
AND230

# Google Maps

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.



Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

# Objectives

1. Explore Android map options
2. Configure an application to use Google Maps
3. Display a map in your application
4. Add markers to a Google Map
5. Adjust the map position and perspective

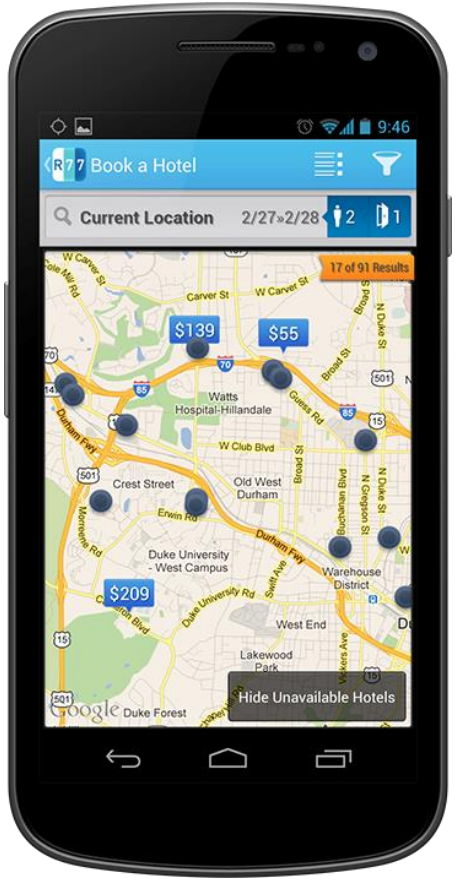
# Explore Android map options

# Tasks

1. Launch the default map application
2. Identify the components of Google Play Services
3. Install Google Play Services on an Android emulator

# Maps in Android

- Map support can enhance applications in a variety of ways
  - interactive points-of-interest
  - point-to-point navigation
  - tour guides
  - current location tracking
  - …



# Launching the "Maps" application

- Android supports launching installed map application through registered Uri – can use this to switch to app and display a specific location

```
private void ShowLocationOnMap()
{
    var seattle = Android.Net.Uri.Parse ("geo:47.6,-122.3");
    Intent intent = new Intent(Intent.ActionView);
    intent.SetData(seattle);
    if (intent.ResolveActivity(this.PackageManager) != null) {
        StartActivity(intent);
    }
}
```

# Launching the "Maps" application

- Android supports launching installed map application through registered Uri – can use this to switch to app and display a specific location

```
private void ShowLocationOnMap()
{
    var seattle = Android.Net.Uri.Parse ("geo:47.6,-122.3");
    Intent intent = new Intent(Intent.ActionView);
    intent.SetData(seattle);
    if (intent.ResolveActivity(this.PackageMa
        StartActivity(intent);
    }
}
```

"geo" MIME scheme, must pass latitude and longitude coordinates

# Launching the "Maps" application

- Android supports launching installed map application through registered Uri – can use this to switch to app and display a specific location

```
private void ShowLocationOnMap()
{
    var seattle = Android.Net.Uri.Parse ("
    Intent intent = new Intent(Intent.Action
    intent.SetData(seattle);
    if (intent.ResolveActivity(this.PackageManager) != null) {
        StartActivity(intent);
    }
```

Should check to see if URI is supported; e.g. if a mapping application is registered



There are other URIs available to launch specific Google Maps features such as turn-by-turn navigation, see: http://bit.ly/gmaps-intents

# Integrating maps

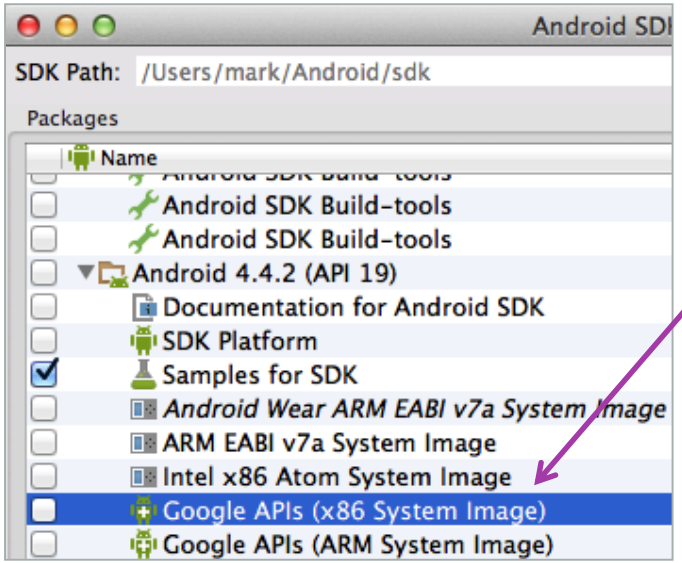
- Several integrated mapping options available for Android
  - Google Maps V2
  - Bing Maps (AJAX)
  - ESRI ArcGIS
  - OMSDroid (open source)
  - MapsForge (open source)
  - Nutiteq
  - Mapbox

# Getting started with Google Maps

- Google ships a variety of additional libraries (.APKs) for Android, one of which is **Google Play Services** which gives the device access to *Google-provided services*

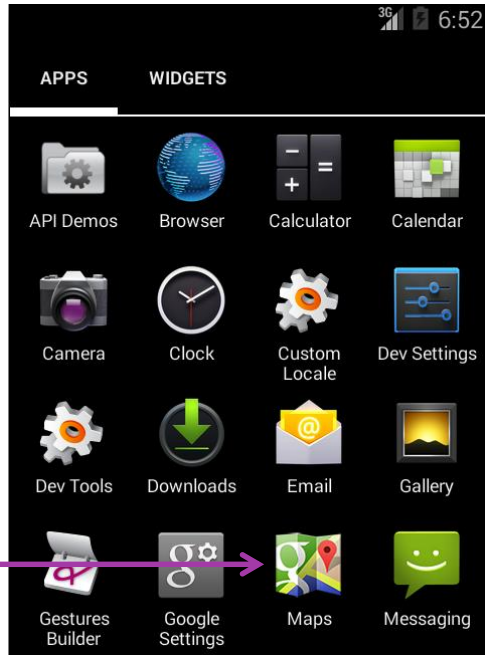
# Using Play Services on an Emulator

- Google includes an emulator image which has Play Services pre-installed



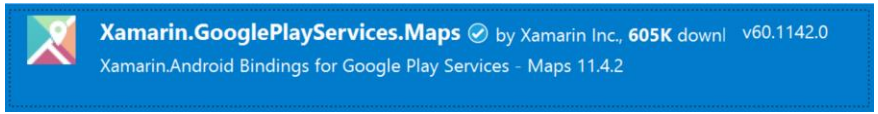

Make sure to use the x86 image!

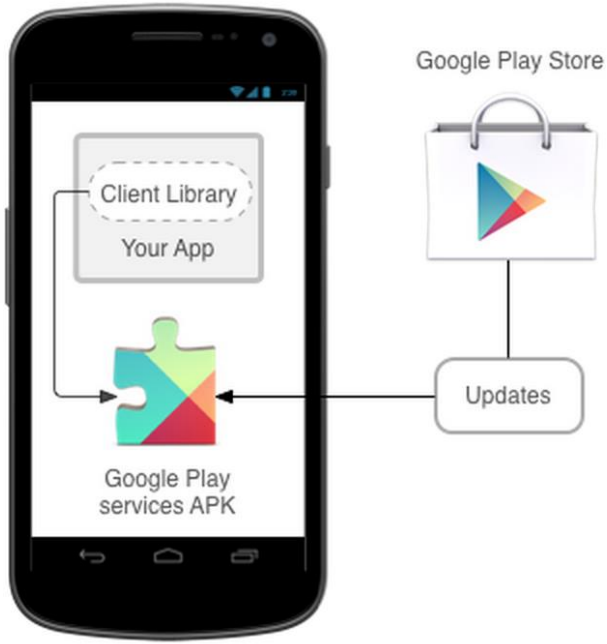
Standard apps, such as **Maps** are installed



# Google Play Services component

- Must add **Google Play Services - Maps** NuGet package to your Xamarin.Android application to access mapping features

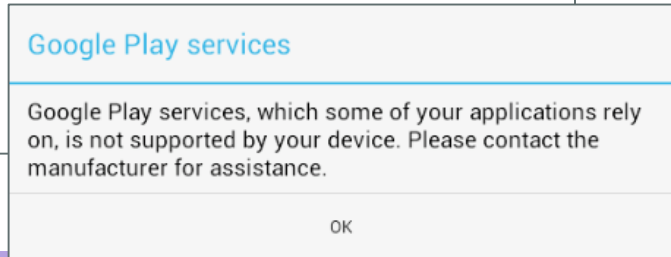


# Google Play services on devices

- Google Play Services is a licensed product that is not present on all physical devices – if it is not installed, then Google Maps will not work properly on the device

```
int error = GoogleApiAvailability.Instance
                    .IsGooglePlayServicesAvailable(this);
if (error != ConnectionResult.Success) {
   var dialog = GooglePlayServicesUtil.GetErrorDialog(
                     error, this, 0, null);
   dialog.DismissEvent += delegate { this.Finish(); };
   dialog.Show();
   SetContentView(new View(this));
}
```



# Flash Quiz

Xamarin University

# Flash Quiz

① How can you ensure there is a map application present?

a) `StartActivity`

b) `ActionView`

c) `ResolveActivity()`

d) `Android.Net.Uri.Parse()`

# Flash Quiz

① How can you ensure there is a map application present?

a) `StartActivity`

b) `ActionView`

c) **`ResolveActivity()`**

d) `Android.Net.Uri.Parse()`

# Flash Quiz

② What optional component must be included in the Android SDK Manager?

a) Google Play Services
b) Android Support Repository
c) Google Repository
d) Google Play Licensing Library

# Flash Quiz

② What optional component must be included in the Android SDK Manager?

a) Google Play Services
b) Android Support Repository
c) Google Repository
d) Google Play Licensing Library

# Flash Quiz

③ Which emulator images have Play Services pre-installed?

a) Android SDK Build-tools
b) Google APIs (x86 System Image)
c) Google APIs (ARM System Image)
d) SDK Platform

# Flash Quiz

③ Which emulator images have Play Services pre-installed?

a) Android SDK Build-tools

b) Google APIs (x86 System Image)

c) Google APIs (ARM System Image)

d) SDK Platform

# Summary

1. Launch the default map application
2. Identify the components of Google Play Services
3. Install Google Play Services on an Android emulator

# Configure an application to use Google Maps

Xamarin University

# Tasks

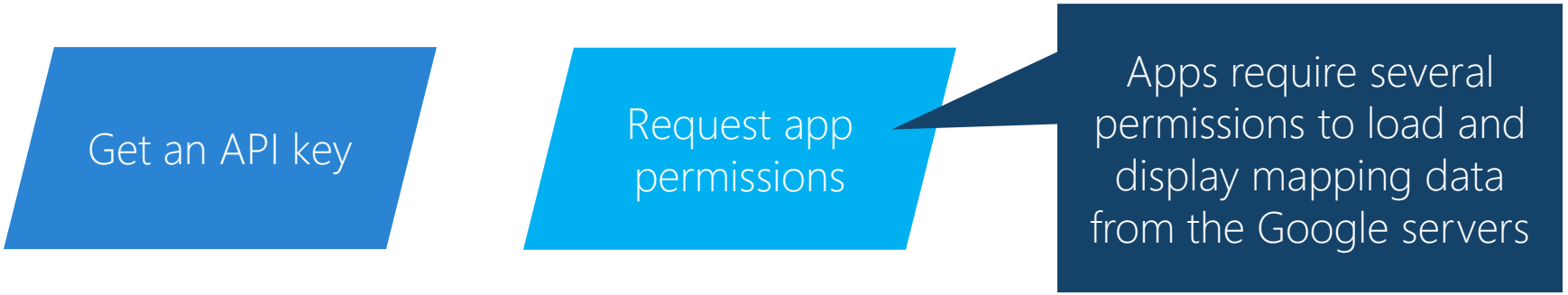
1. Create an API key
2. Register your application with Google Maps API
3. Set required permissions to use location in your application

# Adding support for Google Maps

- There are several required steps to utilize Google Maps features

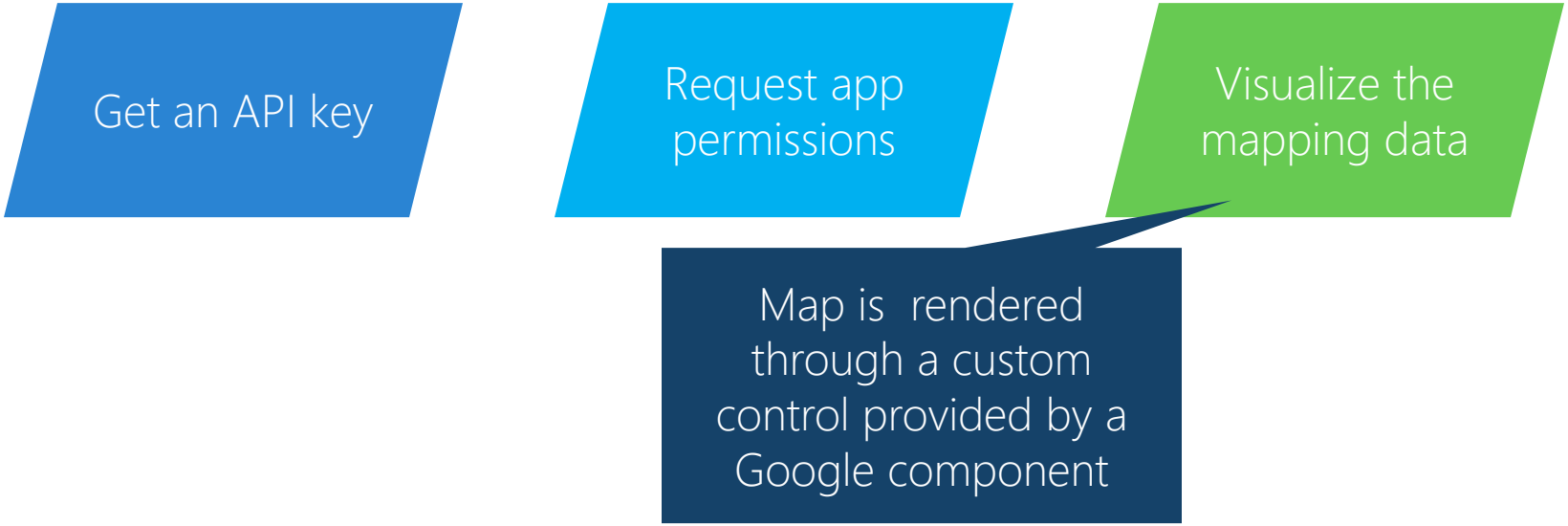
# Adding support for Google Maps

- There are several required steps to utilize Google Maps features



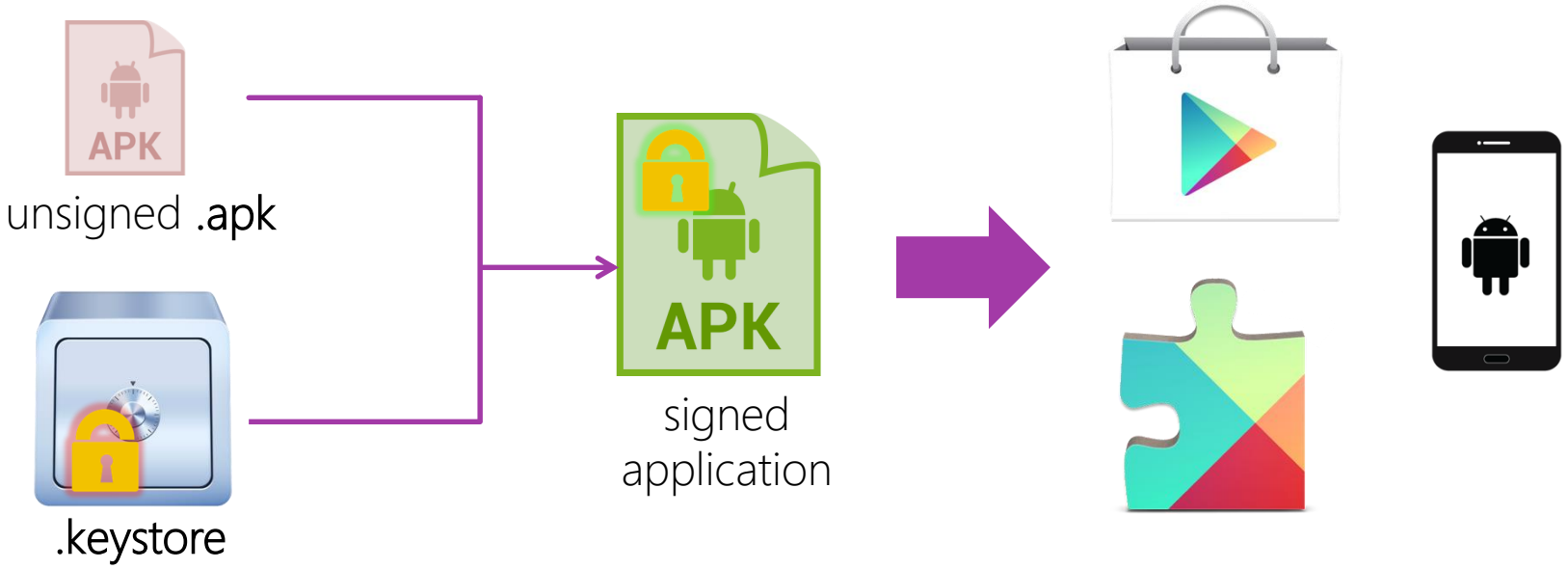
# Adding support for Google Maps

- There are several required steps to utilize Google Maps features



# What is the keystore?

- Keystores provides a unique identifier for the application / publisher; it is a signing certificate added to the app as part of the build process



# What is the keystore?

- Keystores provides a unique identifier for the application / publisher; it is a signing certificate added to the app as part of the build process

# What is the keystore?

- Keystores provides a unique identifier for the application / publisher; it is a signing certificate added to the app as part of the build process

# What is the keystore?

- Keystores provides a unique identifier for the application / publisher; it is a signing certificate added to the app as part of the build process

# Where is the keystore?

- Keystore default location is operating-system specific, and can be configured on a per-project basis; the **debug keystore** is normally at:

Windows

```
%LocalAppData%\Xamarin\Mono for Android\debug.keystore
```

macOS

```
~/.local/share/Xamarin/Mono for Android/debug.keystore
```

# Get an API key: retrieve SHA1 fingerprint

- Google Play Services uses the SHA1 fingerprint of the signing keystore to identify the application; this can be found using the Java **keytool** utility

```
keytool -list -v -keystore [filename]
        -storepass [password]
        -keypass [password]
```

Must include **-v** switch to get entire fingerprint signature



You will need the *debug* keystore fingerprint during development and the *release* keystore fingerprint to submit to the Google Play Store – both must be registered!

# Example: getting the SHA1 fingerprint

```
> keytool -list -v -keystore ~/.local/share/Xamarin/Mono\ for\
Android/debug.keystore -storepass android -keypass android

Certificate fingerprints:
     MD5:  56:37:EB:CB:58:41:2F:0C:84:89:2B:E0:8E:51:D3:95
     SHA1: 1F:6B:A7:6D:E6:8E:87:69:1A:62:44:14:59:4B:9F:4F:3B:EE:35:E9
     Signature algorithm name: SHA1withRSA
     Version: 3
```



Must deploy an app to create the debug keystore in the default location (shown here for macOS), or can create a new keystore (typical for release builds)

# Example: getting the SHA1 fingerprint

```
> keytool -list -v -keystore ~/.local/share/Xamarin/Mono\ for\
Android/debug.keystore -storepass android -keypass android

Certificate fingerprints:
     MD5:  56:37:EB:CB:58:41:2F:0C:84:89:2B:E0:8E:51:D3:95
     SHA1: 1F:6B:A7:6D:E6:8E:87:69:1A:62:44:14:59:4B:9F:4F:3B:EE:35:E9
     Signature algorithm name: SHA1withRSA
     Version: 3
```

Want the entire SHA1 signature – this is how Google will validate the calling application

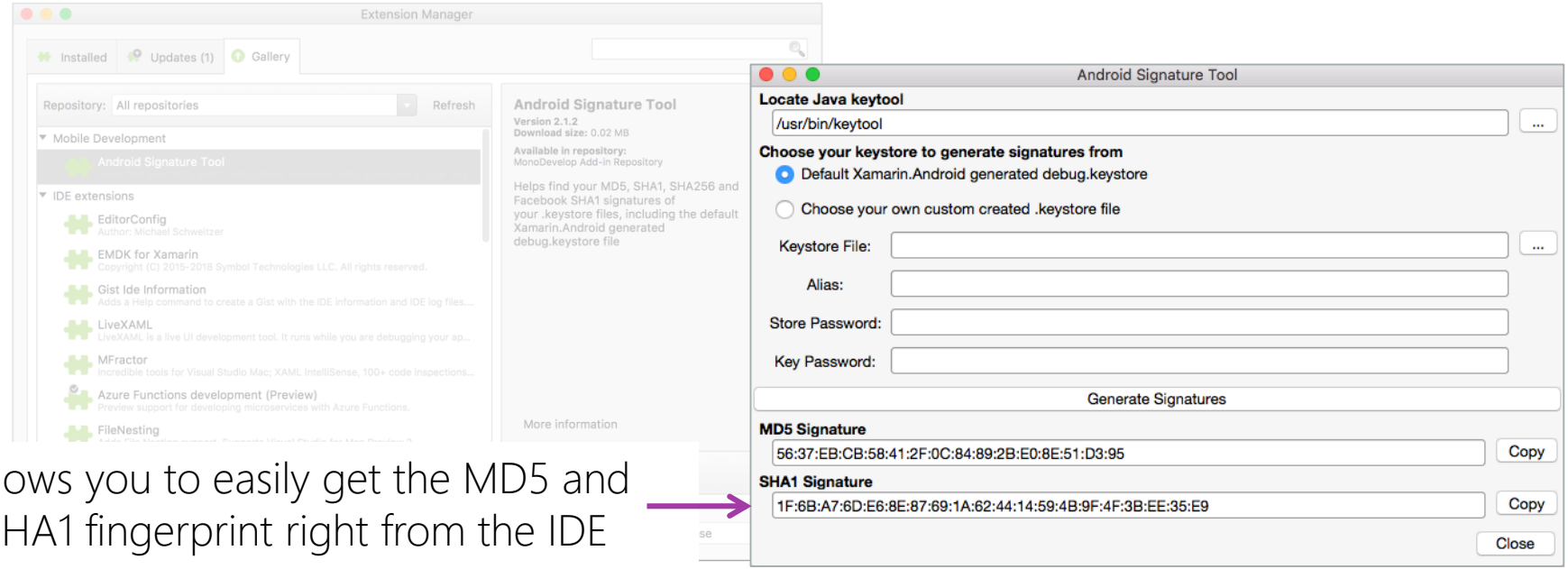
# Example: getting the SHA1 fingerprint

```
> keytool -list -v -keystore ~/.local/share/Xamarin/Mono\ for\
Android/debug.keystore -storepass android -keypass android

Certificate fi
     MD5:  56                                  :8E:51:D3:95
     SHA1: 1F                                  :59:4B:9F:4F:3B:EE:35:E9
     Signature
     Version:
```

Store password and Key password are always "android" for the default debug keystore, these values will change for custom keystores

# Getting the SHA1 fingerprint [macOS]

- Visual Studio for Mac has an optional add-in you can use to find the keystore signature – use **Visual Studio > Extensions...** to install



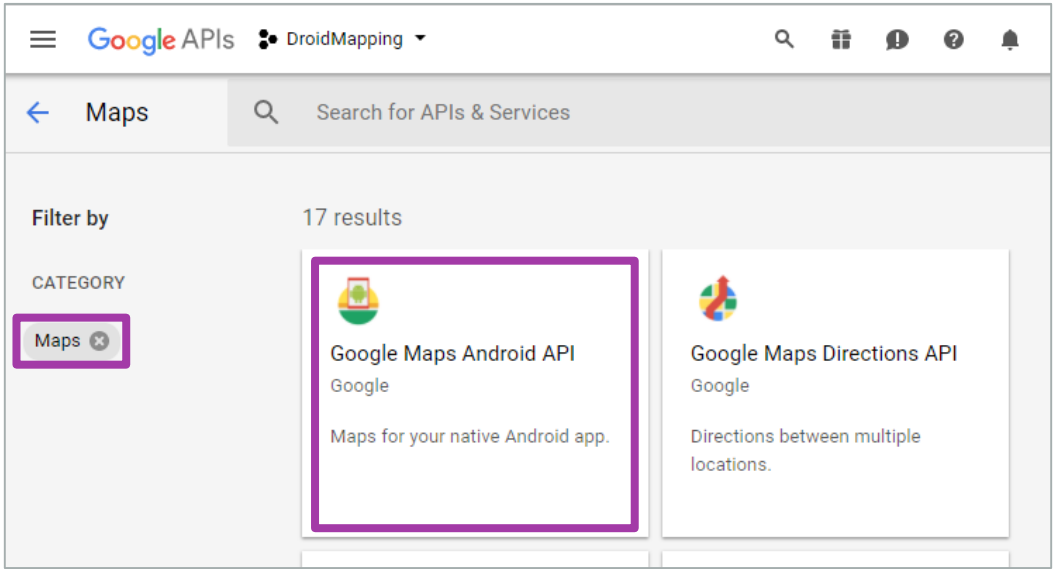

Allows you to easily get the MD5 and SHA1 fingerprint right from the IDE

# Group Exercise

Locate your keystore and SHA1 fingerprint

# Get an API key: create a project

- Must create or have a project in the Google APIs console *and* add the Google Maps Android API v2 service to the existing project

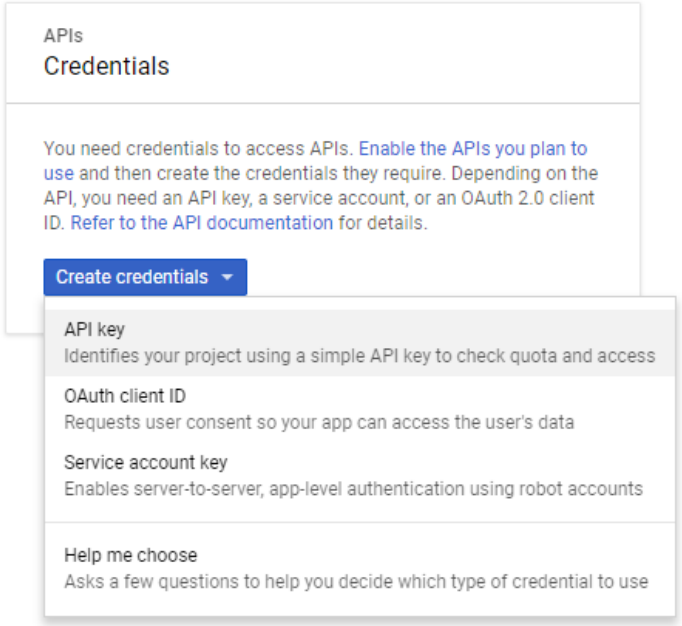



Go to https://console.developers.google.com to access the API console

# Get an API key: register the fingerprint

- Can create a new public API access key for the application in the credentials section

- This is where the SHA1 fingerprint and package name are registered

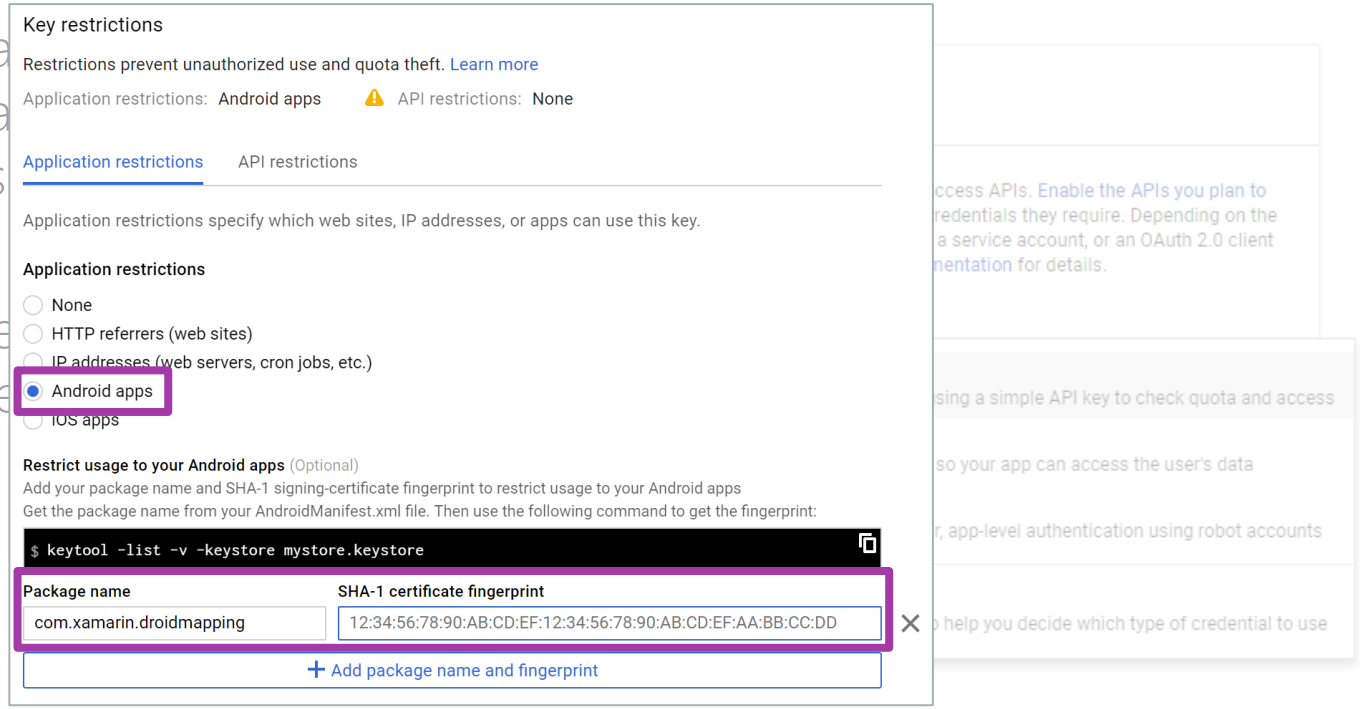





Google allows you to create an unrestricted key that doesn't depend require the SHA1 – this is not recommended – if the key leaked it could be used outside of your application

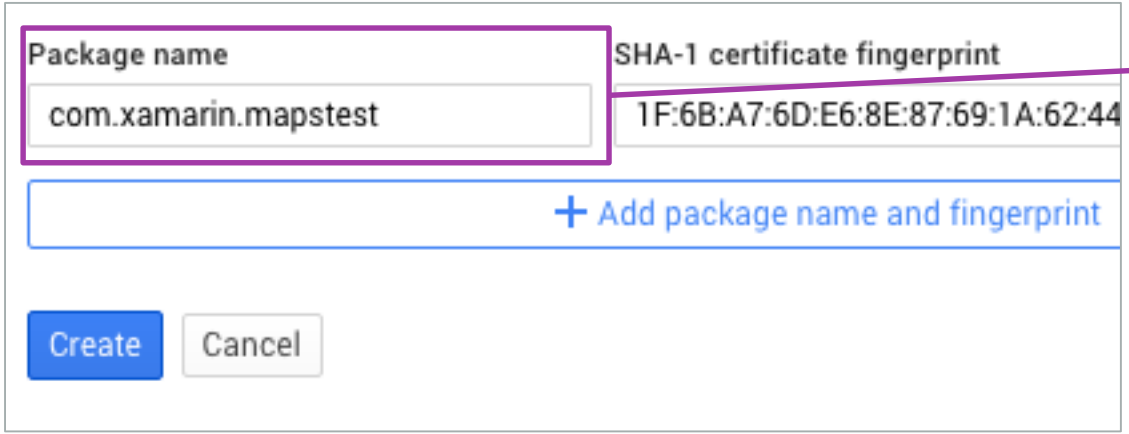
# Get an API key: register the fingerprint

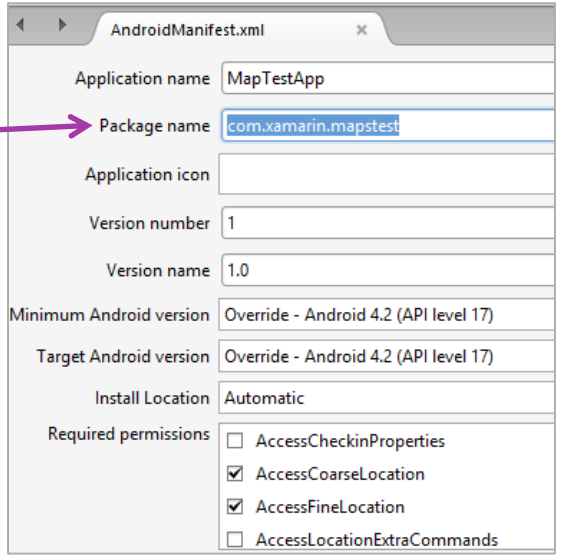
- Can create a key for the a credentials s
- This is where and package



# Synchronizing the package id

- Package Name registered in Google Play console must match the one in the app manifest



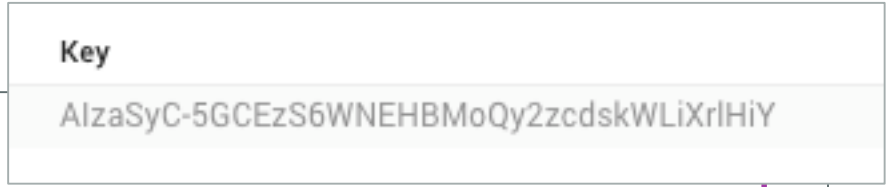



# Get an API key

- Once registered, the app will have an assigned API key – this is what needs to be added to the binary to identify the app to Google Maps

# Assign the API key to the app

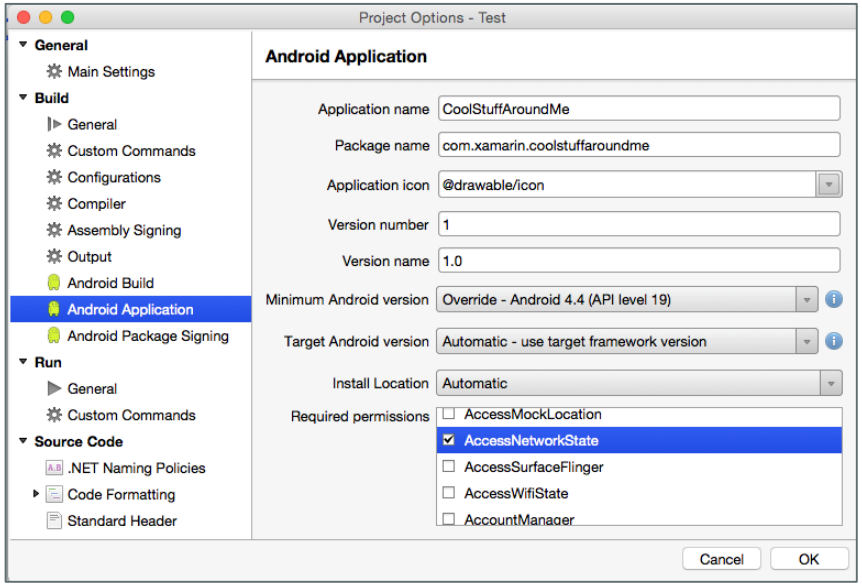
- The API key value must be registered in the Android Manifest under the key **com.google.android.geo.API_KEY**, can either hand-edit the manifest or add it with an assembly-level **`[MetaDataAttribute]`**




```
using Android.App;

#if RELEASE
[assembly: MetaDataAttribute("com.google.android.geo.API_KEY",
                             Value = "release_key_goes_here")]
#else
[assembly: MetaDataAttribute("com.google.android.geo.API_KEY",
                             Value = "AIzaSyC-5GCEzS6WNEHBMoQy2zcdskWLiXrlHiY")]
#endif
```

# Required Android permissions

- Mapping requires several permissions to access network state, internet; optionally, location permissions are required to use the user's location




Permissions can be edited with the project properties GUI

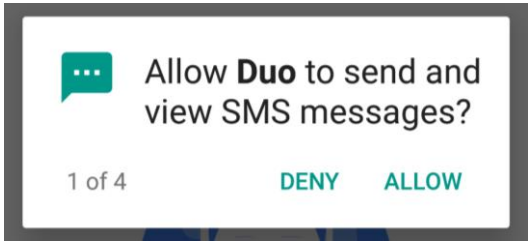
# Android permissions in the manifest

- Permissions set in the GUI are automatically added to the **`AndroidManifest.xml`**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <uses-sdk android:minSdkVersion="15" />
    <application />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
</manifest>
```

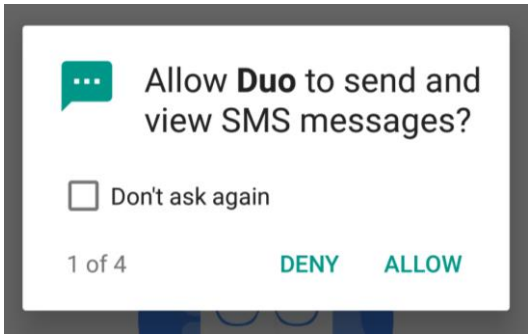
# Android's granular permission model

- Android 6.0 and newer requires explicit approval from the user at runtime for permissions allowing access to a user's personal information



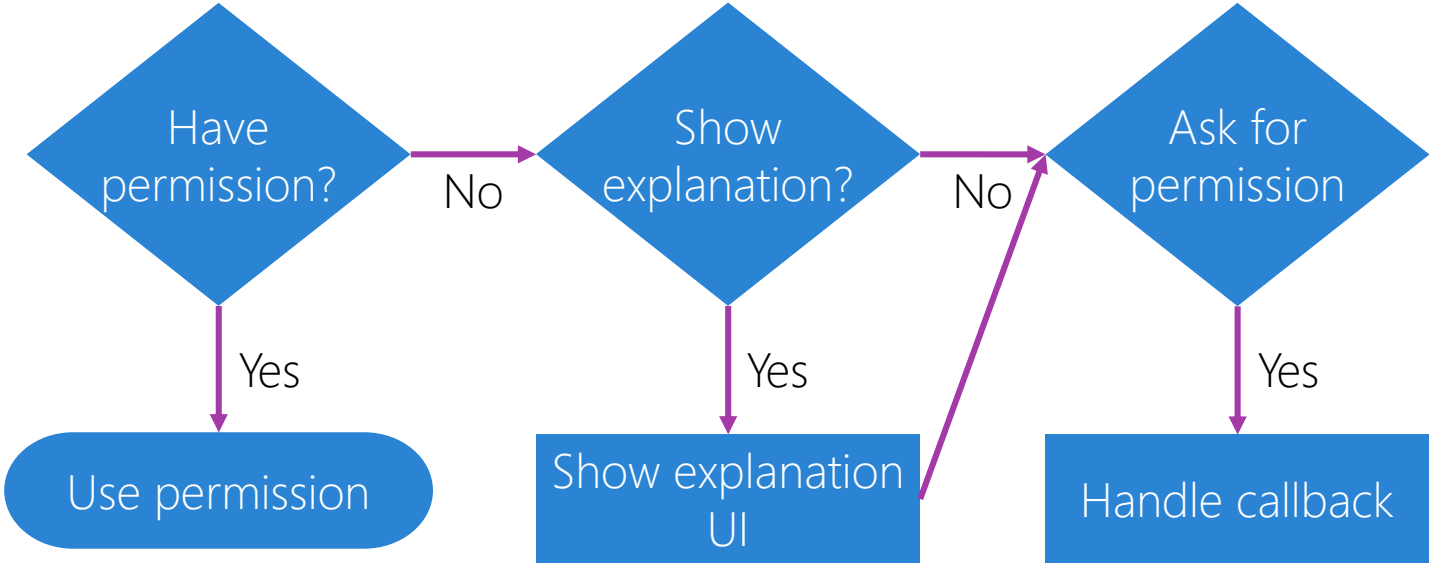

First permission request

Second permission request



# Requesting individual permissions

- To acquire specific permissions, ask Android to present the request and show a custom explanation if they have previously denied



# Request Android permissions at runtime

- Check if a permission was already granted before asking for it from the user at runtime, providing an explanation as needed

```
if (CheckSelfPermission(Manifest.Permission.AccessFineLocation)
    == Permission.Granted && …) {
    // Already have permission!
}
else if (ShouldShowRequestPermissionRationale(Manifest.Permission.AccessFineLocation)
    || …) {
    // Show UI explaining why we need permissions, then request permissions.
}
else {
    RequestPermissions(new[] { Manifest.Permission.AccessFineLocation, … },
        yourPermissionRequestCode);
}
```

# Request Android permissions at runtime

- Users responses to permission requests are verified and handled in the **`OnRequestPermissionsResult`** method Android calls on your activity

```
public override void OnRequestPermissionsResult(int requestCode,
                                                 string[] permissions,
                                                 Permission[] grantResults)
{
    switch (requestCode)
    {
        case yourPermissionRequestCode:
            // Verify they granted permission and handle accordingly.
            break;
    }
}
```

# Flash Quiz

Xamarin University

# Flash Quiz

① You can use the same API key on more than one development machine (True or False)?

a) True

b) False

# Flash Quiz

① You can use the same API key on more than one development machine (True or False)?

a) True

b) False

# Flash Quiz

② How does Google Play Services authenticate your application?

a) Shared password
b) API key + SHA1 keystore fingerprint + package name
c) X.509 certificate
d) No authentication necessary

# Flash Quiz

② How does Google Play Services authenticate your application?

a) Shared password

b) API key + SHA1 keystore fingerprint + package name

c) X.509 certificate

d) No authentication necessary

# Flash Quiz

③ Which permissions must be present in your Android app's manifest to show the user a Google Map? (Select all that apply)

a) INTERNET
b) ACCESS_NETWORK_STATE
c) ACCESS_FINE_LOCATION
d) WRITE_EXTERNAL_STORAGE

# Flash Quiz

③ Which permissions must be present in your Android app's manifest to show the user a Google Map? (Select all that apply)

a) INTERNET
b) ACCESS_NETWORK_STATE
c) ACCESS_FINE_LOCATION
d) WRITE_EXTERNAL_STORAGE

# Group Exercise

Obtain a Google API key for Google Maps V2

# Summary

1. Create an API key
2. Register your application with Google Maps API
3. Set required permissions to use location in your application

# Display a map in your application

Xamarin University

# Tasks

1. Add a Map UI into your app
2. Configure the map view
3. Display current location on the map

# Google Maps Basics

- **`GoogleMap`** is the main class for the Google Maps Android API – it is created automatically when you insert a map into the UI
- **`MapView`** and **`MapFragment`** are used to add the map visualization into the UI screen
- Requires API Level 12 or the Android Support library

# Using MapFragment

- Can insert the map into a layout as a *fragment* – this is the most common approach and manages both the map UI and lifecycle

Main.axml

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          class="com.google.android.gms.maps.MapFragment" />
```

class defines the object which will draw the fragment – in this case **MapFragment**



There is no designer support for this – this must be added by hand into the XML

# Accessing the MapFragment

- Use the **`FragmentManager`** to find and get a reference to the created UI fragment

```
public class MainActivity : Activity
{
   ...
   protected override void OnCreate(Bundle bundle)
   {
      var mf = FragmentManager.FindFragmentById(Resource.Id.map) as MapFragment;
      ...
   }
}
```

Use the assigned **`android:id`** to locate the fragment

# Creating a fragment in code

- **`FragmentManager`** also allows direct insertion of fragments from code

```
MapFragment mapFragment;

protected override void OnCreate(Bundle bundle)
{
    ...
    mapFragment = MapFragment.NewInstance(); // Create fragment
    FragmentTransaction trans = FragmentManager.BeginTransaction();
    trans.Add(Android.Resource.Id.Content, mapFragment);
    trans.Commit();
}
```

Must identify the view parent for the fragment

# Using MapView

- **`MapView`** is a *view*-derived class that can be used to host the map in a child fragment (to avoid fragments-in-fragments)

MapFragment.axml

```
<com.google.android.gms.maps.MapView
      android:id="@+id/mapview"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />
```

**`MapView`** can be placed in any Layout as a child, or used as the root Layout element on it's own (that's essentially what a **`MapFragment`** is)

# Using MapView – manage lifecycle

- Must forward lifecycle events from the fragment owner to the **MapView**

```
public class MapsFragment : Fragment {
   MapView mapView;

   public override View OnCreateView(LayoutInflater inflater, ...) {
      var view = inflater.Inflate(...);
      mapView = view.FindViewById<MapView>(Resource.Id.mapview);
      mapView.OnCreate(savedInstanceState);
      return view;
   }

   protected override void OnResume()  {
      base.OnResume();
      mapView.OnResume();
   }
}
```

Must forward **OnCreate**, **OnResume**, **OnDestroy** and **OnLowMemory**

# Which one should I use?

- Use a **`MapFragment`** if you are hosting the map in an **`Activity`** layout
- Use **`MapView`** if you are hosting the map in a **`Fragment`** layout (such as in a **`ViewPager`**)
- They both produce *exactly the same* visualization

# Getting the Google Map instance

- **MapFragment** and **MapView** will create the **GoogleMap** object automatically; can be retrieved through **GetMapAsync**

```
public class MainActivity : Activity, IOnMapReadyCallback
{
    GoogleMap map;
    public void OnMapReady(GoogleMap googleMap)
        map = googleMap;
    }

    protected override void OnCreate(Bundle bun

        ...
        var mf = FragmentManager.Find                    agment;
        mf.GetMapAsync(this);
        ...
}
```

Map not available until it has been initialized with Google's web services, must pass interface **IOnMapReadyCallback** to **GetMapAsync**

# Getting the Google Map instance

- **`MapFragment`** and **`MapView`** create the **`GoogleMap`** object, can be retrieved through **`GetMapAsync`**

```
public class MainActivity : Activity, IOnMapReadyCallback
{
    GoogleMap map;
    public void OnMapReady(GoogleMap googleMap) {
        map = googleMap;
    }

    protected override void OnCreate(Bundle bundle) {
        ...
        var mf = FragmentManager.FindFragmentById(Resource.Id.map) as MapFragment;
        mf.GetMapAsync(this);
    }
}
```

Common to implement directly in activity

# Getting the Google Map instance

- **MapFragment** and **MapView** create the **GoogleMap** object, can be retrieved through **GetMapAsync**

```
public class MainActivity : Activity, IOnMapReadyCallback
{
    GoogleMap map;
    public void OnMapReady(GoogleMap googleMap) {
        map = googleMap;
    }

    protected override void OnCreate(Bundle
        ...
        var mf = FragmentManager.FindFragment
        mf.GetMapAsync(this);
```

```
) as MapFragment;
```

Can then cache off reference once Map has been created



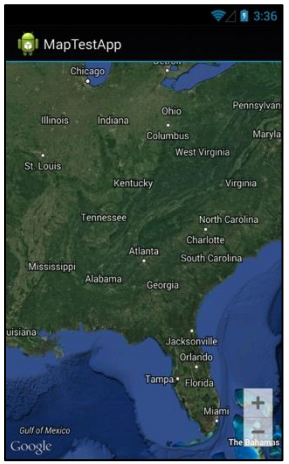
Almost all of the mapping APIs are exposed through the **GoogleMap** instance. You must be on the UI thread to interact with this object

# Changing the map style

- **`GoogleMap`** provides properties to change the type of data displayed, such as the **`MapType`** property which can be one of four values


Standard (default)


Satellite

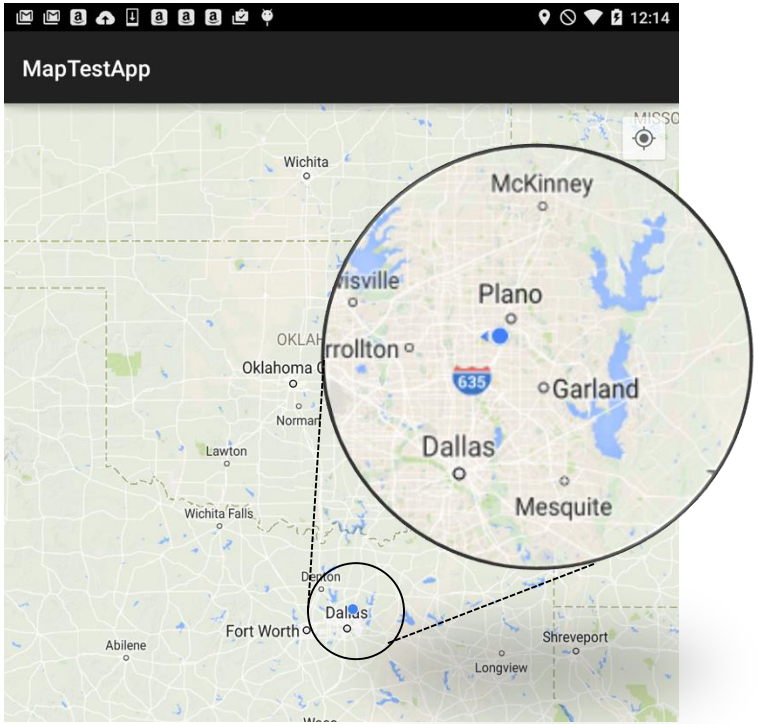



Hybrid


Terrain

# Noting the current location

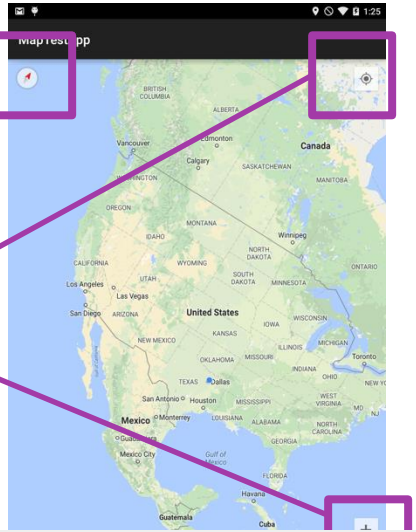
- Can turn on a location marker using the `MyLocationEnabled` property on `GoogleMap`
- Displays a glowing dot indicating the current location of the device, or a chevron if the device is moving
- Requires `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION` permissions



# Use UiSettings class to provide user control

- The **UiSettings** property on the **GoogleMap** turns various UI elements on and off in the display such as a compass and zoom controls

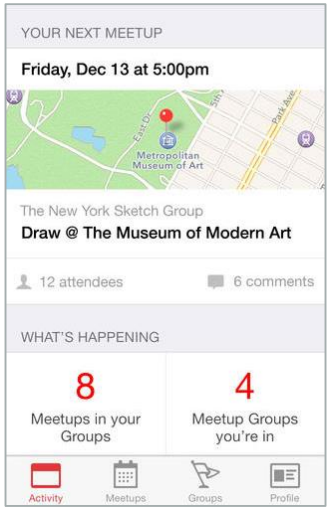
```
GoogleMap map ...
map.UiSettings.CompassEnabled = true;
map.UiSettings.MyLocationButtonEnabled = true;
map.UiSettings.ZoomControlsEnabled = true;
...
```




There is also a **GoogleSettings** class which can initialize the map to a known state when creating the **MapFragment** in code

# Built-in gestures

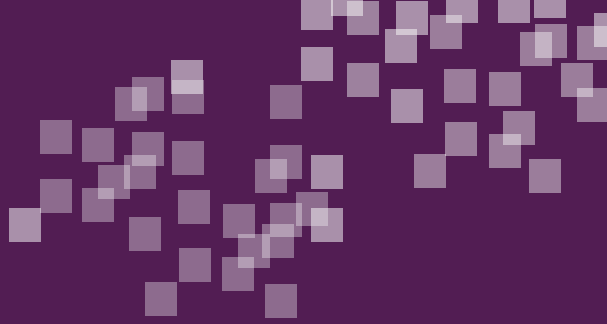
- Map control supports common gestures to zoom, rotate and scroll – these are enabled by default, but can be turned off with **`UiSettings`** to support read-only map views that cannot be altered




```
map.UiSettings.RotateGesturesEnabled = false;
map.UiSettings.TiltGesturesEnabled = false;
map.UiSettings.ScrollGesturesEnabled = false;
map.UiSettings.ZoomGesturesEnabled = false;
```

OR

```
map.UiSettings.SetAllGesturesEnabled(false);
```



# Individual Exercise

Adding support for Google Maps

Xamarin University

# Summary

1. Add a Map UI into your app
2. Configure the map view
3. Display current location on the map

# Add markers to a Google Map

# Tasks

1. Add map markers
2. Display marker details
3. Programmatically interact with the map

# Google Maps markers

- Markers allow you to annotate the map with different colored *pins*

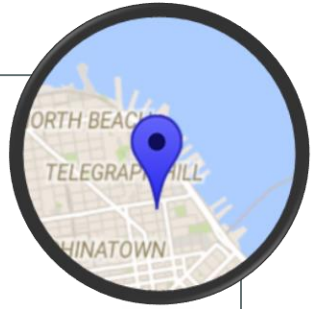



Particularly common to do this when creating static, immovable maps

# Adding a marker to the map

- Markers are created through the **`MarkerOptions`** class and added to the map using the **`GoogleMap.AddMarker`** method

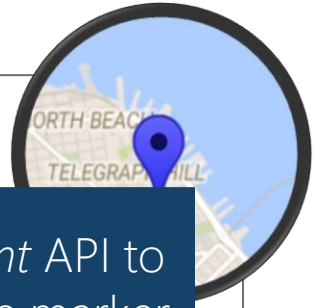
```
GoogleMap map;
...
LatLng Location_Xamarin = new LatLng(37.79, -122.40);
Marker xamarinMarker = map.AddMarker(
      new MarkerOptions()
         .SetPosition(Location_Xamarin)
         .SetIcon(BitmapDescriptorFactory.DefaultMarker(
                     BitmapDescriptorFactory.HueBlue)));
```



# Adding a marker to the map

- Markers are created through the **`MarkerOptions`** class and added to the map using the **`GoogleMap.AddMarker`** method

```
GoogleMap map;
...
LatLng Location_Xamarin = new LatLng(37.79, -1
Marker xamarinMarker = map.AddMarker(
      new MarkerOptions()
         .SetPosition(Location_Xamarin)
         .SetIcon(BitmapDescriptorFactory.DefaultMarker(
                     BitmapDescriptorFactory.HueBlue)));
```

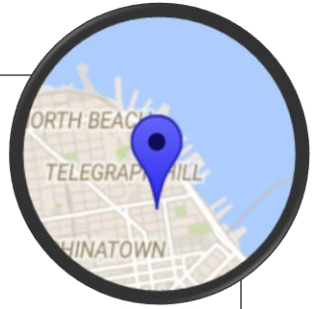


Uses a *fluent* API to describe the marker

# Adding a marker to the map

- Markers are created through the **MarkerOptions** class and added to the map using the **GoogleMap.AddMarker** method

```
GoogleMap map;
...
LatLng Location_Xamarin = new LatLng(37.79, -122.40);
Marker xamarinMarker = map.AddMarker(
      new MarkerOptions()
         .SetPosition(Location_Xamarin)
         .SetIcon(BitmapDescriptorFactory.DefaultM
                  BitmapDescriptorFactory.HueBl
```




Must always set position

# Adding a marker to the map

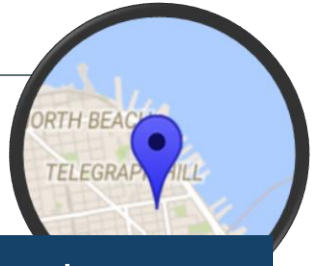
- Markers are created through the **MarkerOptions** class and added to the map using the **GoogleMap.AddMarker** method

```
GoogleMap map;
...
LatLng Location_Xamarin = new LatLng(37.79, -12
Marker xamarinMarker = map.AddMarker(
      new MarkerOptions()
         .SetPosition(Location_Xamarin)
         .SetIcon(BitmapDescriptorFactory.DefaultMarker(
                  BitmapDescriptorFactory.HueBlue)));
```

# Adding a marker to the map

- Markers are created through the **MarkerOptions** class and added to the map using the **GoogleMap.AddMarker** method

```
GoogleMap map;
...
LatLng Location_Xamarin = new LatLng(37.79, -122.40);
Marker xamarinMarker = map.AddMarker(
      new MarkerOptions()
         .SetPosition(Location_Xamarin)
         .SetIcon(BitmapDescriptorFactory.Defaul
                  BitmapDescriptorFactory.Hue
```




Call **AddMarker** to place the **MarkerOption** on the map – returns the marker instance

# MarkerOptions settings

- Various options can be configured through the fluent API exposed by the **`MarkerOptions`**

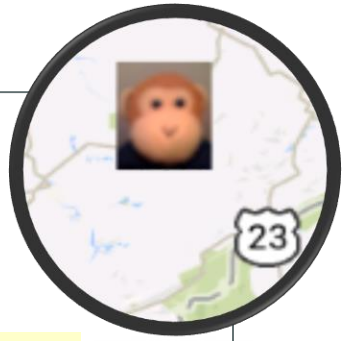
```
new MarkerOptions()
    .SetPosition(Location_Xamarin)
    .SetIcon(BitmapDescriptorFactory.DefaultMarker())
    .SetAlpha(.5f)
    .SetRotation(45.0f);
```

Properties are set on the resulting marker using fluent methods, each method returns the **`MarkerOption`** so you can chain them on a single statement

# Using custom images in the Marker

- Can set the marker to be a custom image (including transparency); use the **Anchor** to adjust the image position relative to the **LatLng** coordinate (centered by default)

```
Marker xamarinMarker = map.AddMarker(
        new MarkerOptions()
            .SetPosition(Location_Xamarin)
            .Anchor(1,1)
            .SetIcon(
          BitmapDescriptorFactory.FromResource(
                Resource.Drawable.Monkey));
```



# Changing Marker properties

- Marker settings can be altered at runtime through the **`Marker`** instance

```
Marker marker = ...

// Change the marker to be green to indicate "no problem"
marker.SetIcon(BitmapDescriptorFactory.DefaultMarker(
                    BitmapDescriptorFactory.HueGreen));
marker.Rotation = 180f;


...
marker.Remove(); // Remove from map!
```

# Info Window

- Can add popup info window to describe details for a given map marker, shown when the user taps the marker (or programmatically)

# Controlling the marker info window

- **`SetTitle`** displays the info window title and **`SetSnippet`** sets a single line of text in a slightly smaller font

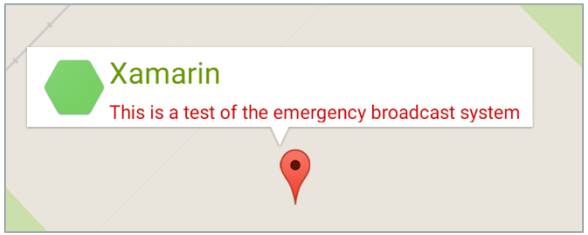
```
var newYorkMarker = map.AddMarker(new MarkerOptions()
            .SetPosition(Location_NewYork)
            .InfoWindowAnchor(.5f,0)
            .SetTitle("The Big Appple")
            .SetSnippet("Get Details!"));

newYorkMarker.ShowInfoWindow();
```

Can request to show the info window immediately

# Customizing the info window

- Can assign **`IInfoWindowAdapter`** implementation to the map to provide complete customization of displayed view, can either provide just the *content* or the complete frame (to customize even the balloon style)

```
map.SetInfoWindowAdapter(
   new CustomMarkerPopupAdapter(LayoutInflater));
```



# Implementing IInfoWindowAdapter

```
public class CustomMarkerPopupAdapter : Java.Lang.Object, GoogleMap.IInfoWindowAdapter
{
   private LayoutInflater layoutInflater;
   public CustomMarkerPopupAdapter(LayoutInflater inflater) {
      layoutInflater = inflater;
   }

   // Used to draw frame + contents
   public View GetInfoWindow(Marker marker) { return null; }

   // Draw only contents
   public View GetInfoContents(Marker marker) {
      var view = layoutInflater.Inflate(...);
      ... // Fill in view from marker data
      return view;
   }
}
```

# Implementing IInfoWindowAdapter

```
public class CustomMarkerPopupAdapter : Java.Lang.Object, GoogleMap.IInfoWindowAdapter
{
    private LayoutInflater layoutInflater;
    public CustomMarkerPopupAdapter(Layou
        layoutInflater = inflater;
    }

    // Used to draw frame + contents
    public View GetInfoWindow(Marker marker) { return null; }

    // Draw only contents
    public View GetInfoContents(Marker marker) {
        var view = layoutInflater.Inflate(...);
        ... // Fill in view from marker data
        return view;
    }
}
```

Must derive from Java.Lang.Object *and* implement interface

# Implementing IInfoWindowAdapter

```
public class CustomMarkerPopupAdapter : Java.Lang.Object, GoogleMap.IInfoWindowAdapter
{
   private LayoutInflater layoutInflater;
   public CustomMarkerPopupAdapter(LayoutInflater inflater) {
      layoutInflater = inflater;
   }

   // Used to draw frame + contents
   public View GetInfoWindow(Marker marker) { return null; }

   // Draw only contents
   public View GetInfoContents(Marker marker) {
      var view = layoutInflater.Inflate(...);
      ... // Fill in view from marker data
      return view;
   }
}
```

Implement *either* **GetInfoWindow** or **GetInfoContents**

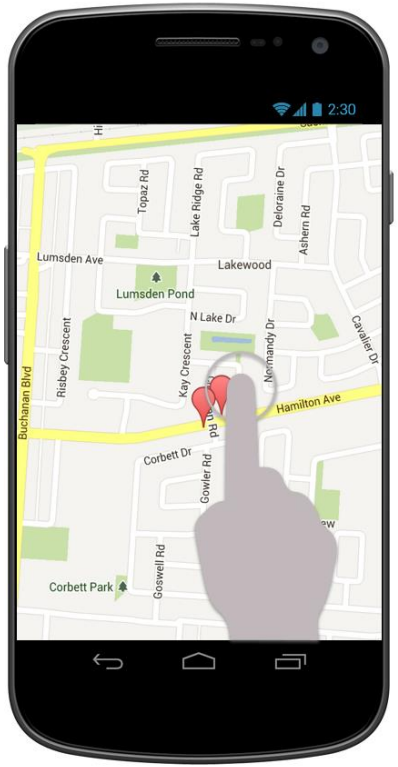
# Implementing IInfoWindowAdapter

```
public class CustomMarkerPopupAdapter : Java.Lang.Object, GoogleMap.IInfoWindowAdapter
{
   private LayoutInflater layoutInflater;
   public CustomMarkerPopupAdapter(LayoutInflater inflater) {
      layoutInflater = inflater;
   }

   // Used to draw frame + contents
   public View GetInfoWindow(Marker marker) { return null; }

   // Draw only contents
   public View GetInfoContents(Marker marker) {
      var view = layoutInflater.Inflate(...);
      ... // Fill in view from marker data
      return view;
   }
}
```

Can return null to get default behavior (e.g. to only draw *certain* markers)

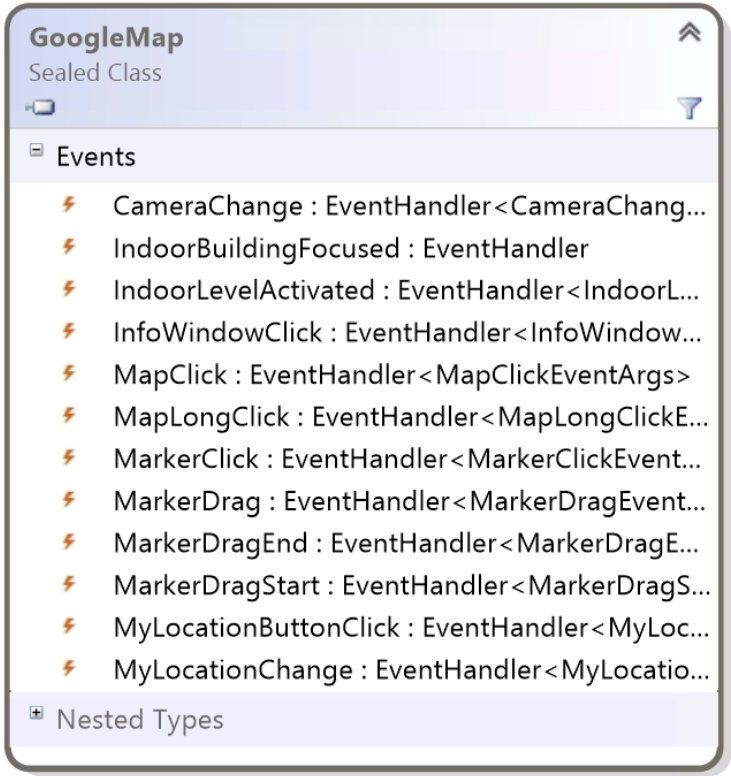
# Responding to interaction

- Several events are exposed on the **`GoogleMap`** to support adding user interactivity
  - click on a **`Marker`**
  - drag a **`Marker`**
  - click on an info window
  - long-click on the map



# Map Events

- Xamarin component exposes map interactivity as .NET events
- Can also use underlying interface notification support if desired (for behavioral reuse)
  - `SetOnMapClickListener`
  - `SetOnMarkerClickListener`
  - …



# Activating the marker

- Google Maps raises the **MarkerClick** event in response to user interaction with a marker, even if an info window is associated with it

```
map.MarkerClick += HandleMarkerClick;
...
void MapOnMarkerClick(object sender, GoogleMap.MarkerClickEventArgs e)
{
    e.Handled = false;
    Marker marker = e.Marker;
    if (marker.Equals(xamarinMarker)) {
        ... // tapped on Xamarin marker
    }
}
```

Passed **MarkerClickedEventArgs** to retrieve the marker which was activated

# Activating the marker

- Google Maps raises the **`MarkerClick`** event in response to user interaction with a marker, even if an info window is associated with it

```
map.MarkerClick += HandleMarkerClick;
...
void MapOnMarkerClick(object sender, GoogleMap.MarkerClickEventArgs e)
{
    e.Handled = false;
    Marker marker = e.Marker;
    if (marker.Equals(xamarinMarker))
        ... // tapped on Xamarin marker
    }
}
```

Handled flag (true by default) decides whether to perform the normal action – if this is true, then the Info Window will *not* be shown!

# Dragging the marker

- Setting a marker as *draggable* will cause the map to raise **MarkerDragStart**, **MarkerDrag** and **MarkerDragEnd** events to track when the user drags the marker around

```
marker.Draggable = true;
...
map.MarkerDrag += OnMarkerDrag;
...
void OnMarkerDrag(object sender, GoogleMap.MarkerDragEventArgs e)
{
    Marker marker = e.Marker;
    LatLng pos = marker.Position;
    ... // Process the drag operation
};
```

Drag events passed event args with marker

# Interacting with the Info Window

- Google Maps raises the **`InfoWindowClick`** event when the user taps on a displayed info window popup for a marker

```
map.InfoWindowClick += HandleInfoWindowClick;
...
void HandleInfoWindowClick(object sender,
                           GoogleMap.InfoWindowClickEventArgs e)
{
    e.Handled = true;
    Marker marker = e.Marker;
    if (marker.Equals(xamarinMarker)) {
        ... // tapped on Xamarin marker info window
    }
}
```

# Interacting with the map directly

- **`GoogleMap`** exposes **`MapClick`** and **`MapLongClick`** to handle click interactions

```
map.MapClick += (object sender, GoogleMap.MapClickEventArgs e) =>
{
    LatLng pos = e.Point;
    ...
};
```

# Interacting with the map directly

- **`GoogleMap`** exposes **`MapClick`** and **`MapLongClick`** to handle click interactions

Can also capture raw motion events from the parent or a transparent overlay and then transform screen coordinates into locations using the **`Projection`** property

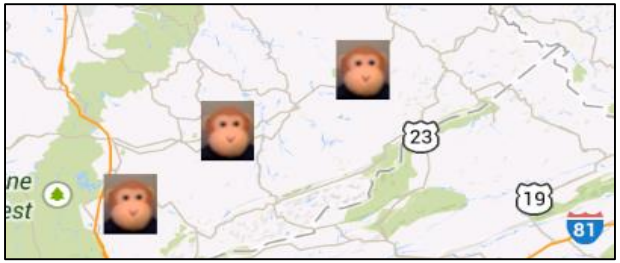
```
List<LatLng> touchPoints = ...;
void OnMotionEvent(object sender, MotionEvent e)
{
    if (e.Action == MotionEventActions.Move) {
       touchPoints.Add (
          map.Projection.FromScreenLocation(
             new Point(
                (int)e.RawX, (int)e.RawY)));
    }
    ...
}
```

# Flash Quiz

Xamarin University

# Flash Quiz

① How were these marker images created?




a) `AddMarker`

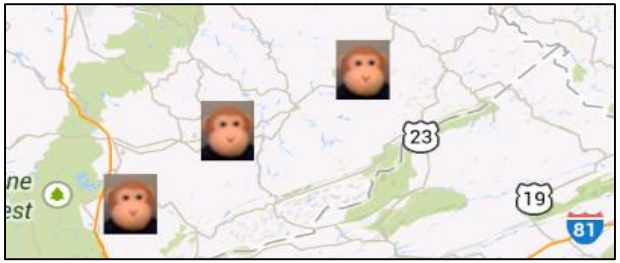
b) `MarkerOptions`

c) Both A and B

d) None of the Above

# Flash Quiz

① How were these marker images created?




a) `AddMarker`

b) `MarkerOptions`

c) Both A and B

d) None of the Above

# Flash Quiz

② Google Maps raises the `InfoWindowClick` event when the user taps on a displayed info window popup for a marker.

a) True

b) False

# Flash Quiz

② Google Maps raises the `InfoWindowClick` event when the user taps on a displayed info window popup for a marker.

a) True

b) False

# Individual Exercise

Working with markers

# Summary

1. Add map markers
2. Display marker details
3. Programmatically interact with the map

# Adjust the map position and perspective

Xamarin University

# Tasks

1. Update the map viewpoint
2. Animate camera position changes

# Google map surface

- Map is drawn as a flat plane on the screen with a camera looking down

- Position of the camera can be adjusted to change the
  - location
  - zoom
  - tilt
  - bearing

# Specifying the camera position

- The **CameraPosition** class identifies the location, bearing, tilt and zoom level; includes a fluent API to generate a specific position

```
CameraPosition pos = CameraPosition.InvokeBuilder()
      .Target(new LatLng(40.7, -74.0))
      .Zoom(20)
      .Bearing(155)
      .Tilt(65)
      .Build();
```

Latitude and Longitude are specified as **LatLng**

# Specifying the camera position

- The **CameraPosition** class identifies the location, bearing, tilt and zoom level; includes a fluent API to generate a specific position

```
CameraPosition pos = CameraPosition.InvokeBuilder()
      .Target(new LatLng(40.7, -74.0))
      .Zoom(20)
      .Bearing(155)
      .Tilt(65)
      .Build();
```

Zoom level at the center of the map, can get valid range with **MinZoomLevel** and **MaxZoomLevel**

# Specifying the camera position

- The **`CameraPosition`** class identifies the location, bearing, tilt and zoom level; includes a fluent API to generate a specific position

```
CameraPosition pos = CameraPosition.InvokeBuilder()
      .Target(new LatLng(40.7, -74.0))
      .Zoom(20)
      .Bearing(155)
      .Tilt(65)
      .Build();
```

Compass measurement clockwise from North

# Specifying the camera position

- The **CameraPosition** class identifies the location, bearing, tilt and zoom level; includes a fluent API to generate a specific position

```
CameraPosition pos = CameraPosition.InvokeBuilder()
      .Target(new LatLng(40.7, -74.0))
      .Zoom(20)
      .Bearing(155)
      .Tilt(65)
      .Build();
```

Viewing angle directly facing earth (down)

# Specifying the camera position

- The **CameraPosition** class identifies the location, bearing, tilt and zoom level; includes a fluent API to generate a specific position

```
CameraPosition pos = CameraPosition.InvokeBuilder()
      .Target(new LatLng(40.7, -74.0))
      .Zoom(20)
      .Bearing(155)
      .Tilt(65)
      .Build();
```

Call **Build** method to generate final position object

# Changing the camera viewpoint

- Camera position is changed by creating an *update request* through the **CameraUpdateFactory** and calling **MoveCamera** on the map

```
void ChangeCamera(CameraPosition pos)
{
    CameraUpdate update = CameraUpdateFactory
                                .NewCameraPosition(pos);
    map.MoveCamera(update);
}
```

**MoveCamera** will *immediately* transition the current view to the specified location identified by the passed **CameraUpdate**

# Adjusting the camera viewpoint

- **`CameraUpdateFactory`** has a variety of *adjustment* methods which apply to the current position

```
CameraUpdate update = CameraUpdateFactory.ZoomIn();
                      .ZoomOut();
                      .ZoomTo(...);
                      .ZoomBy(...);
                      .ScrollBy(x,y);
                      .NewLatLng(...);
                      .NewLatLngBounds(...);
                      .NewLatLngZoom(...);
```

Returned **`CameraUpdate`** can then be used to reposition the map location

# Animating the camera viewpoint

- Can also animate the position change – this will smoothly shift the location in the map from the current location to the specified one

```
CameraUpdate update = ...;
map.AnimateCamera(update);
map.AnimateCamera(update, 2000, null);
```

- `StopAnimation` can be used to halt the animation and immediately transition to the new position – for example if the user taps on the map

```
map.StopAnimation();
```

# Monitoring camera viewpoint changes

- Interactive maps can be adjusted with gestures, can monitor these changes by handling the **CameraChange** event

```
map.CameraChange +=
      (object sender, GoogleMap.CameraChangeEventArgs e) => {
         CameraPosition newPos = e.P0;
         ... // React to change
      };
```

# Flash Quiz

Xamarin University

# Flash Quiz

① What does the **`CameraPosition`** identify?

a) Tilt

b) Zoom

c) Target

d) All the above

# Flash Quiz

① What does the **`CameraPosition`** identify?

a) Tilt

b) Zoom

c) Target

d) All the above

# Flash Quiz

② What class do you use to create an update request?

a) `CameraUpdateFactory`

b) `CameraUpdate`

c) `CameraPosition`

d) None of the above

# Flash Quiz

② What class do you use to create an update request?

a) **`CameraUpdateFactory`**

b) `CameraUpdate`

c) `CameraPosition`

d) None of the above

# Individual Exercise

Changing the map viewpoint

# Summary

1. Update the map viewpoint
2. Animate camera position changes

# Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile