RockMobile

The purpose of this document is to detail the APIs that has been built to aid in developing mobile apps for both iOS and Android devices.

Although multi-platform, naming conventions and design patters will more closely adhere to iOS / CSharp than they will Android.

Table of Contents

RockMobile 1

CoreLocation 1

iBeacon 2

Network 4

HttpWebRequest 4

Math 4

PlatformCommon 4

iOS 4

Droid 4

DroidFontManager 5

PlatformMedia 5

PlatformCamera 5

PlatformImagePicker 6

Profiler 7

Threading 8

UIThreading 8

## RockMobile

The primary problem when developing for multiple platforms is dealing with device specific features in a cross-platform way.

The goal of RockMobile is to provide unified, platform agnostic APIs for features that have different platform implementations.

For example, while both iOS and Android support iBeacon technology, they each have their own implementation with slightly different requirements. The iBeacon API offered by RockMobile abstracts the platform, allowing users to implement iBeacon support once and share the code across platforms.

## CoreLocation

Core Location represents the namespace for items that utilize location-detection technologies, such as GPS, iBeacon and Wifi.

### iBeacon

This namespace contains technology that allows platform agnostic use of iBeacons.

Usage:

Namespace: RockMobile.CoreLocation.iBeacon

Singleton: LocationManager.Instance

1. Create a new BeaconRegion containing the UUID of the iBeacons to monitor.

2. Add callbacks to LocationManager for AddRegionEnteredEvent and/or AddDidRangeBeaconsEvent

3. Call LocationManager’s StartMonitoring and StartRangingBeacons.

**Android Only:**

1. In the MainActivity, derive from IBeaconConsumer.
2. Cast LocationManager to DroidLocationManager and call BindIBeaconManager, passing the MainActivity’s ‘this’.
3. Implement IBeaconConsumer’s “public void OnIBeaconServiceConnect()”

and call DroidLocationManger.OnIBeaconServiceConnect

1. Implement OnResume() and call DroidLocationManager::EnterForegroundMode()
2. Implement OnStop() and call

DroidLocationManager::EnterBackgroundMode()

1. Implement OnDestroy() and call

DroidLocationManager::UnBindIBeaconManager()

For Background Scanning

1. Create a main Application class that derives from both Application and IBootstrapNotifier.
2. For Xamarin to consider your Application that main application,

override the standard constructor and call the base. Additionally, add the Application attribute.

1. Create a RegionBootstrap for any UUID that should be scanned.
2. In OnCreate(), allocate a region for the UUID that should be scanned, and pass it as an argument to a newly created RegionBootstrap object.
3. To control background scan time, get the IBeaconManager Instance in OnCreate, and set the background scan time and frequency.

Notes regarding background scanning:

Implemented via a bootstrapper service that is launched with your application. The user must reboot, plug in, or unplug their device to start the service.

Classes:

Beacon: An object storing the UUID, Major and Minor values.

BeaconRegion: Defines the UUID, and optionally Major, Minor values to “Range”.

RegionEventArgs: Contains the BeaconRegion of the region that was entered/exited.

RegionBeaconRangedEventArgs: Contains the BeaconRegion and associated Beacons that were “ranged”. (Meaning they were within the maximum distance to be discovered.)

LocationManager: The “core” object used for negotiating with the iBeaon technology.

Code Implementation:

iBeacon.cs – This implements the abstracted interface that end-users

should use.

iOS\_iBeacon.cs – This implements the iOS iBeacon API.

Apple’s CoreLocation framework contains CLLocationManager and

dependent classes. For the most part, iBeacon is a simple wrapper for

iOS\_iBeacon.

Droid\_iBeacon.cs – Because Android SDK doesn’t offer an “iBeacon” API, RockMobile uses RadiusNetwork’s SDK for Android. This mimics most of the functionality of the iOS implementation.

Because Android concepts such as ‘Binding’ objects, several additional features had to be added to the Android implementation. These are of course hidden from the iBeacon abstracted API, but include:

OnIBeaconServiceConnect() callback once Binding is finished

Queuing of monitor/ranging requests if binding has not completed.

## Network

Network provides platform abstracted reusable network components.

### HttpWebRequest

This provides quick access for retrieving data from http sockets.

## Math

Math provides helper functions for basic algebraic calculations.

## PlatformCommon

Provides platform specific functionality for common tasks.

**Code in PlatformCommon should only be called in platform specific areas of code.**

Usage:

Namespace: Rock.Mobile.PlatformCommon

Usage: Depends on the object.

### iOS

Methods:

UIFont LoadFontDynamic( string name, float fontSize )

Desc: Loads a font from the application’s asset bundle.

Returns: A reference to the loaded font.

Throws: NSErrorException on any error.

Arguments:

Name – The name of the font file to load, **excluding the extension.**

FontSize – The point size to load for the font.

### Droid

Members:

Public static Android.Content.Context Context

Should be set in the primary Android Activity as soon as possible.

Allows Rock.Mobile Android code to access the application’s context.

Methods:

Public static Bitmap ApplyMaskToBitmap(Bitmap image, Bitmap mask)

Desc: Returns a new bitmap with image applied only where mask allows.

Returns: A new bitmap

Arguments:

Image – The source image

Mask – The image to use as a mask

### DroidFontManager

Provides a central place to load and store fonts loaded dynamically on Android.

This is important because creating a font from an asset is a time consuming process.

Using the manager, if the font is already loaded, a reference to it is simply returned rather than loading it again, cutting down on memory and cpu time.

Usage:

Namespace: Rock.Mobile.PlatformCommon

Singleton: DroidFontManager.Instance

Methods:

Public Typeface GetFont( string fontName )

Desc: Either returns a reference to fontName if it is already loaded, or loads it and then returns a reference to it.

Returns: A reference to the requested font.

Arguments:

FontName – The name of the font to load **excluding the extension.**

## PlatformMedia

Provides platform abstracted access to various media components.

### PlatformCamera

Provides access to a device’s camera thru the platform provided camera image grabber.

Usage:

Namespace: Rock.Mobile.Media

Singleton: PlatformCamera.Instance

Methods:

Bool IsAvailable()

Returns: True if there is a camera attached to the device. False if there is not.

void CaptureImage(object imageDest, object context, CaptureImageEvent response)

Arguments:

imageDest – On Android, a Java.IO.File pointing to the destination for the captured image. On iOS, a string containing the destination for the captured image.

Context – On Android, the application Context. On iOS the parent view controller.

CaptureImageEvent – Delegate that will be called when the image capture is complete.

Returns: Nothing

#### CaptureImageEventArgs

The arguments returned when capturing an image.

Members:

Bool result – True if an image was obtained or the user cancelled the operation. False if there was an actual error.

String imagePath – The location of the captured image. Null if the operation was cancelled.

### PlatformImagePicker

Provides access to a device’s image picker.

Usage:

Namespace: Rock.Mobile.Media

Singleton: PlatformImagePicker.Instance

Methods:

Void PickImage(object context, ImagePickEvent callback)

Desc: Brings up the device’s image picker and returns the image picked by the user.

Returns: Nothing

Arguments:

Context – on Android, the application context. On iOS, the parent view

controller.

Callback – The callback that will be invoked on completion.

#### ImagePickEventArgs

The arguments returned when picking an image.

Members:

Bool result – True if an image was obtained or the user cancelled the operation. False if there was an actual error.

Object image – On Android, the path to the image picked. On iOS, the actual image as a UIImage.

## Profiler

Provides basic cpu time profiling.

Usage:

Namespace: Rock.Mobile

Singleton: Profiler.Instance

Methods:

Void Start(string name)

Desc: Begins a timing operation.

Returns: Nothing

Arguments:

Name: The name of the sample. (Completely arbitrary)

Float Stop(string name, bool printResult = true)

Desc: Stops the sample with ‘name’ provided in Start

Returns: The time of the sample in milliseconds.

Arguments:

Name: The name of the sample provided when Start() was called

printResult: If true, (the default) will print the sample name and time into the

application’s console window.

**Note:** There is a small overhead associated with the sampling itself that can cause the parent of nested samples to take longer than they actually do. This is increased if printing is enabled.

Example:

Start( “SomeTask” ) – 0ms elapsed

Stop( “SomeTask” ) – 5ms elapsed

//Book keeping occurs after “SomeTask” is stopped, so if “SomeTask” has a

nested sample, the following could occur.

Start(“SomeTask”) – 0ms elapsed

Start(“ANOTHER TASK” ) – 1ms elapsed

Stop(“ANOTHER TASK”) – 2ms elapsed

// book keeping occurs within Stop() and may cost cpu cycles.

Stop(“SomeTask”) – 5.5ms elapsed due to the bookkeeping of sampling “ANOTHER TASK”

A good rule of thumb is to not rely on this profiling for EXACT timings, but rather for general timings.

In the above example, it would be acceptable to assume “SomeTask” takes 5-6 ms. It would be a mistake to assume it takes exactly 5.5ms and attempt to optimize based on that.

## Threading

Provides methods/classes associated with threading.

### UIThreading

Provides a method for executing code on the main / UI thread.

Usage:

Namespace: Rock.Mobile.Threading

Methods:

Static void PerformOnUIThread( ThreadTask task )

Desc: Executes “task” on the main thread.

Returns: Nothing

Arguments:

ThreadTask: A delegate that contains work.