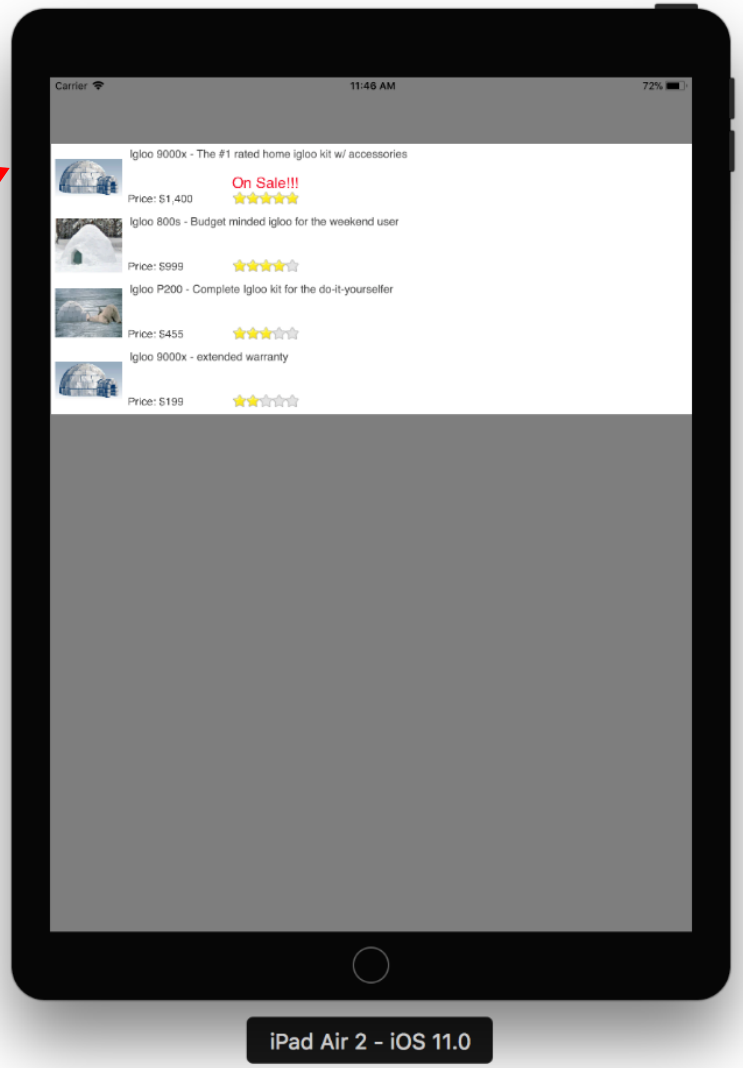
- You'll use the existing service and code from your mobile app, where:
  - The buttons across the top trigger the service call
  - The left navigation bar is a scroll bar with the article titles (use a scroll Box to contain the segment)
  - At the right half we show the title, publisher date, description AND a browser widget showing the article
- Copy/paste the widgets from your mobile app to your tablet form to build the UI
- Re-use AS MUCH code as possible - Name your form, segment, labels, etc., the same as mobile and your UI code will pretty much just work

Note: even though we used buttons, we wanted the user to know what news type was picked: change the skin to show that – not required but a nice touch!



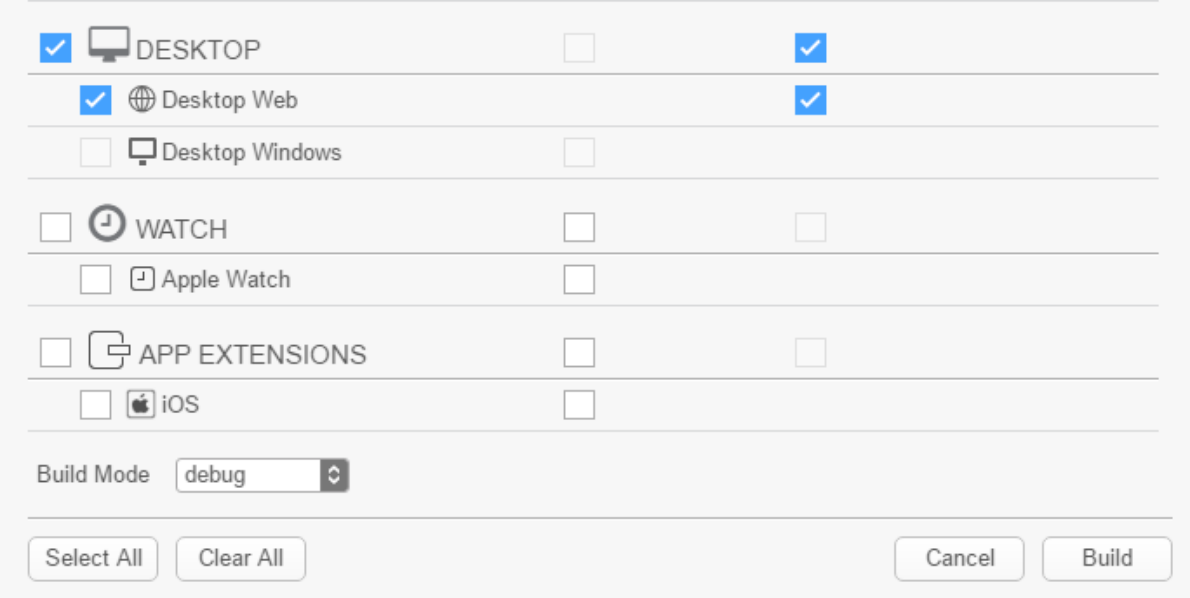
Exercise

- Let's run our Segment mobile app on Tablet
- Copy/paste the widgets from your segment mobile app to your new tablet form to build the UI
- Launch the application on iPad tablet



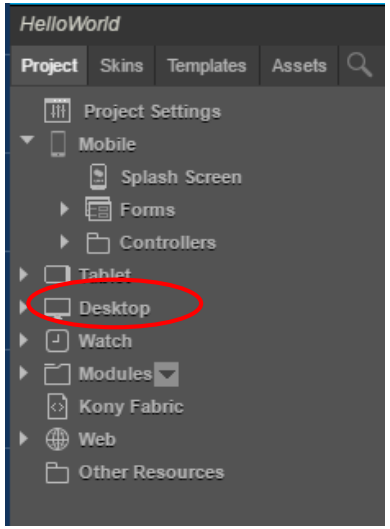
Desktop and Desktop Web

- There are 2 types of applications you can build for the desktop:

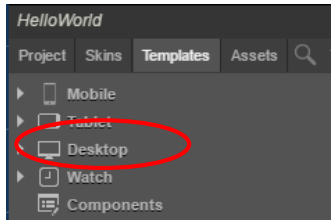


- Native:
  - Windows - an .exe that runs like any other windows app
- Web:
  - SPA - exactly like the mobile SPA only designed to run on desktop browsers

Desktop Web (cont'd)



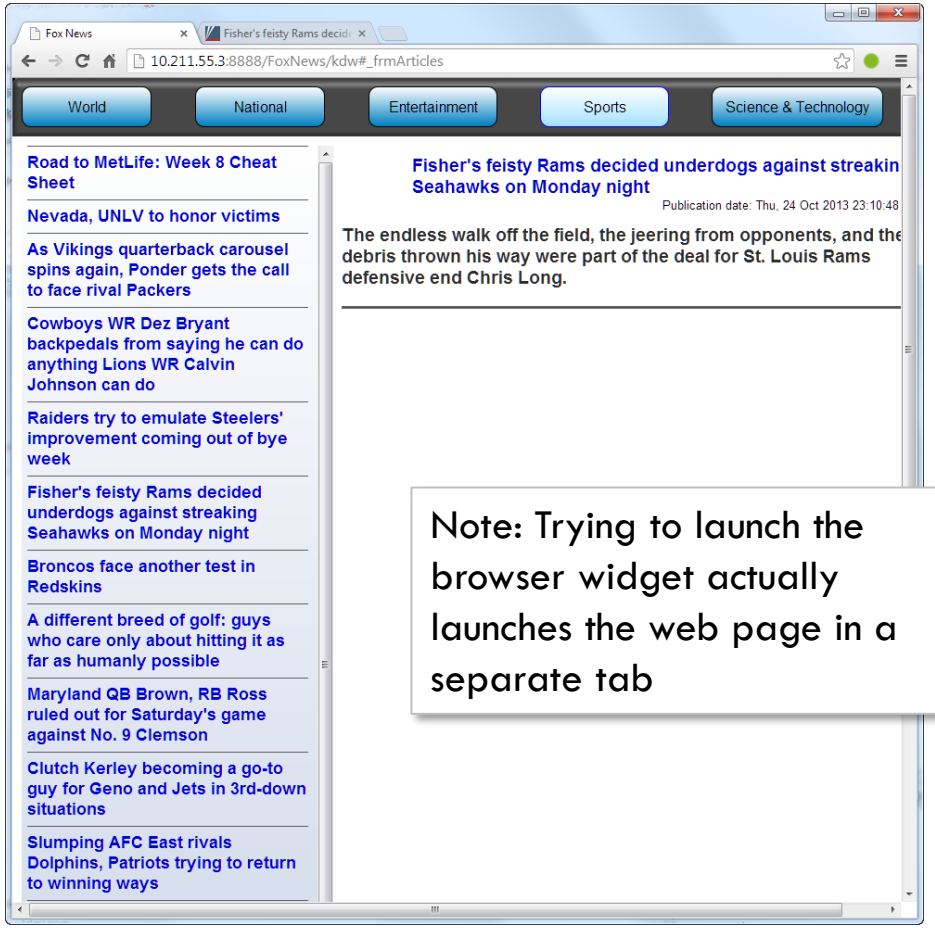
Note that any UI element has separate folders for mobile, tablet and desktop

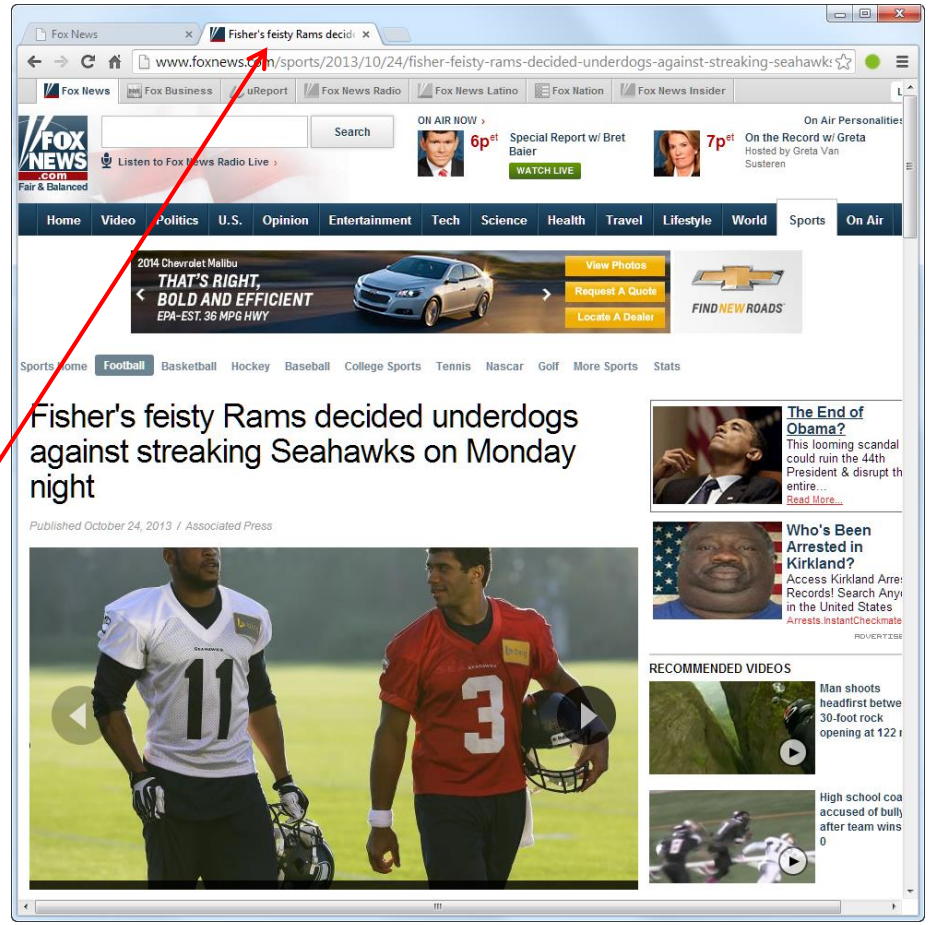


- Desktop Web supports flex layout like SPA. Everything that we discussed in SPA is applicable for Desktop Web also
- Building and Launching for Desktop Web is same as like how we do for SPA. This is the desktop web url: http://<IP or localhost>:<visualizerPort>/<ProjectName>/kdw

Desktop Web (cont'd)

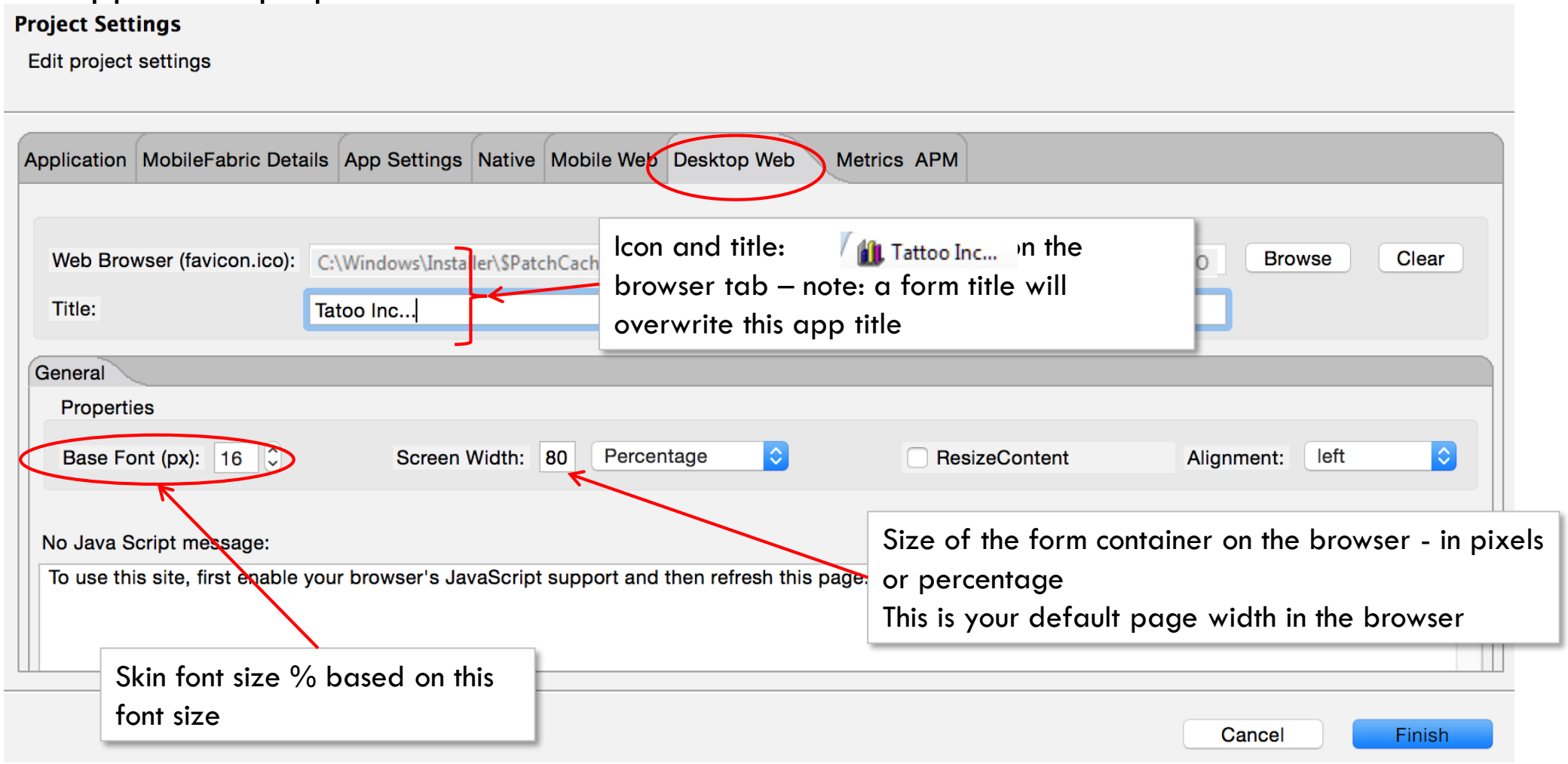
- Look at our tablet FoxNews app and you can recreate Desktop Web app
  - You can reuse your services and business logic to the greatest extent possible





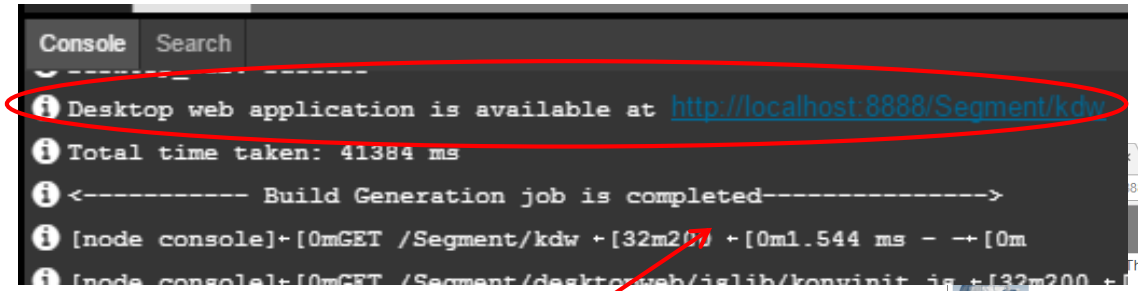
Desktop Web - Unique Properties

- In Application properties:

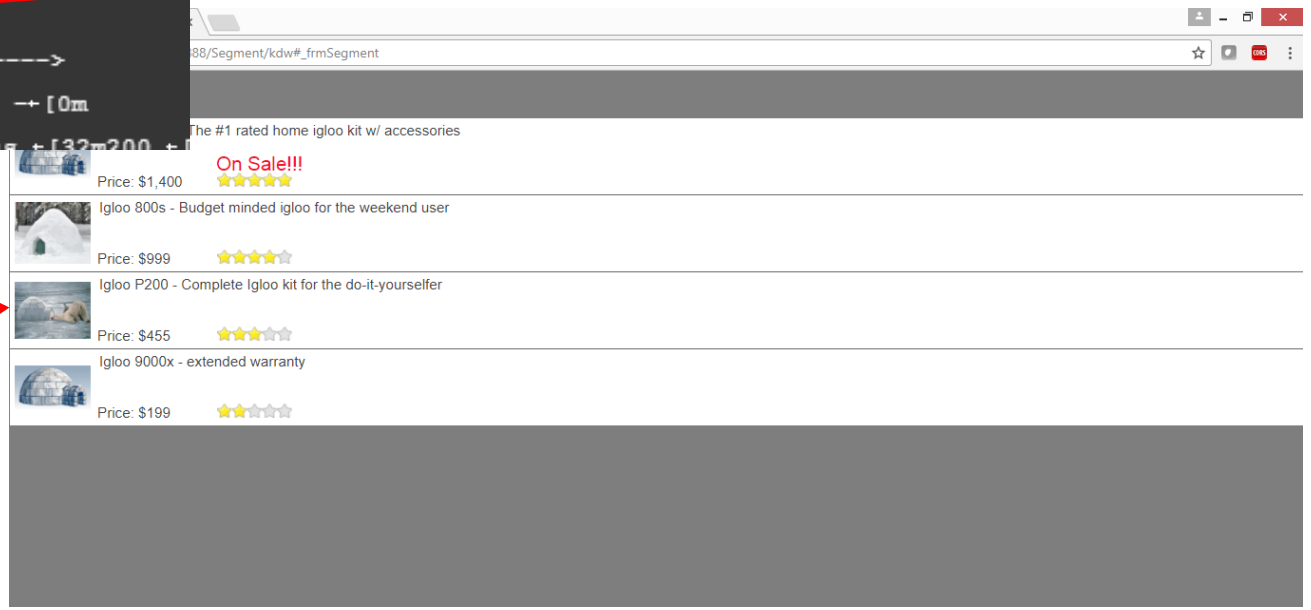


Exercise

- Let's run our Segment mobile app on Tablet
- Copy/paste the widgets from your segment mobile app to your new tablet form to build the UI
- Build the application for Desktop web platform and you can see your Desktop web application URL in Visualizer console after the build success



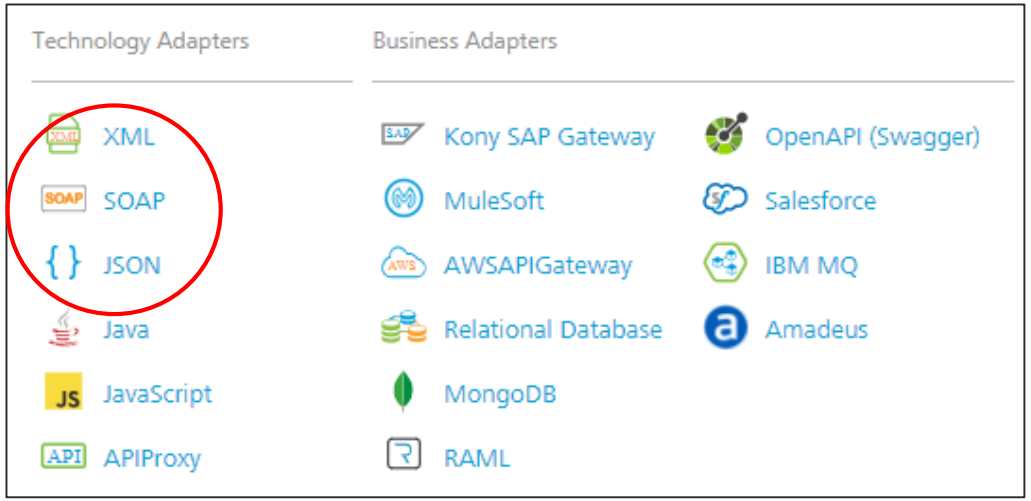
Visualizer automatically launches the app with this URL onto default browser



More Services

SOAP and JSON Web Services

- Here is the list of currently supported Integration types:



  - Note – The SDK coding to invoke the Integration services are the same regardless of the type you are calling
- We'll now cover SOAP and JSON web services
  - These work very much the exact same way as XML web services but each has a slightly different twist we need to cover, Let's talk about SOAP first...

SOAP Integration Services

- SOAP services are unique and are defined using a WSDL file, this file contains all the web services and configuration info for them

- Here is a sample WSDL URL:
  http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL

SOAP Integration Services (cont'd)

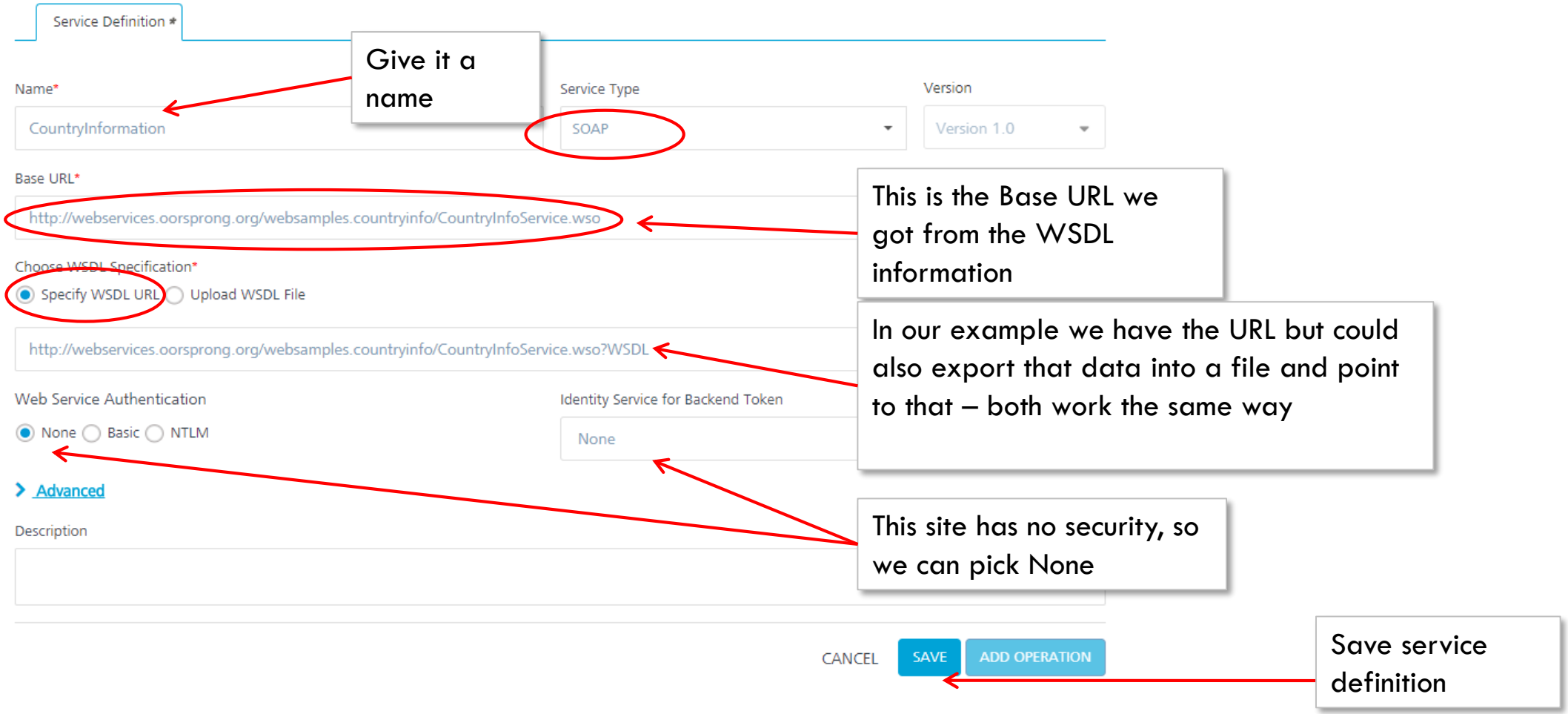
- Inside the WSDL (in XML format), you'll see this at the bottom:

```
<service name="CountryInfoService">
  <documentation>
    This DataFlex Web Service opens up country information. 2 letter ISO codes are used for Country code. There are fu
    Continent and Telephone code.
  </documentation>
  <port name="CountryInfoServiceSoap" binding="tns:CountryInfoServiceSoapBinding">
    <soap:address location="http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso"/>
  </port>
  <port name="CountryInfoServiceSoap12" binding="tns:CountryInfoServiceSoapBinding12">
    <soap12:address location="http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso"/>
  </port>
</service>
```

  - That URL value is the Base URL for our Integration service, and the base URL is same for both sets of services
- Here are the basic steps covered in the next slides…
  - You'll create a Service that points to the **Base URL** and the **WSDL**
  - Kony Fabric provides a list of all web services available in **WSDL**
  - You pick one and configure it as a Service Operation and Repeat, if you need other web services

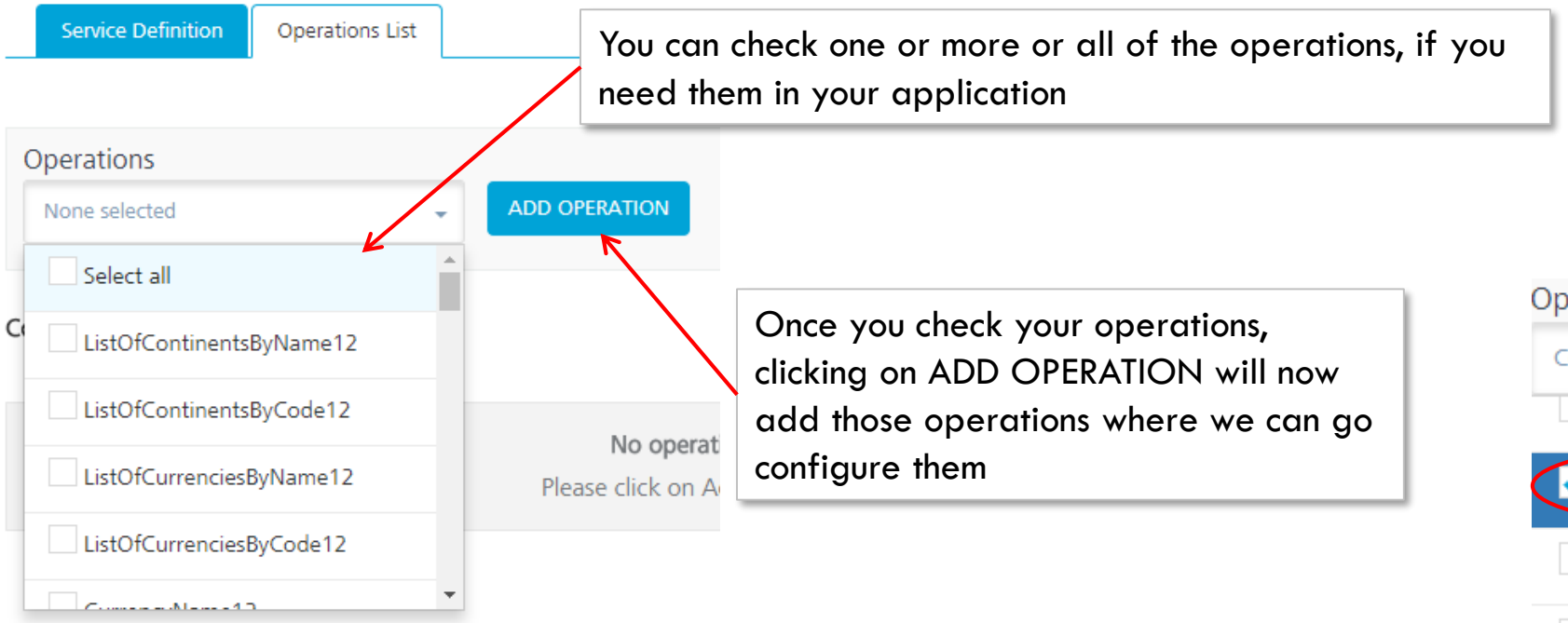
SOAP Service Configuration

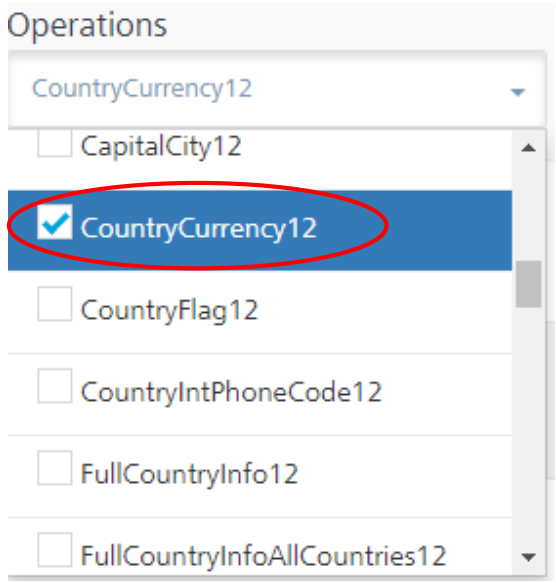
- Service - Definition



SOAP Operation List

- Unlike XML (and JSON), we're given a list of supported operations:

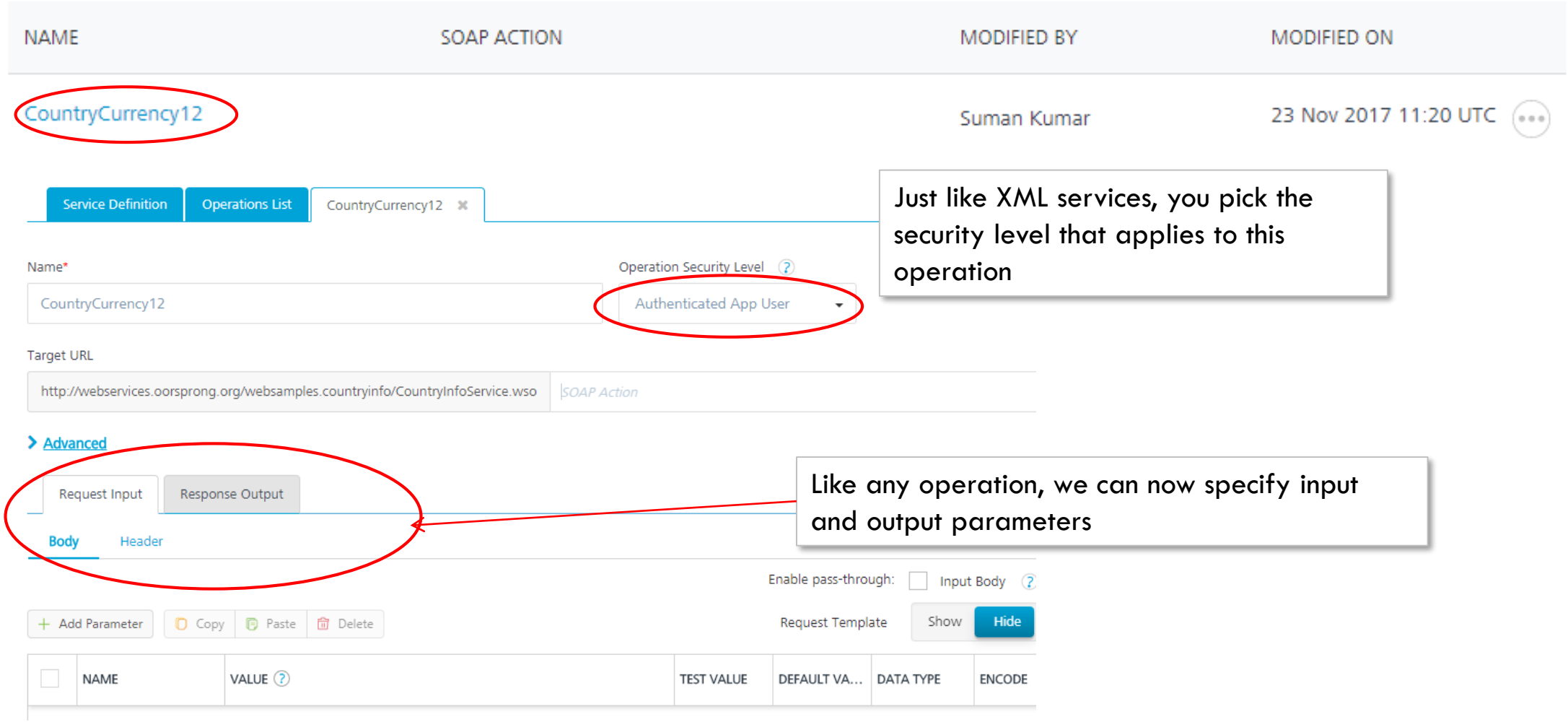




- For our example, we'll just pick one operation: This returns country currency for country code

SOAP Operation Configuration

- Your selected operations are shown below the list:

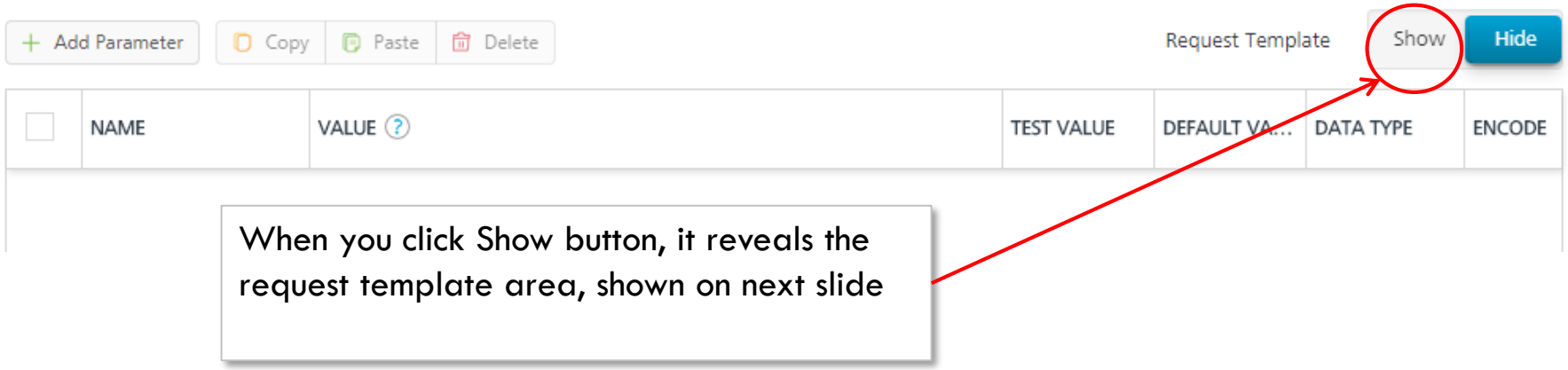


Just like XML services, you pick the security level that applies to this operation

Like any operation, we can now specify input and output parameters

Request Template

- In XML service, the inputs are specified as parameters to the web service call and are passed on as part of the URL
  - Sometimes, the inputs are specified as a part of a request document - a specific package of data with your input values
  - In Kony Fabric, the Request Template is where you specify these input values:



Request Template Configuration

- Click the Request Template to see the existing template

- The template indicates where a value needs to be inserted by using **"?"**
  - We need to replace all these with a hardcoded value OR an input parameter
  - We'll change the **?** to **`$city`** – we'll then have to create an input parameter called `city` and use this to pass values to our service

Configure Request Template

- The request template should look like the below:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:web="http://www.oorsprong.org/websamples.countryinfo">
   <soap:Header/>
   <soap:Body>
      <web:CountryCurrency>
         <web:sCountryISOCode>$country</web:sCountryISOCode>
      </web:CountryCurrency>
   </soap:Body>
</soap:Envelope>
```

Here is the input parameter

- And now we can test it...

	NAME	VALUE	TEST VALUE	DEFAULT VA...	DATA TYPE	ENCODE
☐	country	request	USA		string	☑

SOAP Response

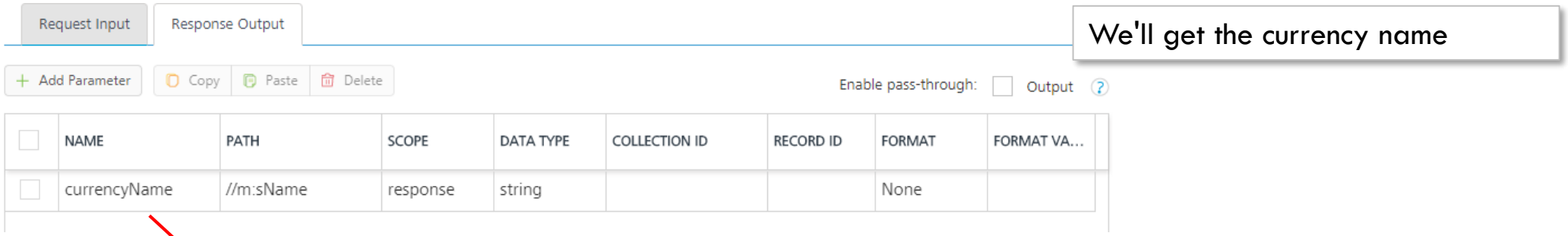
- Here is part of the response (copied outside of Kony Fabric window that is too small to see all the content):

Here is our country currency for given country (test value = “USA”)

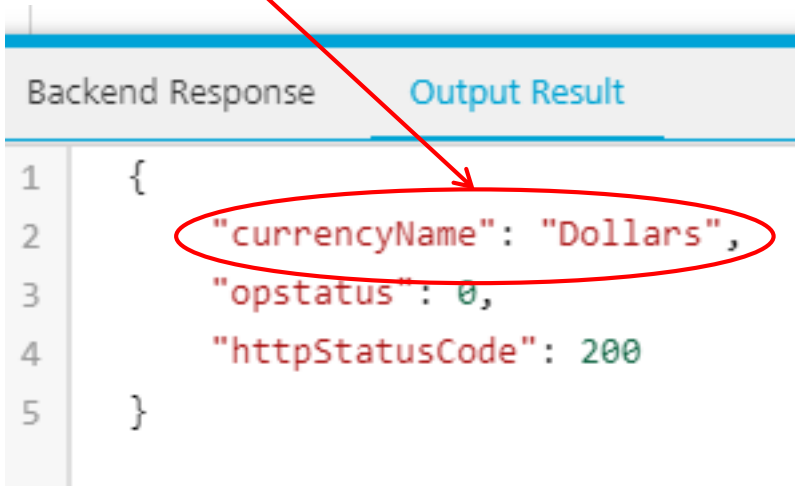
```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Body>
        <m:CountryCurrencyResponse
            xmlns:m="http://www.oorsprong.org/websamples.countryinfo">
            <m:CountryCurrencyResult>
                <m:sISOCode>USD</m:sISOCode>
                <m:sName>Dollars</m:sName>
            </m:CountryCurrencyResult>
        </m:CountryCurrencyResponse>
    </soap:Body>
</soap:Envelope>
```

- We can now configure output parameter to grab the data and check the results

SOAP Sample Result



- We click the Test button to test our Results (what goes back to the mobile application):

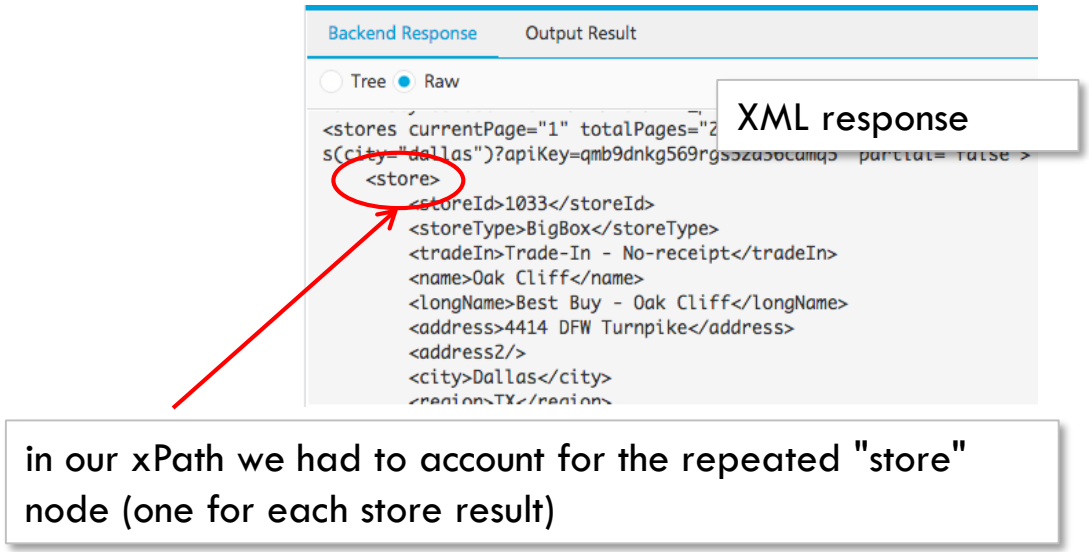


JSON Integration Services

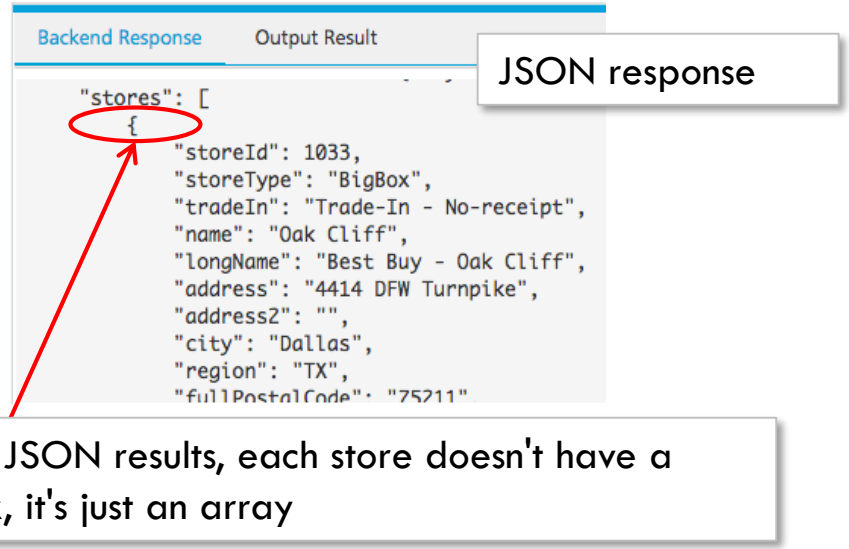
- A JSON service communicates with an external data source using JSON data connector over the HTTP protocol and returns a JSON (is popular over XML for mobile since it has a reduced notation size response in JSON format)
- Differences between XML and JSON notation:
  - A data element:
    - XML: <data>123</data>
    - JSON: {data:"123"}
  - A collection of data elements:
    - XML: <datas>
      <data>123</data>
      <data>456</data>
      </datas>
    - JSON: { datas: [{data:"123"}, {data:"456"}]}

JSON Integration Services

- JSON Integration Services work exactly like XML services with no difference in configuration
- The only difference is instead of using xPath to parse the results, use JSONPath
  - The commands are very similar, but the data structure you're working with is different
  - The BestBuy services we've used in training return both XML and JSON; Just add "&format=json" to any service to get JSON – XML is the default
  - Let's compare examples of the same service returning XML data and JSON data (a list of store locations):



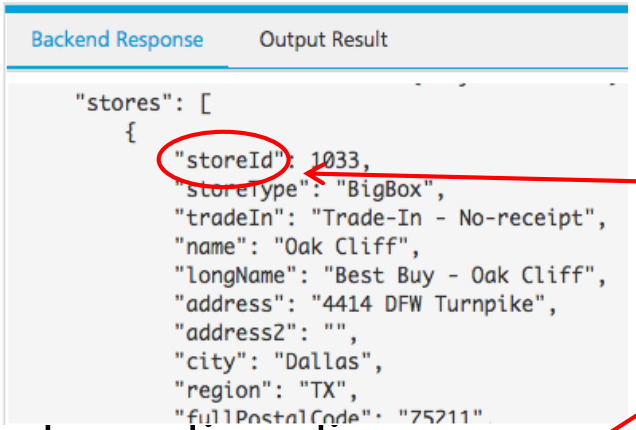
in our xPath we had to account for the repeated "store" node (one for each store result)



In the JSON results, each store doesn't have a prefix, it's just an array

JSON Path

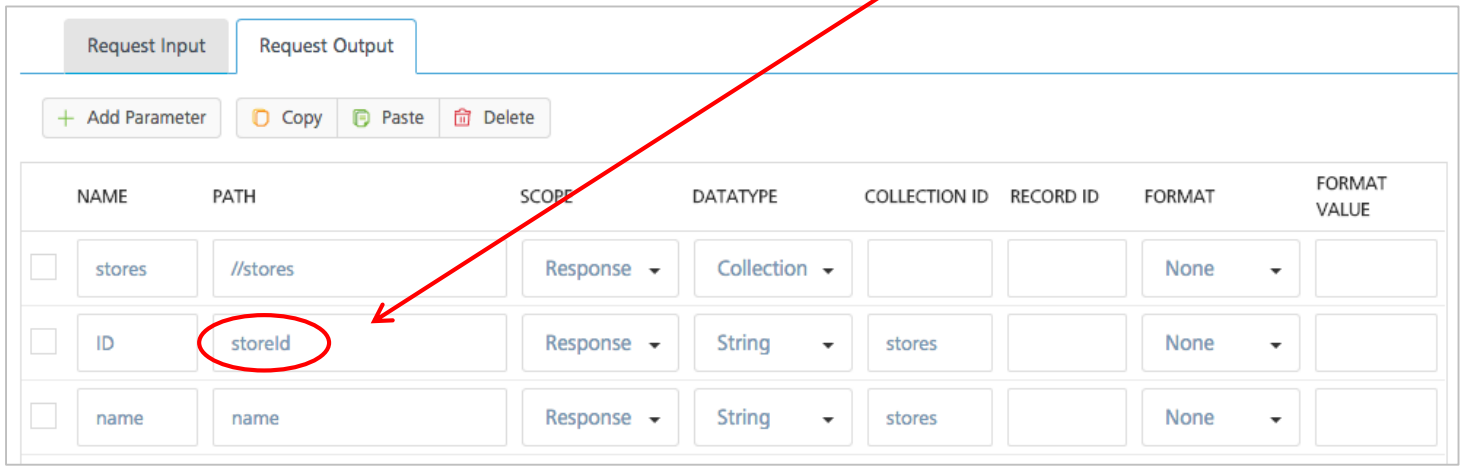
- In our example, let's look at the output parameter configuration to capture the list of returned stores:



Each store is a JSON object in the "stores" array

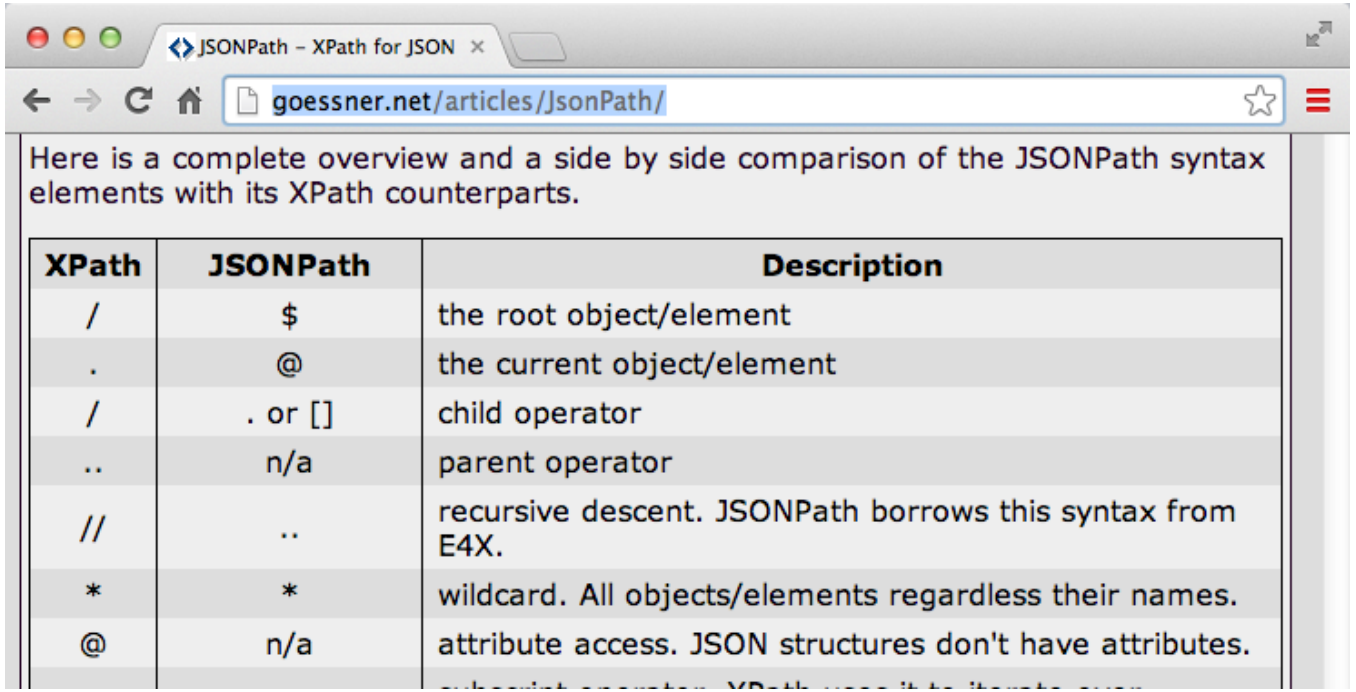
The data for each store is accessed by it's name

- Here is the output parameter configuration:



JSON Path Resources

- We saw that the basic path statements are same as XML
  - There are differences if you start doing more complex work
- There are many sites on the web with examples.
  - For example: http://goessner.net/articles/JsonPath/

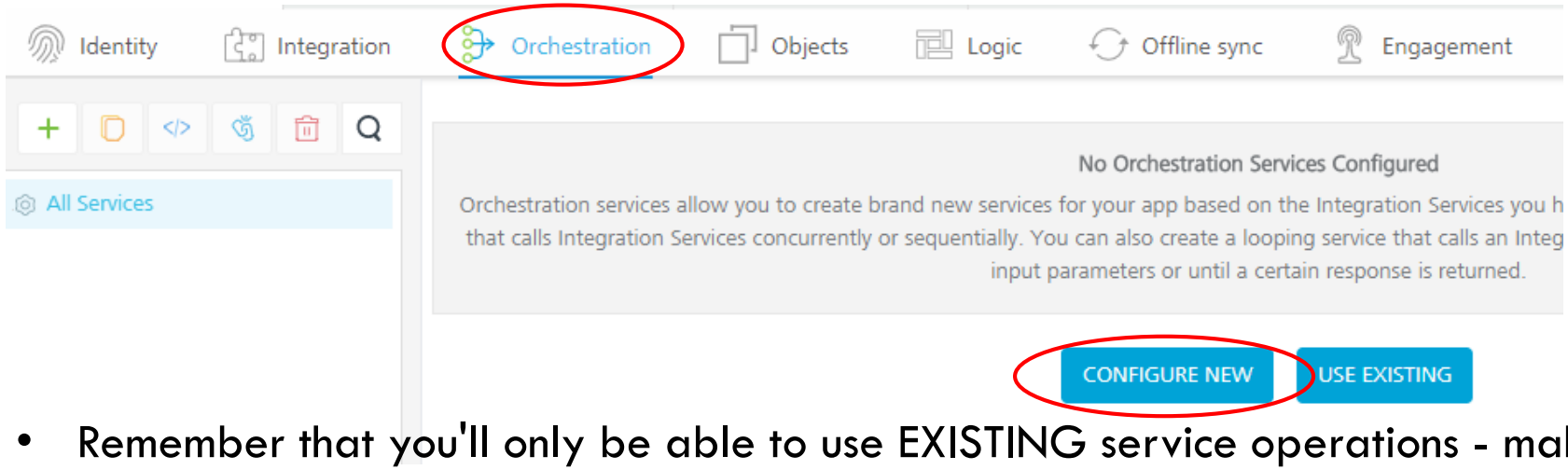


Orchestration Services Overview

- Orchestration services let you use existing Integration services to create more complex data management scenarios
- There are 2 types of Orchestration services:
  - Composite services - where you call more than 1 Integration services either sequentially or concurrently
  - Looping services - where you call 1 Integration service in a loop
- In either case, all the results will be gathered before sending them, as a batch, back to the mobile application
  - The advantage of this approach is that you don't have to manage calling the services from the mobile application
    - A single call to execute the Orchestration service lets Kony Fabric do all the calls and gathering of the data

Orchestration Services

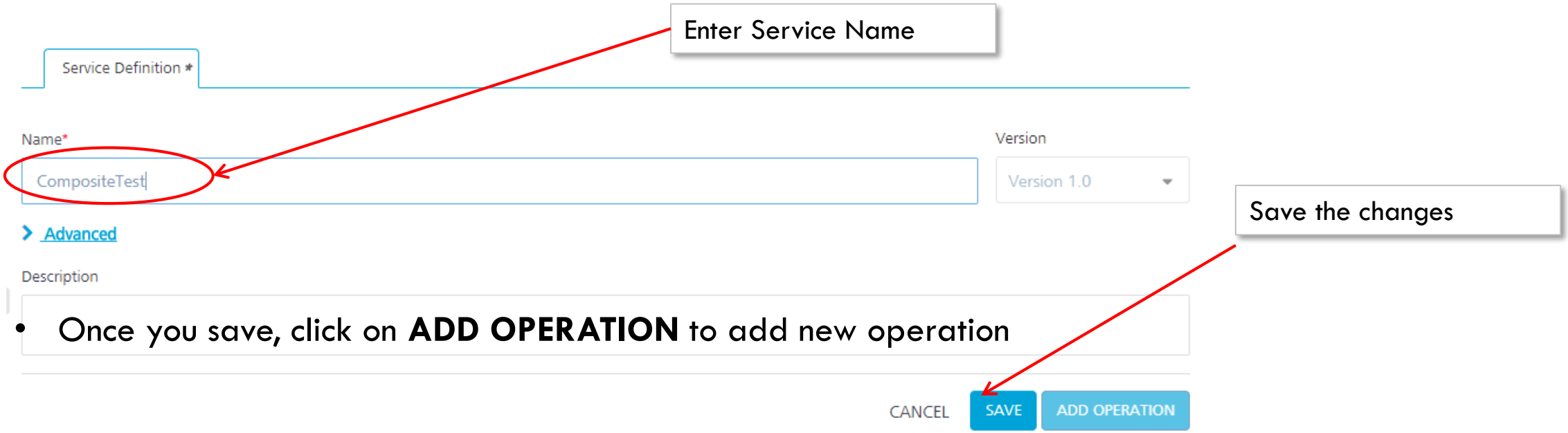
- Orchestration services are configured by assembling already defined Integration service operations
  - Note that you'll have to authenticate at a level that allows all your selected operations to run
- Here's how you create an Orchestration service:



- Remember that you'll only be able to use EXISTING service operations - make sure you've done that before

Composite Service Configuration

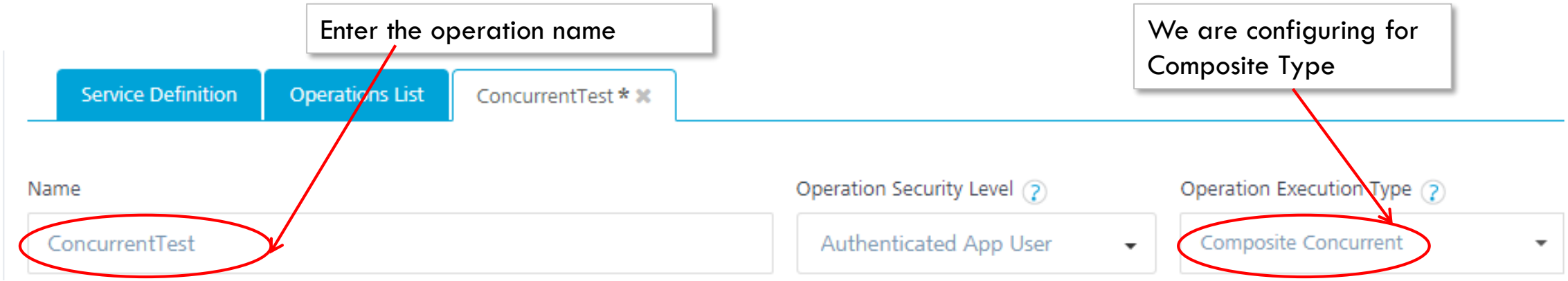
- Here is the first part of the configuration:



- Once you save, click on **ADD OPERATION** to add new operation

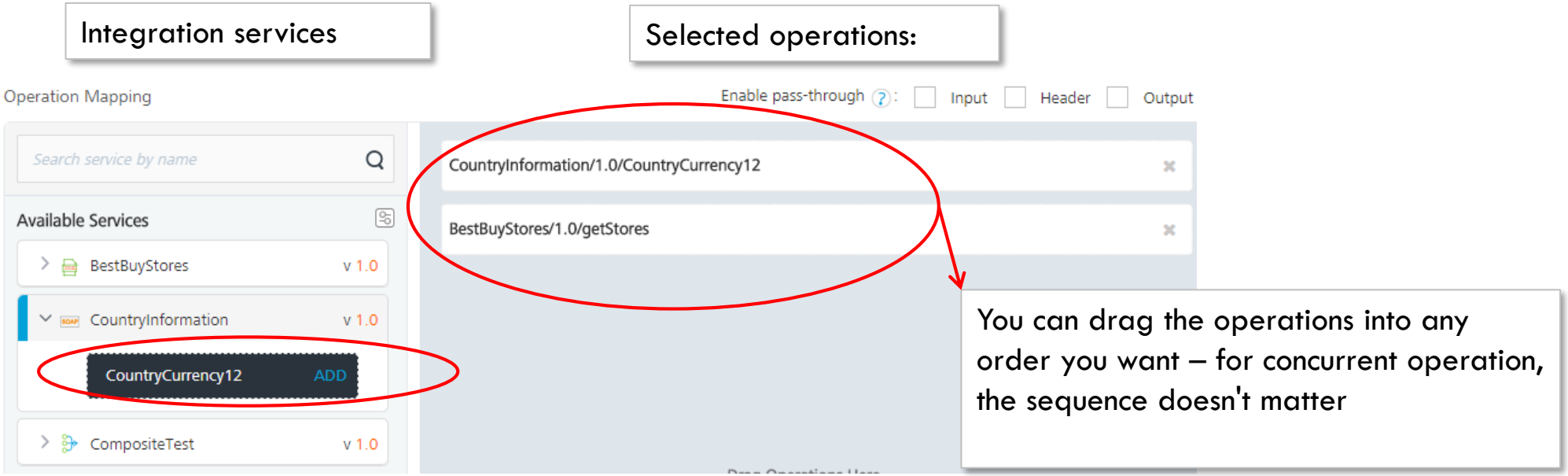
Composite Service - Operation Configuration

- There are 2 choices for Service Execution Mode:
- Concurrent – Runs all the selected operations at the same time
  - Results from all operations are returned to the client when the last operation has finished running
- Sequential – Runs the selected operations in a sequence, one after another; This is used to take the output of one service and feed it as input to another
  - Results from the last service in the sequence are returned to the client



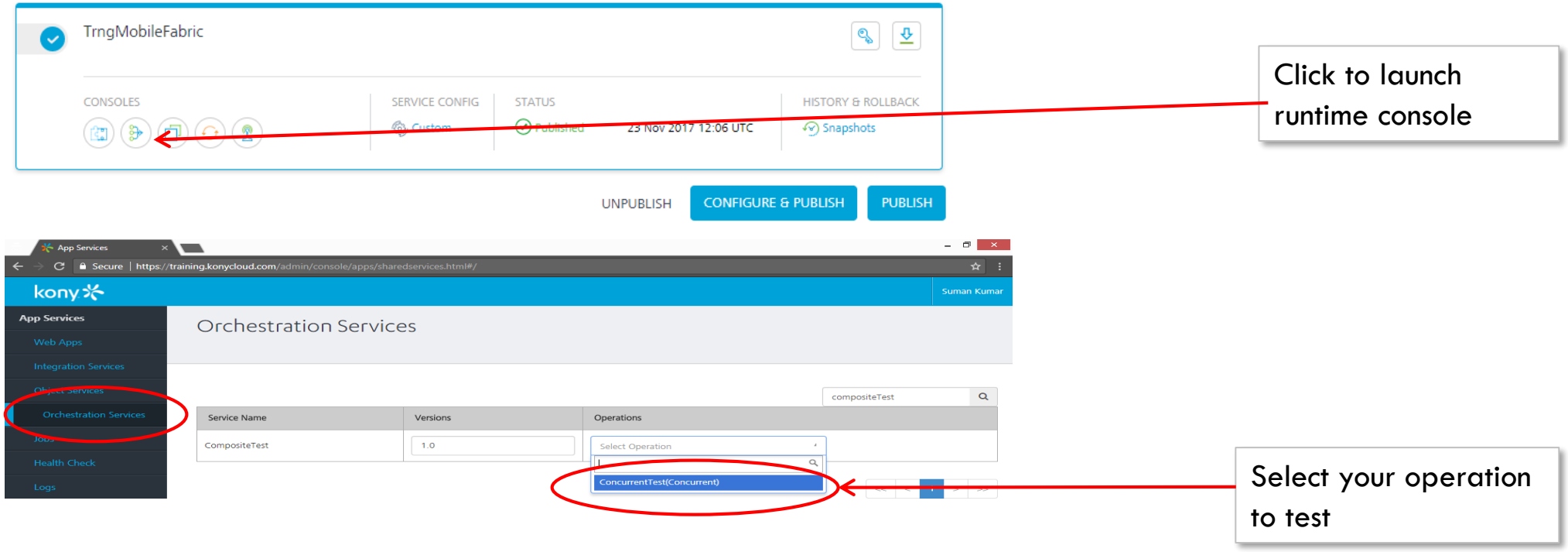
Concurrent Service Configuration

- Configuring a concurrent service:
- Here is how you add your operations to your composite service:



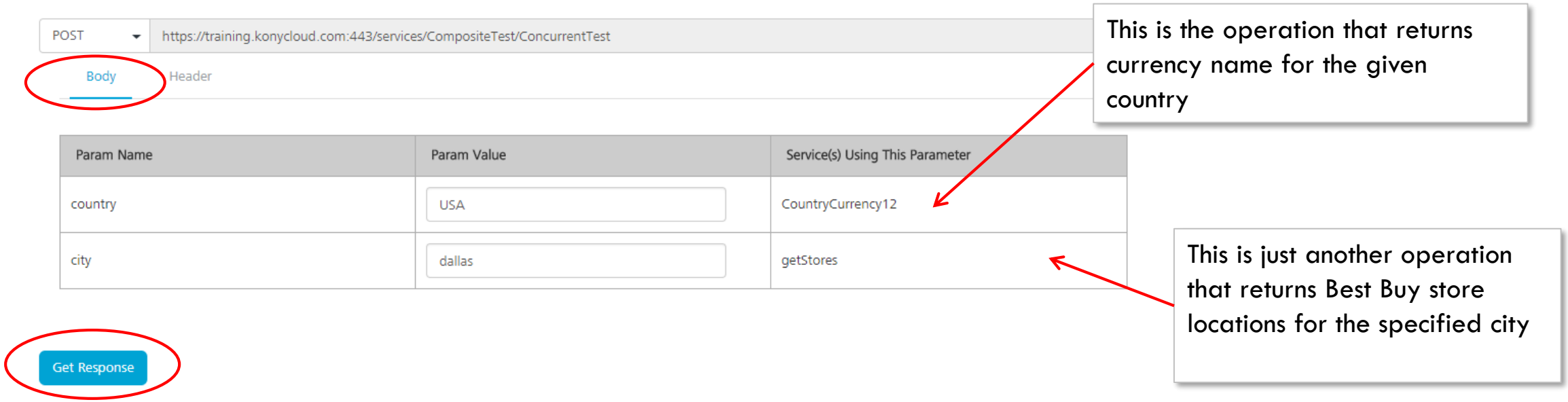
Concurrent Service Testing

- There is no way to test your composite service from the design screens (like we do for operation definition)
  - You must first publish your app
  - On the Publish tab, click the runtime console - is for Orchestration:



Concurrent Service Testing (cont'd)

- If there are any input parameters, you need to specify values:



- When we get the response (shown on next slide) - you'll have ONE result set that contains all the returned data for each operation
  - Note: if you have output parameters with the same name - you'll lose data
- Let's look at the data...

Concurrent Service Response

- The response can be huge, here is ours:

Response Body


```
{
   "currencyName":"Dollars",
   "stores":[
   {
      "name":"PARK LANE TX"
   }
,
   {
      "name":"BBA035DAL1"
   }
,
   {
      "name":"DALLAS GALLERIA"
   }
```

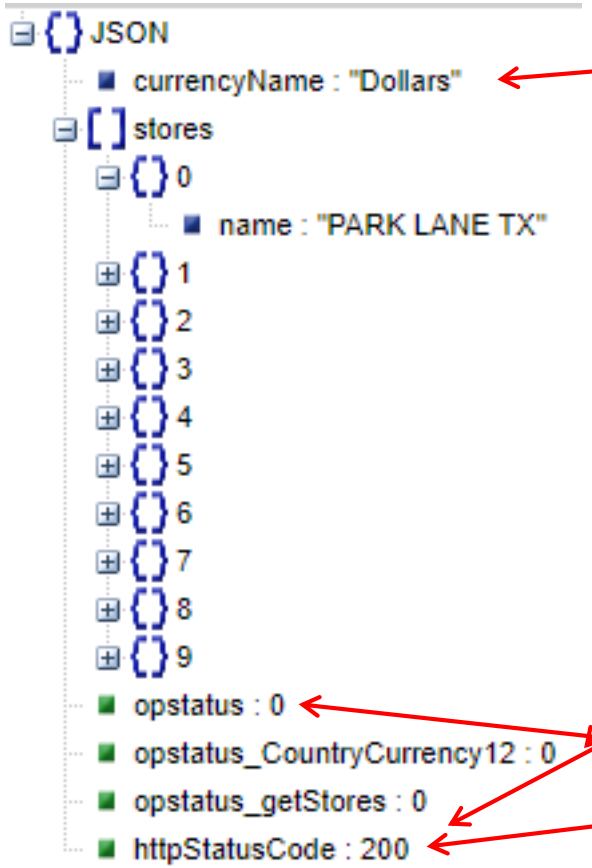

The runtime response will always be in JSON format

We start off with our currency data – but there is way too much data to show in this window…

- It's hard to show/see what is going on with these results
- I like to use a JSON viewing tool that helps me see the data and it's structure better
  - Find the tool at:
  - http://www.jsoneditoronline.org/
- Using that tool, let's examine what data we got back…

Concurrent Service Response (cont'd)

- Here's what the data looks like in the tool: Leaving the collections of data (stores and articles) compressed to show the data structure



Here is currency name for the given city

Here is our collection of stores

This is the individual opstatus for our CountryCurrncy12 operation

This is the individual opstatus for our getStores operation

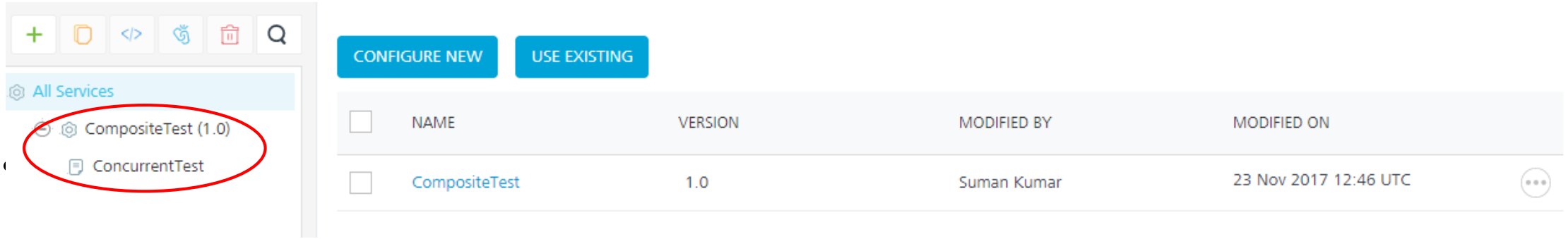
This opstatus and status code are for the composite service - 0 and 200 means it executed successfully

Note: The order is not guaranteed.

Invoking Orchestration Services

- When calling Integration services, we would first get an instance of our service definition and then call our operation

- Calling an Orchestration service works exactly the same way:



Invoking Orchestration Services (cont'd)

- Below is the sample code:
  - When using the SDK, the service name is “CompositeTest” and operation name is “ConcurrentTest” in this example:

```
//here is where we left off – getting our service definition
integrationObj = client.getIntegrationService("CompositeTest");

operationName =  "ConcurrentTest";
data= {"newsType": "national","city":"dallas"};

headers= {};
integrationObj.invokeOperation(operationName, headers, data, opSuccess, opFailure);
```

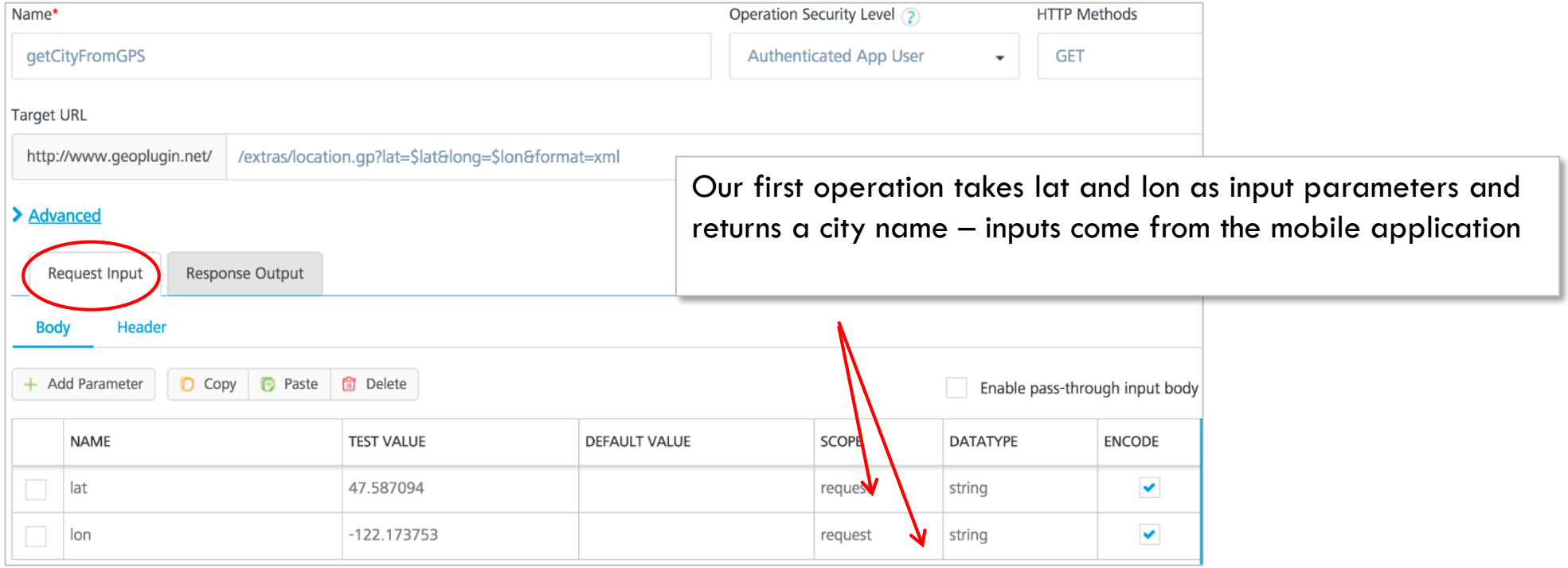
  - So, it's really no different than calling a normal Integration service

Sequential Composite Services

- When you set your Composite service to Sequential, they will now run one after another
  - This means that the FIRST operation will be given input parameter values
  - The output of first operation becomes the input for second operation and so on...
  - The last operation's outputs will be returned as the Sequential service final output
- In this case, Kony Fabric is handling the input/output parameters for those operations
  - We need to configure our services to indicate this by setting the scope to Session and
  - The output parameter name from the first operation must match the input parameter name from the next operation
- Let's look at an example:
  - Operation #1: takes GPS coordinates as inputs and returns the city name as an output
  - Operation #2: takes a city name input and returns a list of store locations in that city

Configuring First Operation

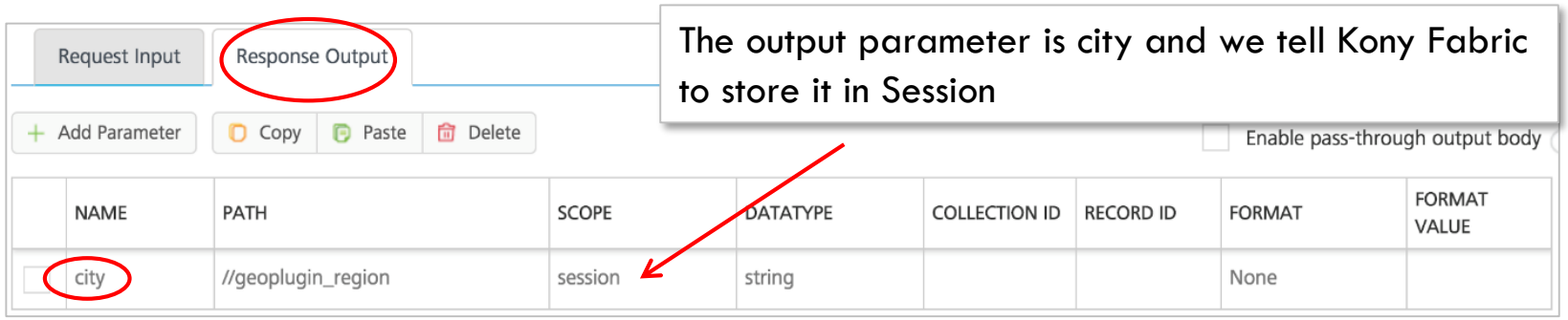
- First Operation: **Request Input**



Our first operation takes lat and lon as input parameters and returns a city name – inputs come from the mobile application

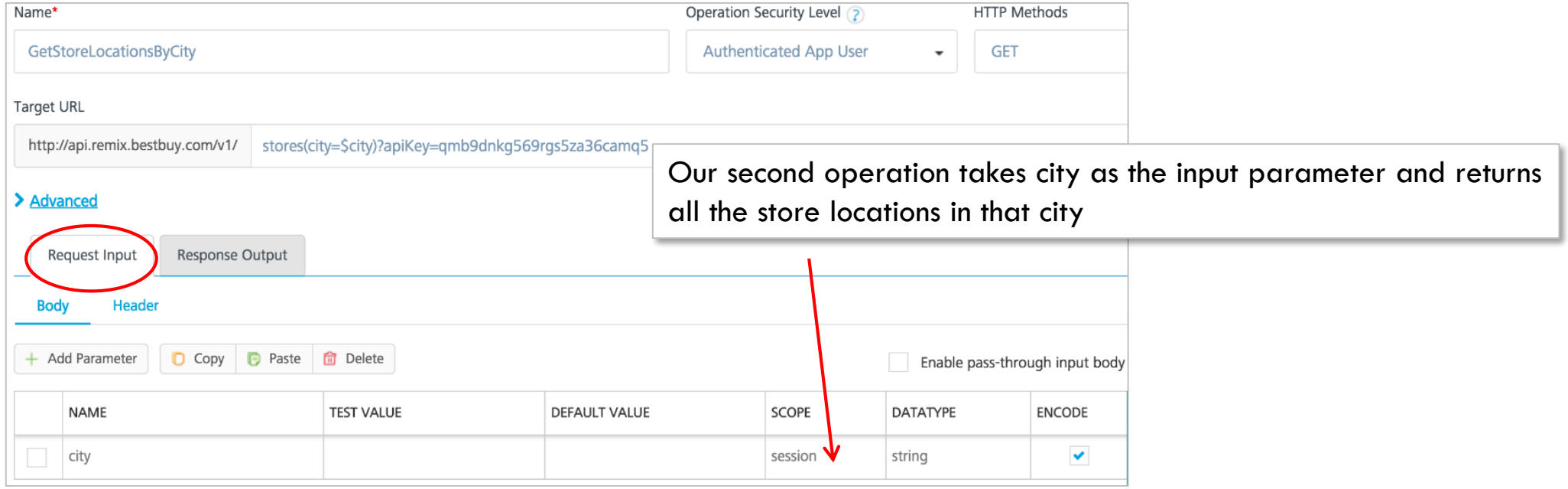
Configuring First Operation

- First Operation: **Response Output**



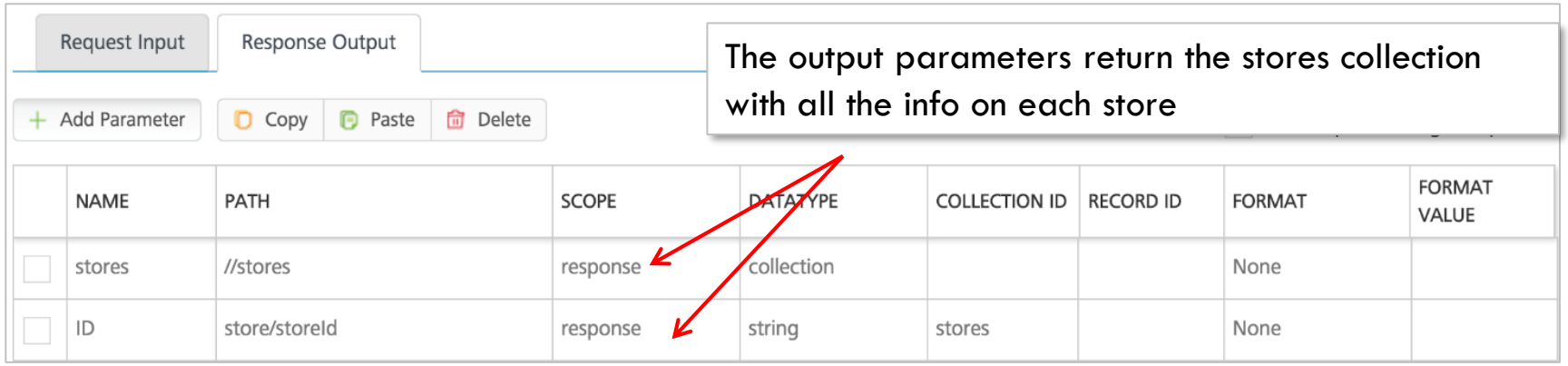
Configuring Second Operation

- Second Operation: **Request Input**



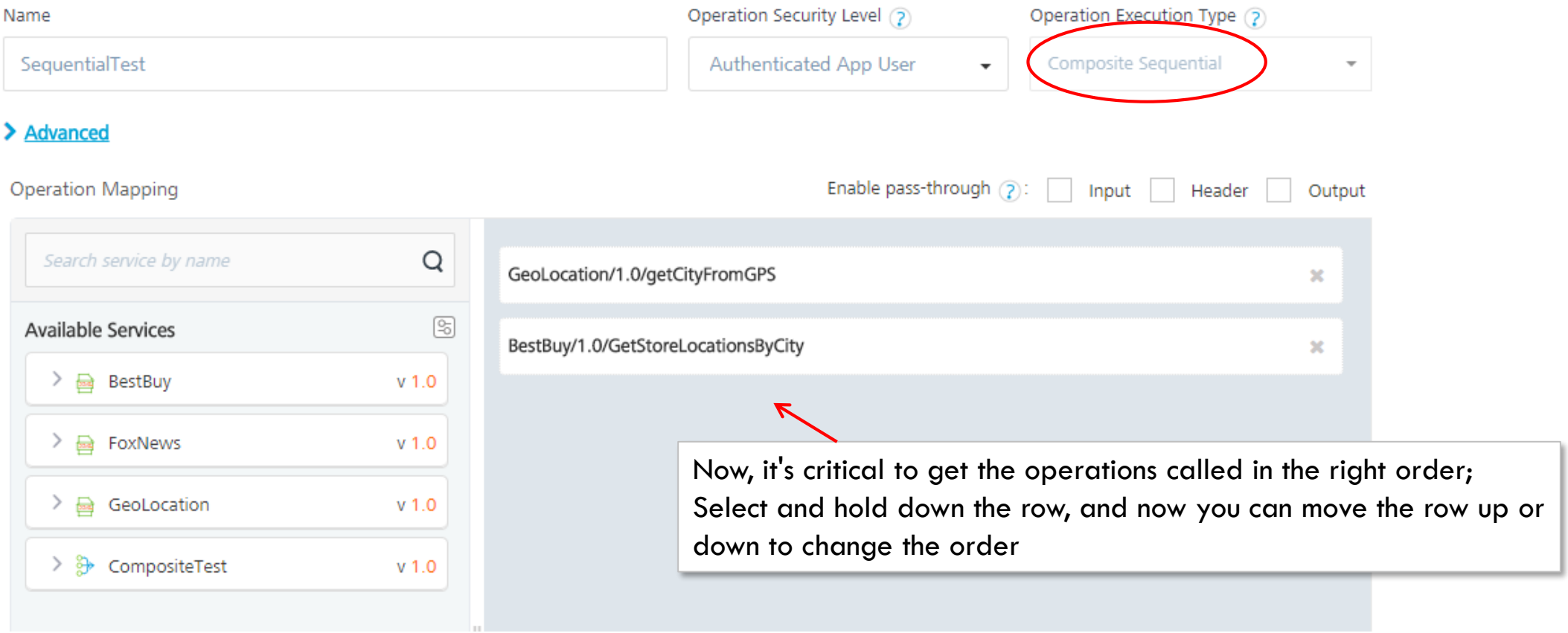
Configuring Second Operation

- Second Operation: **Response Output**



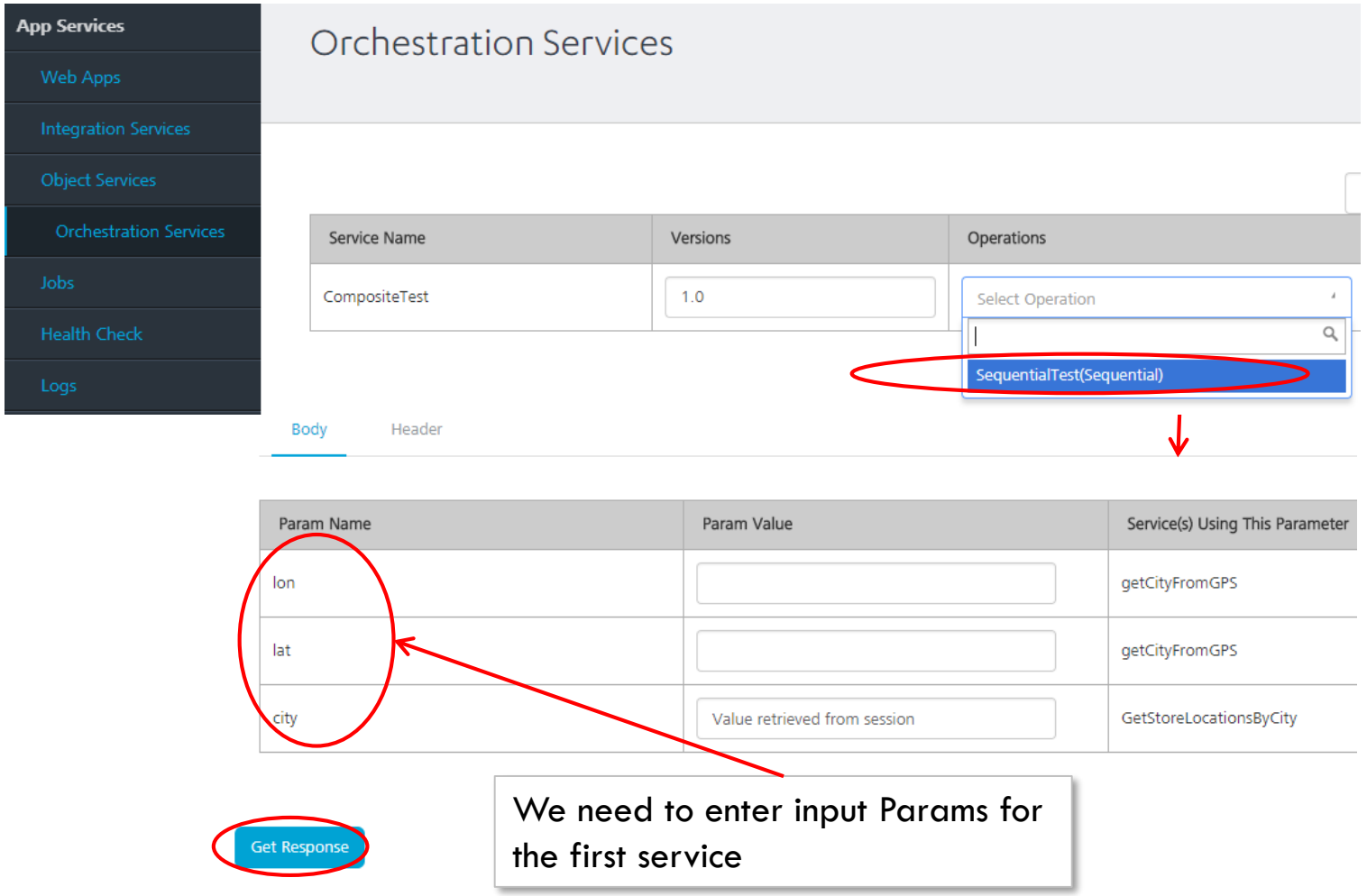
Sequential Service Configuration

- With the Integration service operations properly configured, we can now create our Orchestration service:



Testing in Runtime Console

- Now, we can test the sequential composite service in the runtime console

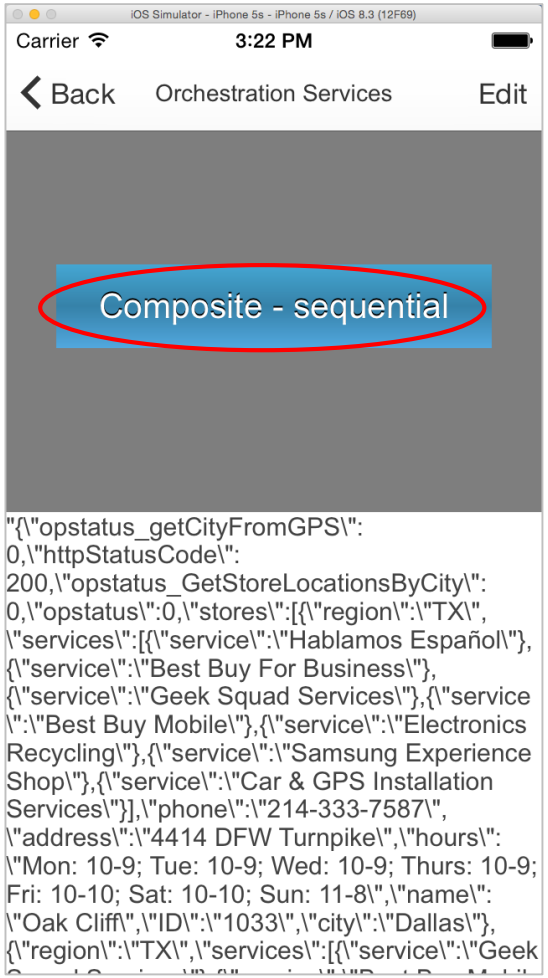


Composite Sequential - Exercise

- Let's try it!
- You'll first need to get your Integration services configured - use the following information:
  - GetCityFromGPS as an XML Service:
    - http://www.geoplugin.net/extras/location.gp?lat=$lat&long=$lon&format=xml
    - lat and long are your input parameter names
    - Output parameter (as SESSION) named city Here is the xPath: //geoplugin_region
  - GetStoresInCity as a XML Service:
    - http://api.remix.bestbuy.com/v1/stores(city=$city)?apiKey=<your_key>
    - city is the only input parameter (as SESSION)
    - replace <your_key> with a valid BestBuy developer key – get one at: http://developer.bestbuy.com/
    - create output parameters for the returned store location information (ID, address, hours, phone, etc.,)
  - Note: test each with Request/Response first and then set to Session for publish

Exercise – Composite Sequential

- Now you can configure your Composite service as sequential
  - First call GetCityFromGPS
  - Then call GetStoresInCity
  - Publish your App
- Now go add a new button for Composite – sequential and call your service
  - In your mobile application, you'll have to pass in some GPS coordinates
    - Try: 32.778820/-96.796914 for Dallas, Texas (lots of Best Buy stores there)
  - log the outputs so you can see your results



Server Side Programming

kony Stay Ahead

Server Side Programming Overview

- Very often in our applications we will come across situations where we want to manipulate data before we send it to the backend datasource and also after we get the data from the backend.
- You have a choice:
  - Do the processing on the device OR
  - Use Preprocessor / Postprocessor on the server
- Preprocessor
  - Used to manipulate the data before we call be backend service.
- Postprocessor
  - Used to manipulate or add new data to the response data that we get from the backend service.
- Preprocessor and Postprocessor can be achieved using
  - Java
  - JavaScript

Preprocessor - Use cases

- Some uses of the Preprocessor are:
  - Input fields validations - validate inputs are proper
  - Add some data from the session - retrieve data stored on the server and use it as input parameters
  - Reading some data from a properties file on the server and use it as input parameters
  - Logging some/all the inputs - use for reporting purposes
  - Implement version control - don't let the user continue using the app if there is a newer mandatory version available
  - Session recycle - clearing out session during login/logout service calls
- A Preprocessor is invoked before a service execution and can return without executing service if needed
- Preprocessors can be implemented in **Java** or **JavaScript**

DataPreProcessor2 Interface

- **com.konylabs.middleware.common.DataPreProcessor2** is the interface provided to implement a Preprocessor
- Here are the standard imports, the interface and the method to be overridden:

```
import com.konylabs.middleware.controller.DataControllerRequest;
import com.konylabs.middleware.controller.DataControllerResponse;
import com.konylabs.middleware.dataobject.Result;
import java.util.HashMap;
import com.konylabs.middleware.common.DataPreProcessor2;
public abstract interface DataPreProcessor2
{
  public abstract boolean execute(HashMap inputMap,
                                  DataControllerRequest request,
                                  DataControllerResponse response,
                                  Result result) throws Exception;
}
```

Note: the imports come in for you when you implement the interface - we're showing them here so you're aware of them - they're added automatically for you

Override Methods in DataPreProcessor2

- The interface contains a single method that you need to override: **execute**

  public boolean execute(HashMap inputMap, DataControllerRequest request, DataControllerResponse response, Result result)throws Exception

  inputMap: contains the input parameters from the client

  request: the object that gives you access to session data

  response: used to set device-headers and device-cookies

  result: contains any values returned to the device IF the service is NOT called

  - The return value should be true if you want the service to be called or false if you want to abort the service call
  - Note: if returning false, you'll want to communicate data back to the device. This is why the result object is available in this method

Preprocessor - inputMap Parameter

- **inputMap** is a Hash Map based implementation of the Java HashMap type hence you can use all the HashMap APIS for soring and retrieving data
- inputMap contains all the service input parameters that are sent from the device:
  - Here's a simple example that simply print out all the values

```
Iterator inputIt = inputMap.entrySet().iterator();
while(inputIt.hasNext()) {
        Map.Entry pairs = (Map.Entry)inputIt.next();
        LOG.debug("    &&&&&  KEY : " + pairs.getKey());
        LOG.debug("    &&&&&  VALUE : " + pairs.getValue());
}
```

- NOTE: The actual service call will use the data in inputMap. Change the data here if you want the service to use your new value

Preprocessor - DatacontrollerRequest

- This object gives us access to the request configuration data and it includes the following:
  - Attributes - key/value pairs of data including:
    - The server's Cache ID
    - The encoding scheme
    - Anything you want - a place to store data to communicate with the Postprocessor
      - You can add attributes and/or read attributes as part of your logic scheme
  - Parameters - the input parameters and their data.
    - This also includes: App ID, client platform, app version, etc - you can use this for reporting purposes
  - Session - access to the user's session on the server that is active beyond the single service call

DataControllerRequest - Attributes

- Let's take a look at Attributes first... here are the get methods for attributes:

  Iterator<String> getAttributeNames(): is used to return all the attribute key names

  Object getAttribute(String name) : is used to return the data for the attribute under the key name

  Boolean containsKeyInRequestContext(String name): is used to check if the key specified by name is found in the attributes

- Here is how we set attributes:

  void setAttribute(String key, Object value) : is used to store the specified value under the key in attributes

  E.g., request.setAttribute("nowTime", System.currentTimeMillis());

DataControllerRequest - Parameters

- The parameters collection contains a copy of the original input parameters as well as some other information about the service call
- The parameters methods work the same as for attributes. Here are the methods:

  Iterator<String> getParameterNames() : is used to return all the parameter key names.

  Object getParameter(String name) : is used to return the data for the parameter under the key name

  Boolean containsKeyInRequest(String name) : is used to check if the key specified by name is found in the parameters

  - Note: the attributes used the containsKeyInRequestContext method - the word "Context" is added to indicate it's part of the service call's "session"

DataControllerRequest - Session

- The request object was for the scope of the service call
  - E.g., we can communicate between the preprocessor and the postprocessor by setting/getting attributes
- Session is for the scope of the entire user's session - namely for as long as the user is using the app/mobile web site
  - By default session expires after a period of no activity - at this point the session is deleted
  - If the user starts using the app/mobile website again, a new session will be created
  - Session timeout is configured in the server properties
- Storing excess data in the session will impact performance of the application.
- Any custom object placed in the session must be serializable. If data is common across the user base, then store the data in application context (i.e., Servlet Context).

DataControllerRequest - Session

- Session is accessed through one of the request object methods:

  Session **getSession()** : Returns the session associated with the request. if there is no session creates a new session and returns its reference. For example:

  For example: Session = request.getSession();

- Let's look at the methods for Session:

  containsKey (String key) - checks to see if there is an attribute for the key - it's best practice to check first, before accessing the data

  getAttribute(String key) - retrieves the attribute value for the key

  setAttribute(String key, Object value) - sets the attribute value for the key

  removeAttribute(String key) - removes the key from Session and all data associated with it

  invalidate() - will delete the user's session and all data associated with it.

  to immediately re-create it use **getSession(true)**

Session Example Code

```
//get the current session
Session session=request.getSession();
//see if there is an attribute for "city"
Boolean flag = session.containsKey("city");

if (flag) {LOG.debug("city is: " + session.getAttribute("city"));};
//change the value to Boston (or create it if the key doesn't exist)
session.setAttribute("city", "Boston");
//now remove that attribute from session
session.removeAttribute("city");

//delete the user's session
session.invalidate();
//start a new user session
session=request.getSession(true);
```

Preprocessor - response and result

- It's possible that the Preprocessor doesn't call the service (returns false) and still returns a result.
  - E.g., implementing a caching scheme
    - First time, you call the service that returns 1000 pieces of data so you return the first 100 and put the rest in session
    - Each subsequent call retrieves the next batch of 100 records from session without calling the service
- The returned result object can contain valid data
- The response object is also returned to the client containing data like the status, headers, cookies, etc
- We will discuss the methods available in response and result object during Postprocessor

Exercise - Preprocessor

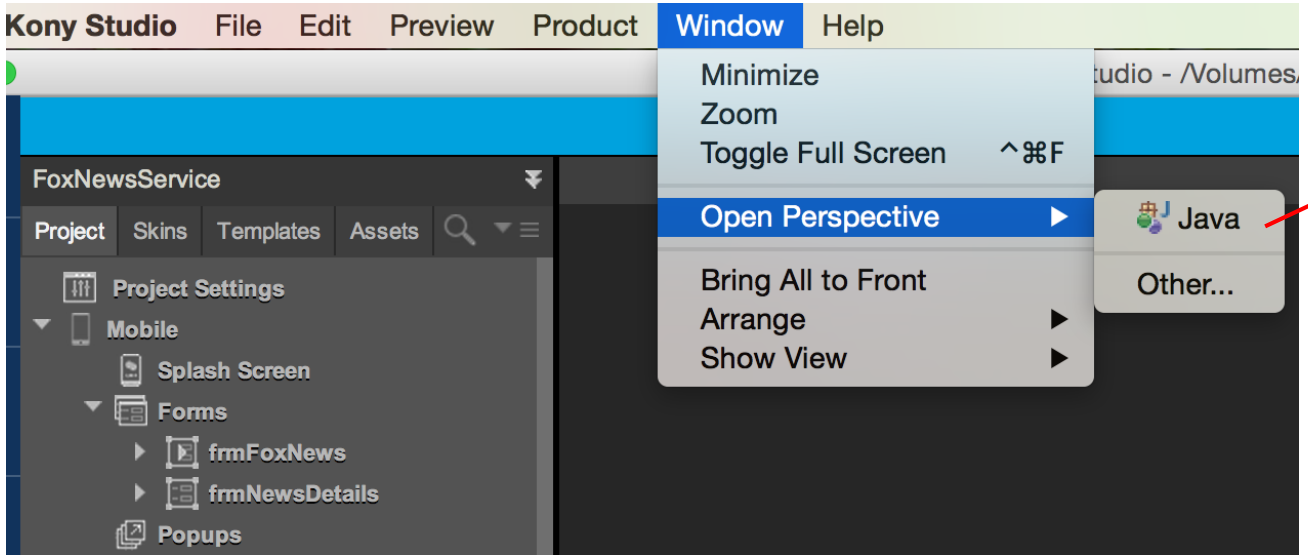
- Let's use FoxNews to test how session works in the Preprocessor
  - Remember that Fox News takes one input parameter - newsType
- Here is the logic that we need to implement
  - If there is a valid newsType, use it and store it in session (overwriting any previous value)
  - If there is NO newsType passed in, then we need to check the value in session:
    - If there is a value, use it to invoke service and then also remove it from session (so you only get 1 attempt without passing in a valid type)
    - If there is NO value, then don't execute the service (i.e., return false) and return no results
    - Make use of logging to check your logic

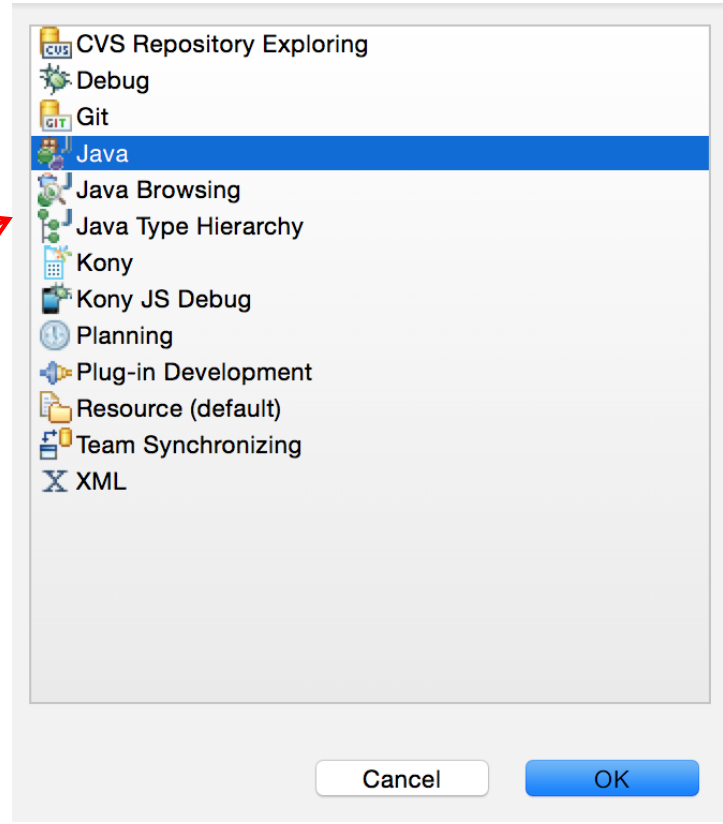
Exercise - Java Preprocessor Execution Steps

- Create Java Project
- Add necessary jar files to the project
- Create a new java class implementing the DataPreProcessor2 interface and override the **execute** method
- Import the required packages
- Write your logic in the **execute** method
- Create a Jar file of the preprocessor java project
- Add the jar file at service level
- Call the java class in your operations
- Publish the app and run your client application

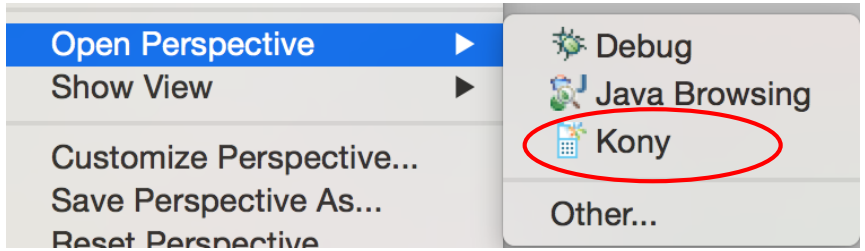
Preprocessor - Java Perspective

- Open "Java" perspective:



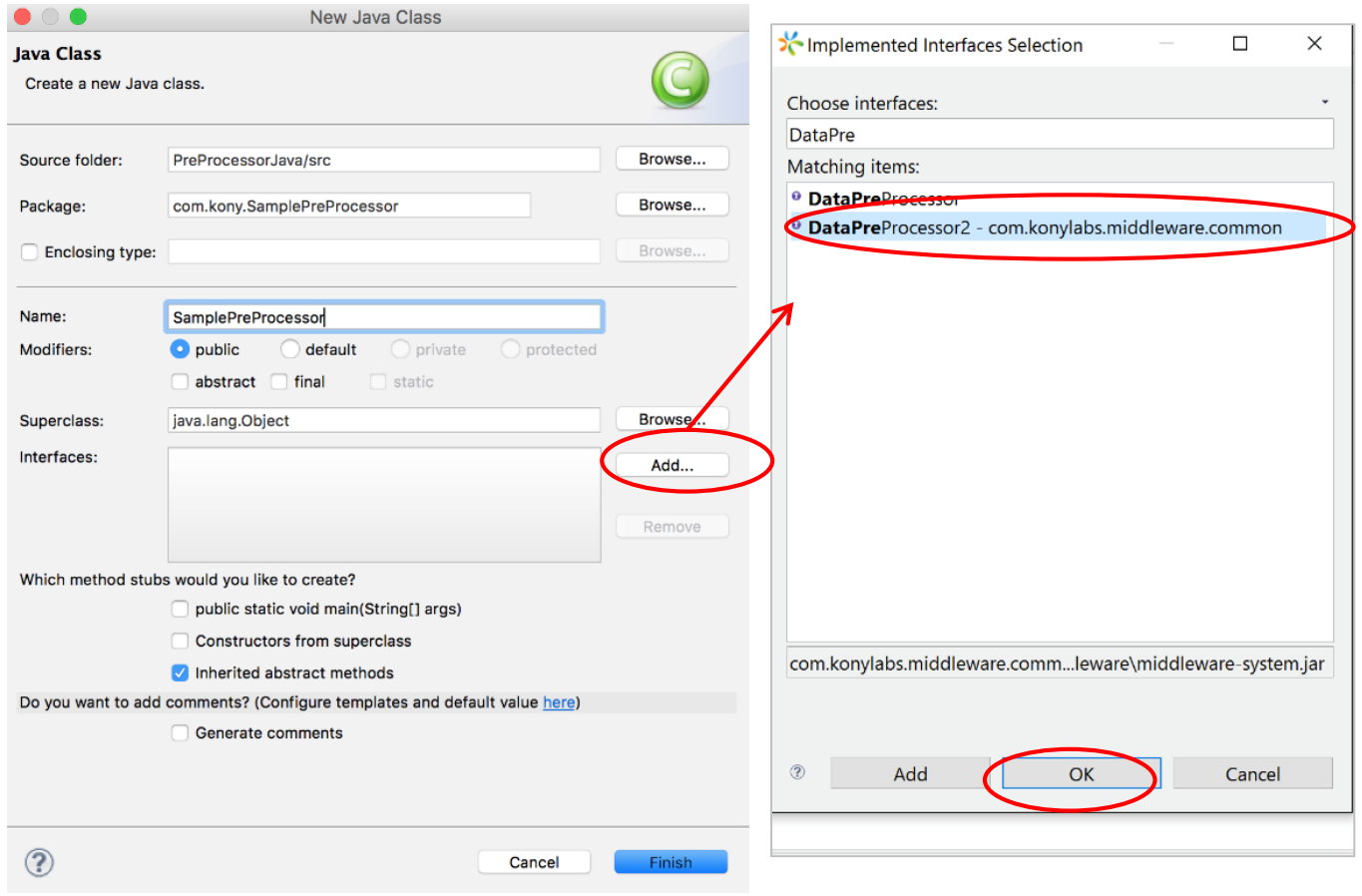


- To switch back to Kony perspective:



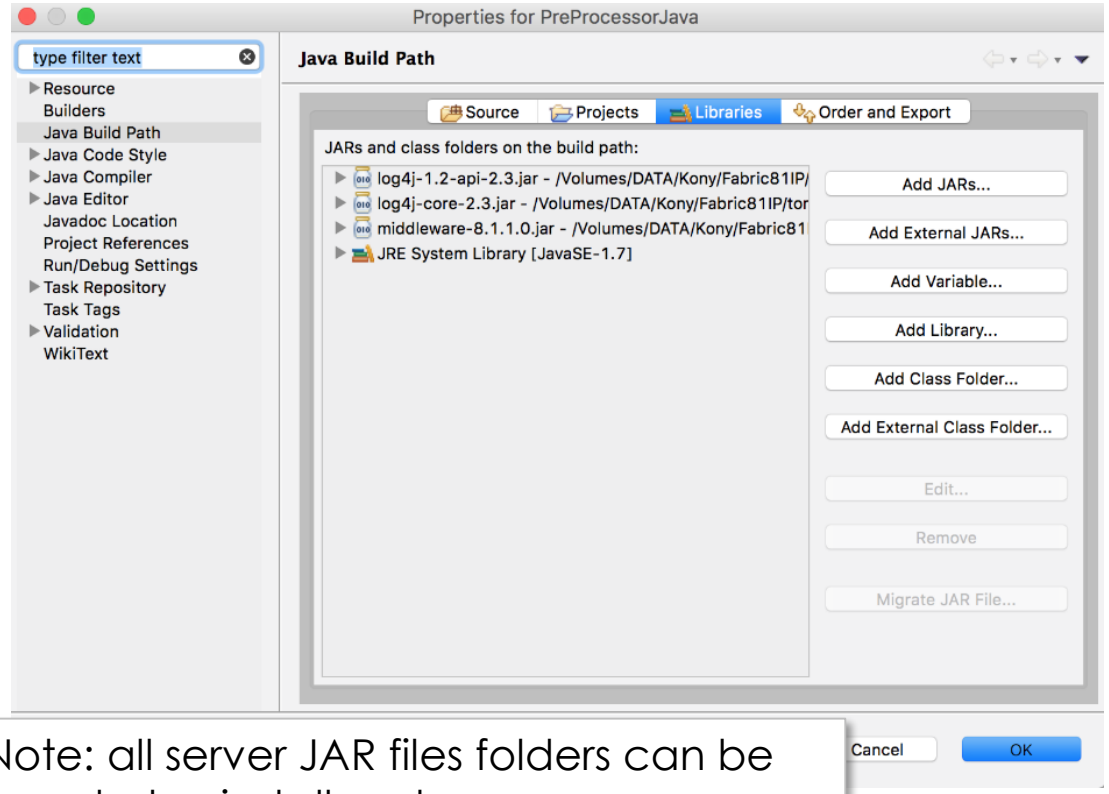
Implementing Interfaces

- Create a Java Project
- When creating your new class, you can add the interface you want to implement
- The interfaces we'll use are:
  - DataPreProcessor2
  - DataPostProcessor2
- Starting to type in the interface name shows the filtered list - pick the one you want when you see it



Adding External Jar files

- When creating the new Java project, you'll be able to specify external references
- We need to add:
  - **middleware-8.1.1.0.jar** (in the server's /tomcat/webapps/services/WEB-INF/lib folder)
  - **log4j-1.2-api-2.3.jar** (in the server's /tomcat/webapps/services/WEB-INF/lib folder)
  - **log4j-core-2.3.jar** (in the server's /tomcat/webapps/services/WEB-INF/lib folder)
- After creating the project, you can get back to this dialog by:
  - right-clicking the project...



Note: all server JAR files folders can be found at: <install root location>/tomcat/services/WEB-INF/lib for a default local Kony Fabric server installation

Packages and Classes

- Here is the hierarchy of objects you see in a completed project that has one of each type of classes (a Preprocessor and a Postprocessor):

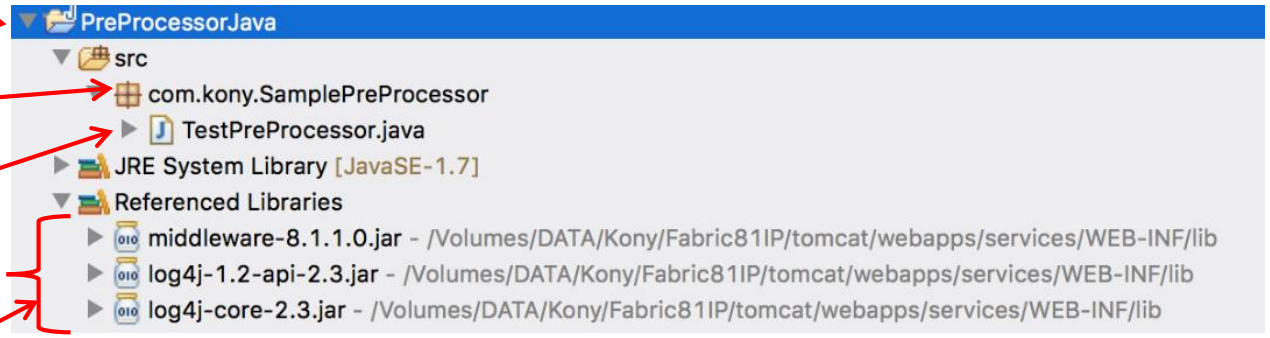
Java Project

Java Package

Java Classes (where our code goes)

Referenced external jar files –libraries

Example Java project structure



- Our job will be to create this project structure so we can write our code

Default Argument Names

- When implementing an interface, eclipse will give you default argument names (arg0, arg1, etc)
  - For example, here's what we get when we implement the interface:

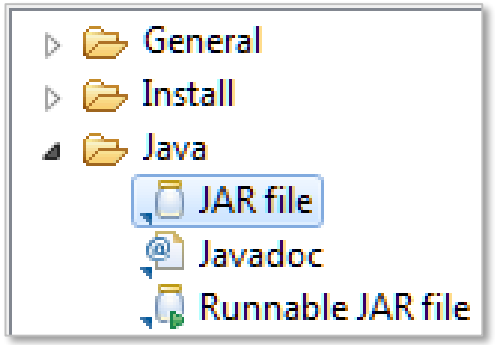
```
@Override
public boolean execute(HashMap arg0, DataControllerRequest arg1,
        DataControllerResponse arg2, Result arg3) throws Exception {
    // TODO Auto-generated method stub
```

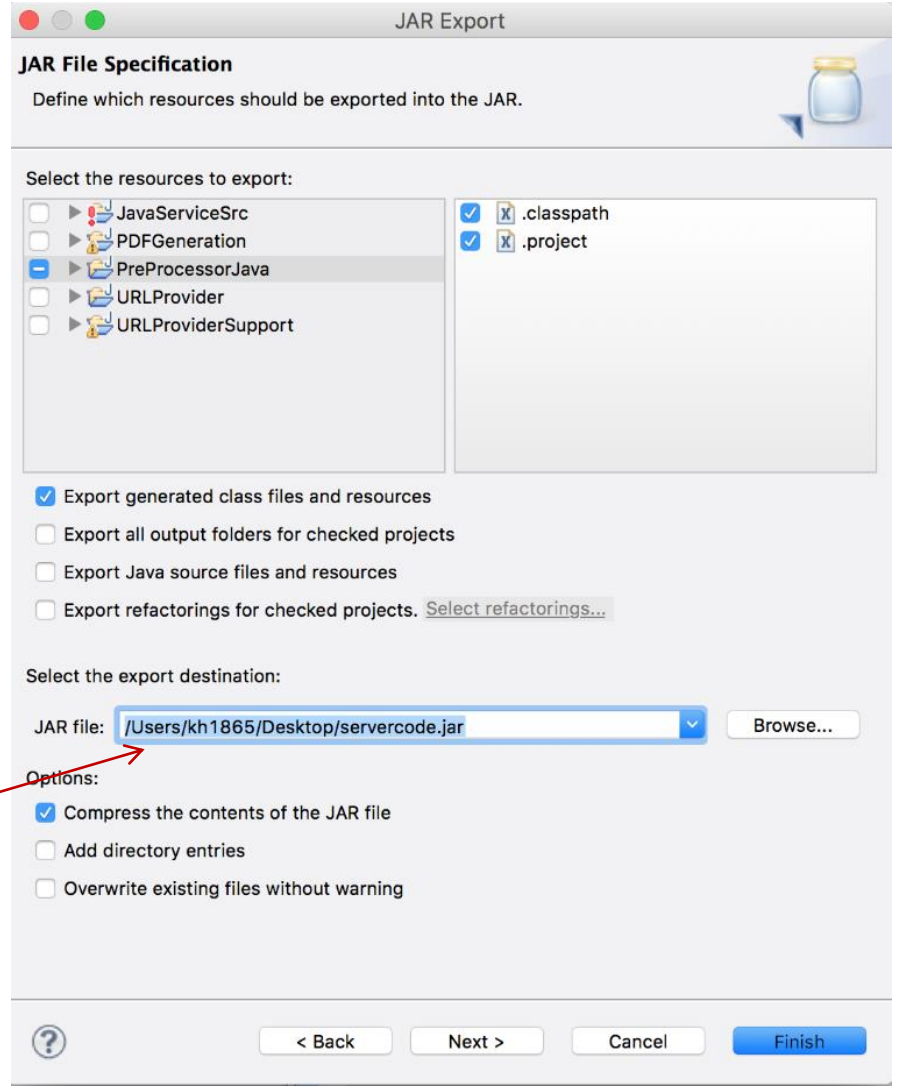
- We will give them more meaningful names so in our code it's clear what we are doing
  - For example, here we've given the arguments better names
  - Note: we'll use friendlier names in all our slides going forward

```
@Override
public boolean execute(HashMap inputMap, DataControllerRequest request,
        DataControllerResponse response, Result result) throws Exception {
    // TODO Auto-generated method stub
```

Creating/Re-creating the JAR File

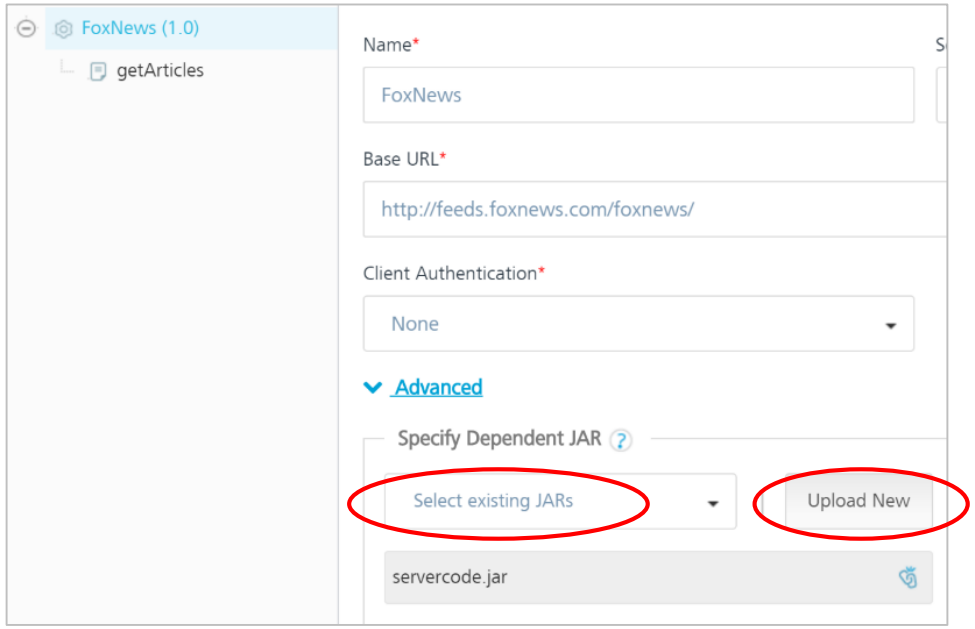
- When you are done with your code, to deploy it, you'll need to create a JAR file
- Right click the project and choose Export...
- From there pick Java -> JAR file:
- Pick a name and Browse where you want to put it
  - Next time it'll pre-populate that name and location or you can change it if you want it elsewhere

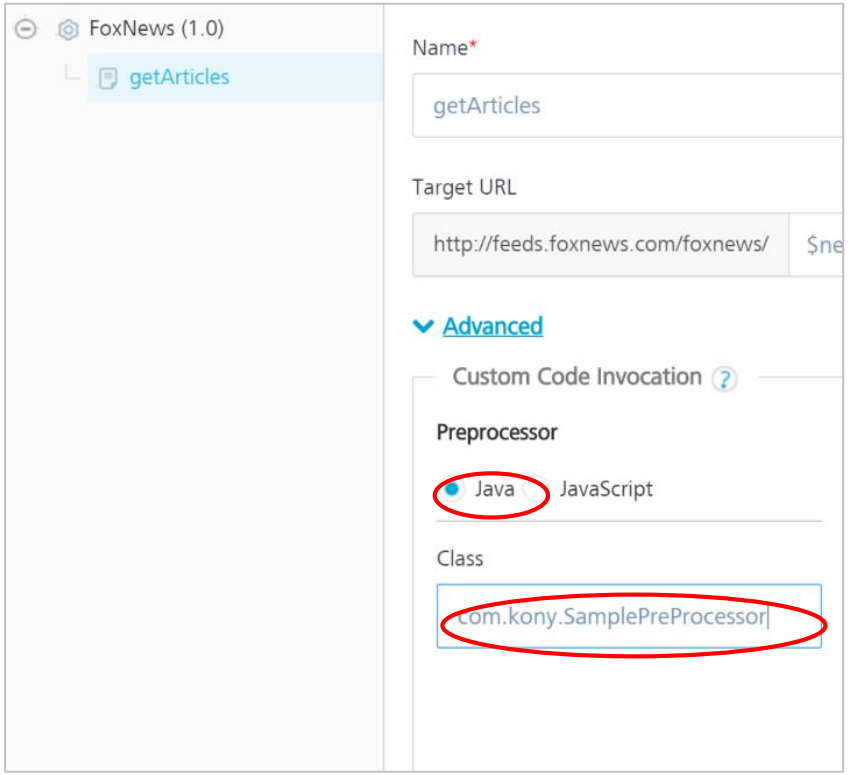




Uploading JAR File In Kony Fabric

- Import or Open the appropriate Fabric App in the Kony Fabric
- Open the Service definition Page and upload the jar file that you created

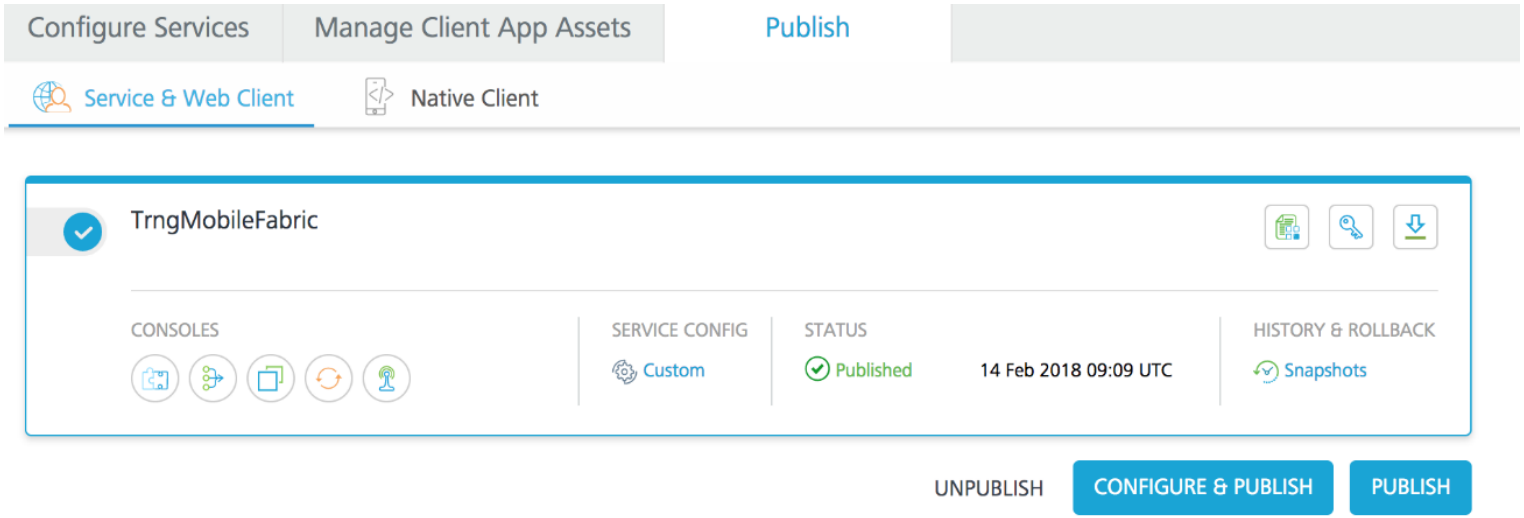




- Go to your operation, click on advanced and choose Java option under Preprocessor and give the filly qualified name of the Java class

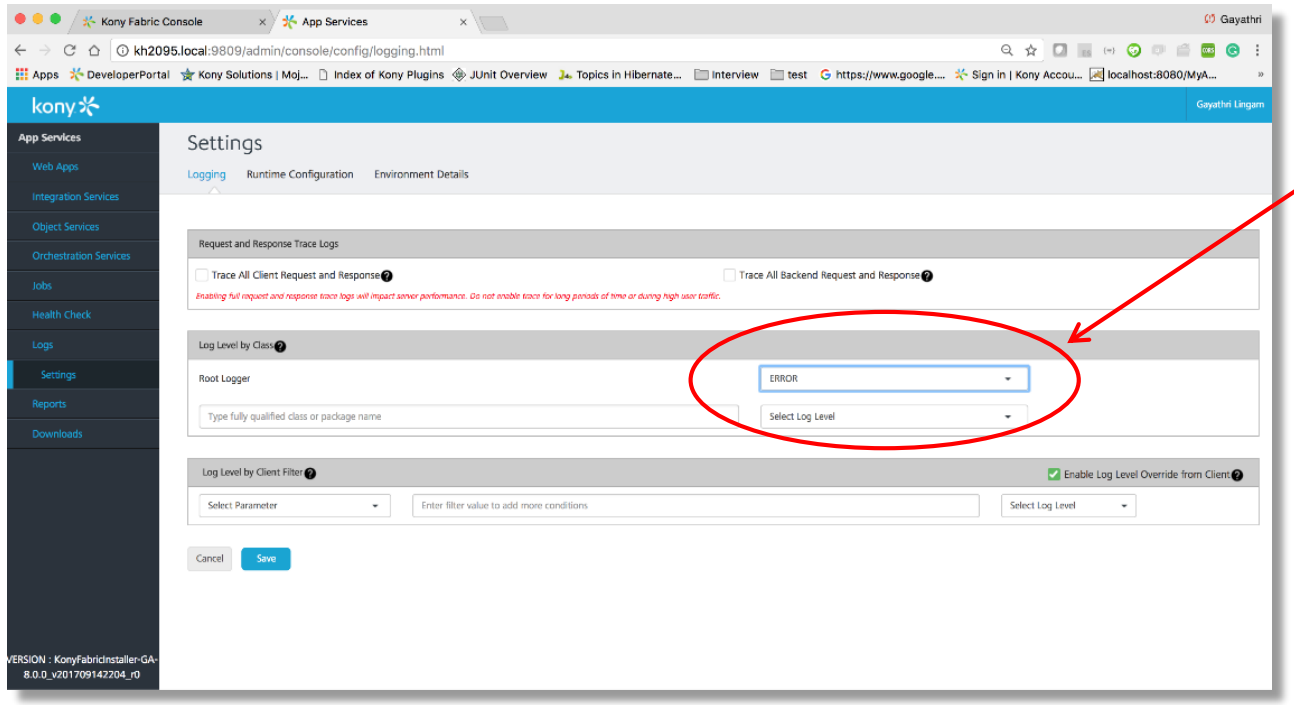
Uploading JAR File In Kony Fabric - Publish

- Once the JAR file is uploaded, you must publish the changes in Kony Fabric:



Logging and Log4j Properties File

- Log4j is a popular logging package for Java
  - The package is distributed under the Apache Software License & Kony Fabric uses Log4j by default
- Here is the log level configuration for the Kony Fabric server: Go to App Services runtime console and change the log level.



Change the Server Log Level to DEBUG and other options to Enabled

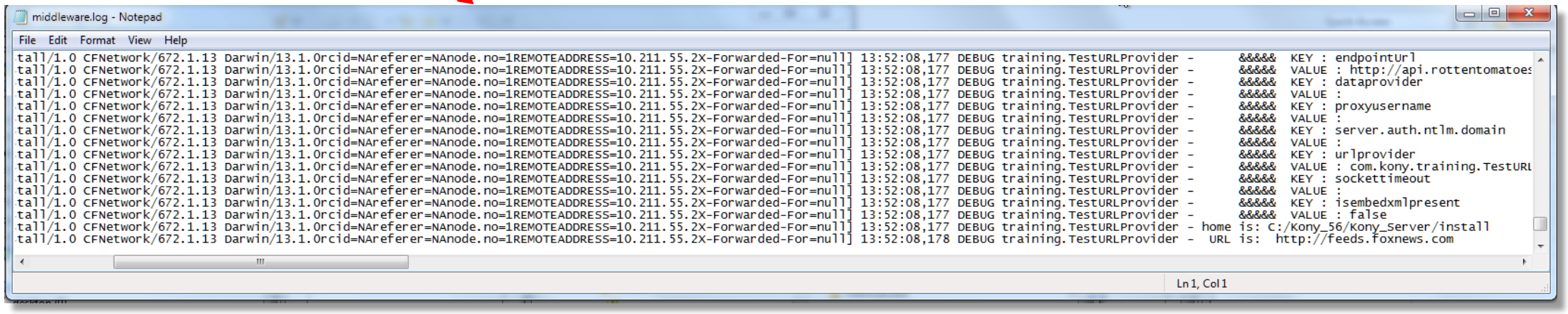
Log4j Properties File

- The configurations are kept in a properties file called: server-log4j2.xml
- The location for this file in Kony Fabric Server is:
  - <KonyFabricInstallation-Folder> → /KonyFabric/middleware_home/middleware/ middleware-bootconfig/server-log4j2.xml
- Log4j Configuration for Kony Apps
  - Sample code showing best practices:

```
private static final Logger LOG = Logger.getLogger(MyProvider.class);
//Declare the check for debug to improve performance
private static final boolean isDebugEnabled = LOG.isDebugEnabled();
//Code...
if (isDebugEnabled) {LOG.debug("In Execute now...");};
//Code continues...
```

Log files - Kony Fabric Server

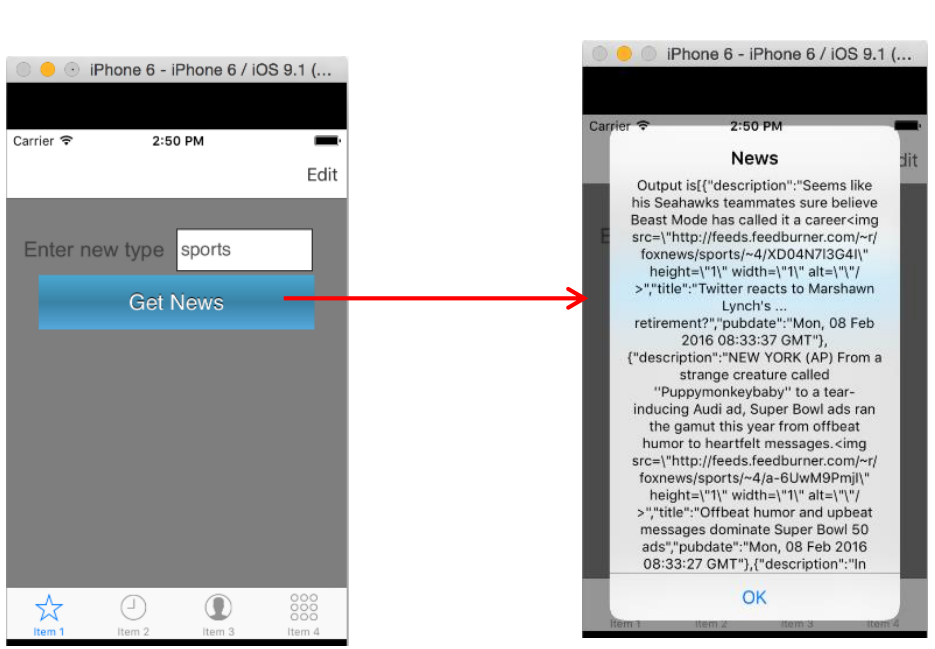
- Here is where the log lives on our server:
  - <KonyFabricInstallation-Folder> → /KonyFabric/logs/middleware.log



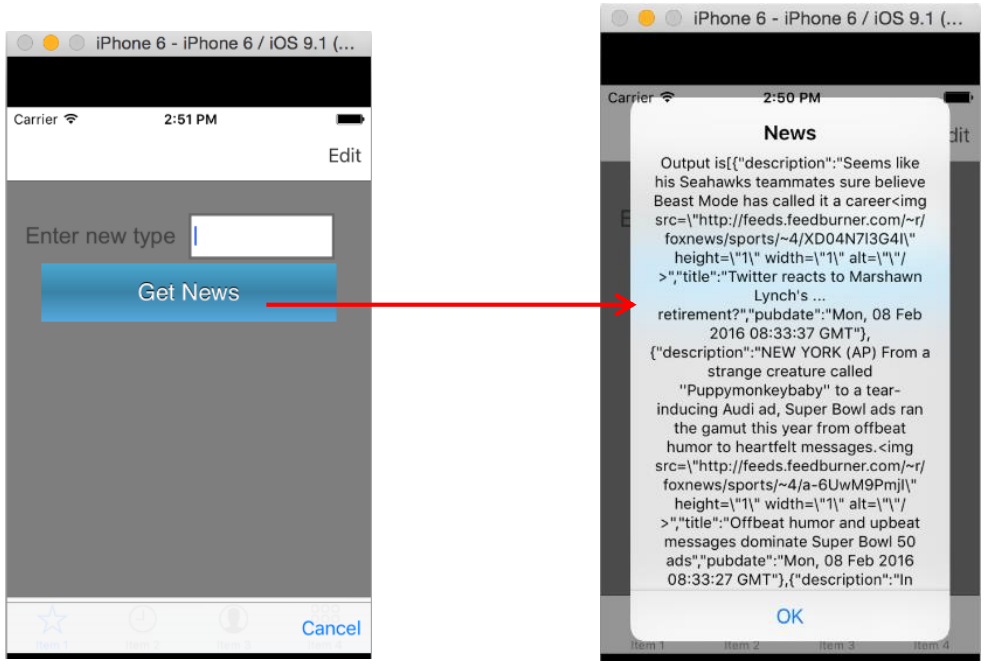
Clearly the server logs more info for each line - this is the default logging format

Exercise - Preprocessor

- When we pass a valid value, we get a proper response



- 2nd time when we do not pass a value, we get the response as it takes input from the session



- Here are the logs

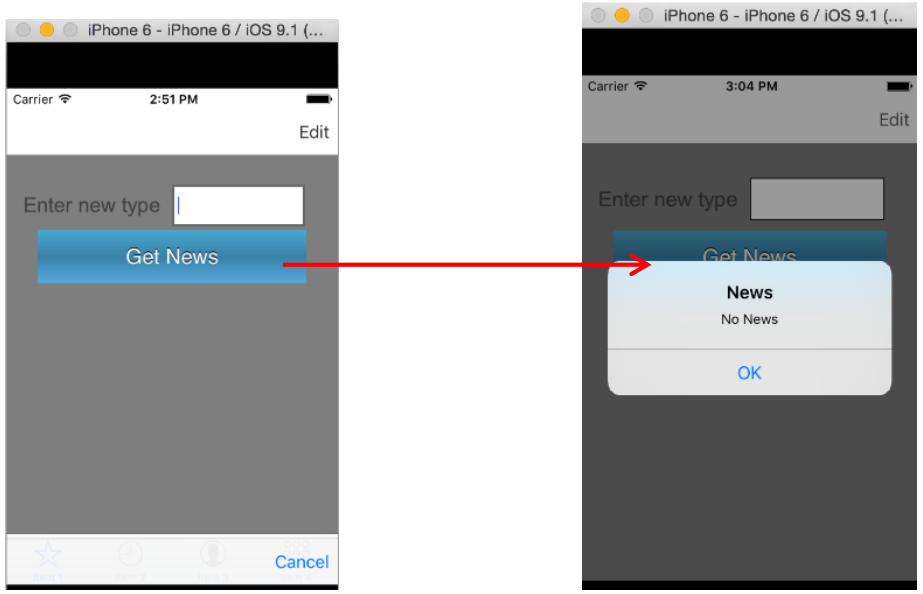
```
[appID=FoxNews/ntenantId=LocalDevEnv/nrequestID=12985700-30f7-4023-a4ac-e8afdad5f00b/nUA=Debugger-FoxNewsServi
.0 CFNetwork/758.1.6
Darwin/14.5.0/nrcid=NA/nreferer=NA/nSESSIONID=336C154CA24F00A539261298CE464B3A/nREMOTEADDRESS=10.10.12.75/nX-F
arded-For=null][KonyServer][DEBUG][08 Feb 2016 14:45:46,418]-DEBUG-processors.TestPreProcessor - Setting
TYPE IN SESSION with value: sports
```

Exercise - Preprocessor (cont'd)

- Here's how to test the application
  - Pass a valid value for type (E.g., "entertainment") and get response
  - Send an empty value i.e., either empty string or null and get response
  - Get response again (with empty string or null value).This time there will be no news as the session is also empty
- Let's see the application execution and also the logs

Exercise - Preprocessor (cont'd)

- 3rd time when we do not pass a value, we will get empty response as there is nothing in session.



- Here are the logs

```
[appID=FoxNews/ntenantId=LocalDevEnv/nrequestID=2e626470-fe51-40fb-92b7-094114736e1d/nUA=Debugger-FoxNewsServic/1
.0 CFNetwork/758.1.6
Darwin/14.5.0/nrcid=NA/nreferer=NA/nSESSIONID=336C154CA24F00A539261298CE464B3A/nREMOTEADDRESS=10.10.12.75/nX-Forw
arded-For=null][KonyServer][DEBUG][08 Feb 2016 14:49:01,524]-DEBUG-processors.TestPreProcessor - Value of
type in session is: null
```

Preprocessor using JavaScript

- Overview
  - Uses the java.script.ScriptEngine(JSR 223) to execute the JavaScript
  - Java 8 uses Nashorn engine. Previous versions uses Rhino engine
  - Support for node.js programming through Avatar,js library
- Advantages
  - Service definition and custom logic is at single place
  - No restart of server required on change of custom logic

Preprocessor using JavaScript

- The following objects are pushed to the script engine context
- Custom script i.e., preprocessor script code can refer them with handle name as variable

Handle name	Corresponding Java class
serviceInputParams	Java.util.Map
request	com.konylabs.middleware.controller.DataControllerRequest
logger	logger
result	com.konylabs.middleware.dataobject.Result
response	com.konylabs.middleware.controller.DataControllerResponse

- We shall discuss result and response during Postprocessor
- Lets discuss serviceInputParams, request and logger

Preprocessor - serviceInputParams

- **serviceInputParams** contains all the service input parameters that are sent from the device:
- The actual service call will use the data in `serviceInputParams`. Change the data here if you want the service to use your new value
  - Here's a sample code to get and set the input parameter

```
var news=serviceInputParams.get("newsType")
if(news==""){
  logger.debug("newsType is empty");
  serviceInputParams.put("newsType", "sports");
}
```

Using the logger object we can print the logs

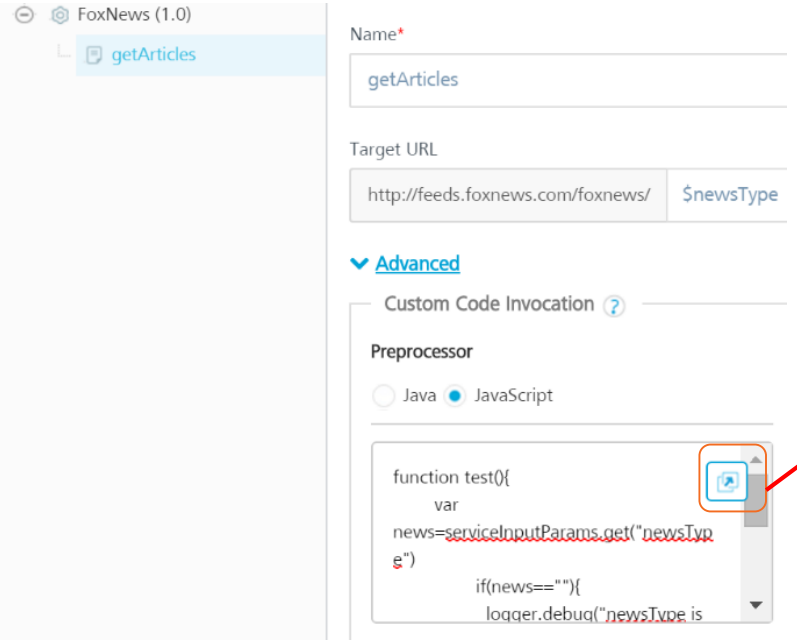
Preprocessor - request

- **request** object gives us access to the request configuration data and it includes the following:
  - Attributes, Parameters, & Session
- This object contains the same methods as of DataControllerRequest object in java.
- Few examples

```
request.getAttribute("attributeName")
request.getParameter("newParam")
request.containsKeyInRequest("newParam")
request.setAttribute("newAtt", "newAttValue");
request.getSession();
```

Preprocessor - configuring JavaScript code

- In case of JavaScript, we need to configure the code under the operations
- Go to **Advanced** under operation, and under **Preprocessor**, select **JavaScript** and configure code
- Publishing and calling the service from the application is same as earlier




```
function test(){
    var news=serviceInputParams.get("newsType")
        if(news==""){
          logger.debug("newsType is empty");
          serviceInputParams.put("newsType", "sports");
        }
}
test();
```

In case of JavaScript, We need to define a function and invoke the function

Clicking on the icon opens a windows to configure the code

Exercise - Preprocessor (cont'd)

- Test the application as we tested for Java
  - Pass a valid value for type (E.g., "entertainment") and get response
  - Send an empty value i.e., either empty string or null and get response
  - Get response again (with empty string or null value).This time there will be no news as the session is also empty
- Observe the application execution and also the logs

Postprocessor

Postprocessor - Use Cases

- Some real time uses of the postprocessor are:
  - Logging some/all the output data - use for reporting purposes
  - Adding new output parameters to the result to communicate other information back to the client
  - Modifying the outputs - for example, changing the data format as required by the application i.e., mm-dd-yyyy to dd-mm-yyyy.
  - Caching data for a user to create a pagination scheme for large result sets
  - Performing calculations on the data to provide calculated results to the client Postprocessors are custom java classes that run on the Kony server

DataPostProcessor2 Interface

- **com.konylabs.middleware.common.DataPostProcessor2** is the interface provided by middleware to implement a postprocessor.
- Here are the standard imports, the interface and the method to be overridden:

```
import com.konylabs.middleware.controller.DataControllerRequest;
import com.konylabs.middleware.controller.DataControllerResponse;
import com.konylabs.middleware.dataobject.Result;
import java.util.HashMap;
public interface DataPostProcessor2
{
  public object execute(Result result, DataControllerRequest request,
                        DataControllerResponse response) throws Exception;
}
```

Override Method in DataPostProcessor2

- The interface contains a single method that you need to override: `execute`
  - public object execute(Result result, DataControllerRequest request, DataControllerResponse response) throws Exception
    - **result:** the object that must be returned in this method - it contains the data to send back to the client
    - **request:** the object that gives you access to session data
    - **response:** used to set device-headers and device-cookies
    - Returns a Result object
      - **Note:** use the request object that is shared by preprocessor for communication between these classes

Result class

- Before implementing Postprocessor, we need to understand the following data objects and their respective methods exposed by Kony:
  Result: the complete output data structure sent back to the client
  Dataset: a collection in our output parameters
  Record: a different type of collection in the output parameters
  Param: an individual piece of data
- Here are some rules
  Param: leaf data element - you'll always access a specific piece of data as one of these parameter objects
  Dataset: always consist of Record objects.
  Record: can consist of **Param, Record** and/or **Dataset** objects - you can have "infinite" nesting of data within a Result object
- Let's look at some examples…

Result class example

Raw data from service call:

```
<?xml-stylesheet href="/v1/xsl/xml_pretty_pr
<stores currentPage="1" totalPages="2" from="1" to="10" total="12" queryTime="0.002" totalTime="
canonicalUrl="/v1/stores(city="dallas")?apiKey=qmb9dnkg569rgs5za36camq5" partial="false">
    <store>
        <storeId>58</storeId>
        <storeType>BigBox</storeType>
        <tradeIn>Trade-In - No-receipt</tradeIn>
        <name>Park Lane</name>
        <longName>Best Buy - Park Lane</longName>
        <address>9378 N Central Expy</address>
        <address2/>
        <city>Dallas</city>
        <region>TX</region>
        <fullPostalCode>75231</fullPostalCode>
        <country>US</country>
        <lat>32.876068</lat>
        <lng>-96.768257</lng>
        <hours>Mon: 10-9; Tue: 10-9; Wed: 10-9; Thurs: 10-9; Fri: 10-10; Sat: 10-
        <hoursAmPm>Mon: 10am-9pm; Tue: 10am-9pm; Wed: 10am-9pm; Thurs: 10am-9pm;
        <gmtOffset>-5</gmtOffset>
        <services>
            <service>Geek Squad Services</service>
            <service>Best Buy Mobile</service>
            <service>Best Buy For Business</service>
            <service>Apple Shop</service>
            <service>Magnolia Design Center </service>
            <service>Pacific Kitchen & Home</service>
            <service>Camera Experience Shop </service>
            <service>Windows Store</service>
            <service>Samsung Experience</service>
            <service>LG Experience </service>
```

Our Output Parameter configuration:

	NAME	PATH	SCOPE	DATATYPE	ID	RECORD ID
	stores	//stores	response	collection		
	storeId	store/storeId	response	string	stores	
	name	store/name	response	string	stores	
	hourInfo	store	response	collection	stores	
	hours	hours				hourInfo
	hoursAmPm	hours/				hourInfo
	offset	gmtOffset	response	string		hourInfo
	services	store/services	response	record	stores	
	service	service	response	string	services	

We'll group our store info into a Dataset

We'll group our hourInfo into a Record

We'll group our services into a nested Dataset

Result class example

	NAME	PATH	SCOPE	DATATYPE	COLLECTION ID	RECORD ID
	stores	//stores	response	collection		
	storeId	store/storeId	response	string	stores	
	name	store/name	response	string	stores	
	hourInfo	store	response	collection	stores	
	hours	hours	response	string		hourInfo
	hoursAmPm	hoursAmPm	response	record		hourInfo
	offset	gmtOffset	response	string		hourInfo
	services	hoursAmPm	response	string	stores	
	service	gmtOffset	response	string	services	

We'll group our store info into a Dataset with an ID of "stores"

We'll group our hour info into a Record with an ID of "hourInfo"

We'll group our services info into a Dataset with an ID of "services"

```
{   "stores": [{
        "name": "Park La
        "storeId": "58",
        "hourInfo": {
            "hours": "Mon: 10-9; Tue: 10-9; Wed: 10-9; Thurs: 10-9; Fri: 10-10; Sat: 10-10; Sun: 11-8",
            "offset": "-5",
            "hoursAmPm": "Mon: 10am-9pm;                                    ri: 10am-10pm; S
        }
        "services": [{"service": "Geek Sc
        }, {
            "service": "Best Buy Mobile"
        }, {
            "service": "Best Buy For Business"
        }, {
            "service": "Apple Shop"
        }, {
            "service": "Magnolia Design Center "
        }, {
            "service": "Pa
        }, {
            "service": "Ca
        }, {
            "service": "Wi
        }, {
            "service": "Samsung Experience"
        }, {
            "service": "LG Experience "
        }, {
            "service": "Sony Experience "
        }, {
            "service": "Electronics Recycling"
        }, {
            "service": "Car & GPS Installation Services"
        }]
    }, {
        "name": "DAL - Dallas Love Field",
```

Result class - structure

- Let's consider this result as an example for looking at the various methods we have to manage the data
- We'll first cover the various methods for accessing the data then we'll cover methods for changing, adding and/or removing data. Let's discuss first in Java and JavaScript later


```
{   "stores": [{
        "name": "Park Lane",
        "storeId": "58",
        "hourInfo": {
            "hours": "Mon: 10-9; Tue: 10-9; Wed: 10-9; Thurs: 10-9; Fri: 10-10; Sat: 10-10; Sun: 11-8",
            "offset": "-5",
            "hoursAmPm": "Mon: 10am-9pm; Tue: 10am-9pm; Wed: 10am-9pm; Thurs: 10am-9pm; Fri: 10am-10pm; S
        }
        "services": [{"service": "Geek Squad Services"
        }, {
            "service": "Best Buy Mobile"
        }, {
            "service": "Best Buy For Business"
        }, {
            "service": "Apple Shop"
        }, {
            "service": "Magnolia Design Center "
        }, {
            "service": "Pacific Kitchen & Home"
        }, {
            "service": "Camera Experience Shop "
        }, {
            "service": "Windows Store"
        }, {
            "service": "Samsung Experience"
        }, {
            "service": "LG Experience "
        }, {
            "service": "Sony Experience "
        }, {
            "service": "Electronics Recycling"
        }, {
            "service": "Car & GPS Installation Services"
        }]
    }, {
        "name": "DAL - Dallas Love Field",
```

Result class - Methods

- Here are the methods for the Result class:

Method Summary	
Dataset	findDataset(String id) Find a Dataset with the given id.
Param	findParam(String id) Find a Param with the specified name.
ArrayList<Dataset>	getDataSets() Get the list of Datasets.
ArrayList<Param>	getParamList() Get the list of Params.
Record	getRecordById(String id) Find a Record with the given id.
ArrayList<Record>	getRecords() Get the list of Records from the Result.
void	removeParam(Param param) Remove the specified Param from the Result.
void	setDataSet(Dataset dataset) Add a new Dataset to the Result.
void	setDataSets(ArrayList<Dataset> datasets) Update the Datasets with the given list.
void	setParam(Param param) Add a new Param to the Result.
void	setParamList(ArrayList<Param> params) Update the Params with the given list of Params.
void	setRecord(Record record) Add a new Record to the Result.
void	setRecords(ArrayList<Record> records) Update the Result with given list of Records.

These find & get methods that take an ID search the whole result object whereas the "list" methods operate at the top level of the result

You can only remove parameters, use the set list methods to overwrite the result if needed

These set methods that take an object are to add single items whereas the ones that take the ArrayList allow you to populate sets of data

Dataset Class

- As we saw earlier, Datasets contain ONLY Records so all methods are for records only:

Method Summary	
String	getId()
Record	getRecord(int index)
Record	getRecordOrder(int order)
ArrayList<Record>	getRecords()
void	setId(String id)
void	setRecord(Record rec)
void	setRecords(ArrayList<Record> records)

- Note: getRecordOrder() is not intended to be used by the developer, it's an internal platform method

Record class

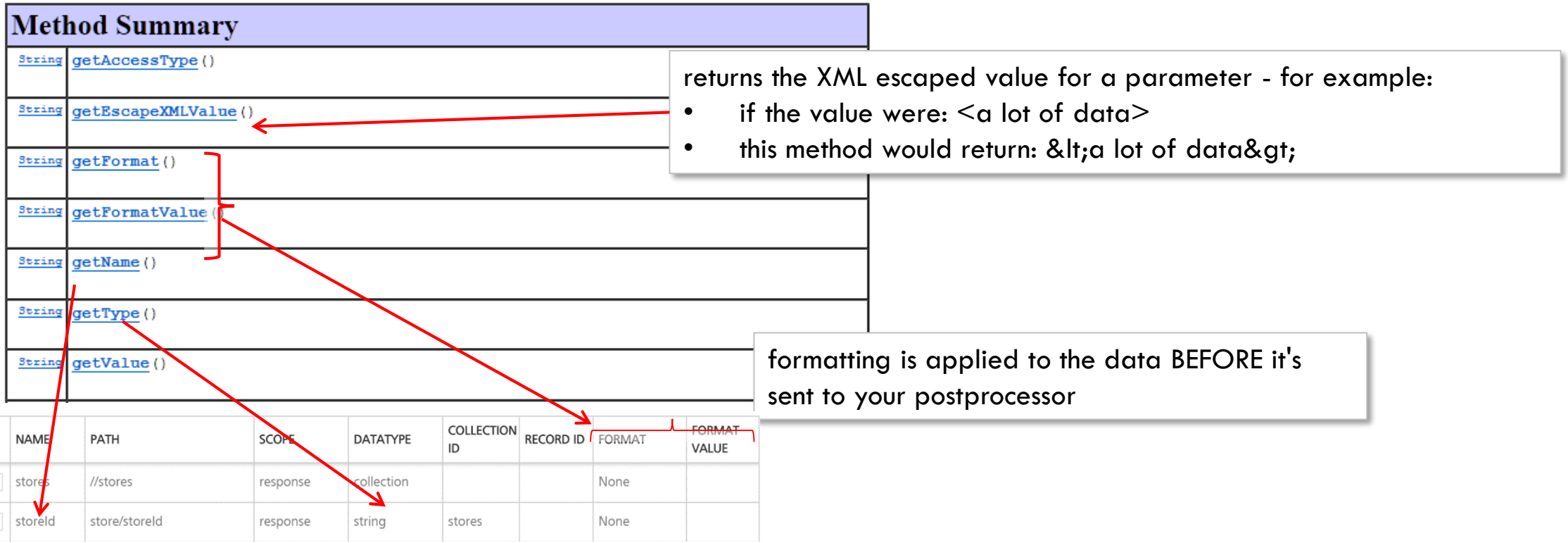
- **Records** can contain **Datasets,** other **Records** and **Params**
- Here are the "get" methods to access those (and they are very similar to what we saw earlier for the Result object):

Method Summary	
int	compareTo(Record o)
Dataset	getDatasetById(String id)
ArrayList<Dataset>	getDatasets()
String	getId()
Param	getParam(String name)
ArrayList	getParams()
Record	getRecordById(String id)
ArrayList<Record>	getRecords()

- **compareTo** is not intended to be used by the developer, it's an internal platform method

Param class Methods

- **Param** is the object that contains the actual data elements defined in Output parameters
- Here are the "get" methods for Params:



- getAccessType is not intended to be used by developer, it's an internal platform method

Example - Getting All Our Data

```
Dataset ds = result.findDataset("stores");
Iterator<Record> reclt = ds.getRecords().iterator();
while (reclt.hasNext()){
        Record rec = reclt.next();
        Iterator<Param> paramIt = rec.getParams().iterator();
        while (paramIt.hasNext()){
                Param pm = paramIt.next();
                LOG.debug("   &&&  param name: "+pm.getName()+"  param value: "+pm.getValue());
        }
        Record recHourInfo = rec.getRecordById("hourInfo");
        Iterator<Param> paramHourIt = recHourInfo.getParams().iterator();
        while (paramHourIt.hasNext()) {
                Param hrPm = paramHourIt.next();
                LOG.debug("   &&&  param name: "+hrPm.getName()+" param value: "+hrPm.getValue());
}
Dataset servicesDs = rec.getDatasetById("services");
        Iterator<Record> servicesReclt = servicesDs.getRecords().iterator();
        while (servicesReclt.hasNext()){
                Record svcRec = servicesReclt.next();
                Param svcPm = svcRec .getParam("service");
                LOG.debug("   &&&  param name: "+svcPm.getName()+" param value: "+svcPm.getValue());
        }
}
```

Result Methods - Changing Data

- So far, we've only accessed the data objects and printed them out in the logs
- A REAL postprocessor usually changes the data somehow…
- Here are the methods for setting data in results:

Method Summary	
void	removeParam(Param param) Remove the specified Param from the Result.
void	setDataSet(Dataset dataset) Add a new Dataset to the Result.
void	setDataSets(ArrayList<Dataset> datasets) Update the Datasets with the given list.
void	setParam(Param param) Add a new Param to the Result.
void	setParamList(ArrayList<Param> params) Update the Params with the given list of Params.
void	setRecord(Record record) Add a new Record to the Result.
void	setRecords(ArrayList<Record> records) Update the Result with given list of Records.

- You can add/remove individual `Datasets`, `Params` and `Records` or replace the entire collection

Param constructors

- When creating new parameters in code, you'll use one of the following constructors:

Constructor Summary
Param()
Param(String name, String value, String type)
Param(String name, String value, String type, String format, String formatValue)
Param(String name, String value, String type, String format, String formatValue, String accessType)

- The Constructors are used to create and fully populate a **Param** object

  format can be: "date", "currency" or "number"

  type can be: "string", "number" or "boolean"
- For example:
- **Param** myColor = new **Param**("mycolor","green","string"); will create a new parameter with a name of "mycolor", a value of "green" and the data type is string

Param Methods - Changing Data

- We already saw how to "get" all the Param info - there are equivalent "set" methods
- Here are the methods for setting data in Params:

Method Summary	
void	setAccessType(String accessType)
void	setFormat(String format)
void	setFormatValue(String formatValue)
void	setName(String name)
void	setType(String type)
void	setValue(String value)

- Typical use:
  - First use the **getParam** method to retrieve the parameter
  - Then use the set methods to change the data

Record Methods - Changing Data

- Here are the methods for setting data in Records:

Method Summary	
boolean	removeParam(Param param) Removes the given Param from the Params of the record.
boolean	removeParamsByName(String name) Removes the given Param from the Params of the record.
void	setDataset(Dataset dataset)
void	setDatasets(ArrayList<Dataset> datasets)
void	setID(String id)
void	setParam(Param param)
void	setParams(ArrayList params)
void	setRecords(ArrayList<Record> records)

- Since **Record** can contain **Datasets,** other **Records** and **Params,** we find similar methods to the **Result** set methods we just saw
- **setID** - we saw using the **recordID** output parameter to set the **ID** - this is how we do it in code
- No constructor variances - example: **Record** myRecord = new **Record();**

Dataset Methods - Changing Data

- Datasets can only have Records in them so the list of methods is short:

Method Summary	
void	setId(String id)
void	setRecord(Record rec)
void	setRecords(ArrayList<Record> records)

- Like Records, you can specify the ID by using setID OR in the Constructor

Constructor Summary
Dataset()
Dataset(String id)

- E.g., Dataset stores = new Dataset("stores"); will create a new dataset with an ID of "stores"

Example - Building a Result Set

- Let's look at an example:
- A simple example showing how to build up a result set using some of the methods we just covered
- Here is the same example using the Dataset setRecords method:

```
result = new Result();
result.setParam(new Param("type","Painting","string"));

Dataset ds = new Dataset("Colors");

Record rd = new Record();
rd.setParam(new Param("color","blue","string"));
rd.setParam(new Param("rank","17","number"));
ds.setRecord(rd);

rd = new Record();
rd.setParam(new Param("color","green","string"));
rd.setParam(new Param("rank","4","number"));
ds.setRecord(rd);

rd = new Record();
rd.setParam(new Param("color","yellow","string"));
rd.setParam(new Param("rank","9","number"));
ds.setRecord(rd);

result.setDataSet(ds);

return result;
```

Postprocessor using JavaScript

- The following objects are pushed to the script engine context
- Postprocessor script code can refer them with handle name as variable

Handle name	Corresponding Java class
request	com.konylabs.middleware.controller.DataControllerRequest
logger	logger
result	com.konylabs.middleware.dataobject.Result
response	com.konylabs.middleware.controller.DataControllerResponse

- We have already discussed request and logger during Preprocessor
- Let's discuss result and response in Postprocessor

Postprocessor - result

- **Result** gives us the access to the complete output data structure sent back to the client
- This object contains the same methods as of **com.konylabs.middleware.dataobject.Result** object in java
- Few examples

  getDataSets()

  getRecords()

  getParamList()

  getDatasetById("dataSetId")
- We can also create a Dataset object, Record object and Param objects in JavaScript and add it to the **result** object

Example - Building a Result Set

- Let's look at an example building a result set in JavaScript:

```
var varDataSet= new com.konylabs.middleware.dataobject.Dataset("Colors");

var varRecord1 = new com.konylabs.middleware.dataobject.Record();
var varParam1 = new com.konylabs.middleware.dataobject.Param();
var varParam2 = new com.konylabs.middleware.dataobject.Param();
varParam1.setName('color');
varParam1.setValue('blue');
varParam2.setName('rank');
varParam2.setValue('4');

varRecord1.setParam(varParam1);
varRecord1.setParam(varParam2);

var varRecord2 = new com.konylabs.middleware.dataobject.Record();
var varParam3 = new com.konylabs.middleware.dataobject.Param();
var varParam4 = new com.konylabs.middleware.dataobject.Param();
varParam3.setName('color');
varParam3.setValue('yellow');
varParam4.setName('rank');
varParam4.setValue('6');

varRecord2.setParam(varParam3);
varRecord2.setParam(varParam4);
varDataSet.setRecord(varRecord1);
varDataSet.setRecord(varRecord2);
result.setDataSet(varDataSet)
```

Postprocessor - response

- response gives us the access to the response that is sent by the backend system.
- This object contains the same methods as of com.konylabs.middleware.controller.DataControllerResponse object in java
- Few examples

  setDeviceHeaders(headerInfo);

  getHeaders()

  getResponse()

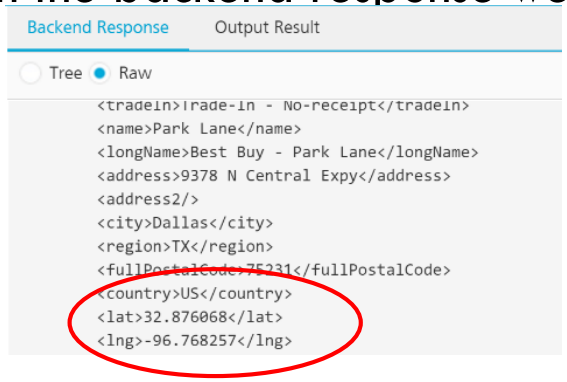
  getCookies()

Postprocessor Guidelines

- Perform all the formatting, conversions, sorting, and so on in the Postprocessor to keep the UI code simple
- Check for opstatus (success or failure) value in the Postprocessor before processing any data
- Use log4j debugging facilities instead of System.out.println or other custom framework
- Write all the processing and business logic at the server side either in Pre/Post processors
- When using log4j in the Postprocessors, check for the debug level before calling loggers
- Postprocessors are singleton classes. Any user specific data should not be stored in class variables, otherwise it will result in abnormal behavior. Common Data can be stored in class variables
- In a Preprocessor or a Postprocessor we cannot call a service
- You cannot retry the service if you want to call the service again if the service failed

Exercise - Postprocessor

- One of the best uses for the postprocessor is to configure the service results data so that it can be immediately consumed by the UI
- Lets say our mobile applications requires latitude and longitude of each store in our BestBuy store locator service
- In the backend response we are getting lat and lng in each store as shown below



- In lat and lng, after the decimal there are multiple digits, but our mobile application expects 2 digits after the decimal for lat and lng

Exercise - Postprocessor Solution

- Java code

```
LOG.debug("############ In  TestPostProcessor2.execute() ############");
Dataset storesDS = result.getDatasetById("stores");
List<Record> datasetRecords = storesDS.getAllRecords();
if(null != datasetRecords && datasetRecords.size() > 0) {
for(Record datasetRecord : datasetRecords) {
Param tempLatParam = datasetRecord.getParam("lat");
tempLatParam.setValue(tempLatParam.getValue().substring(0,5));
Param tempLngParam = datasetRecord.getParam("lon");;
tempLngParam.setValue(tempLngParam.getValue().substring(0,6));
}
}
LOG.debug("------> MY JSON RESULT " + ResultToJSON.convert(result));
return result;
```

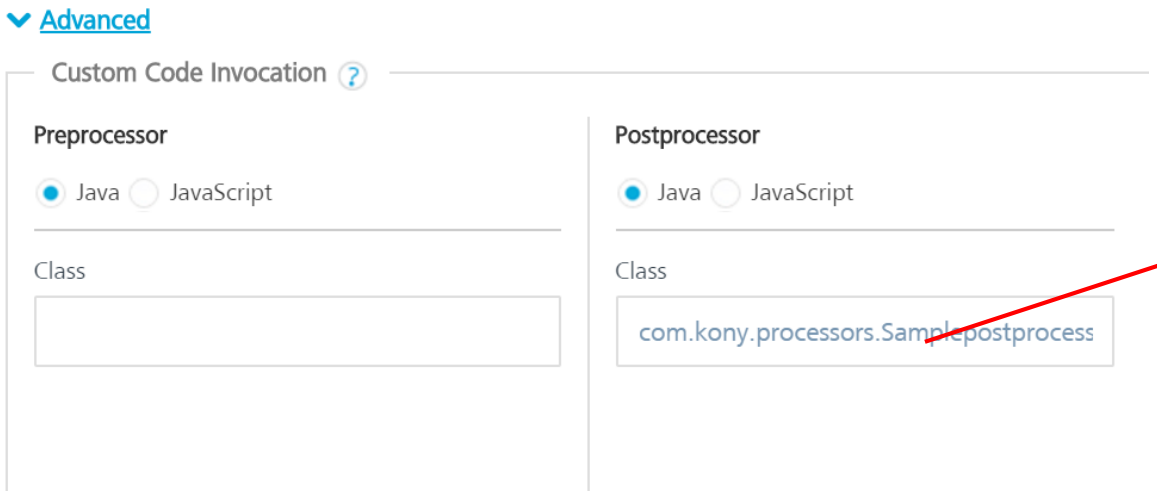
Exercise - Postprocessor Solution

- JavaScript code

```
function processOutput(){
        logger.debug("######### In JavaScript Postprocessor #########");
        var storesDS = result.findDataset("stores");
        var sizeOfRecords = storesDS.getRecords().length;
        for (i = 0; i < sizeOfRecords; i++) {
                var datasetRecord = storesDS.getRecord(i);
        logger.debug("######### Modifying lat and lon params for Record: " + i);
                var tempLatParam = datasetRecord.getParam("lat");
                tempLatParam.setValue(tempLatParam.getValue().substring(0,5));
                var tempLngParam = datasetRecord.getParam("lon");;
                tempLngParam.setValue(tempLngParam.getValue().substring(0,6));
        }
}
processOutput();
```

Postprocessor - configuring the class

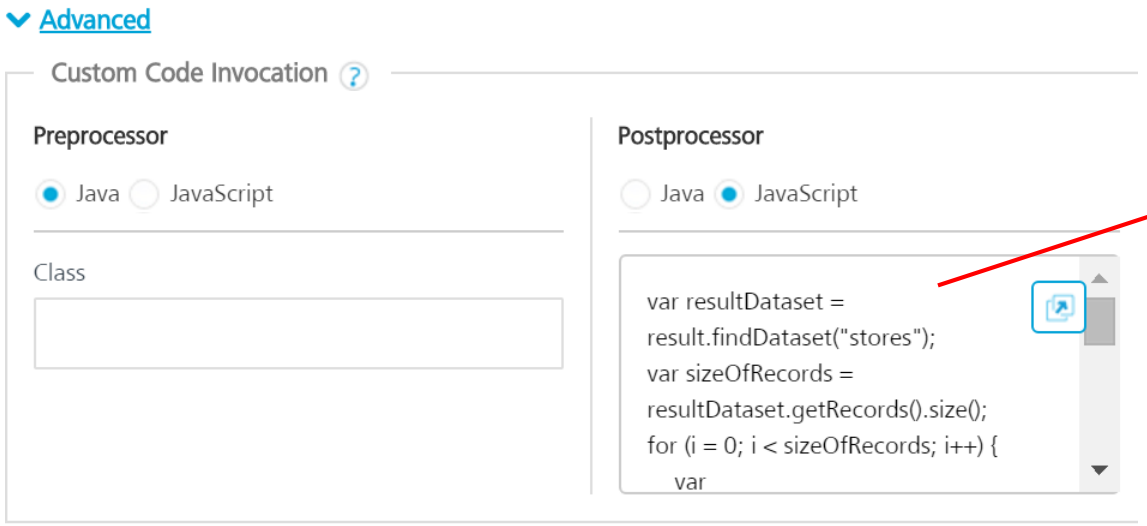
- If Postprocessor is implemented in Java, then we have to create a jar and upload it to the service
- After uploading the jar , we have to configure the class for the operation



Go to **Advanced** under operation, and under **Postprocessor,** select **Java** and configure the class name

Postprocessor - Configuring JavaScript code

- If Postprocessor is implemented in JavaScript, then we have to configure JavaScript code

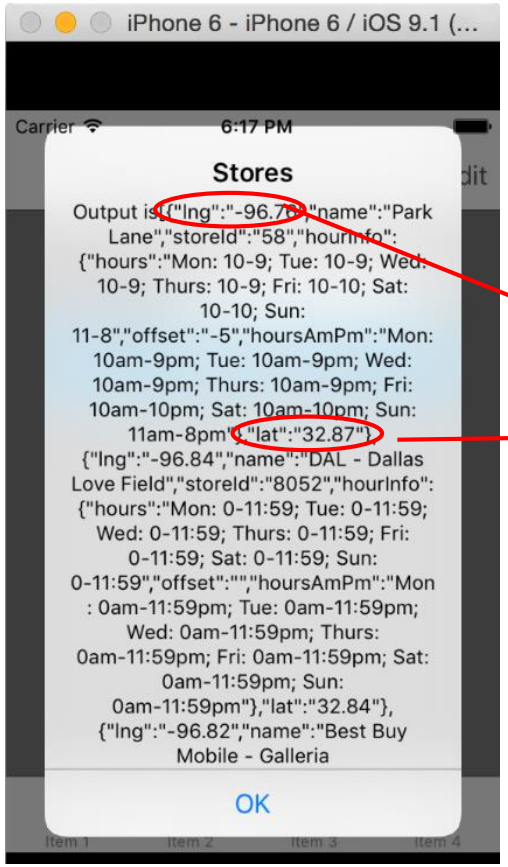


Go to Advanced under operation, and under Postprocessor, select JavaScript and configure the JavaScript code.

- Publishing and calling the service from the application is same as earlier

Exercise - Postprocessor

- Here is the output in the application



Latitude and Longitude is trimmed in the application

Java Service - Overview

- Very often in our applications we will come across situations where we want to connect to some data that is not exposed as a web service
- In these situations, you'll need to create a new custom connector, written in Java, that runs on the server
- As long as you can create what you need in Java - you'll be able to access that data
- Java services will be created like our Preprocessors, Postprocessors, etc
- Java services will be configured in Kony Fabric console
  - The jar file(s) will need to be uploaded to the Kony Fabric server
- Java services will be used like any other service:
  - You'll be able to attach Preprocessors, Postprocessors, etc
  - It will work on the client like any other service

Java service - Interface

- The Java Service will implement the com.konylabs.middleware.common.JavaService2 Interface
- Here is what it looks like when we create a class that implements this interface:

```
public class TestJavaService implements JavaService2 {
    public Object invoke(String methodID, Object[] objectArray,
    DataControllerRequest request, DataControllerResponse response)
                        throws Exception {
                        // TODO Auto-generated method stub
                        return null;
                }
}
```

- Note the invoke method is of type Object - namely we'll return our data like we did in our Postprocessor - this must be the Kony Result object

Java service - Implementation

- As you find from the above interface, we have to override the invoke method and add our code

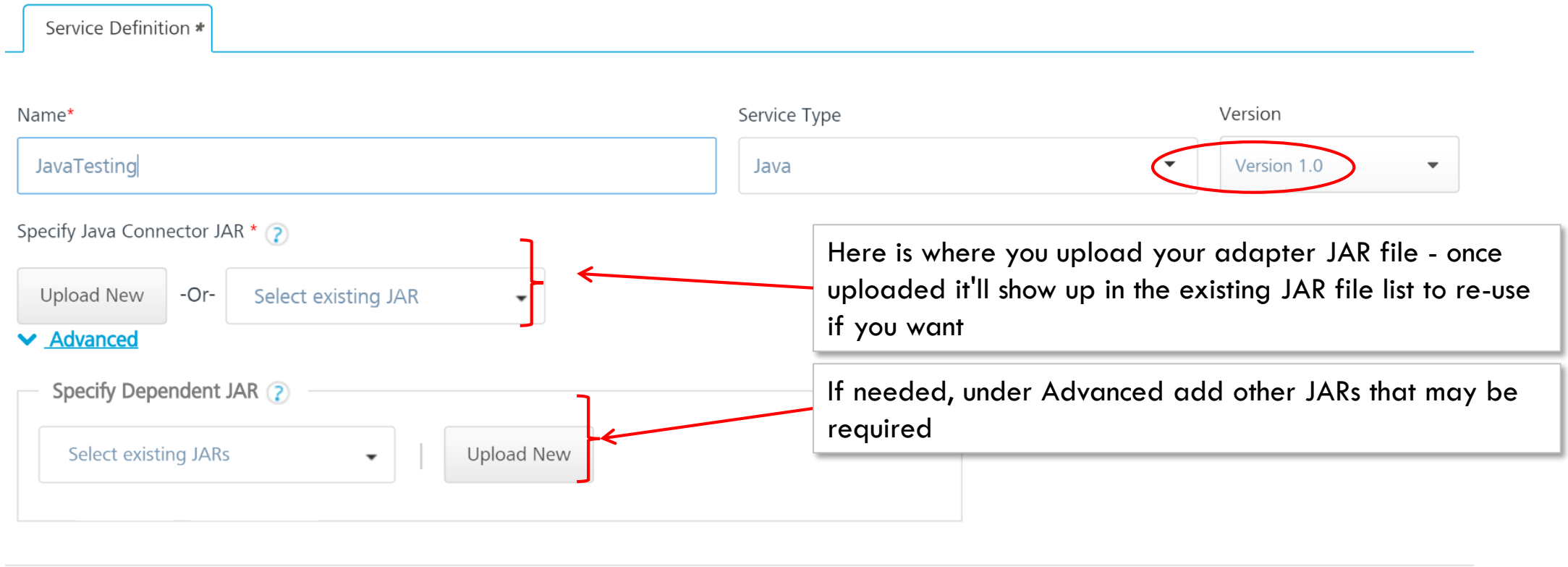
  public Object invoke(String operationID, Object[] objectArray,

  DataControllerRequest request, DataControllerResponse response)

  - operationID - is the unique ID which is used to identify a service.
  - Note: you'll be able to configure multiple services using your custom Java Service - you'll be passed in the ID so you'll know which branch of code to use
  - objectArray[0] - is a HashMap that contains all the configuration information about the service (like we saw in the request parameters)
  - objectArray[1]– is also a HashMap that contains all the input parameters and values sent from the device.
  - request - the same request object we saw earlier in preprocessor, postprocessor, etc...
  - response - the same response object we saw earlier in preprocessor, postprocessor, etc...

Java Service Configuration

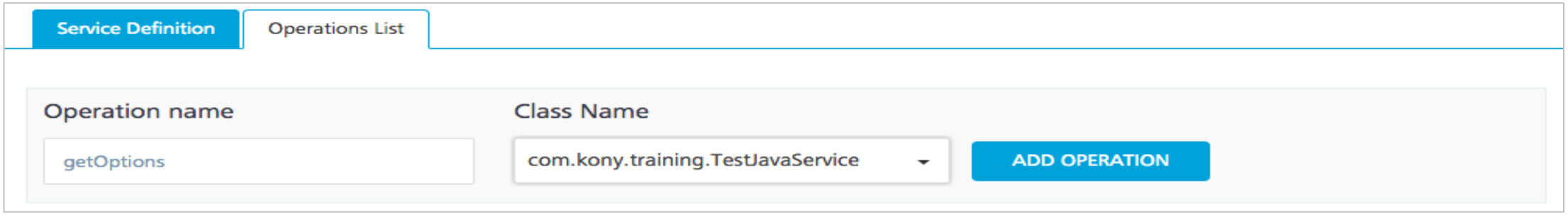
- Here is how we configure a Java Integration Service:



- Once you save, you can now add Operations...

Operation Id

- In the Java service invoke method, the first argument is operationID - a String that's is sent from the UI which is used to identify a operation



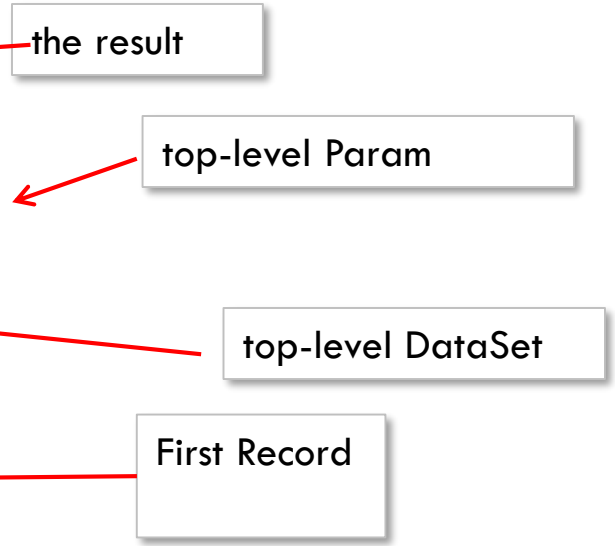
- Do not create multiple Java service implementation classes - create one class that uses the operationID. For example.

```
if ("getOptions"==operation) {
        //do something and returns options data}
else if ("getCustomer"==operation) {
        //do something and return customer data}
else {
        //perhaps this is an error condition}
```

Java service - An Example

- What you DO in your Java Service is write whatever code you need to get the data
- So we'll ignore all that code and just focus on returning the data from the Java service
- Our example will simply return a few hardcoded values
- Code for our example

```
Result result = new Result();
if("getOptions".equalsIgnoreCase(operationID)){
        result.addParam(new Param("type","Painting","string"));
        Dataset ds = new Dataset("Colors");
        ArrayList<Record> rds = new ArrayList<Record>();
        Record rd = new Record();
        rd.addParam(new Param("color","blue","string"));
        rd.addParam(new Param("rank","17","number"));
        rds.add(rd);
```



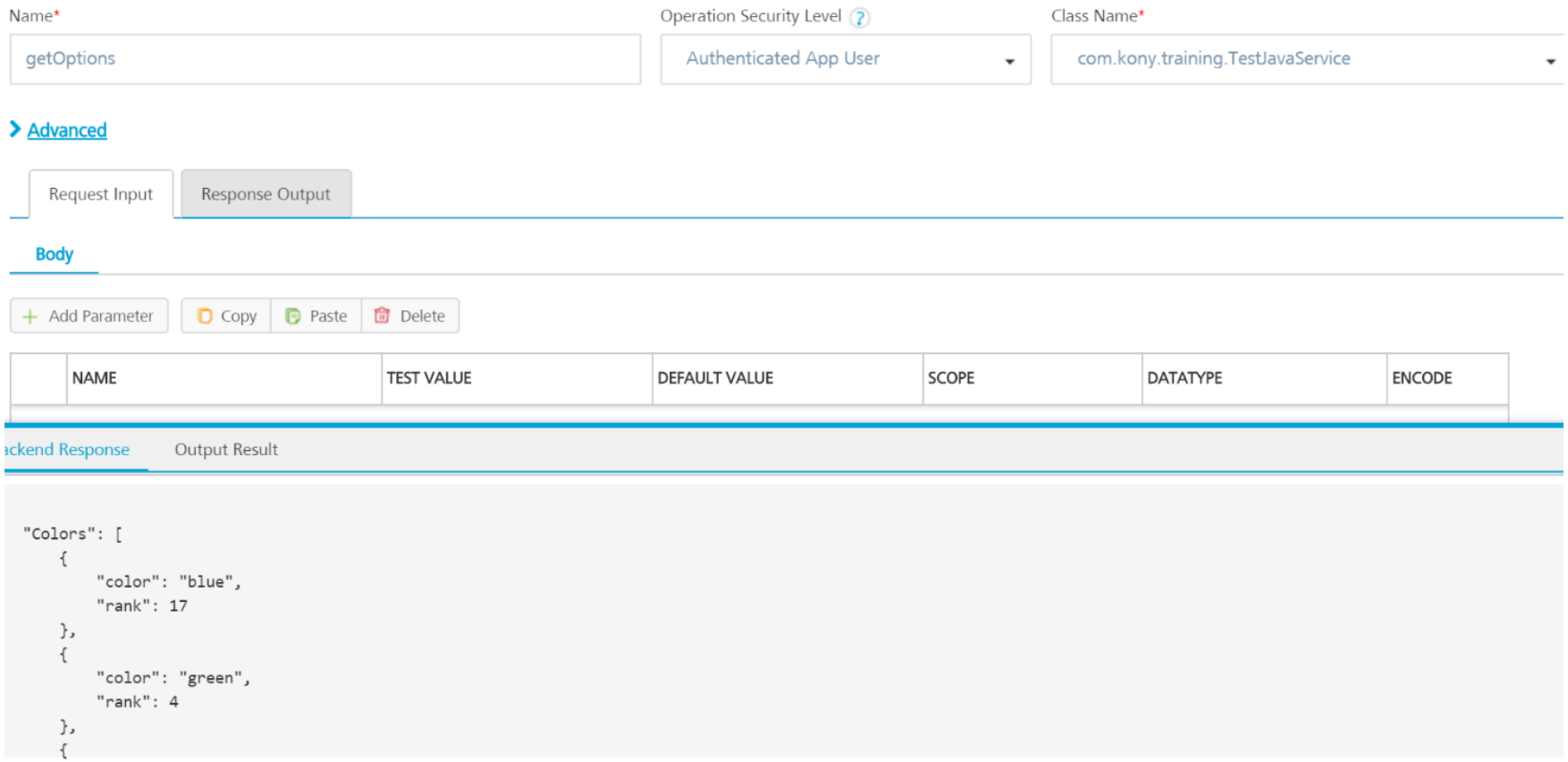
Java service - An Example

```
            rd = new Record();
            rd.addParam(new Param("color","green","string"));
            rd.addParam(new Param("rank","4","number"));
            rds.add(rd);
            rd = new Record();
            rd.addParam(new Param("color","yellow","string"));
            rd.addParam(new Param("rank","9","number"));
            rds.add(rd);
            ds.addAllRecords(rds);
            result.addDataset(ds);
            result.addParam(new Param("opstatus","0","string"));
        }
        return result;
```

- We have already seen how to configure the Java service as part of integration service.Publishing and using it in the application is same as any other integration service.

Exercise - Java Service

- Create a java service class with the code that we have discussed, and configure it in MF console and test it. Publish it and test from the application.



Server Side Programming

kony Stay Ahead

Component Without Contract

Component Overview

- Components are the truly 'self-contained' and reusable entity which enhances master feature-set with these additional capabilities:
  - Code Modules
  - Fabric Services
  - Contract to simplify usage and enable Rapid Mobile App Development
  - Reusability in same project and across the projects in workspace
  - Share with wider audience via Marketplace or local share functionality
  - Download & use components (created by Marketplace Assets team and other developers in your projects)

Types of Components

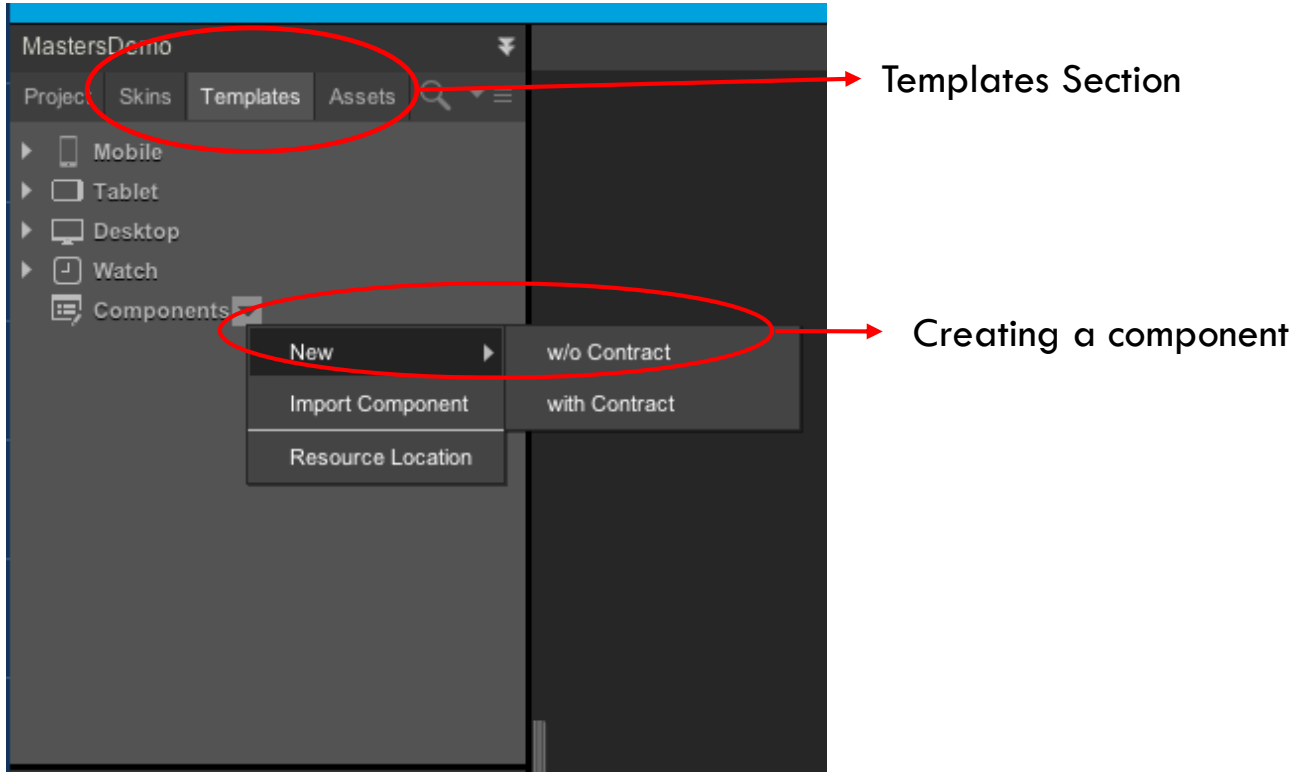
- Component are TWO types:
  - Component without Contract
    - Default Master Widget created from existing Kony widgets
    - This retains all the capabilities of existing Masters with MVC adoption
    - Lot of other new features are also possible with these Masters
  - Component with Contract
    - User widget that app developers can create a master with existing Kony widgets
    - Define a contract around the master to abstract out the internal details of the Master

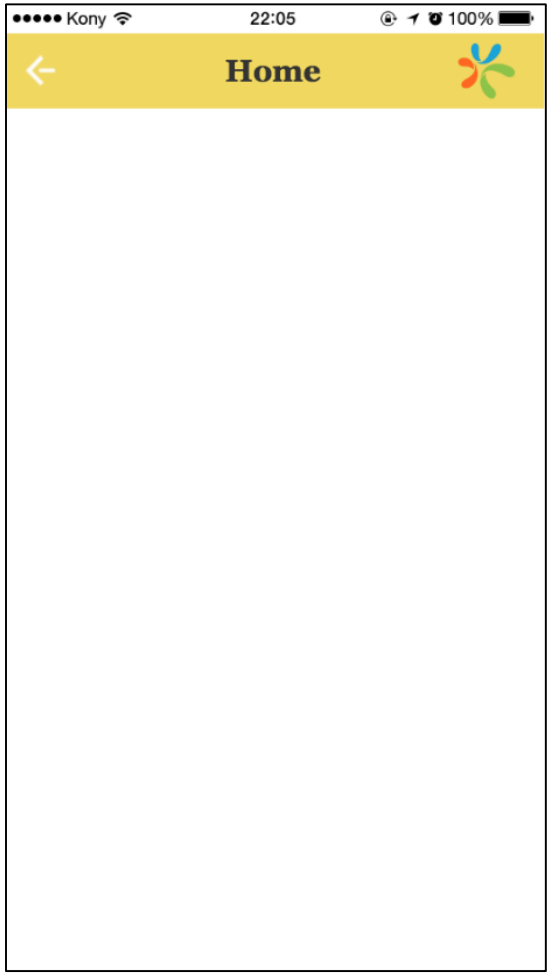
Component without Contract - Overview

- Component without Contract will have it's own UI in view files and Business Logic in Controller files
- The entire UI hierarchy and Business Logic are available or accessible to its parent container
- Master in MVC format can be converted to User widget
- Component without Contract will have it's own namespace and a class name
- this.view.parent will NOT provide access to its parent view in controller, but widget.parent in parent controller will provide its parent view
- Template can be Flex container or Flex scroll container
- Existing Masters created in version 7.x will continue to work in the similar way in both MVC and non-MVC projects
- Component without Contract can be exported and imported
- Component without Contract can be used in both MVC and non MVC projects

Creating a Component (with example)

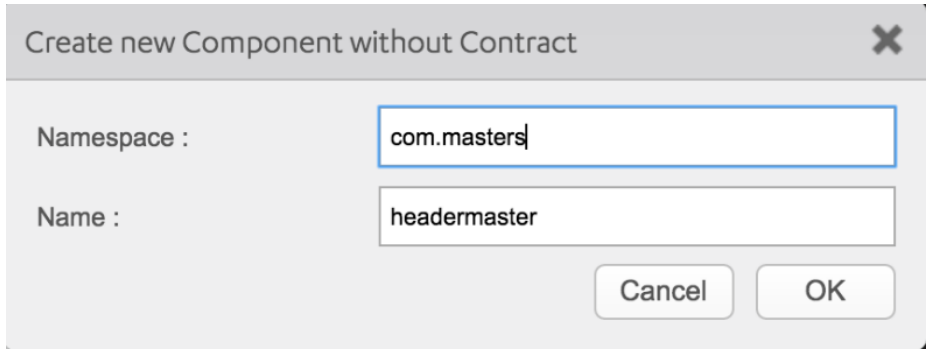
- Steps to create a simple header
  - Go to the **Templates** section and click on the **Components**





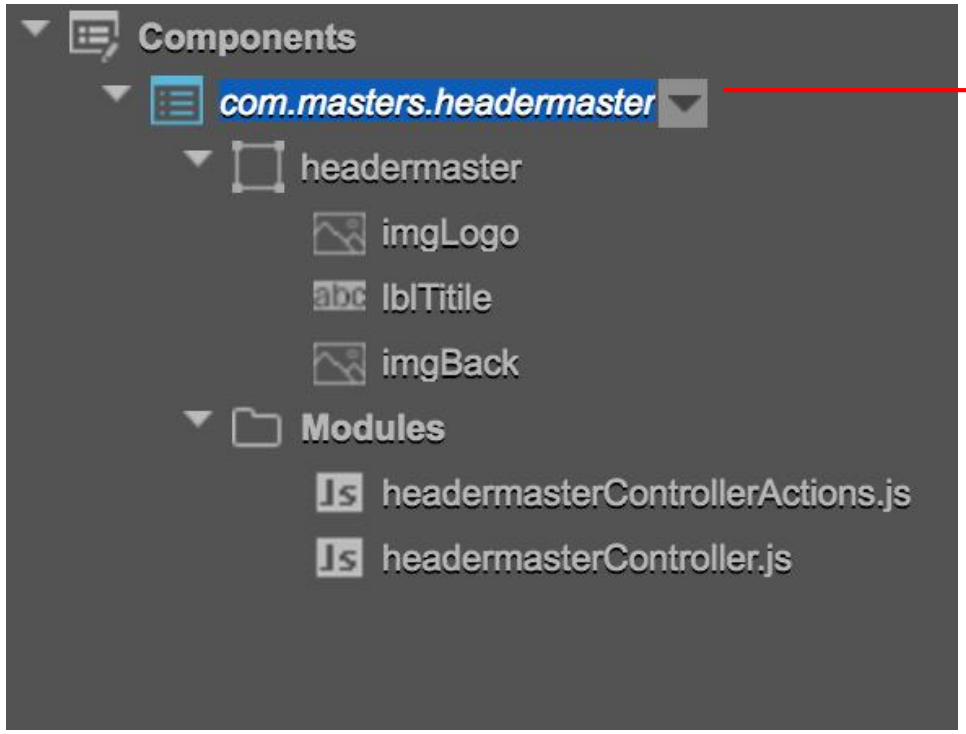
Creating Component without Contract

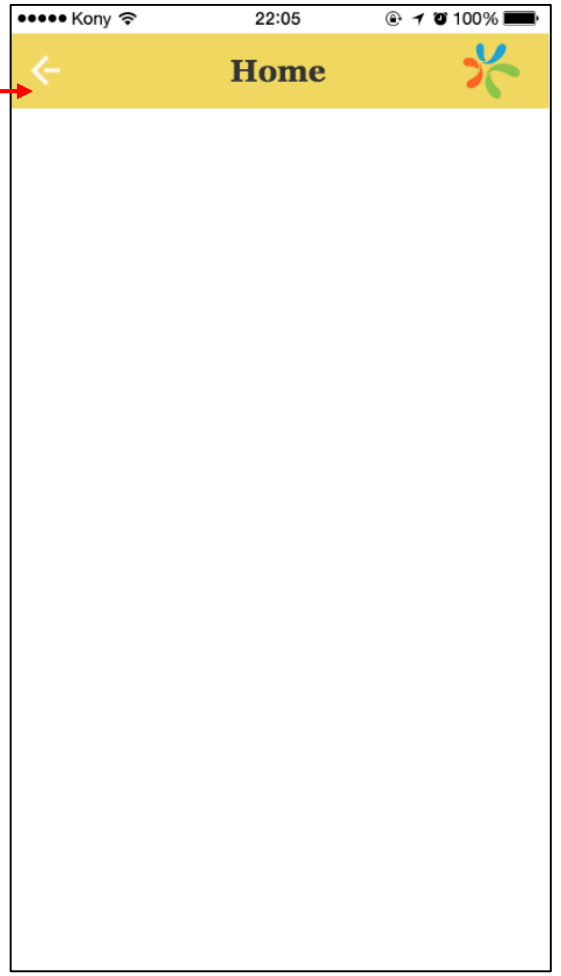
- While creating a component, provide values in the namespace and the name in the corresponding text boxes



- Once the component is created, the related controller and ControllerActions will get created under modules folder
- The naming convention is <component_name> + <Controller>.js and <component_name> + <ContollerActions>.js

Creating Component UI





Controllers & ControllerActions

- Controllers
  - The controller & controllerActions be in require JS format
  - Controller will have all the coding components
- ControllerActions
  - ControllerActions.js will have all the action sequences associated with the component

Creating Module

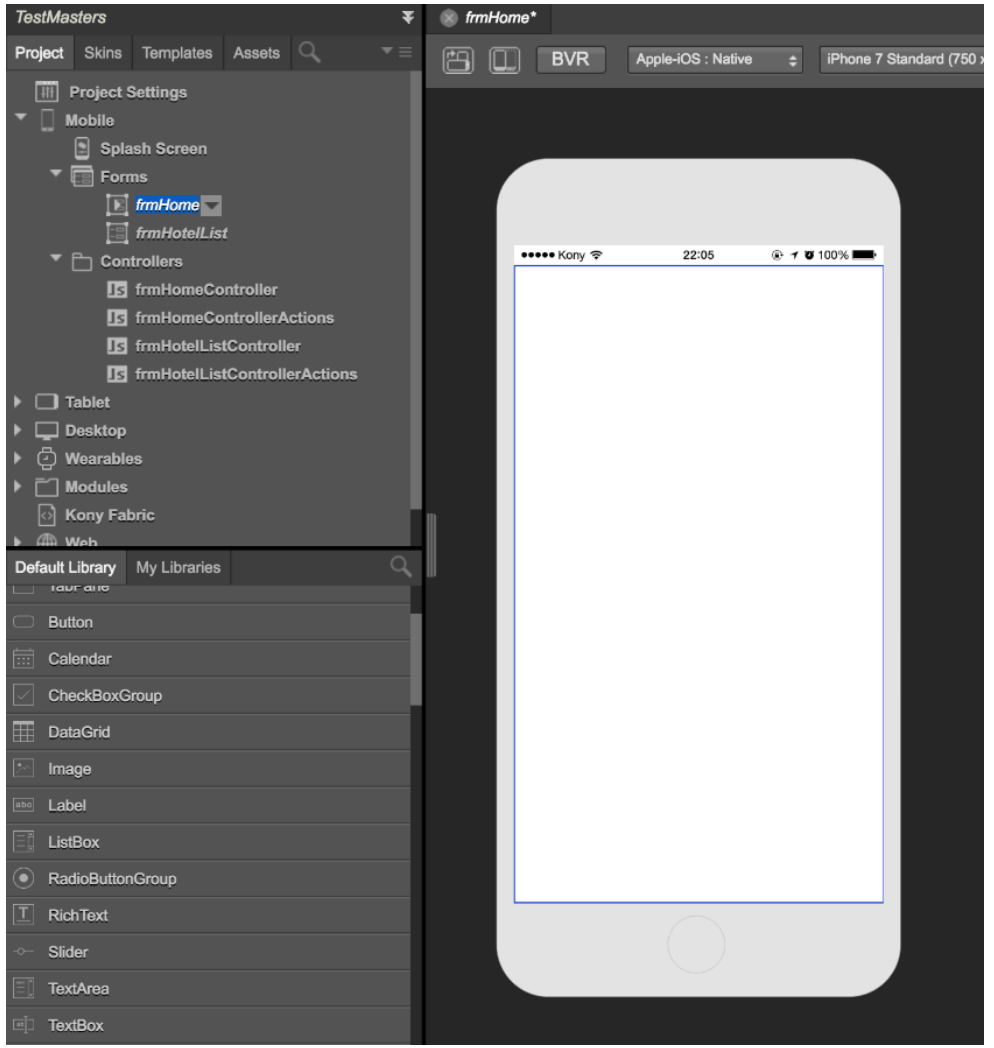
- The header has a back icon, then the icon is clicked, it has go to the previous page. We will implement the logic in Component
  - Get the id of the current form
  - Create Navigation Object, and then navigate to the previous form
- To do so, go to the controller and add the following logic
- Using Invoke Action, add the logic to the back icon

Creating a Module

```
define(function() {
return {
        /*** @function navigateToPreviousForm
        * The function will be assigned to the onClick event of the imgBack.
        * This function will be called when the back button is clicked to
        * navigate back to the previous form.
        */
        navigateToPreviousForm: function () {
            kony.print ("*** Entering into navigateToPreviousForm ***");
            var currentFormID = kony.application.getCurrentForm().id;
            if("frmHotelDetails" === currentFormID) {
                  var navigateToFrmHotelList = new kony.mvc.Navigation("frmHotelList");
                  navigateToFrmHotelList.navigate(null);
            } else if ("frmHotelList" === currentFormID) {
                  var navigateToFrmHome = new kony.mvc.Navigation("frmHome");
                  navigateToFrmHome.navigate();
            }
            kony.print ("*** Exiting out of navigateToPreviousForm ***");
          }
    };
});
```

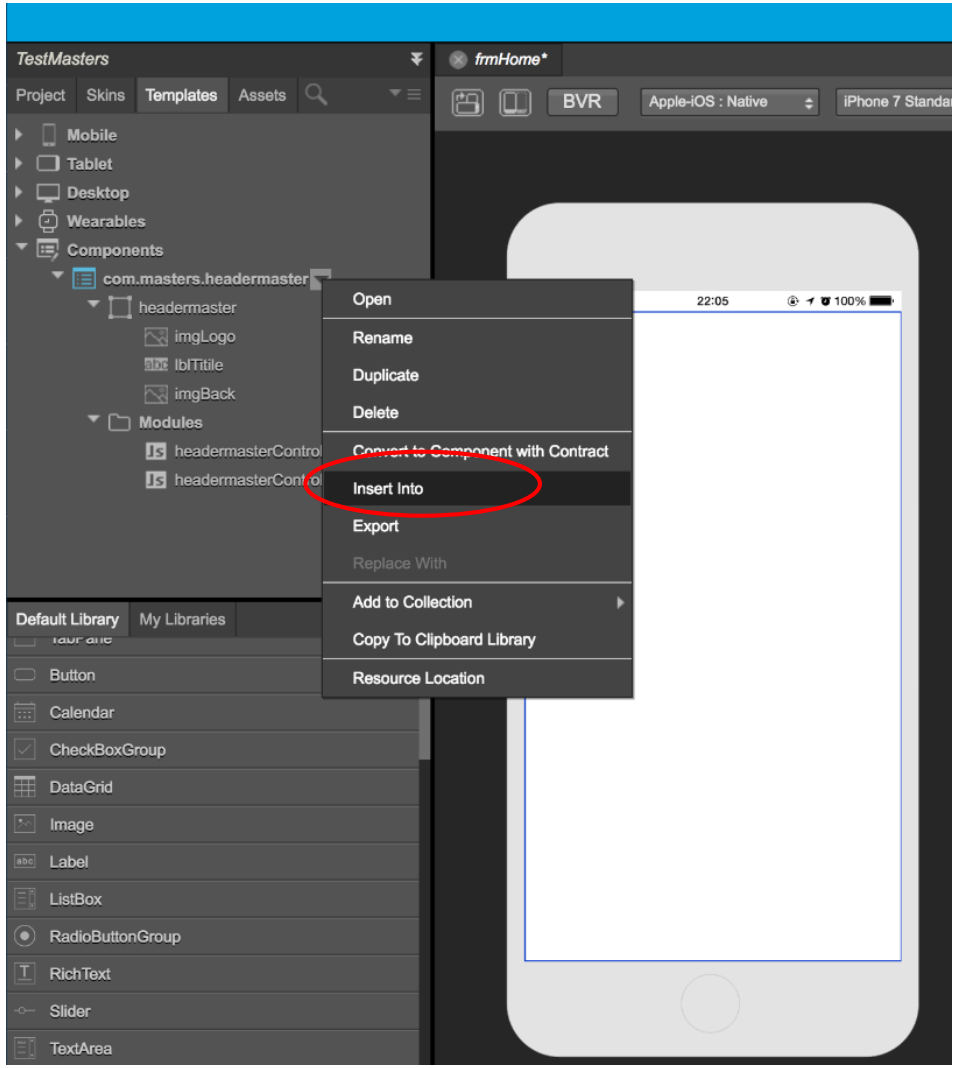
Adding Component to the Form

- Create the following forms
  - frmHome
  - frmHotelList



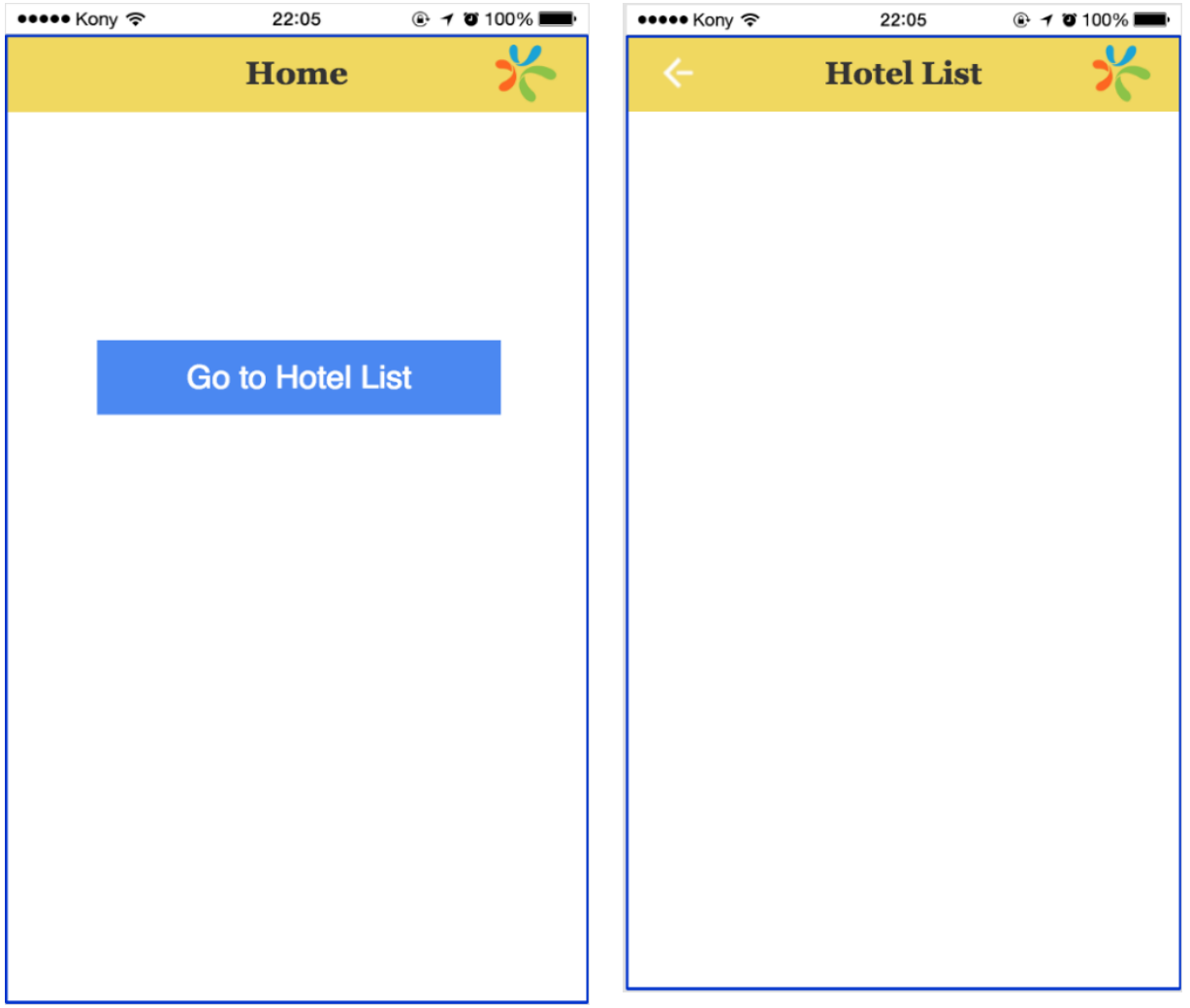
Adding Component to the Form

- To add a component to the form
  - Select the form
  - Go to templates tab → components →
  - Right click on your component (or) click on context menu → choose “insert into” option
  - The component will get added to the form
  - Do this exercise for ”frmHome” and “frmHotelList”

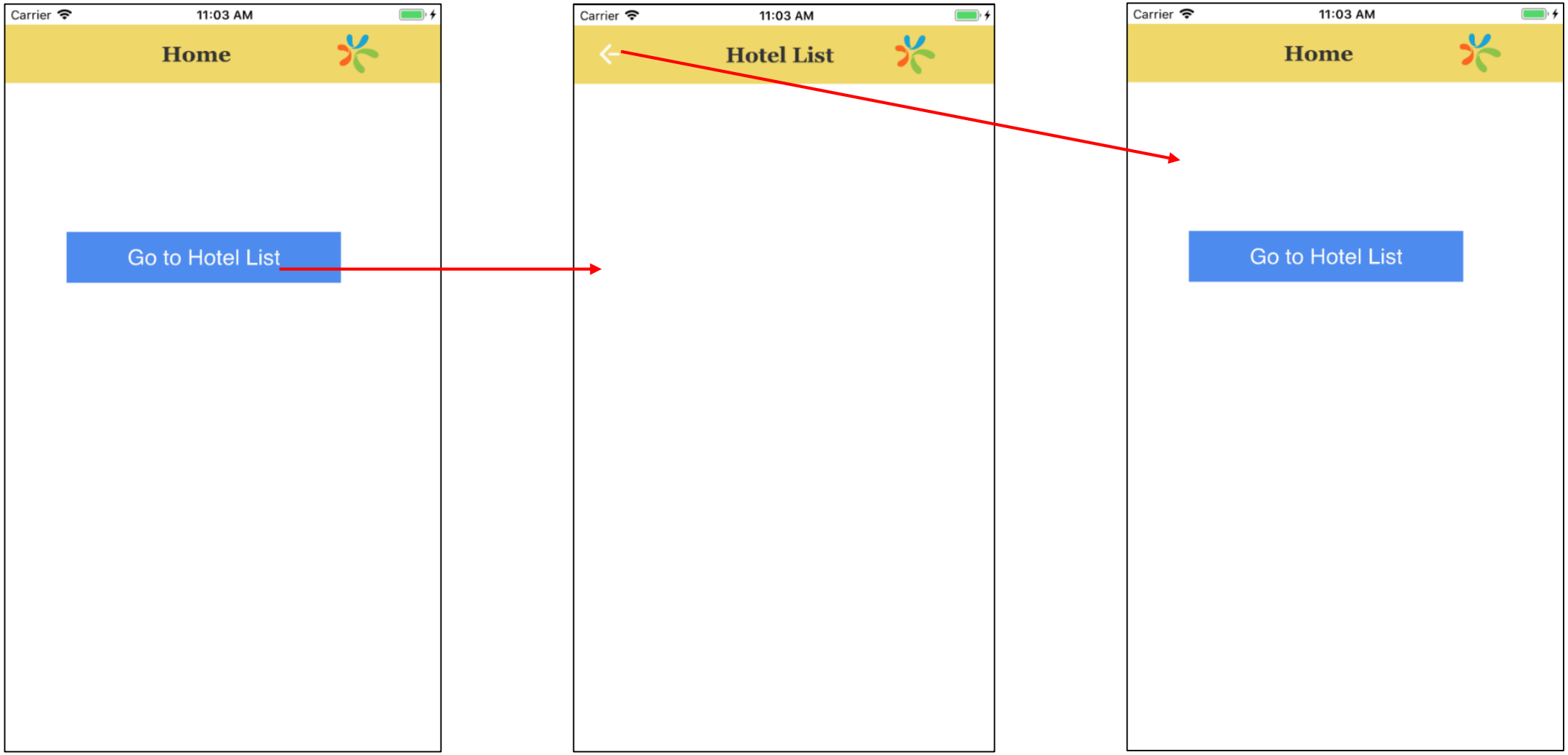


Customizing the Component

- Make the Back icon invisible on the header for ”frmHome”, as it is a startup form
- Add a button to “frmHome” for navigation
- Change the header title of “frmHotelList”



Exercises Execution



Editing Components in Form

- Everything of Component is directly available to its parent
- Component's UI can be used as it is or it can be customized to meet the app requirement
- After the Component is added to the form, any additional functionality code can be added through the forms controllers
- The newly added functions can be called "this.view.functionName()"

Components With Contract

Component Overview

Components are the truly 'self-contained' and reusable entity which enhances master feature-set with these additional capabilities -

- Include Code Modules
- Include Kony Fabric Services
- Include 'contract' to simplify usage and enable Rapid Mobile App Development
- Reuse in same project
- Reuse across projects in workspace
- Share with wider audience via Marketplace or local share functionality
- Download & use components created by Marketplace Assets team and other developers in your projects

Components Overview

- User widget is created only in MVC format
- User Widget/Component will have UI in View files and Business logic in Controller. Additionally, configuration or contract is placed in config file. The config file is placed under
  - <workspace>/<projectName>/userwidgets/<userwidget_namespace>/uwProperties.json
- User Widget Visual hierarchy and business logic are not directly available to its parent container
- Any portion of View and/or business logic can be exposed to its parent via contract
- User widget will not be converted to Master
- App developers can override constructor call
- executeOnParent is not available on user Widgets. Events is the only way to communicate with its parent
- Components can be used in Non-MVC projects as well

Component Creator

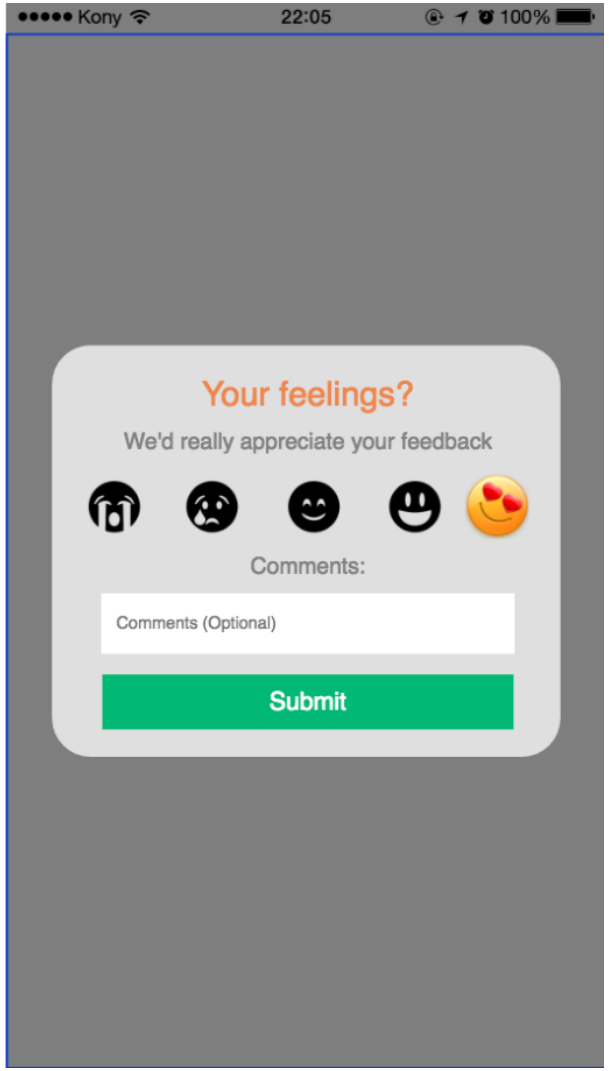
User widgets has two paradigms:

- Creator
  - Creates a user widget and share the widget for other app developers
  - Exposes the properties/events/methods via contract for the consumer
- Consumer
  - Consumer will import the component and use it just like any other widget during app development
  - Consumer can use the exposed properties/events/methods to make any changes as per his requirements

Create Feedback Component

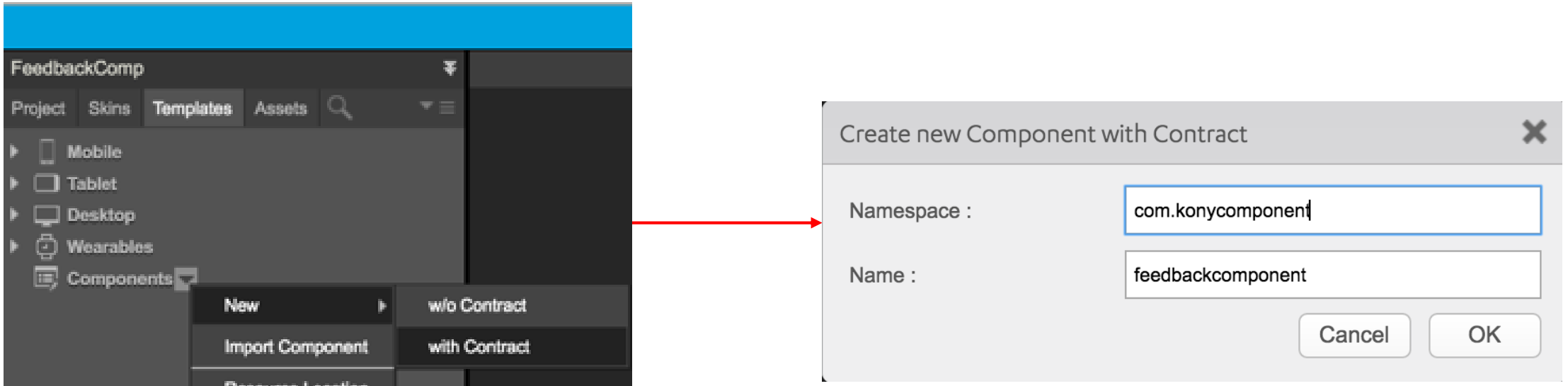
Feedback Component App - Features

- The feedback app will be used to provide feedback in graphical way using images
- It also gives you an option to enter comments in text format
- As part of this app, you will create the feedback component as shown in the screenshot
- When you select a feedback smiley, rest of the smileys become black
- Apart from that, you will be able to add some additional comments in the comments box and submit the feedback
- In the next few minutes, you will learn how to consume a component and how to customize the given component UI and how to use the properties exposed by the component



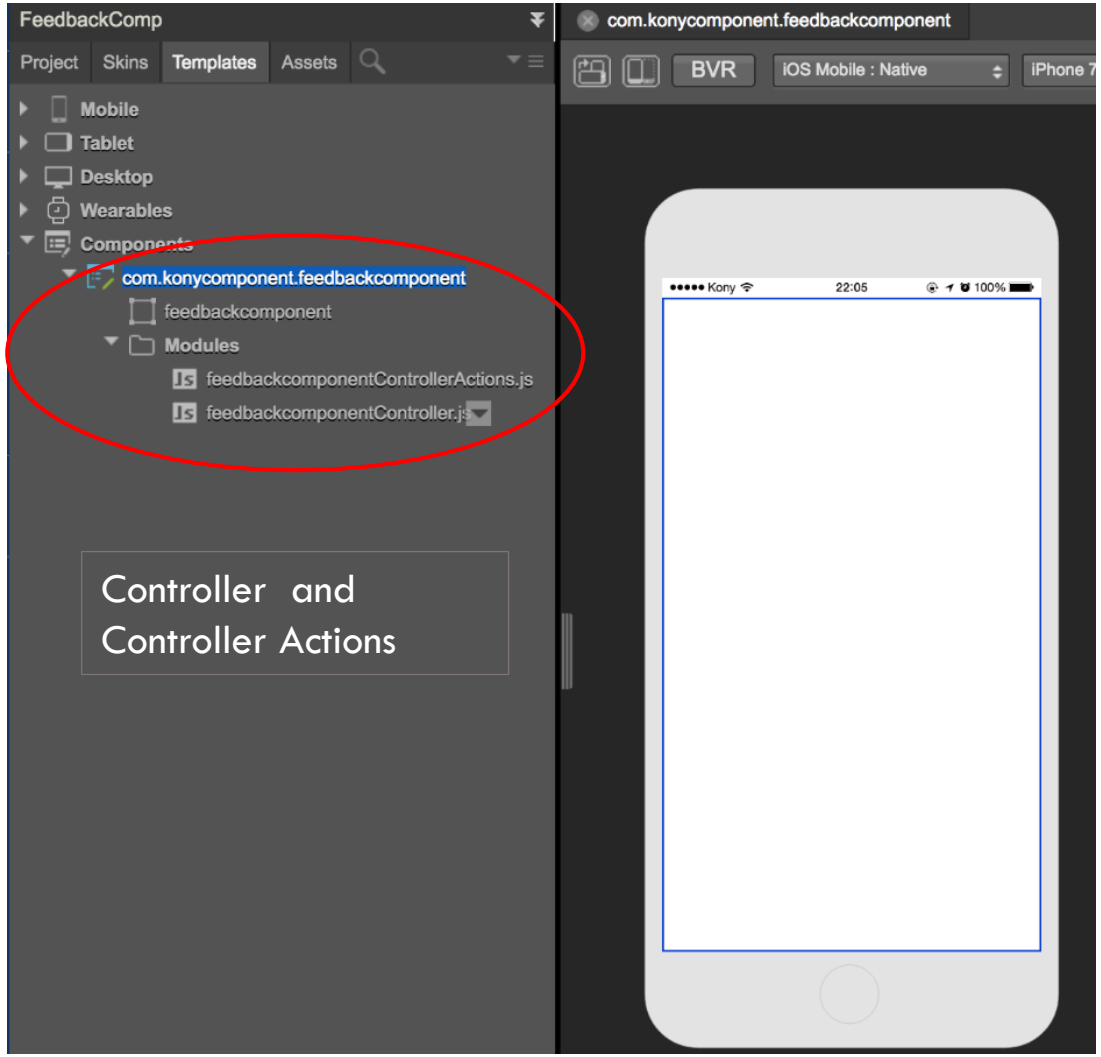
Feedback Component App - Features

- Go to project explorer and click on the templates tab
  - Hover over Components, click its context menu arrow, hover over New, and then select with Contract
  - A dialog box appears
  - Enter a namespace and a name for the component

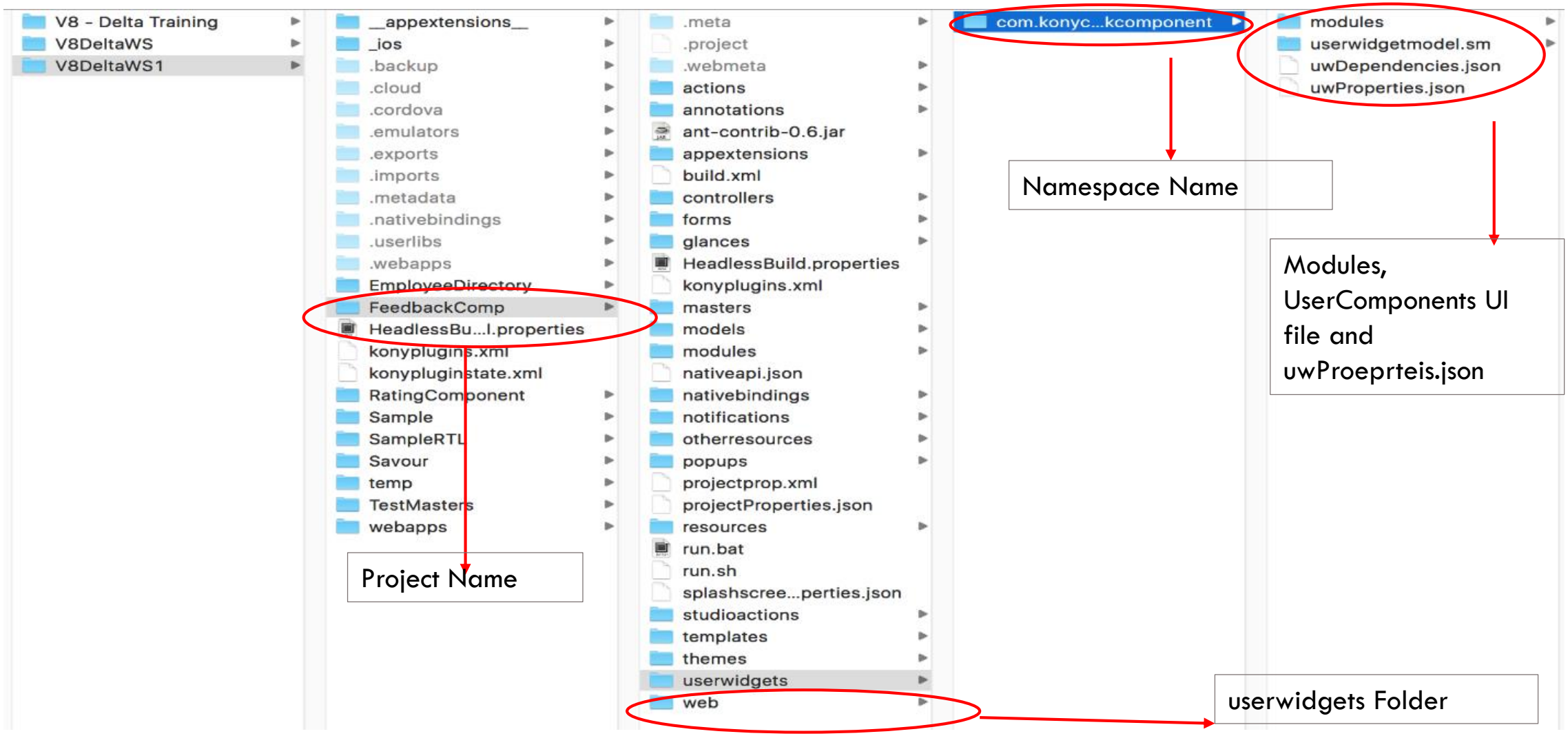


Creating Feedback Components

- Kony Visualizer creates a new component, including a FlexContainer to hold all widgets that you would add to the component
- It also creates module node containing the component's *controller* and *actions* controller JavaScript modules

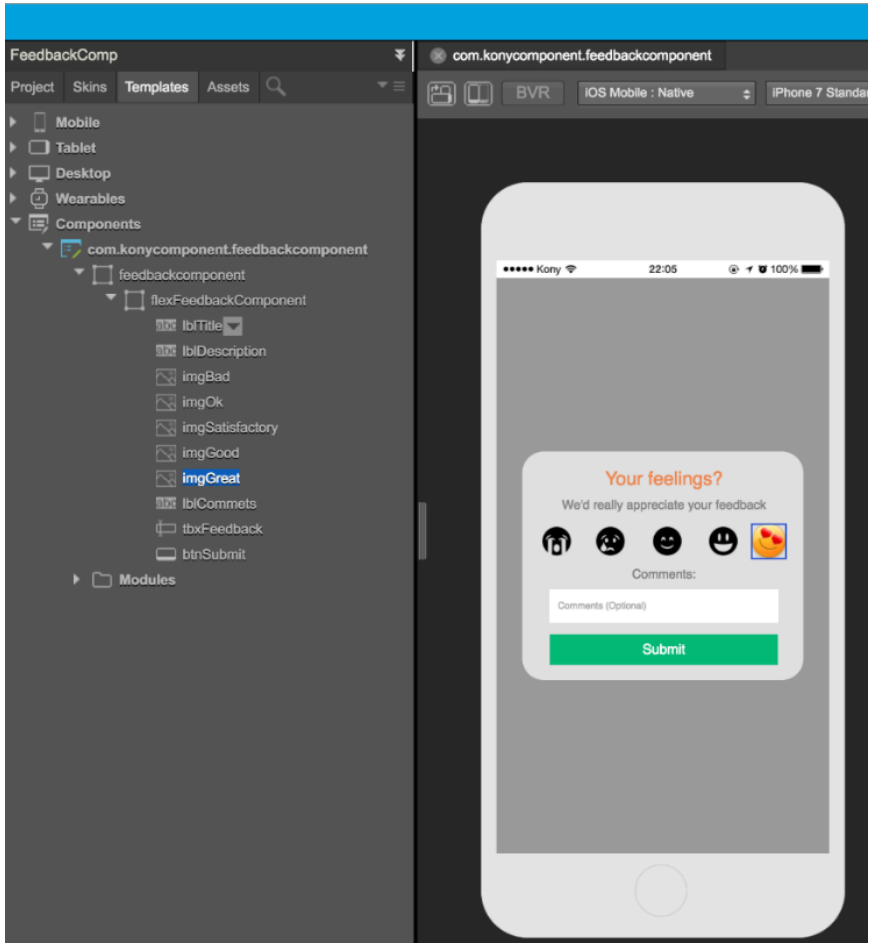


Component Folder Structure



Feedback Component UI

- Create a simple UI as shown in the screenshot



Creating Feedback Component

- What does contract means?
  - Visual hierarchy and business logic are not directly available to its parent container
  - Hence, User widgets will expose few properties/events and APIs for consumer
- Contract is a thin wrapper which can be used to:
  - Mask component internals
  - Expose a simplified set of configurations
  - Allow easy extension and update of component by other users
- All such components have 3 configuration levels for component consumer:
  - Properties
  - Events
  - Methods

Feedback Component - Manage Properties

- Components expose the attributes which can be used to configure look, features or any other characteristic of component
- Any Component will have 2 types of properties and APIs
  - Pass-through properties
    - Pass through properties are the default properties of the underlying widgets used in the component
    - All properties like Basic, Flex Layout, Layout, Platform Specific and events of the widgets are supported as pass-through properties
    - Creator can expose few properties for consumer to change while using it

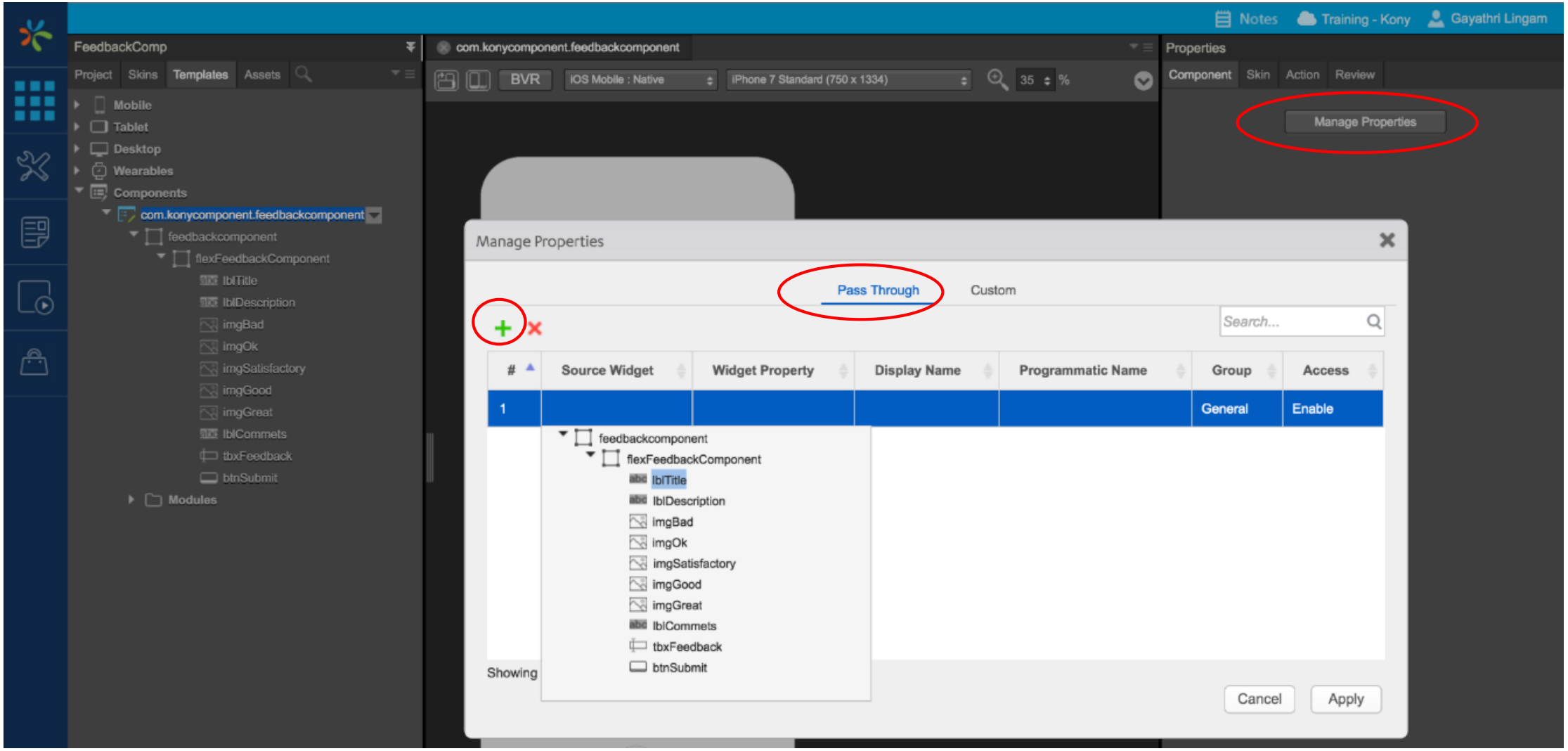
Feedback Component - Manage Properties

- Custom Properties
  - Apart from pre-defined properties, component creator can define few additional attributes which are not directly associated with any widget. They are called Custom Properties.
  - The creator will take care of the implementation part, if any custom property is exposed
  - The customer attributes can be of different types:
    - Boolean
    - List
    - String
    - HTML
    - List Selector
    - Data Grid

Feedback Component - Manage Properties

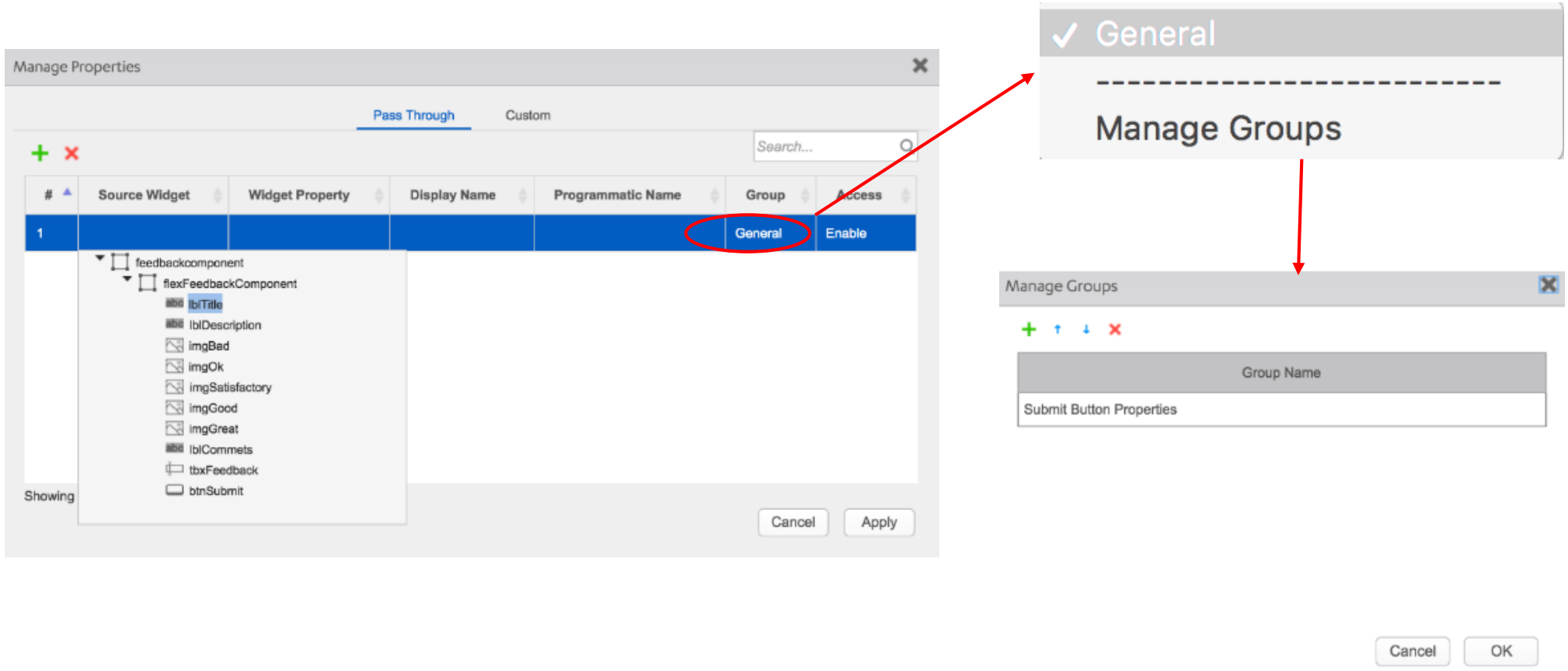
- Feedback Component's pass-through properties
  - lblTitle - Text
  - lblDescription - Text
  - tbxFeedback - Placeholder
  - btnSubmit - Text and skin
  - flxFeedbackComponent - Skin
- Feedback Component's custom properties
  - _rating - number - used to define rating based on the image chosen by the user
  - _feedback - String - used to capture the descriptive feedback

Feedback Component - Pass-through Properties

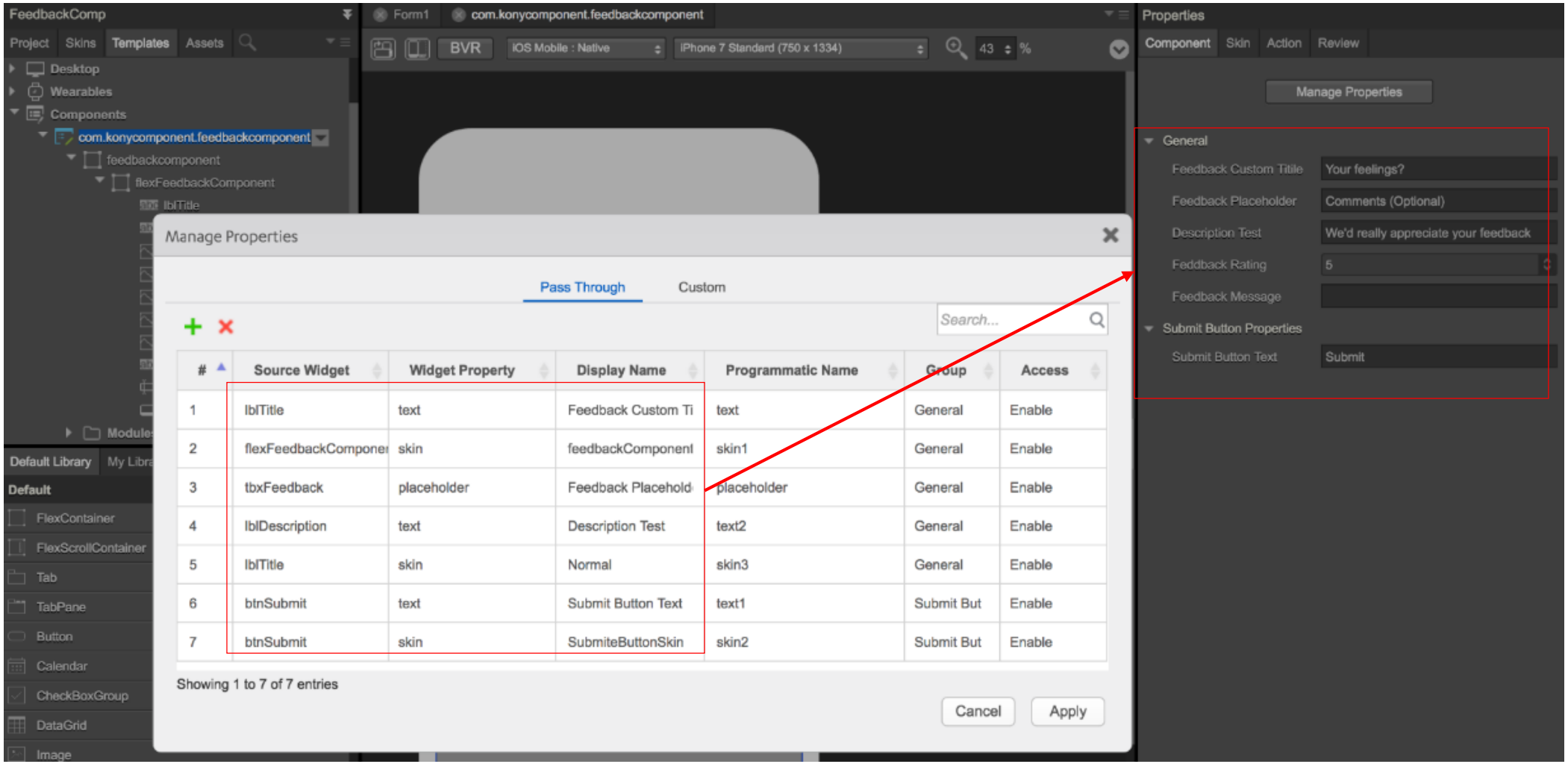


Feedback Component - Manage Groups

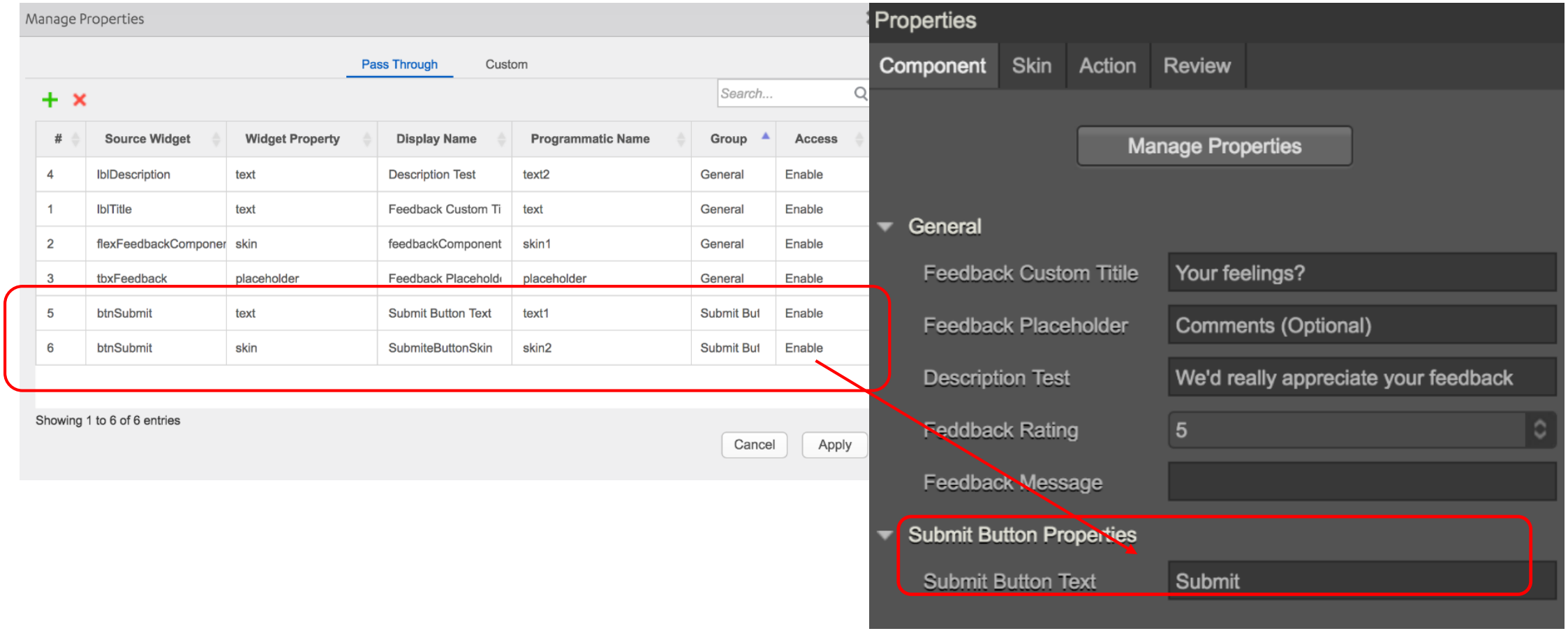
- Manage Groups will be used to group the exposed properties to achieve more granularity



Feedback Component - Pass-through Properties



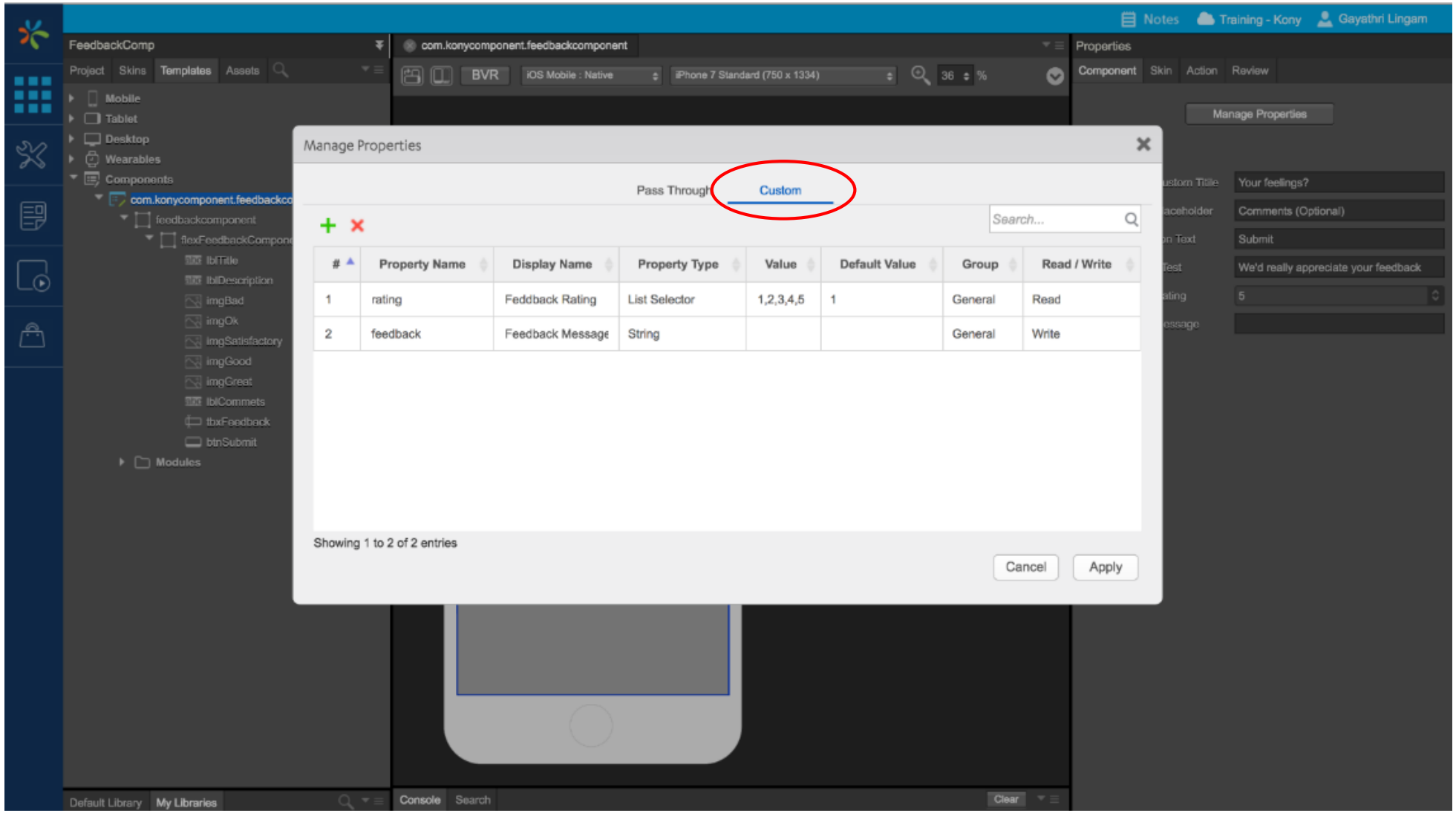
Feedback Component - Manage Groups



Feedback Component - Pass-through Properties

- As shown in earlier slide, you can define all the required widget predefined properties of the underlying widget
- Similarly, when you define a skin, the entire skin property will be exposed to the consumer
- There is no option to expose specific property of the skin object. For e.g., if you want to expose only background of the skin property, you won't be able to do it
- When skin property is exposed, you may not be seeing under manage properties button, rather, you will see them under skin tab when you consume the component

Feedback Component - Custom Properties



Feedback Component - Custom Properties

- When custom properties are defined, they need to be initialized in the relevant controller's constructor, and setter and getter needs to be defined as shown below:

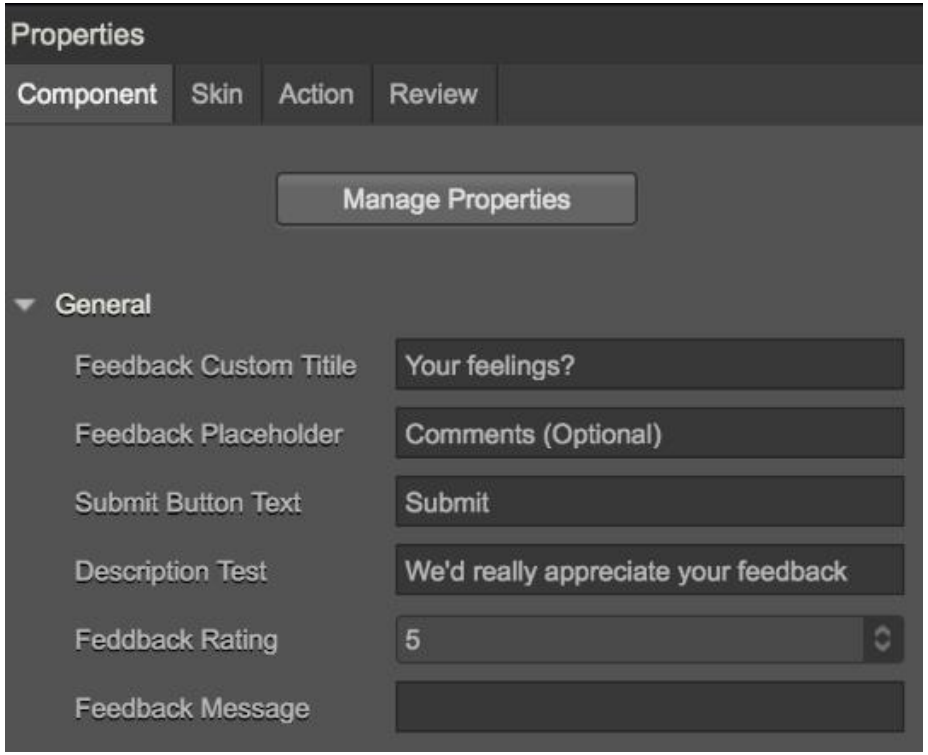
```
constructor: function(baseConfig, layoutConfig,
pspConfig)
{
 kony.print ("*** Entering into constructor ***");
_rating=0;
_feedback="";
kony.print ("*** Exiting out of constructor ***");

 }
```

```
initGettersSetters: function() {
defineGetter(this, "rating", function() {
        kony.print ("*** Entering into getRating ***");
        kony.print ("*** Exiting out of getRating ***");
        return this._rating;
});
defineSetter(this, "rating", function(rating) {
        kony.print ("*** Entering into setRating: "+rating+" ");
        this._rating = rating;
        kony.print ("*** Exiting out of setRating ***");
});
}
```

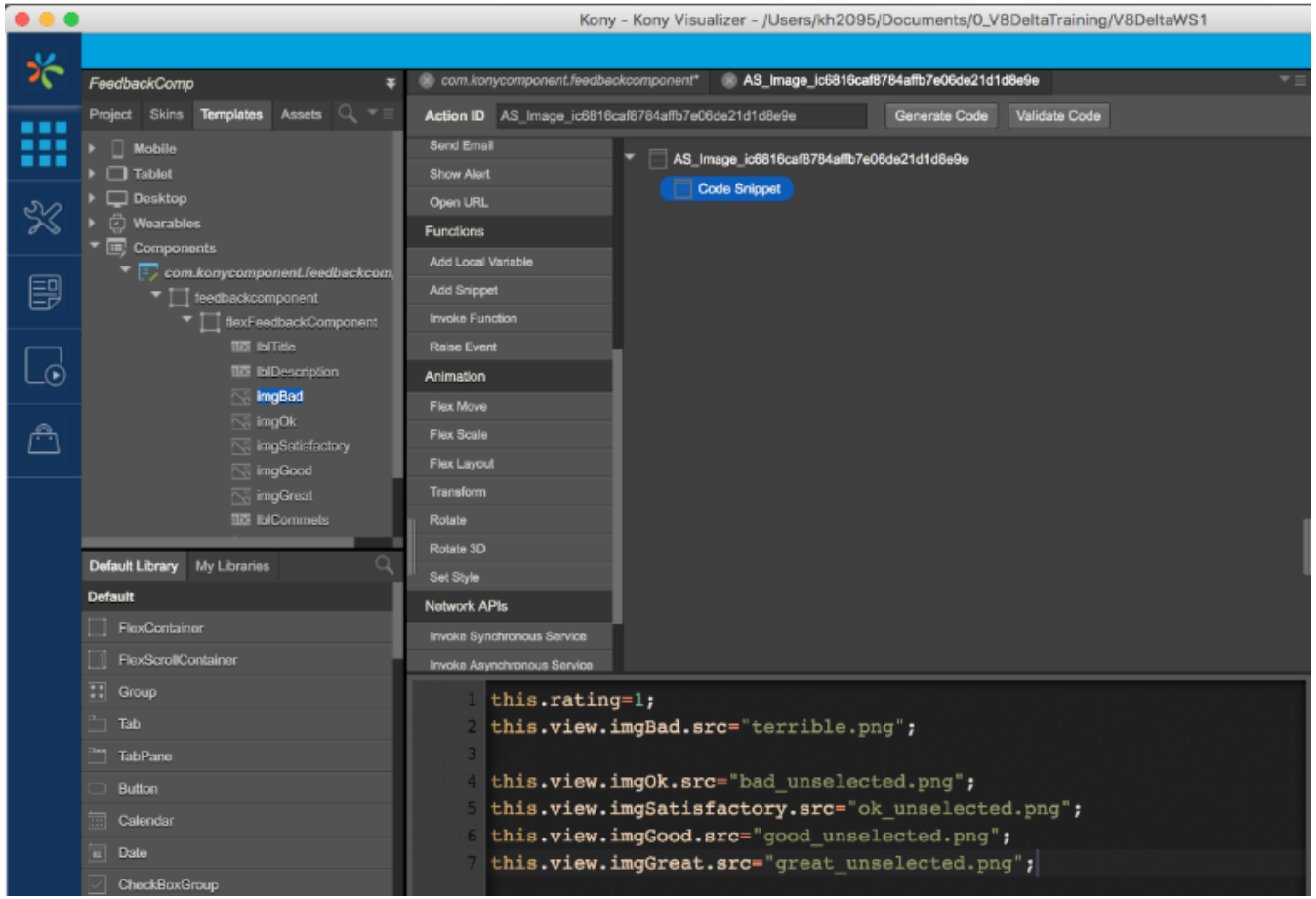
Feedback Component - Custom Properties

- The screenshot shows the list of properties that needs to be exposed for the consumer
- In this way, you will be able to expose the properties of the underlying widgets



Feedback Component - Adding App Behavior

- The UI is ready now, hence you need to add the functionality
  - The selected smiley should turn into yellow smiley and rest should turn into black
  - To do so, we have add the code snippet to each image



Feedback Component - Adding App Behavior

- Code Explanation

```
this.rating=1;
this.view.imgBad.src="terrible.png";
this.view.imgOk.src="bad_unselected.png";
this.view.imgSatisfactory.src="ok_unselected.png";
this.view.imgGood.src="good_unselected.png";
this.view.imgGreat.src="great_unselected.png";
```

- In this code, we are setting the rating value as 1 for bad
- Similarly, you need to set for other feedback levels

  - 1 - Bad
  - 2 - Ok
  - 3 - Satisfactory
  - 4 - Good
  - 5 - Great

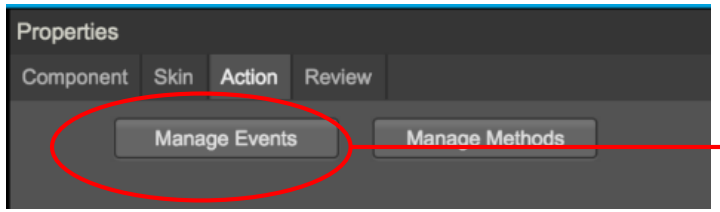
Feedback Component - Adding App Behavior

- We have two types of images in the shared assets. One for selected feedback and another for unselected feedback:
  - All the yellow smileys are for the selected feedback
  - All the black smileys are for the un-selected feedback
  - For e.g., terrible.png is a yellow smiley used for selected feedback and terrible_unselected.png is a black smiley used for unselected feedback
  - You need to set the images accordingly

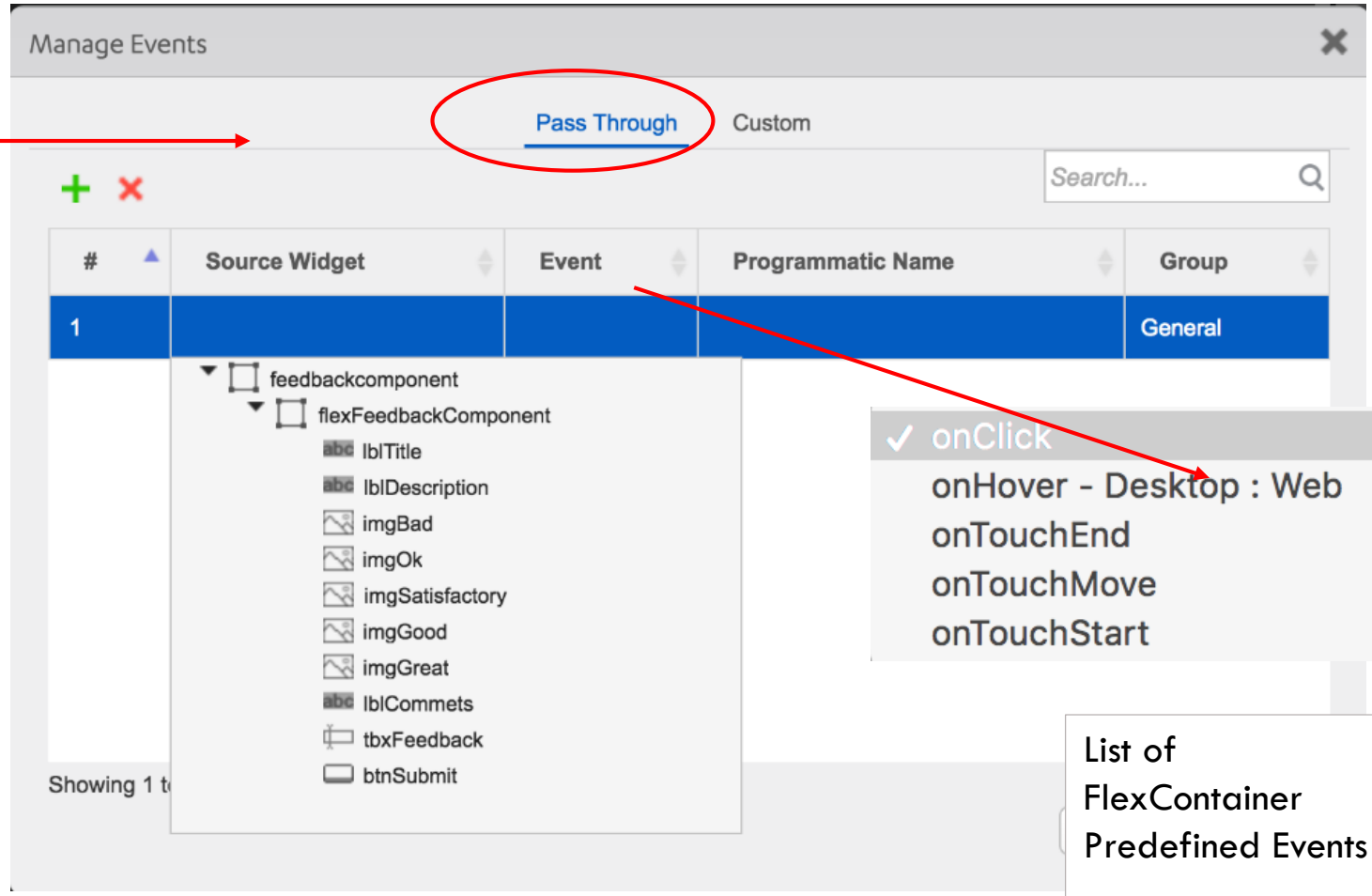
Feedback Component - Manage Events

- Event is a specific action occurrences which can be used by the component consumer to attach his custom logic
- The Manage Events or Component Events are of 2 types:
  - Pass-through events
    - Default events of the underlying widgets provided by the platform. For e.g. For Button, we have onClick(), onTouchEnd(), onTouchMove(), etc.
    - If the creator of the component wants consumer to implement any custom logic for the any of these events, then it can be exposed as a contract; In this case, the consumer will be able to see the exposed events and implement custom logic
  - Custom Events
    - Custom Events are not part of the regular events that platform exposes for any widget
    - Example: Normally, post login behavior is defined in the successcallback method; When it comes to component, the consumer will not have access to it unless the successcallback is exposed as custom event; Consumer can make use of such custom events to implement custom logic for the added components

Feedback Component - Pass-through events



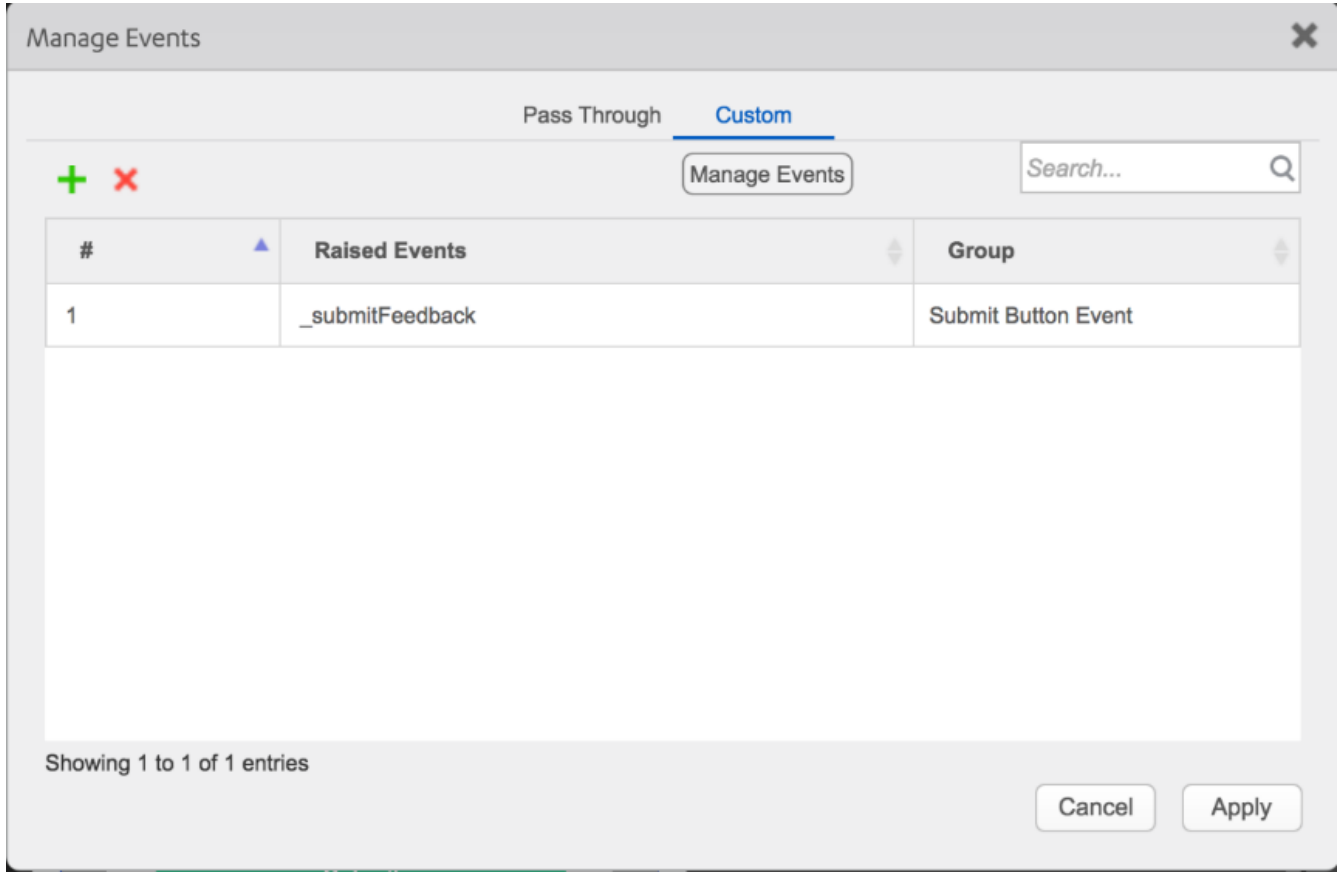
Clicking on "Manage Events" will open, Manage Events Window
Then, we can expose the predefined
events of the underlying widgets to the user



List of FlexContainer Predefined Events

Feedback Component - Custom Events

- Custom Events are not part of the regular events that platform exposes for any widget
  - Example: Normally, post login behavior is defined in the successcallback method; when it comes to component, the consumer will not have access to it unless the successcallback is exposed as custom event; consumer can make use of such custom events to implement custom logic for the added components



Feedback Component - Submit Button Behavior

- Implement the following code for the submit button

```
/**
 * @function _submitFeedback
 *
 * This function in mapped to the onClick of the Submit button on the component
 * This function internally invokes the method which consumer has mapped in the
 * submitFeedback event that this component exposes
 */
_submitFeedback: function () {
  kony.print ("*** Entering into _submitFeedback ***");
  if (null !== this.submitFeedback && undefined !== this.submitFeedback) {
    this.submitFeedback();
  } else {
    alert ("You have not defined any function against submitFeedback. \n" +
           "The default implementation of this component is to display the selected rating and feedback. \n "+
           "Rating: "+this._rating+" \n Feedback: "+this._feedback);
  }
  kony.print ("*** Exiting out of _submitFeedback ***");
}
```

Registering code for the custom event using it's raised event name. When the consumer defines any custom logic for submit button, it will execute the defined logic or it will execute the default implementation

Feedback Component - Manage Methods

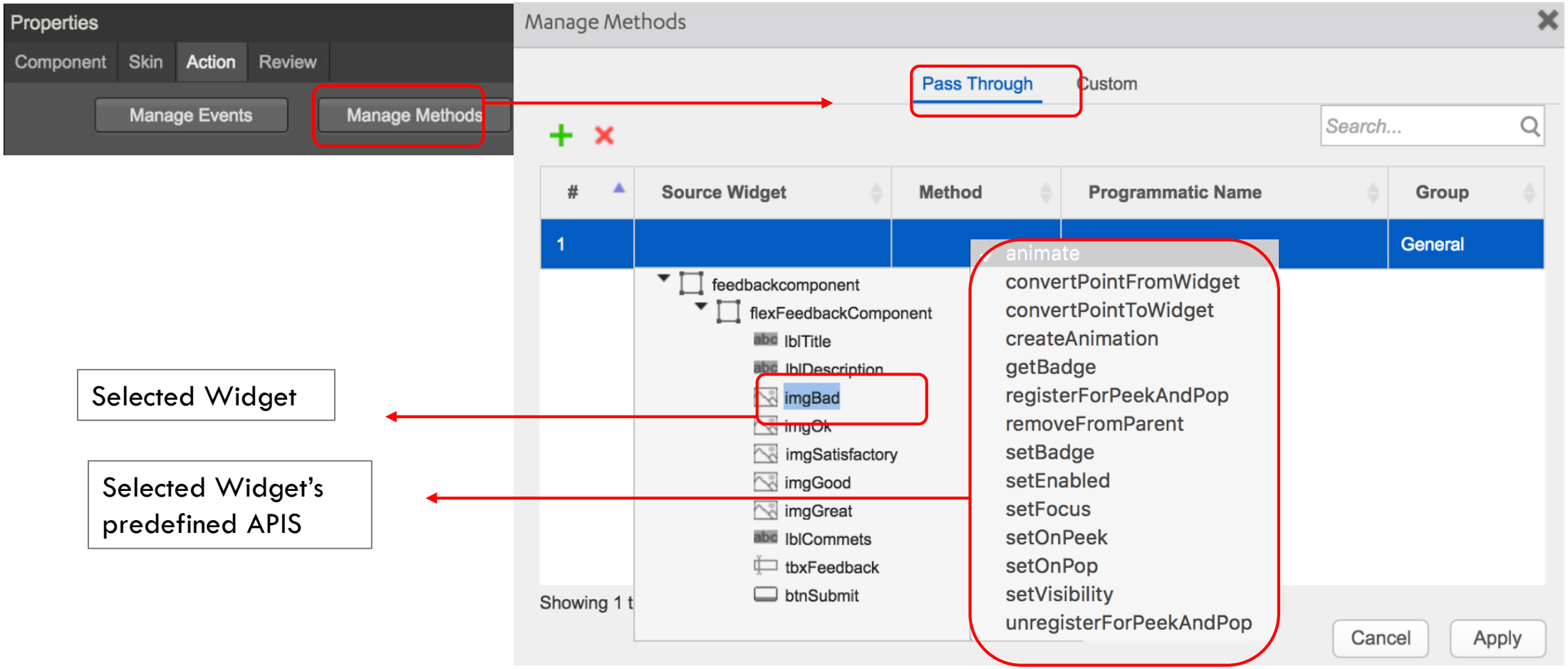
- Component creator can expose APIs which can be used by the component consumer
- The list of exposed methods/APIs will be shared in the documentation by the creator
- The consumer can make use of them, if required
- The Manage Methods / Component methods are of 2 types:
  - Pass-through Methods
  - Custom Methods

Feedback Component - Manage Methods

Let us understand the Pass-through and Custom methods

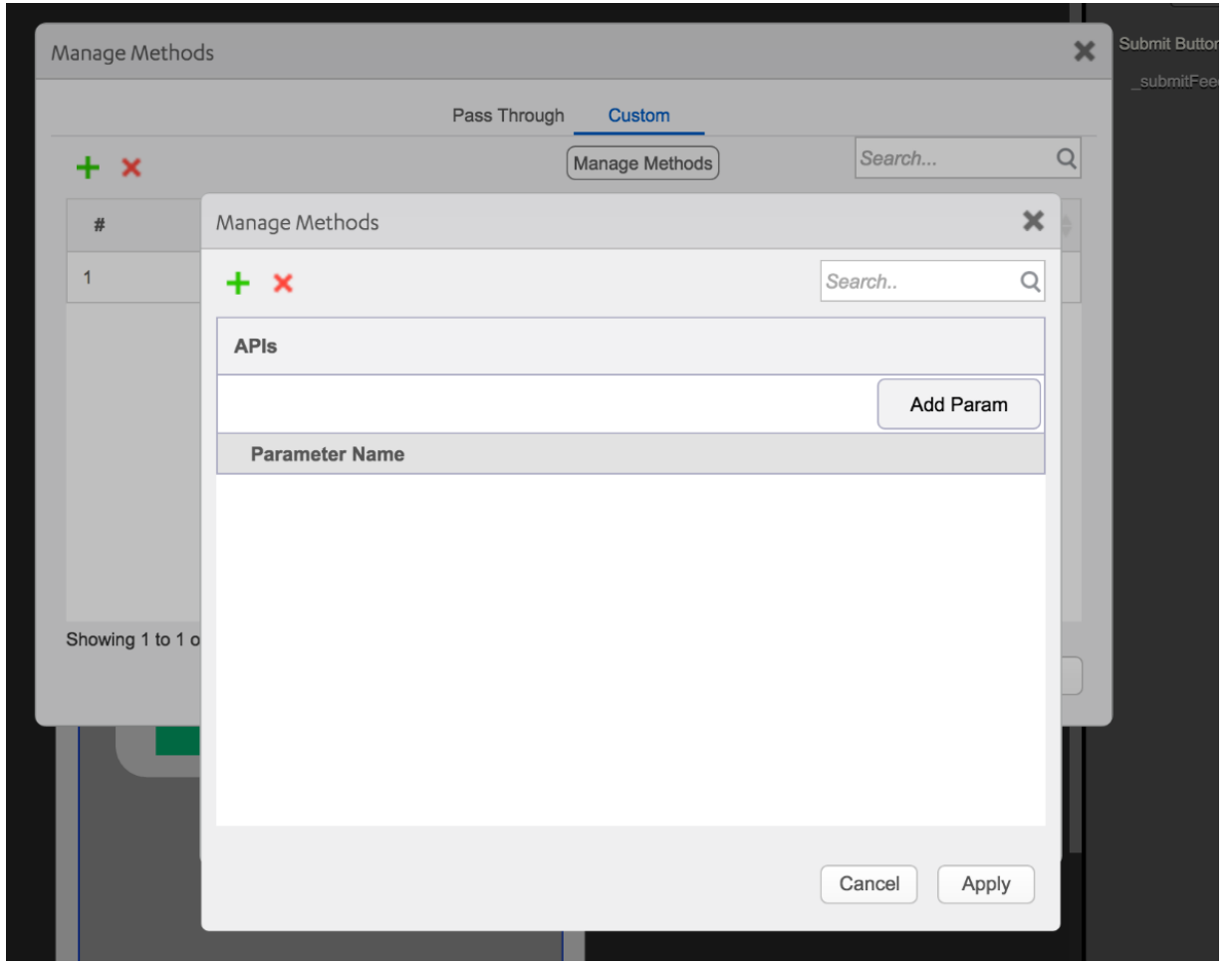
- Pass-through Methods
  - Default methods/API's of the underlying widgets provided by the platform. For e.g., for Button, we have setVisibility and setFocus APIs defined for the widgets
  - If the creator of the component wants consumer to invoke the available APIs, then it can be exposed via contract. In this case, the consumer will be able to access the exposed APIs
- Custom Methods
  - Custom Methods are not part of the regular APIs that platform exposes for any widget. E.g., if the creator wants to expose an API for setting background skin for the FlexContainer or Custom API for any login service, then it can be exposed via contract though the custom API. Consumer can make use of such custom APIs to implement custom logic for the added components
  - Unlike a pass-through method, a custom method has no built-in behavior. Creator must define the method's behavior programmatically and share the API details in the documentation for the consumer.

Feedback Component - Pass Through Methods



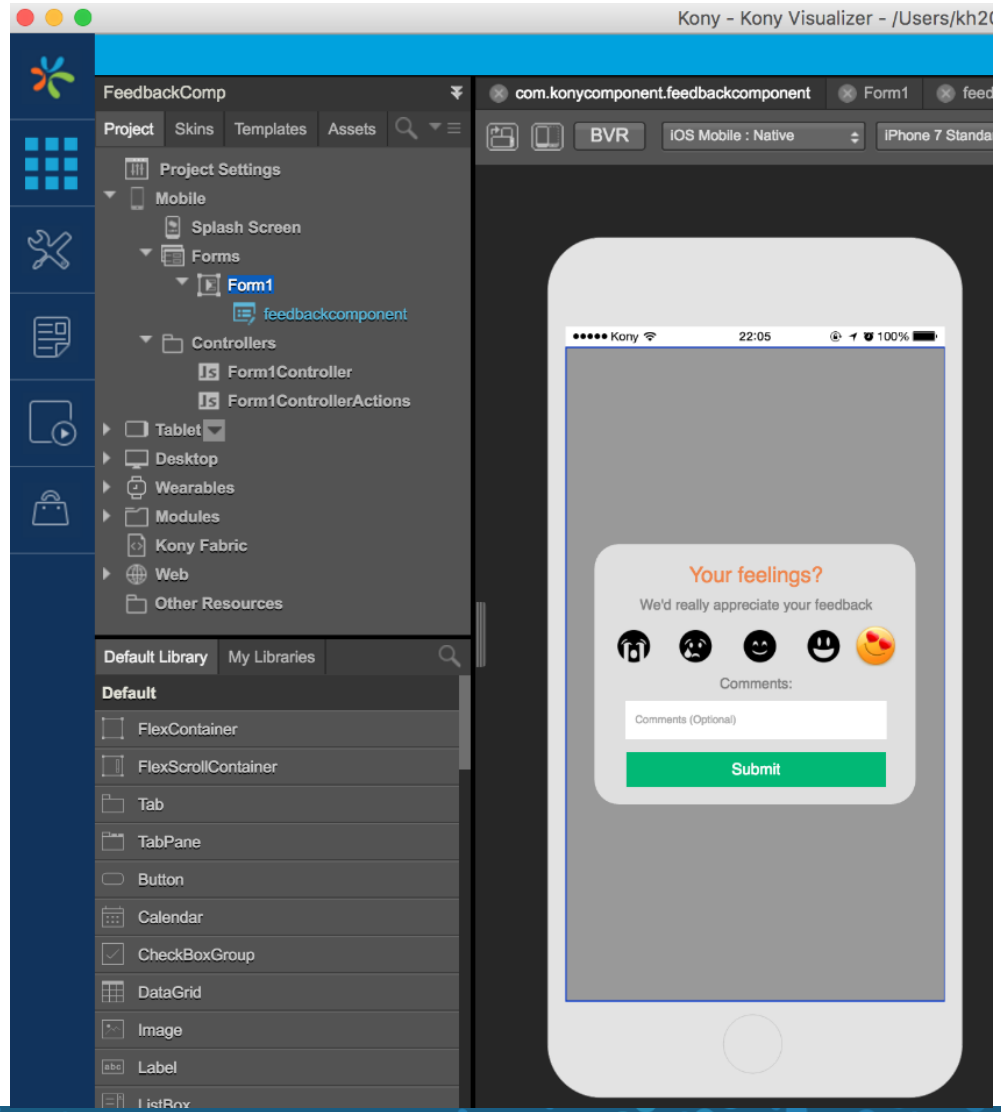
Feedback Component - Custom Methods

- Custom Methods are defined under Custom tab
- Add a custom method and define the parameters using Manage Methods option
- You wont be able to see these custom methods anywhere in the component
- The creator should create proper document of these custom methods and it's usage and parameters
- The consumer should read through the given documentation for the details

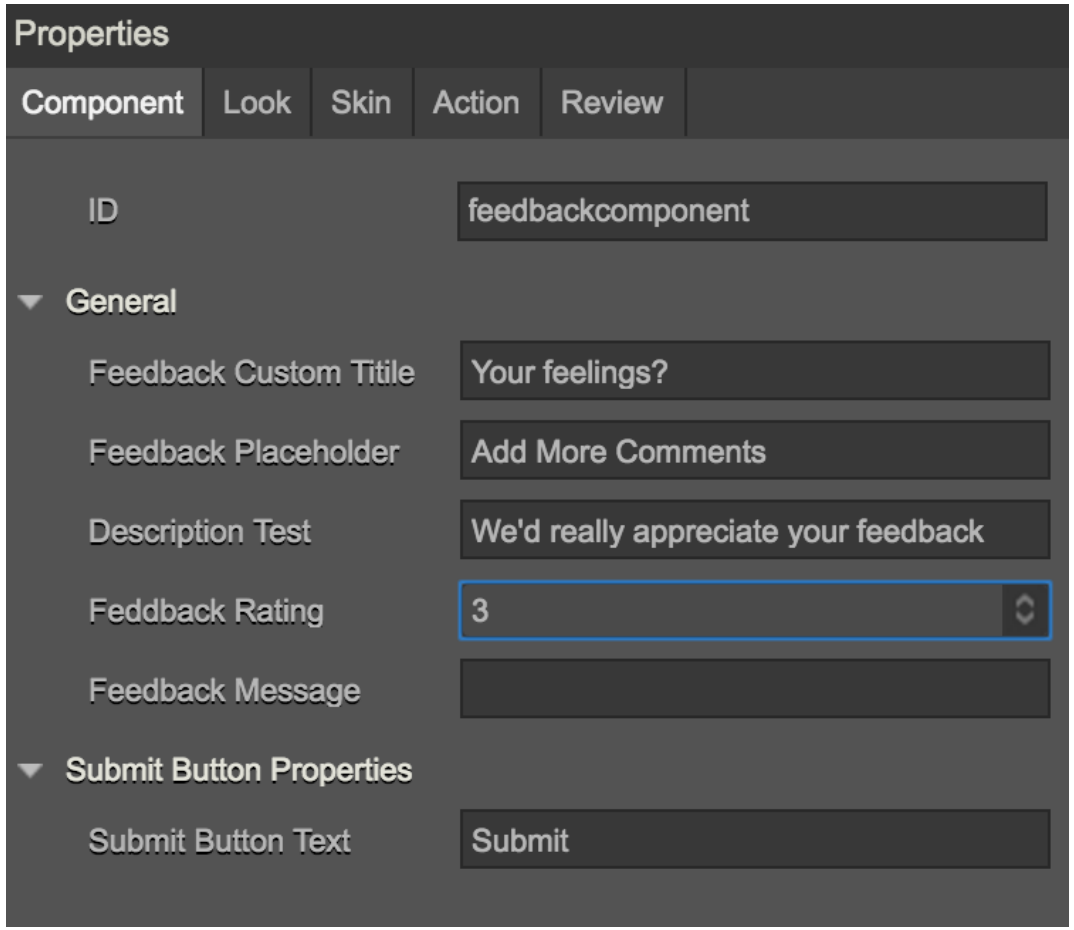


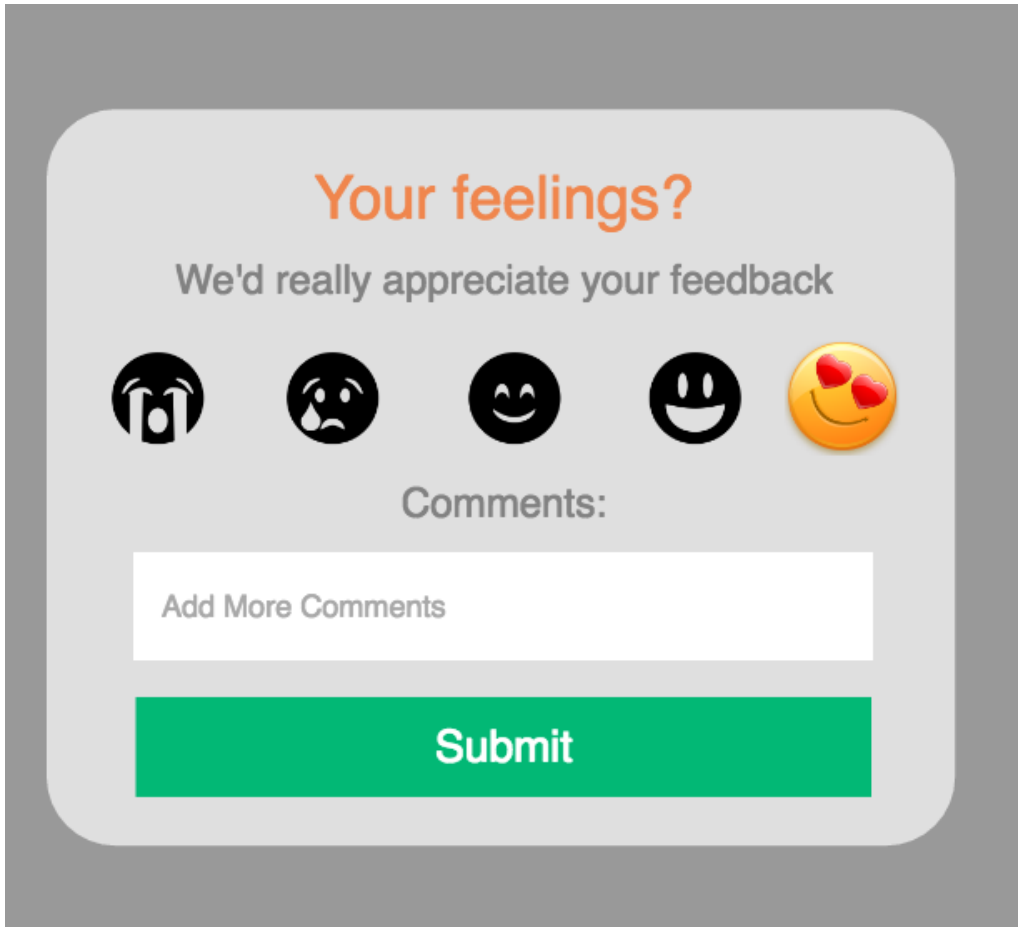
Consuming Feedback Component

- Create a form
- Drag and drop the component to the target form
- You won't be able to see the widget hierarchy, however, you can see all the exposed properties under Component Tab under Properties tab



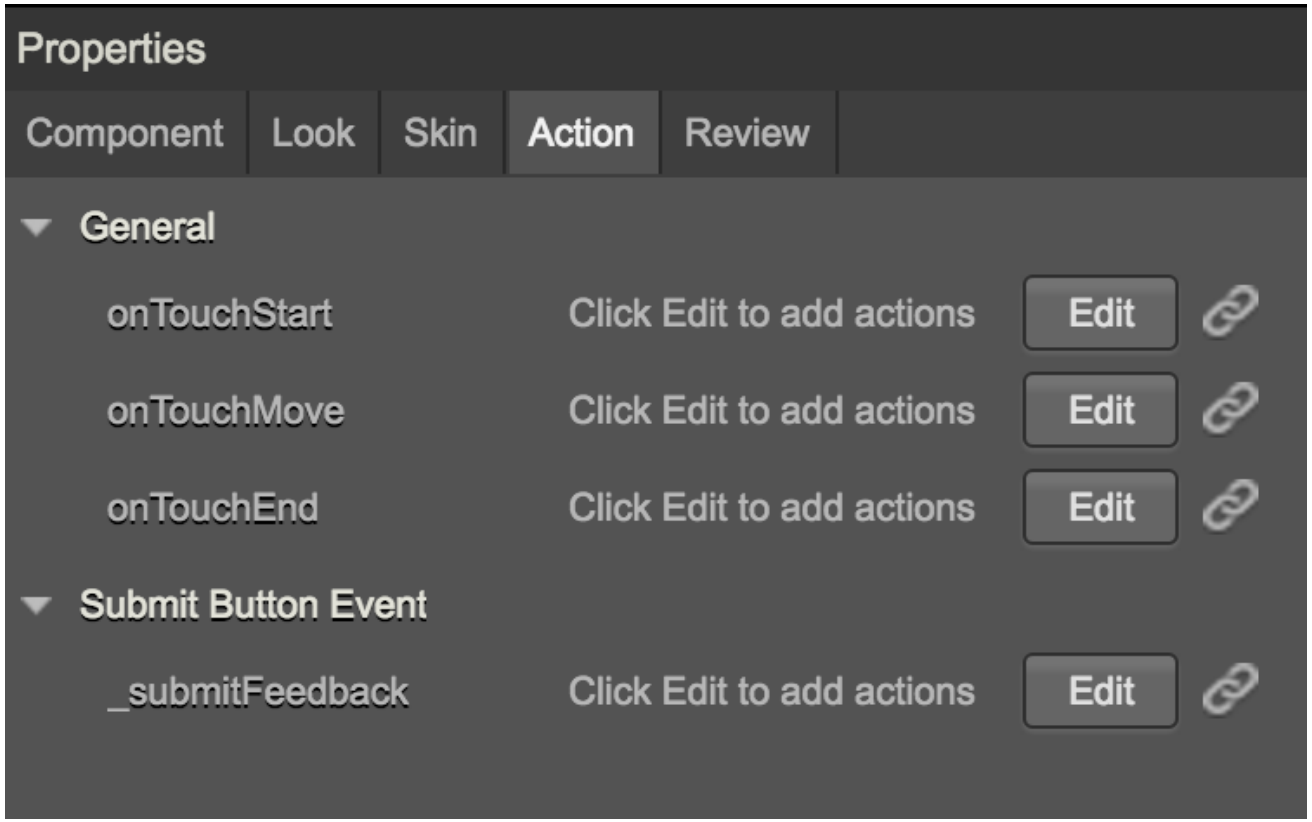
Consuming Feedback Component - Properties



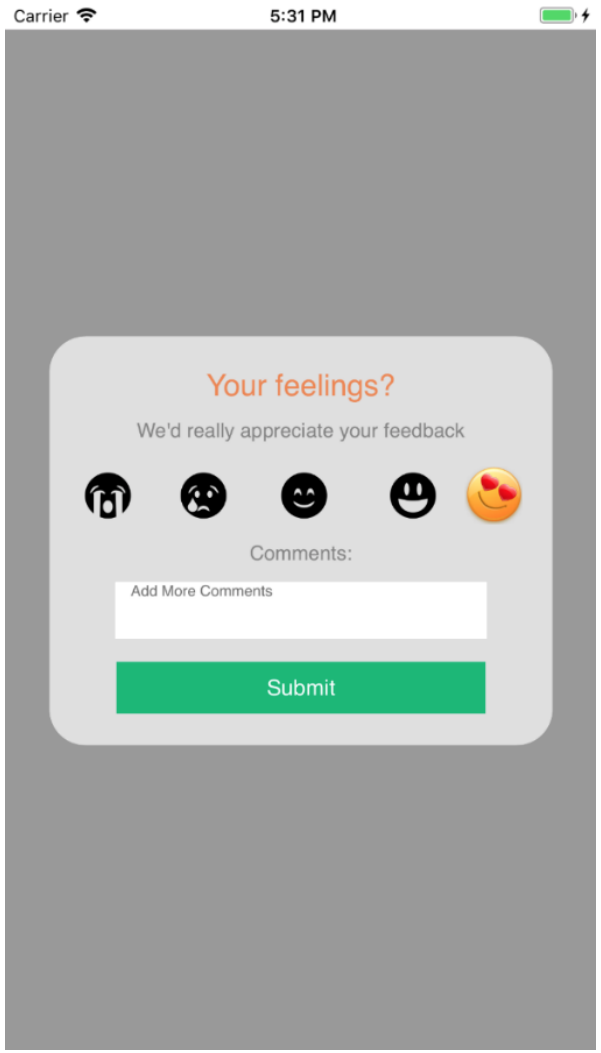


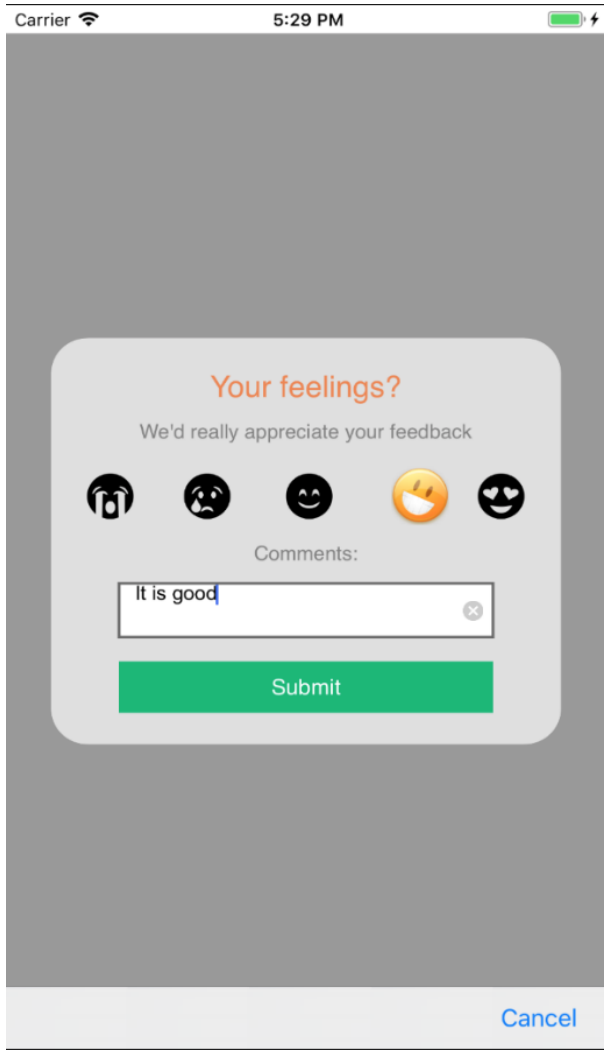
Consuming Feedback Component - Events

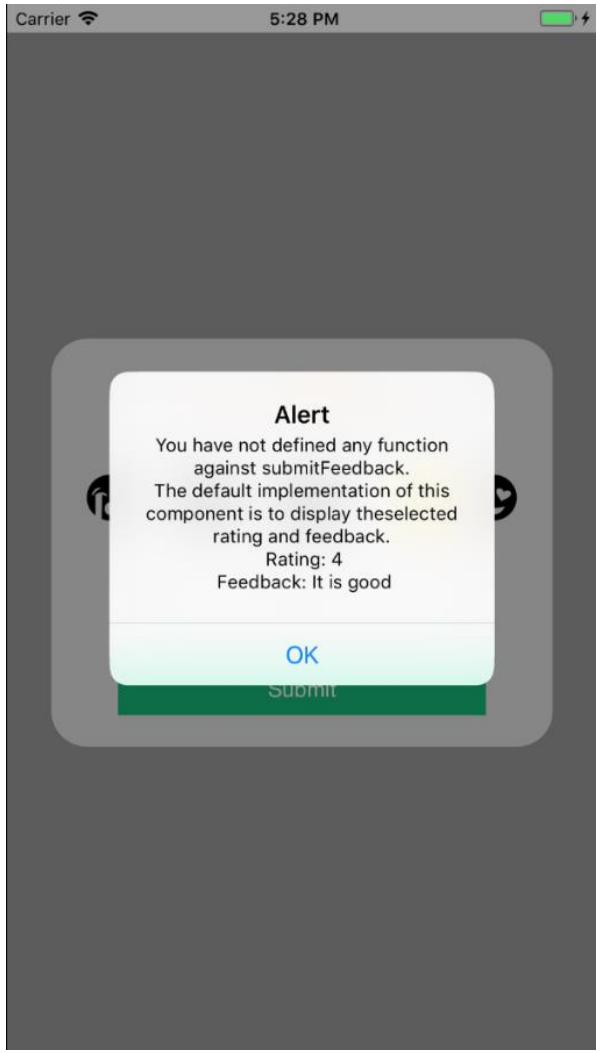
- You can add your own implementation to the exposed events
- When you don't provide your implementation, the default creator's implementation will be executed



Consuming Feedback Component - Run the App

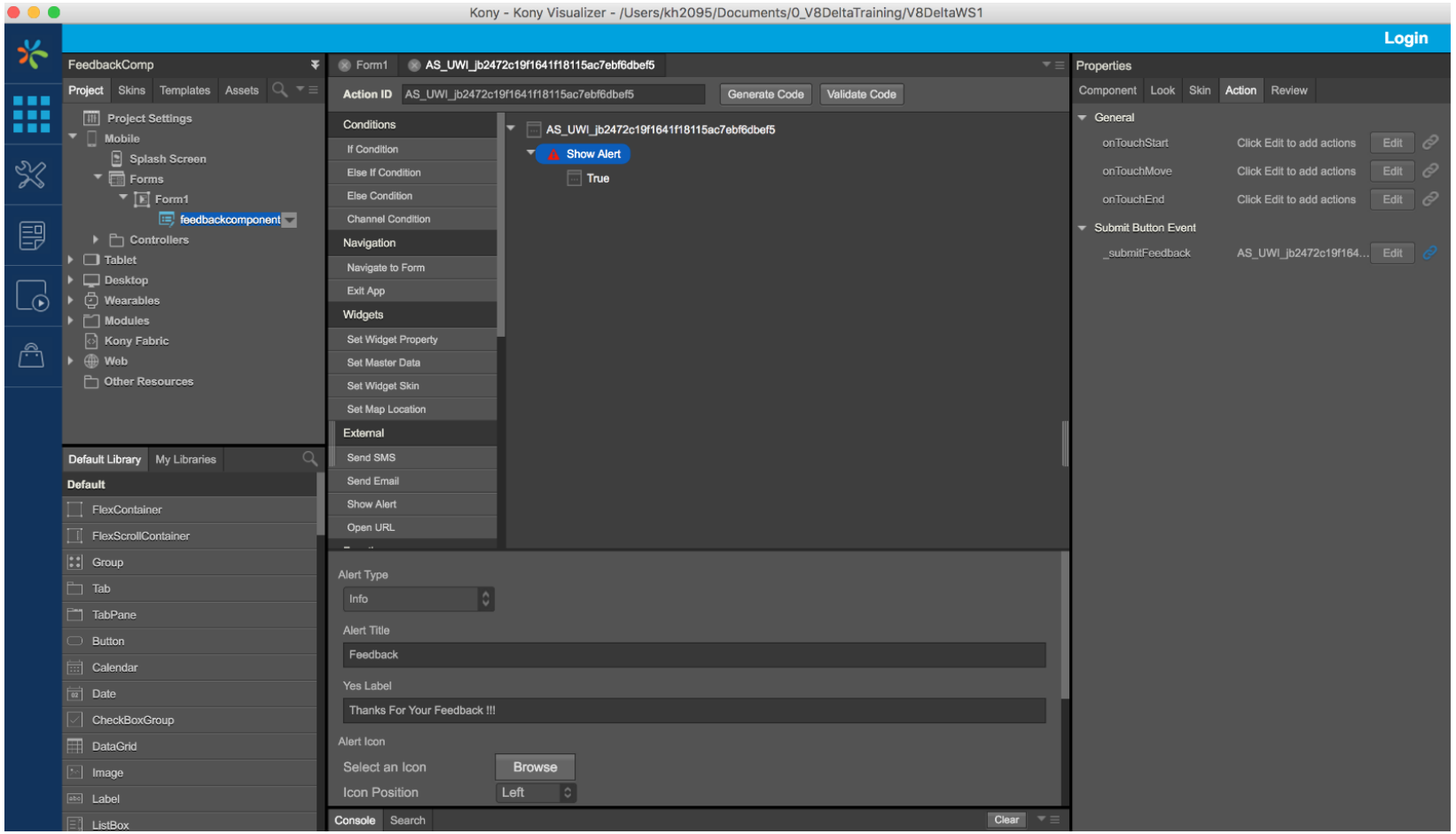




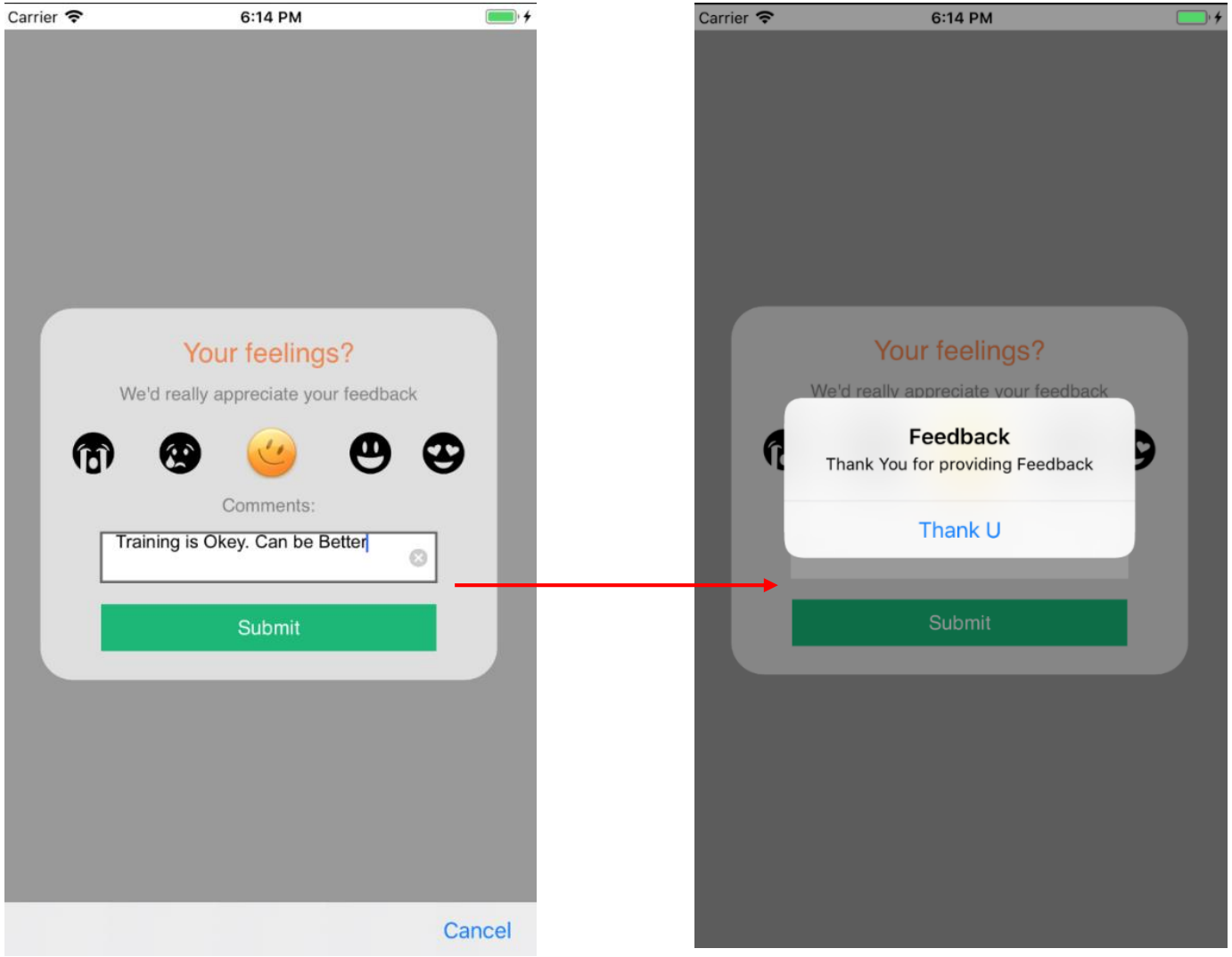


Consuming Feedback Component

- Adding Consumer Implementation
- Add an alert message to _submitFeedback Event
- Run the App and see the difference



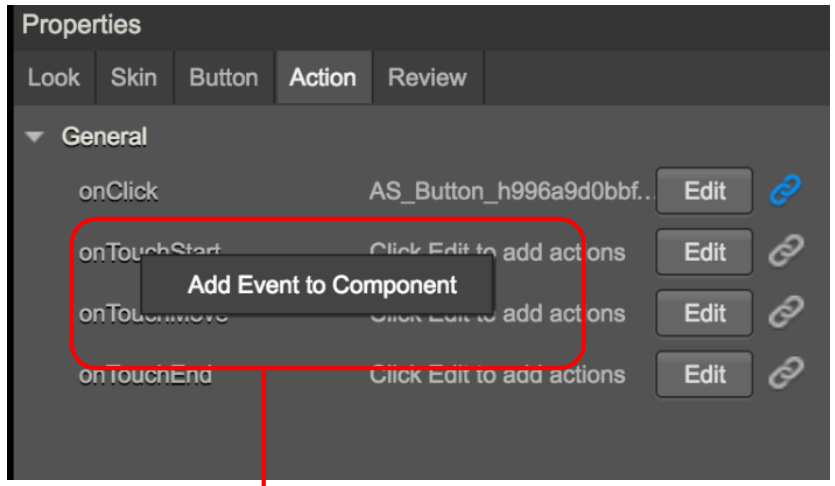
Consuming Feedback Component - Run the App

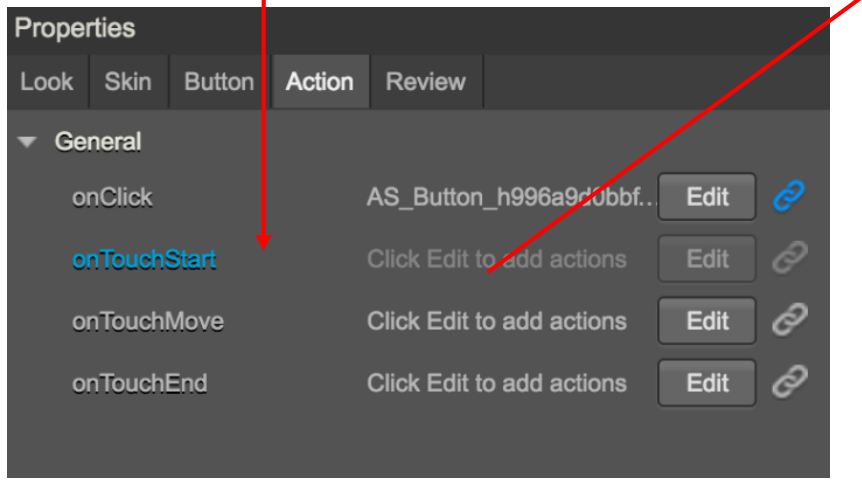


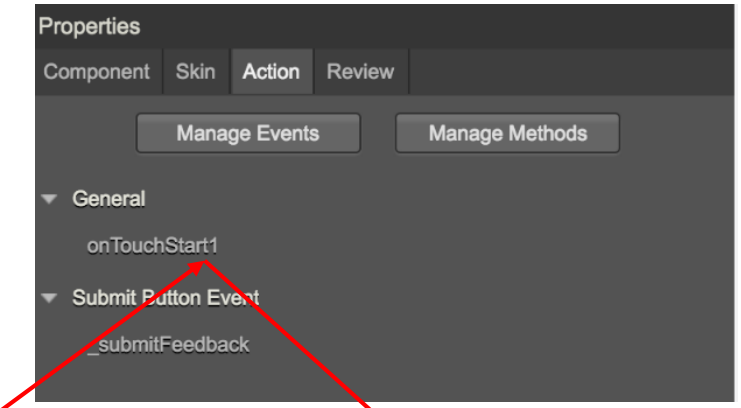
Additional Features in Component Creation

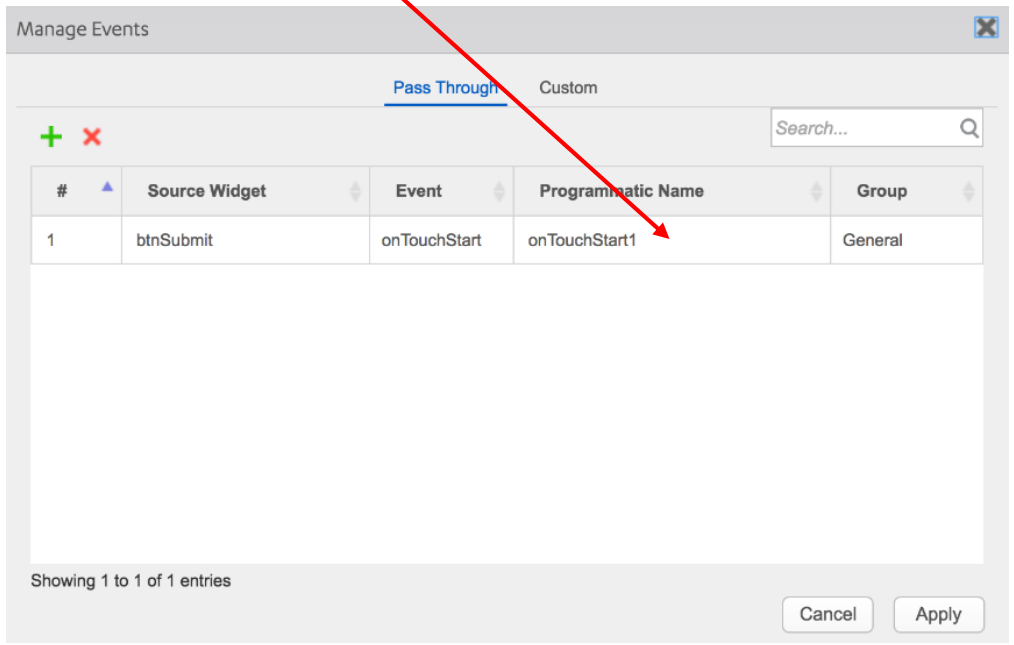
Manage Events - An Alternate Way

- Pass through events - an another way









Custom Events and API's

- The custom events can be implemented at consumer side. The custom Events will not have any build-in behavior
- The consumer can implement the customer function in controller or add action sequence using the action editor
- Custom API's are similar in terms of usage however, the creator has to share the API signature and input params with the consumer. In order to user the custom methods, the user have additional control on the form
- The consumer can just call the custom API and make use of the default behavior
- The consumer can call custom API using the following syntax:
  - *this.view.<componentname>.api();*

Component Creation - Exposing Widgets

- The creator of a component can expose any widget that is added to the Component, including the container widgets
- Right click on any widget in the Canvas and select an option to “Expose Widgets”
- In other words, the Expose Widget property is added to the Context Menu when the User right clicks on a widget within a Component
- When the user selects the expose widget property in the Context Menu, the property of Export Widgets in the Property panel will be set to “Yes”
- When a Container widget is exposed by the Component Creator, all its child widgets will be exposed too
- E.g., this should be an additional level of choice - let’s say, I am using a flex container only as a design element, and I want it to be exposed, but I want it lock for all other purposes other than skin and some flex properties.

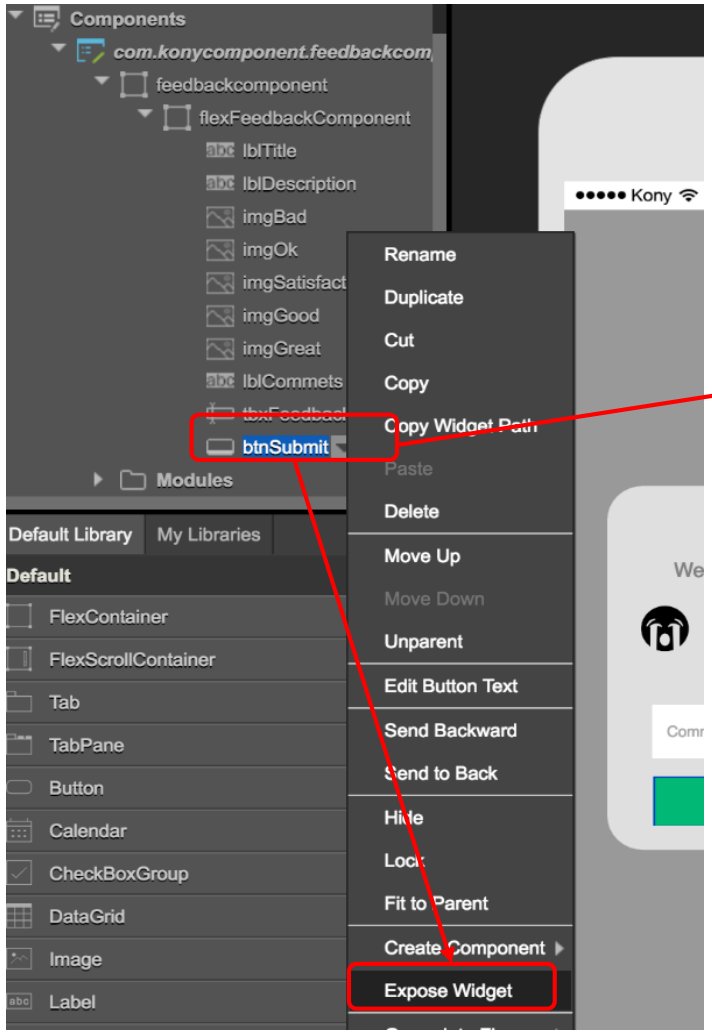
Component Creation - Exposing Widgets

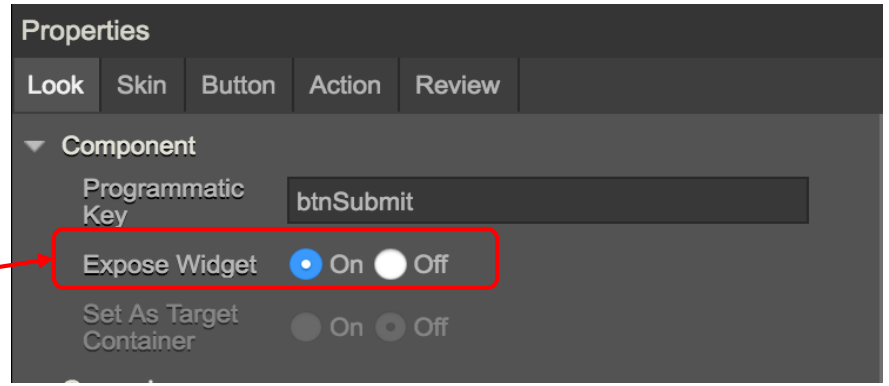
- The child widgets can be set to not be exposed at their individual level
- The Components Creator will still have the option to set “Expose Widget” property for the child widgets of an exposed Container widget to “No”

Container Widget	Child Widget
Expose Widget = Yes	Expose Widget = Yes
Expose Widget = Yes	Expose Widget = No
Expose Widget = No	Expose Widget = Yes
Expose Widget = No	Expose Widget = No

- For the Consumer of the Components, if the child widget of a Container is locked, the consumer will still be able to modify the properties of the Container widget but not the child widgets
- A Component may have some of its child widgets in a container exposed while some of them may not be exposed

Component Creation - Exposing Widgets

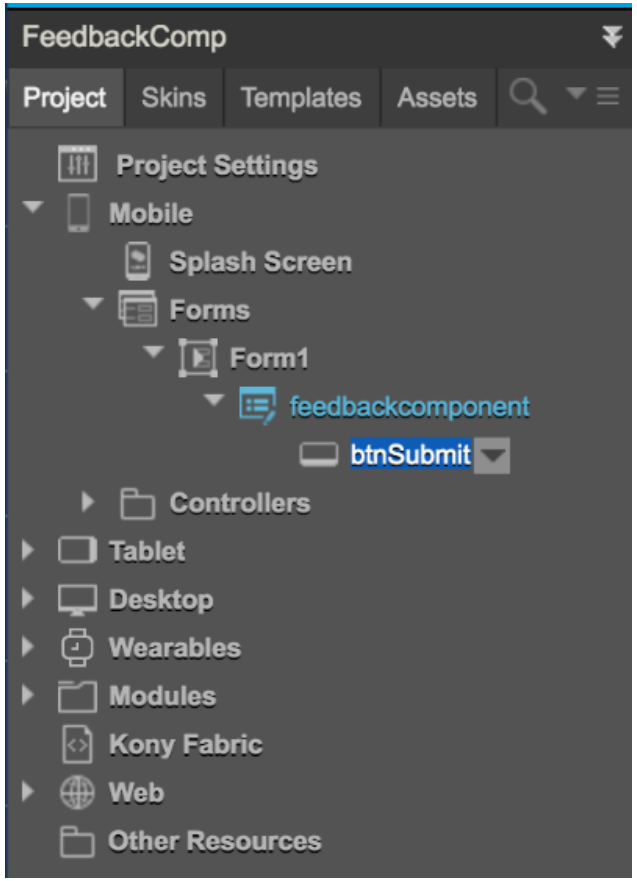


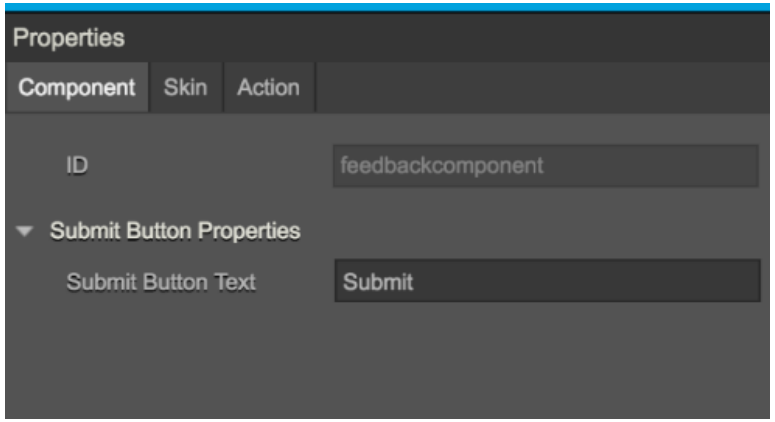


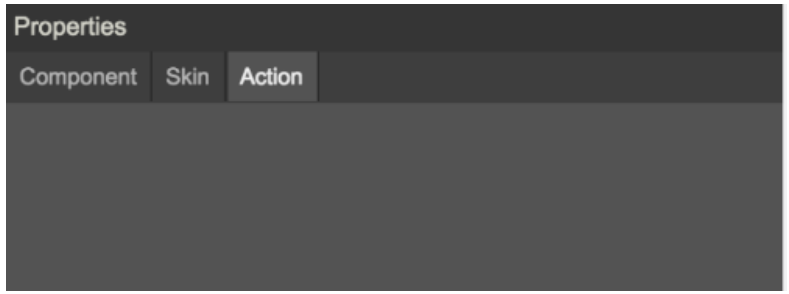
Expose Widgets - Widget Properties & Actions

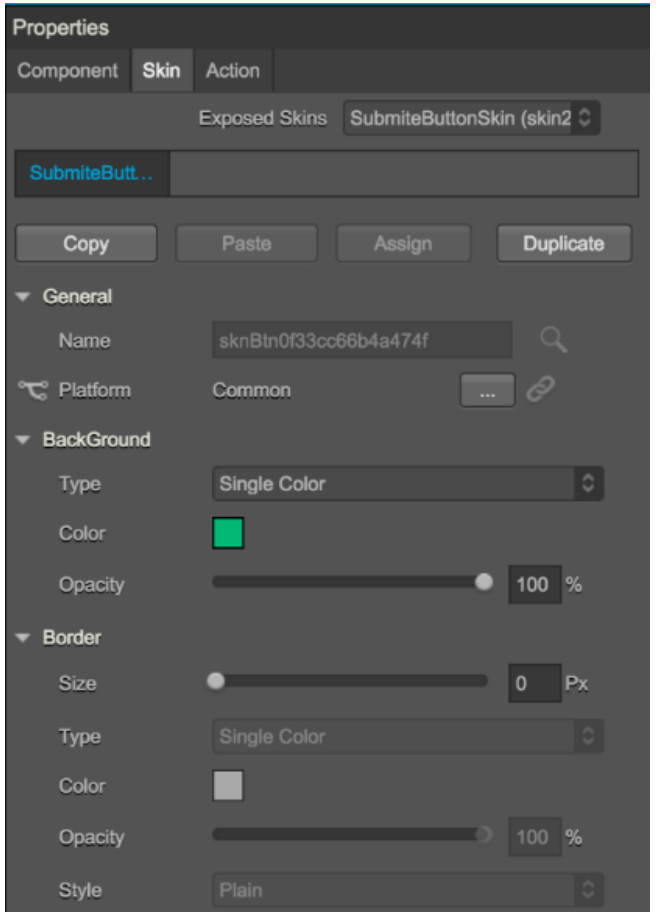
- For an exposed widget, the Component Creator should be able to expose properties, methods & events as pass-through similar to any other widget within the component
- For a Consumer, in a Component, the properties of an exposed widget will be shown under the Component Property Tab along with other pass-through properties as well as under the exposed widget properties when the exposed widget is selected
- At the exposed widget level, only the exposed widget specific pass-through properties are shown
- Similarly, at the Component level all the pass-through events are shown and at the exposed widget level only the exposed widget specific pass-through actions are shown

Component Consumer - Exposing Widgets







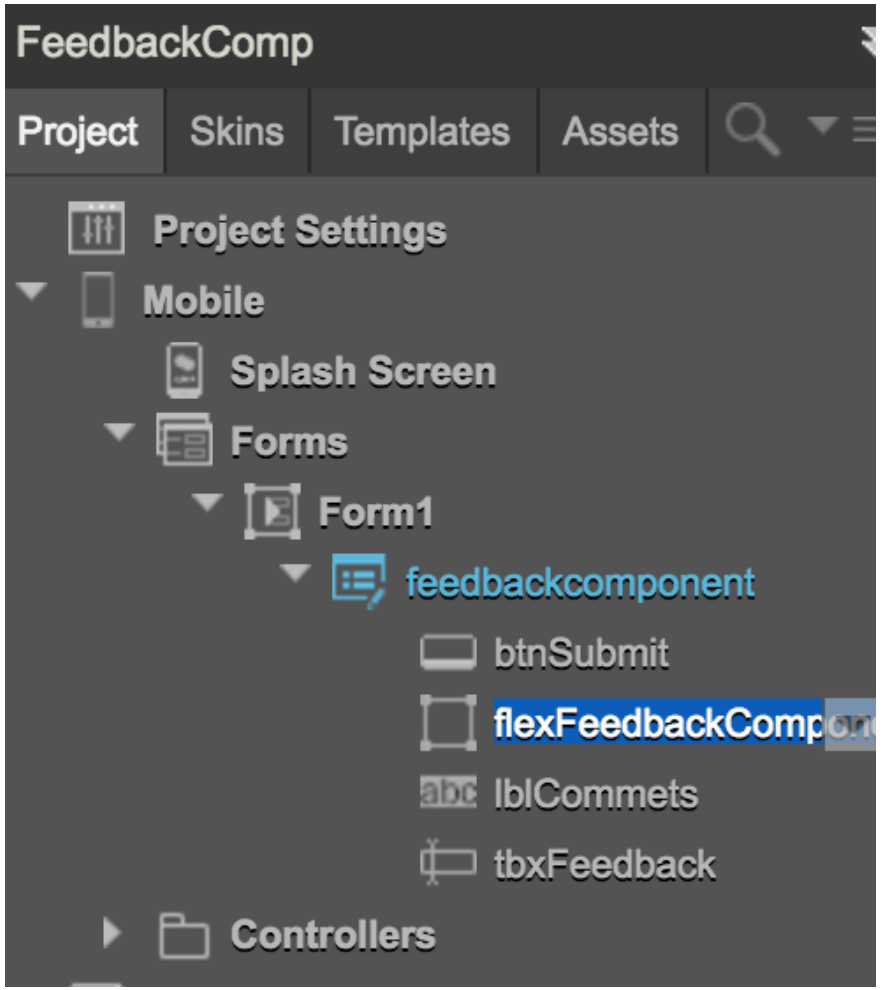


The btnSubmit buttons exposed properties, skin and exposed events

Component Consumer - Exposing Widgets

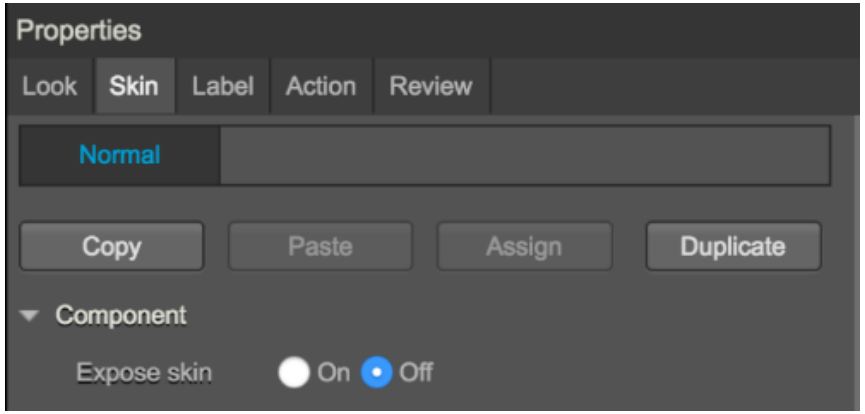
- In the consumer of the component, only the exposed widgets are visible under the Project tree
- Consumer of the component will be able to modify the exposed properties, the widget's skin and all the exposed events
- On the Visualizer Canvas, the hierarchy will not be maintained when the parent widget and it's child widget is exposed

  E.g., shown in the screenshot



Expose Widgets -Widget Skins

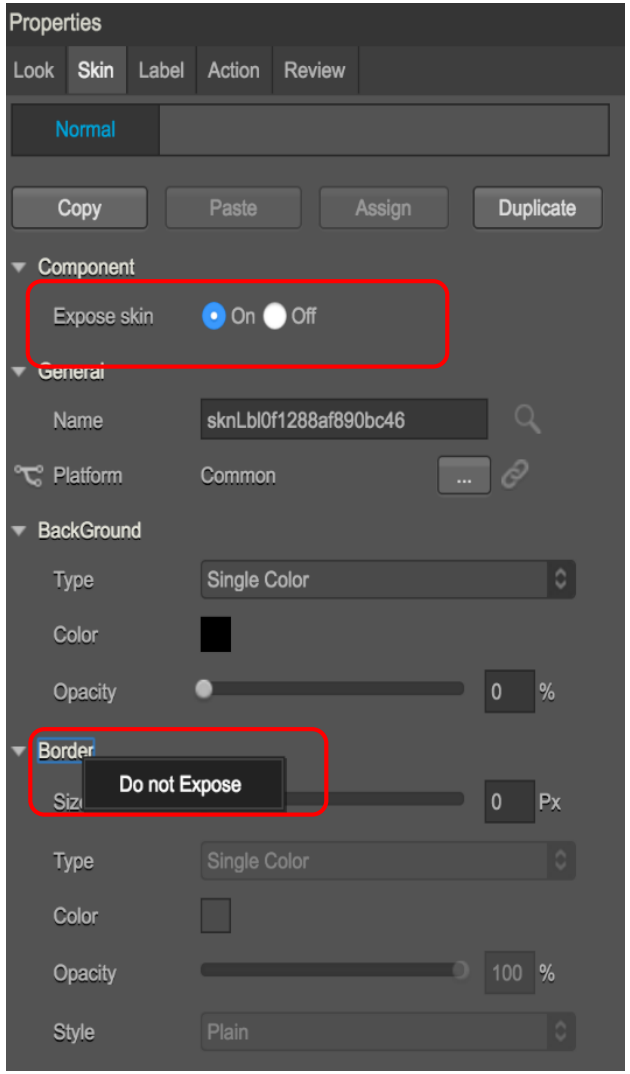
- For an exposed widget, the Component Creator can expose Skins. The skins can be exposed using the "Expose Skins" property under the Skins Tab in the Properties Palette.
- The value of the Expose Skins property will be Boolean (Yes/No) using a radio button. The default value of the Expose Skins property will be "No".



- The Component creator can choose "Yes" property if he wishes to share the skin with the Consumer.

Expose Widgets - Widget Skins

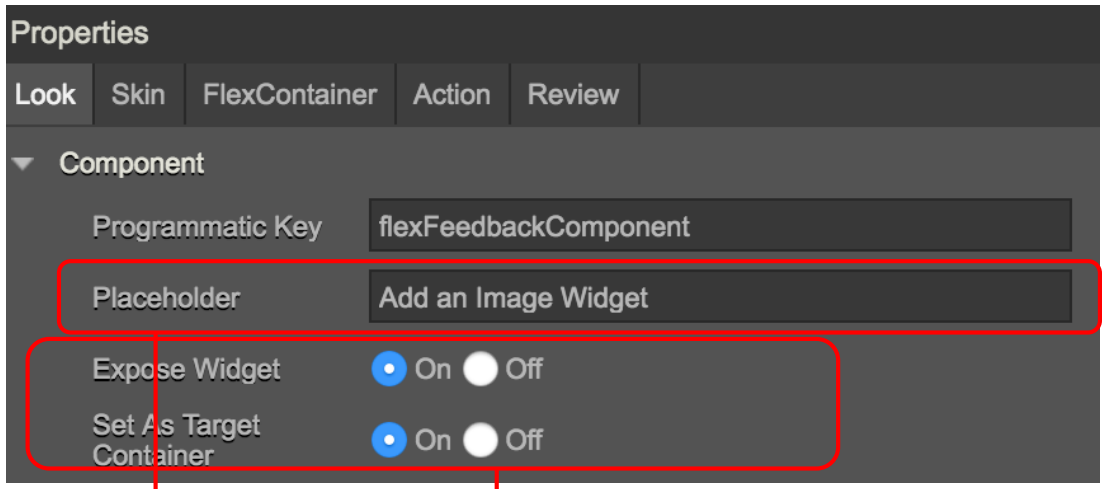
- The Component Creator will also have an option to not expose certain sub properties of the skins. The creator should be able to right click on the skin and mark as “Do not Expose”.
- For the Consumer of the Component, only the skins & skin properties which have been exposed by the Component Creator for the exposed widget are visible. All other skin properties & skins if not exposed are not available for modification.

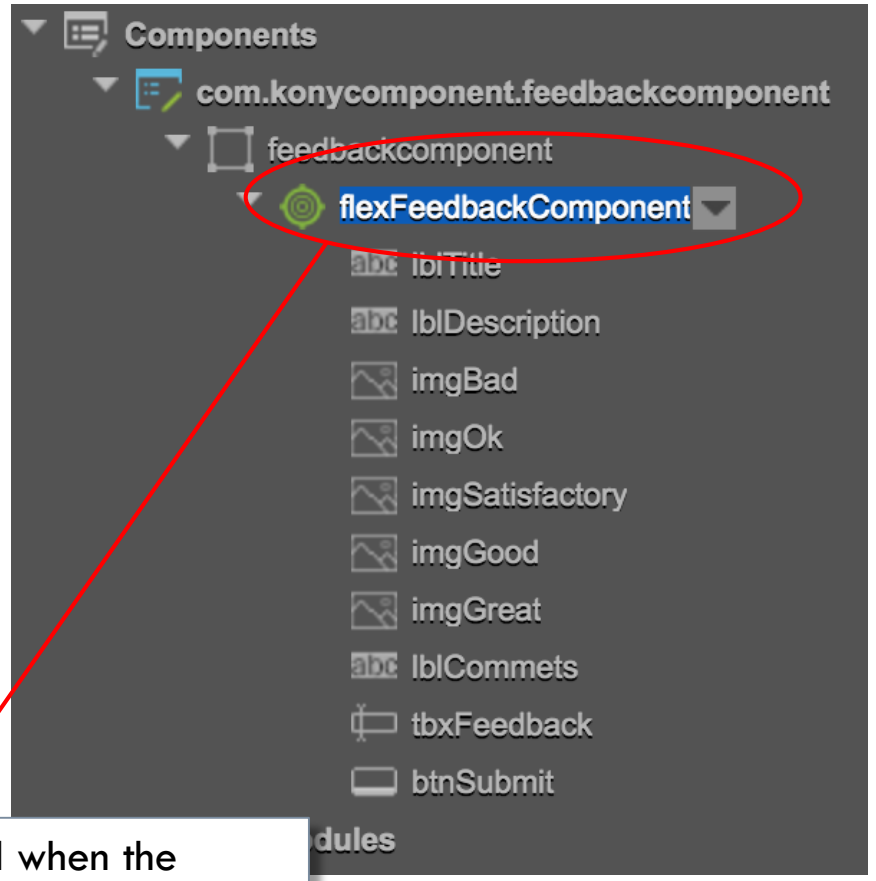


Expose Widgets - Target Container

- A Target Container is a **Container widget** which can contain other Child Widgets
- When a Component Creator marks a Container widget as “Expose Widget” = Yes, the component Creator can also mark that Container widget to be a target Container to accept more widgets within it
- This should be exposed through a property in the Property Palette for the exposed widget using “Set As Target Container”
- The value of the “Set As Target Container” property will be Boolean (Yes/No) using a radio button.property. The default value of this property will be “No”
- The “Set As Target Container” property will only be visible (available) when a Container widget is marked as “Expose Widget” = Yes. This property should appear just below the “Expose Widget” property in Container Widgets
- In other words, a container widget cannot accept child widgets at Consumer side unless it is marked as an exposed widget and marked to accept child widgets within the component.
- If the “Accept Child Widgets” property is marked as “Yes”, the Container widget is open to accept widgets to be placed inside the Container widget by the Consumer of the Component.

Expose Widgets - Target Container





A Container Widget is Exposed and Set As Target Container

Placeholder within the Container to provide further information and direction to the Consumer

The green icon is created when the Container Widget is Exposed and Set As Target Container

Expose Widgets -Target Container Placeholder

- All the Container widgets should be applicable to be exposed as Target Containers.
  - Flex Container
  - Flex Scroll Container
  - Tab / Tab Pane
- When a Container is exposed as Target Container, it should be able to accept any applicable Child widgets or a set of Child widgets, other containers and even Masters and other Components within the target container
- When a Container is marked as Target Container, the Component Creator can add a Placeholder within the Container to provide further information and direction to the Consumer of the Component to where the child widgets can be added or what type of child widgets can be added

Target Container - Consumer View

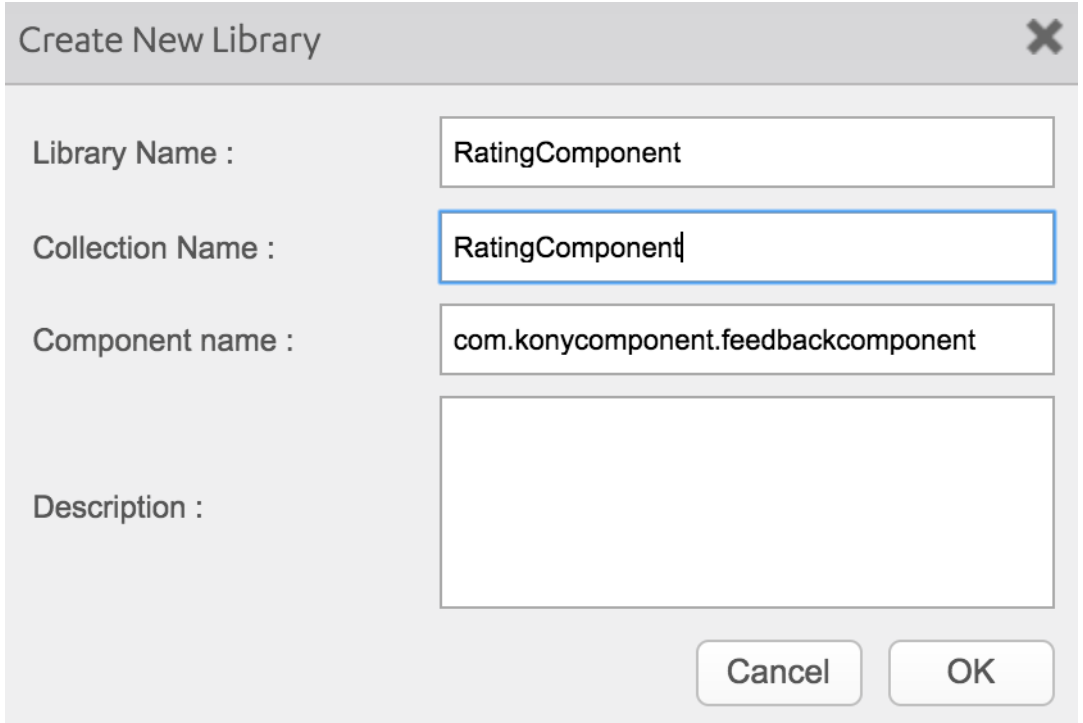
- When a Component Consumer views, a Component marked as a Target container, they see the following:
  - The Target Container Widget is exposed within the Component (available in the project tree and can be selected inside the Component)
  - The Target Container Widget has a placeholder providing guidelines to use the Container
  - The Consumer should be able to Drag & Drop other widgets within the Target Container Widget
  - When dragging widgets onto the Target Container, the target Container borders should get highlighted as they do today in terms of a flex container
- Additionally, once the Consumer drops a widget within a target container, the consumer has all the properties available for the dropped widget as he would normally have for that widget like the layout properties, PSP's, skins, Actions, etc. the only difference would be that the target container would be its Parent Container and all layouts would be w.r.t the Target Container

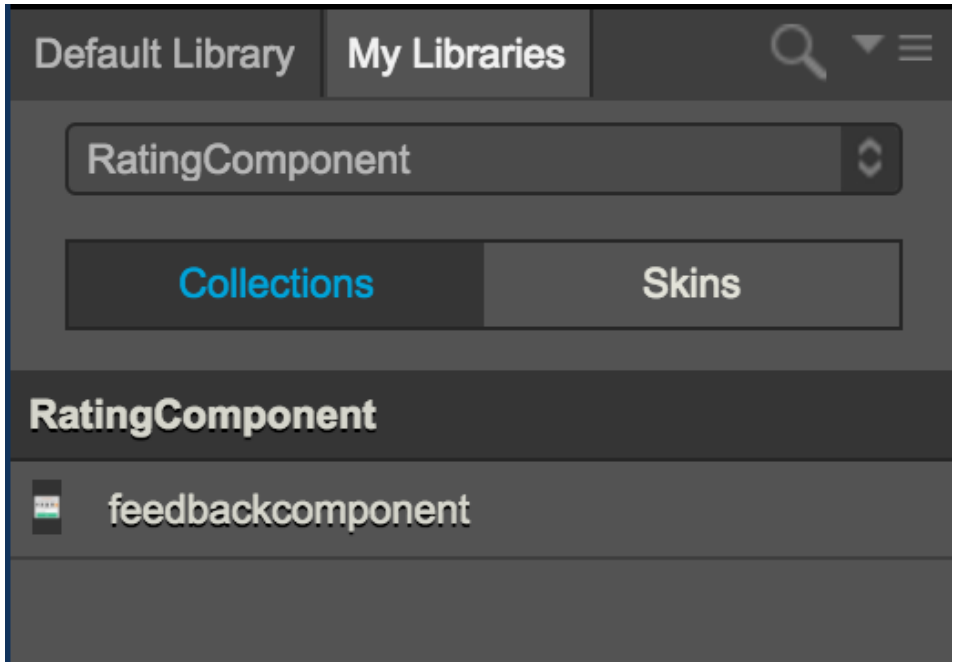
Target Container - Accept Masters

- A Component Consumer can add Masters into the Target Container
  - The Component Consumer can add a master into the Target container through
  - Dragging a Master into a Target Container
  - Using the “Insert Into” option on the master when the target container is selected

Component Upload to Marketplace

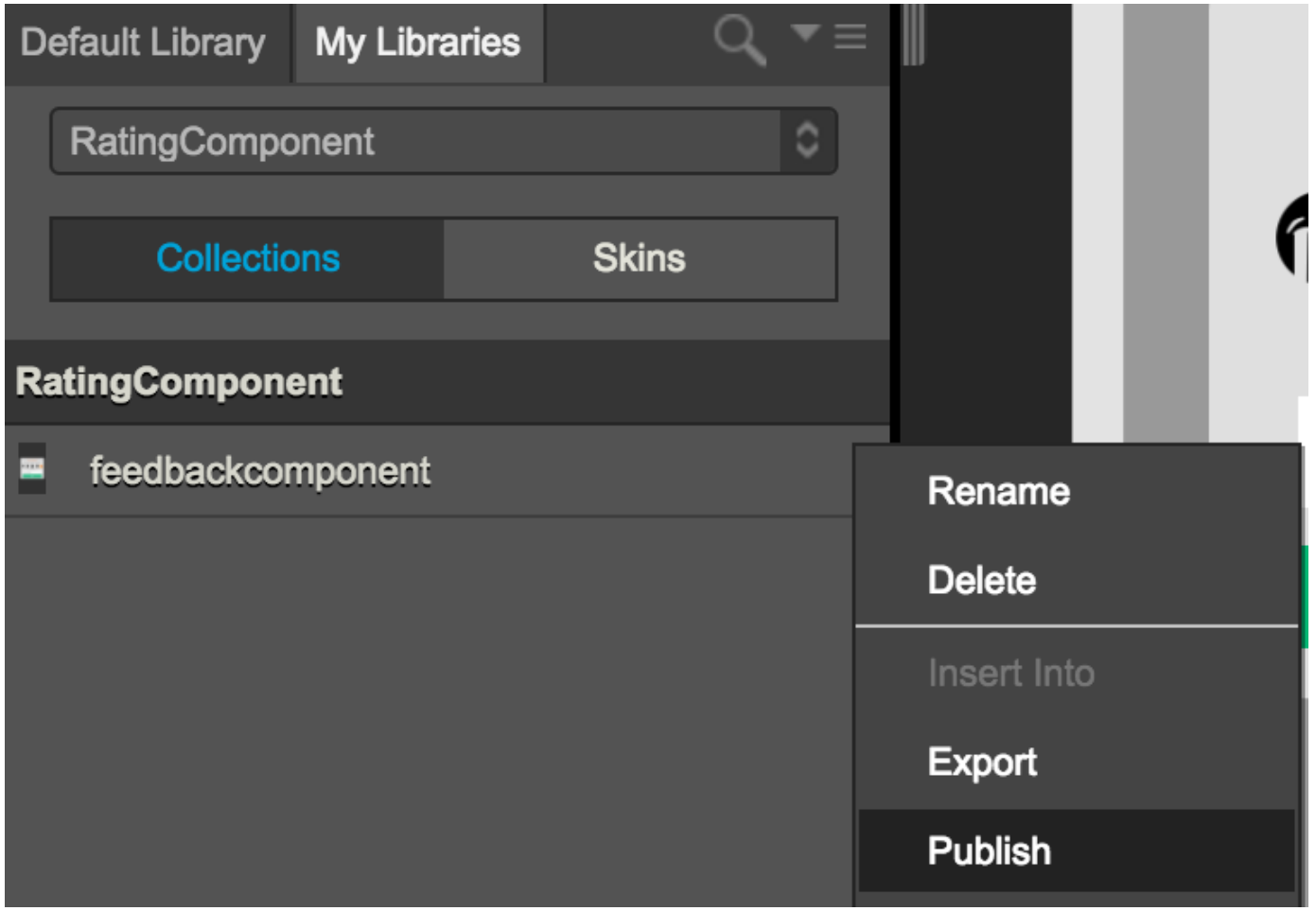
- To publish the component to the marketplace follow the following steps:
  - Create a component
  - Create a new Library/ use exiting Library and add it under a collection





Component Upload to Marketplace

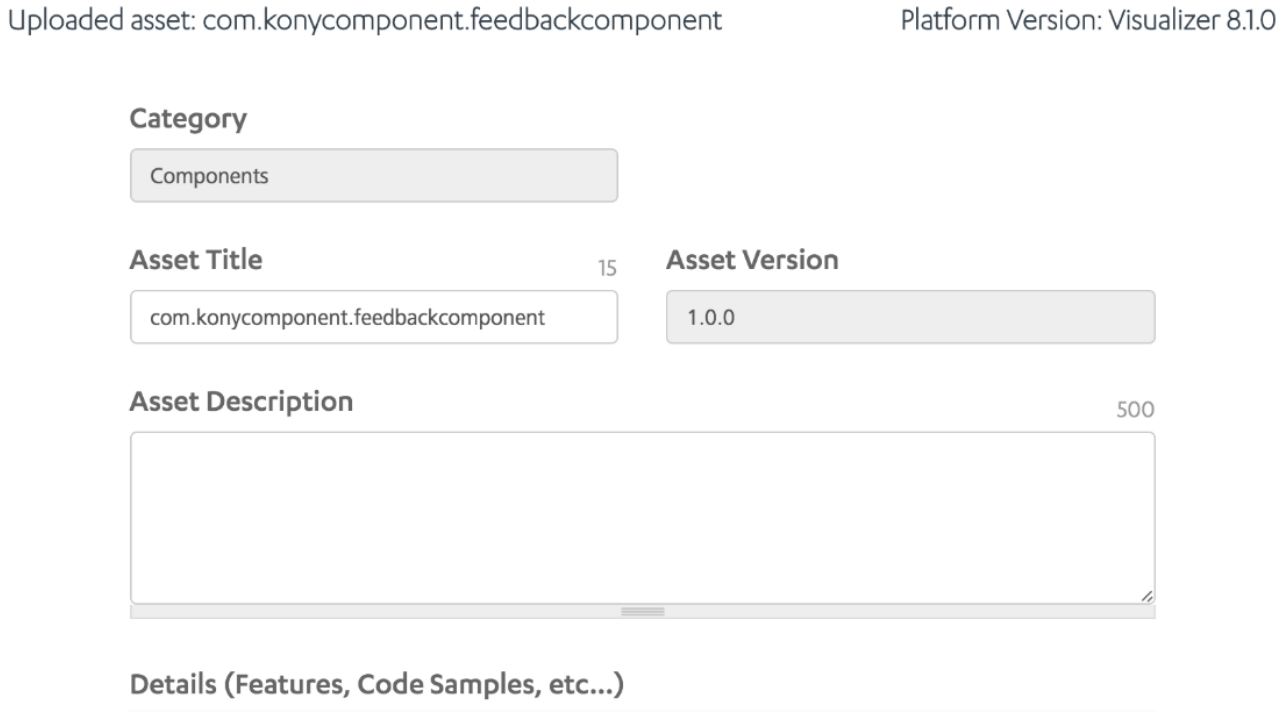
- Right click on the collection item and choose “Publish” option
- This will take you to cloud login page and login with your cloud credentials
- Post successful login, it will open the form for you to enter few details before you submit the component



Component Upload to Marketplace

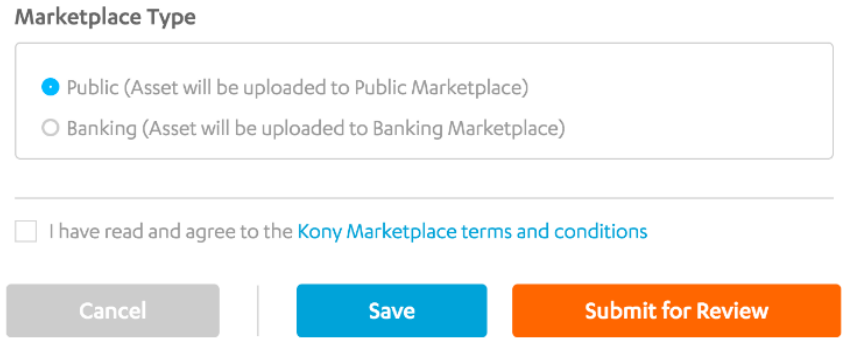
You need fill few details, such as:

- Assent Description
- Details of the feature/code sample etc.
- Assent Display View
- Developers Guide Documentation
- Kony Development Link
- External Links any
- Assent Requirement and additional Info
- Marketplace Type
  - Public
  - Banking



Component Upload to Marketplace

- After filling all the details, click on the terms and condition and then submit it for review
- Kony has set of protocol to be followed and when your component is in compliance with the protocol, your component will be approved and will be added to marketplace
- Creator will be notified by the Kony team on the whole process



Private Marketplace

- A lot of the enterprise customers have their own library of reusable components and other artifacts which are their intellectual property
- These enterprises might not want to share these items on Kony marketplace with the public. For such enterprise customers we need to expose a private section of marketplace which all the users within that organization (identified by Kony Accounts credential) will have access
- If the Component Creator's Kony cloud ID has access to the Private Marketplace, the user can publish the Component to the Private Marketplace. The process of publishing the Component is as same as Public Marketplace
- While uploading the Component, the user will be given an option to choose if he/she wants to upload the component to Private or Public Marketplace
- The uploaded Component will be restricted to their organization if they chooses to publish to the Private Marketplace

References

References

- Components Overview
  - http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/C_ComponentsOverview.htm
- Creating a Component
  - http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#C_CreatingComponent.htm
- Using Components
  - http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Content/C_UsingComponents.htm
- Private Marketplace
  - http://docs.kony.com/konylibrary/visualizer/visualizer_user_guide/Default.htm#C_UsingComponents.htm%23Private

Thank You