iOS Dev Accelerator Week3 Day2

- Custom App URL Schemes
- OAuth
- NSUserDefaults
- UISearchBar
- Mobile Tuesday : Android Views and Grid View

Custom URL Scheme

Registering a custom URL scheme

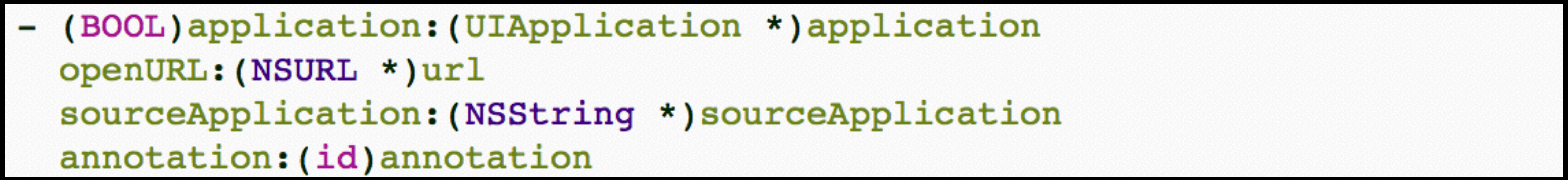
- Go to your info.plist and choose Add Row
- Select URL types from the drop down (its an array with one dictionary)
- In the item 0 dictionary, add an entry for the key ‘URL Identifier’
- This entry will be the name of the custom url scheme you are defining. It is recommended to ensure uniqueness that you use reverse DNS ‘com.yourCompany.yourApp’
- Add another entry from the drop down called ‘URL Schemes’ which is an array
- For Item 0, give it the string you want to be your URL Scheme. For example, if you type MyApp. The custom URL for your app will be MyApp://

Try it out!

- Launch your app after editing your plist with your URL scheme.
- Now launch Safari from your simulator, and enter in your app’s URL into the URL bar.
- Your app should now open.

Parsing information that is passed with URL’s

- Often times, and especially with OAuth, your app’s url will be called with extra parameters. Typically this is a token or flag.
- There is a method you can implement in your app delegate to intercept these URL calls:



```
- (BOOL)application:(UIApplication *)application
  openURL:(NSURL *)url
  sourceApplication:(NSString *)sourceApplication
  annotation:(id)annotation
```

- We will pass the URL they passed back to us to our network controller for parsing

OAuth

OAuth

- OAuth is is an authentication protocol that allows two apps to interact and share resources.
- There are 3 actors in the OAuth workflow:
  - The service provider: Ex: GitHub
  - The consumer: Ex: Our app
  - The user
- The main benefit is that the user never shares his account and password with the consumer app.

OAUTH Workflow

- Step 1: The user shows intent by attempting an action from the consumer app to the service provider (aka GitHub)
- Step 2: Our app (The consumer) redirects the user to the service provider for authentication
- Step 3:The user gives permission to the service provider for all the actions the consumer should be able to do on his behalf (ex: posting to his timeline, accessing his twitter photos, etc)
- Step 4:The service provider returns the user to the consumer app, with a request token
- Step 5:The consumer now sends the request token, together with its secret key to the service provider in exchange for an authentication token
- Step 6: The user performs actions and passes the authentication token with each call to prove who he is

Callback URL

- The callback URL is just the callback entry point of your app.
- The service provider performs an HTTP redirect to get the user back to the consumer app.
- In addition to the URL of your app, the callback url will have the authorization code appended to it. It is up to your app to parse this out and use it in completing the OAuth workflow.
- All apps can be launched from either another app or from the browser itself.

Demo

NSUserDefaults

- “NSUserDefaults allows an app to customize its behavior based on user preferences”
- Think of it as an automatically persisting plist that is easily modified in code.
- Use the standardUserDefaults class method to return the shared defaults object.
- Setting values inside of it is as easy as these methods:
    - setBool:ForKey:
    - setObject:ForKey:
    - setInteger:ForKey:
- Call synchronize() after saving key in defaults.

UISearchBar

UISearchBar

- The UISearchBar class is a text field based control.
- The search bar provides a text field for text input, a search button, a bookmark button, and a cancel button.
- It relies on it’s delegate to perform searches when one of its buttons is pressed.
- You can also embed a search bar into a tableview by dragging it into the tableview on storyboard.

Demo

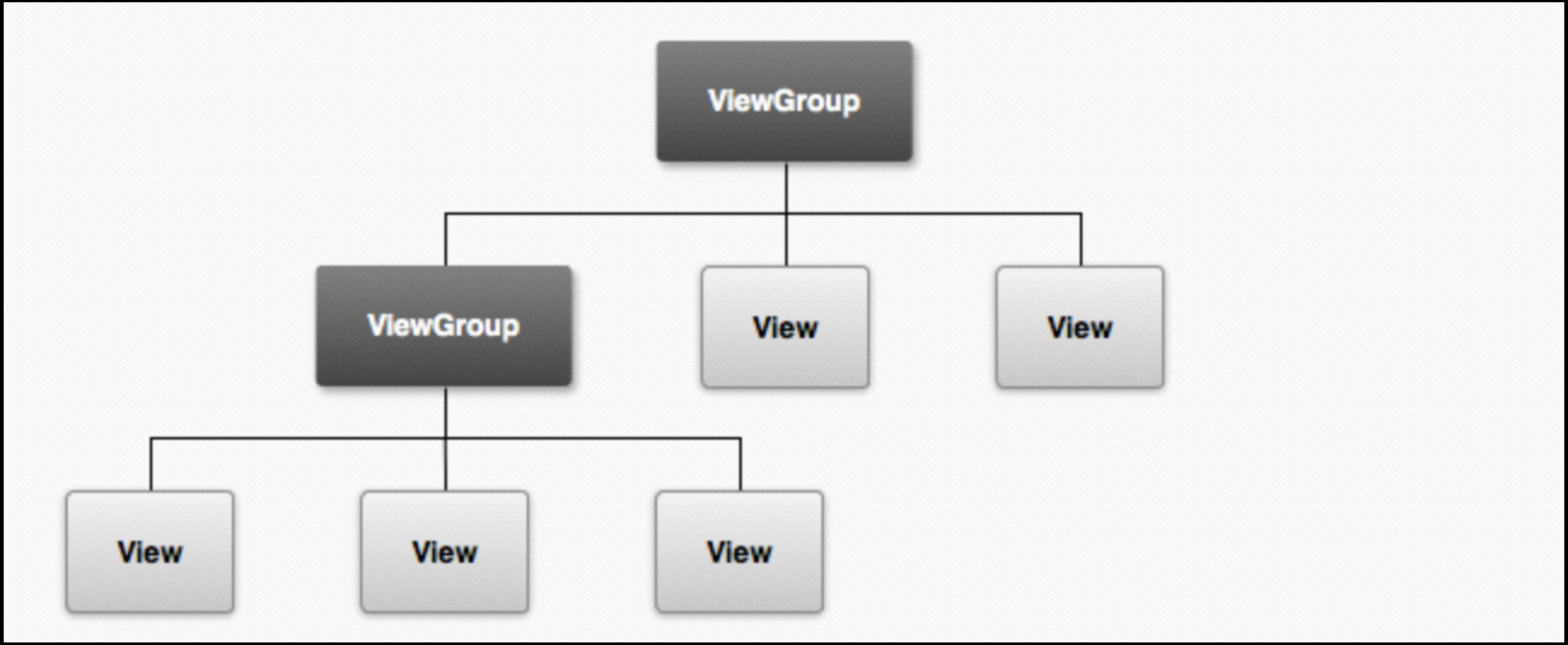
Try Android Tuesday

Android Interface and CollectionView

Android views

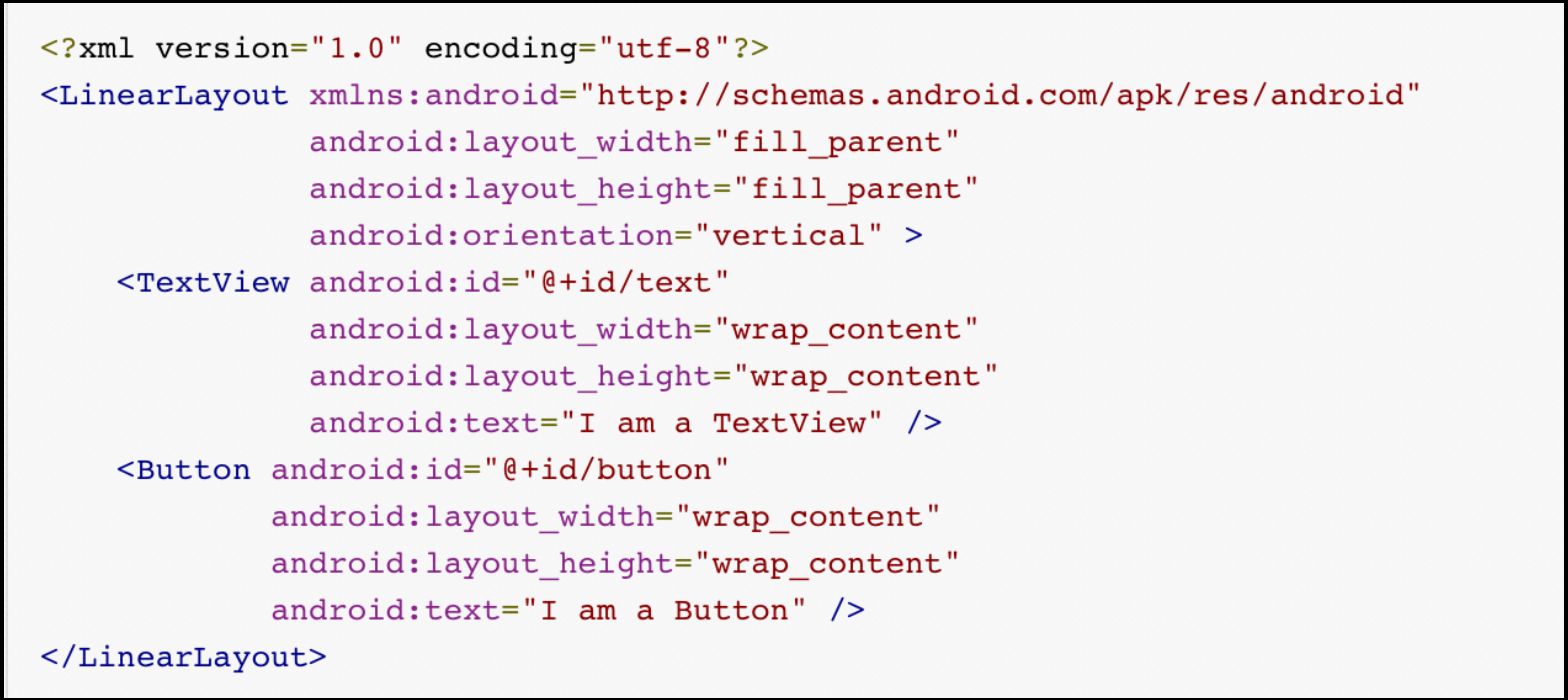
- All user interface elements in android are built with View or ViewGroup Objects.
- A View object is an object that can draw something on screen and that can be interacted with.
- A ViewGroup is an object that holds other Views and ViewGroups and defines the layout of the interface.

Android views



Android Interface Layout

- Like iOS, you can create all your views programmatically in Android, but it recommended against doing so.
- Instead, Google recommends you use XML to layout your interface:



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="I am a Button" />
</LinearLayout>
```

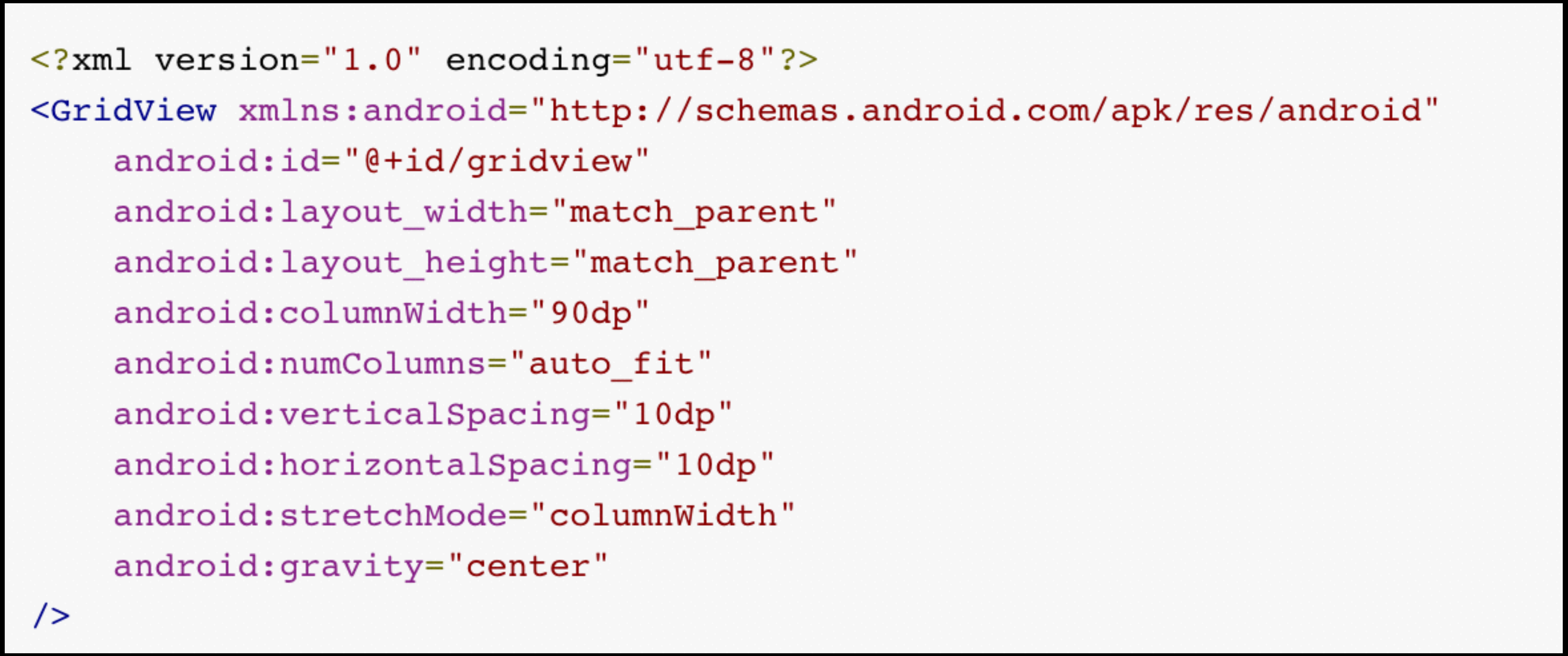
Loading the XML

- You will essentially define an XML for each Activity (aka view controller) in your android app.
- In that activity’s onCreate() - which is just like viewDidLoad - you call setContentView() on your Activity and pass in a reference to your XML:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Android Grid View

- Used for the same purpose as a collection view in iOS (grid based collection)
- Grid View is a ViewGroup.
- Uses a ListAdapter, just like the List View, to display items.
- Can be placed on screen via XML:



```xml
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

Android Grid View

- Used for the same purpose as a collection view in iOS (grid based collection)
- Grid View is a ViewGroup.
- Uses a ListAdapter, just like the List View, to display items.
- Can be placed on screen via XML:

```xml
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

Android Grid View

Below is an Activity that displays the Grid View from the previous XML example

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v, int position, long id)
            Toast.makeText(HelloGridView.this, "" + position, Toast.LENGTH_SHORT).sho
        }
    });
}
```