Rock.Mobile

The purpose of this document is to detail the APIs that have 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

[Rock.Mobile 2](#_Toc418761842)

[Animation 2](#_Toc418761843)

[SimpleAnimator\_Float 2](#_Toc418761844)

[SimpleAnimator\_RectF 3](#_Toc418761845)

[SimpleAnimator\_PointF 3](#_Toc418761846)

[SimpleAnimator\_SizeF 3](#_Toc418761847)

[SimpleAnimator\_Color 3](#_Toc418761848)

[CoreLocation 4](#_Toc418761849)

[iBeacon 4](#_Toc418761850)

[Graphics 5](#_Toc418761851)

[Util 6](#_Toc418761852)

[IO 6](#_Toc418761853)

[FileCache 6](#_Toc418761854)

[AssetConvert 7](#_Toc418761855)

[Math 7](#_Toc418761856)

[Util 7](#_Toc418761857)

[PlatformMedia 8](#_Toc418761858)

[PlatformCamera 8](#_Toc418761859)

[PlatformImagePicker 8](#_Toc418761860)

[Network 9](#_Toc418761861)

[HttpRequest 9](#_Toc418761862)

[Util 10](#_Toc418761863)

[PlatformSpecific 10](#_Toc418761864)

[Android 10](#_Toc418761865)

[Util 10](#_Toc418761866)

[FontManager 11](#_Toc418761867)

[MaskLayer 11](#_Toc418761868)

[CircleView 12](#_Toc418761869)

[AspectScaledImageView 12](#_Toc418761870)

[Core 13](#_Toc418761871)

[ListAdapter 13](#_Toc418761872)

[public abstract class ListItemView 13](#_Toc418761873)

[WebLayout 13](#_Toc418761874)

[NotificationBillboard 14](#_Toc418761875)

[Util 15](#_Toc418761876)

[iOS 15](#_Toc418761877)

[FontManager 15](#_Toc418761878)

[Util 16](#_Toc418761879)

[WebLayout 16](#_Toc418761880)

[NotificationBillboard 17](#_Toc418761881)

[PhoneNumberFormatterDelegate 17](#_Toc418761882)

[PickerAdjustManager 18](#_Toc418761883)

[KeyboardAdjustManager 19](#_Toc418761884)

[UIScrollViewWrapper 19](#_Toc418761885)

[Util 20](#_Toc418761886)

[DateTimeExtensions 20](#_Toc418761887)

[CGObjectExtensions 20](#_Toc418761888)

[Profiler 20](#_Toc418761889)

[Threading 21](#_Toc418761890)

[UIThreading 21](#_Toc418761891)

[Strings 22](#_Toc418761892)

[Parsers 22](#_Toc418761893)

[StringExtensions 22](#_Toc418761894)

## Rock.Mobile

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

The goal of Rock.Mobile 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.

## Animation

Animation provides simple linear animators for common types, such as float, RectF, PointF, etc. Given a start and end value, the animator will interpolate across the range over a given time. Optional callbacks can be added for tick and completion.

Usage:

Namespace: Rock.Mobile.Animation

Classes:

### SimpleAnimator\_Float

Methods:

SimpleAnimator\_Float( **float** start,

**float** end,

**float** duration,

**AnimationUpdate** update,

**AnimationComplete** complete)

Constructor used to initialize the animator.

void Start( )

Called to begin the animator. If delegates were provided to the constructor, they will be called as the animation progresses.

### SimpleAnimator\_RectF

Methods:

SimpleAnimator\_RectF( **RectangleF** start,

**RectangleF** end,

**float** duration,

**AnimationUpdate** update,

**AnimationComplete** complete)

Constructor used to initialize the animator.

void Start( )

Called to begin the animator. If delegates were provided to the constructor, they will be called as the animation progresses.

### SimpleAnimator\_PointF

Methods:

SimpleAnimator\_PointF( **PointF** start,

**PointF** end,

**float** duration,

**AnimationUpdate** update,

**AnimationComplete** complete)

Constructor used to initialize the animator.

void Start( )

Called to begin the animator. If delegates were provided to the constructor, they will be called as the animation progresses.

### SimpleAnimator\_SizeF

Methods:

SimpleAnimator\_SizeF( **SizeF** start,

**SizeF** end,

**float** duration,

**AnimationUpdate** update,

**AnimationComplete** complete)

Constructor used to initialize the animator.

void Start( )

Called to begin the animator. If delegates were provided to the constructor, they will be called as the animation progresses.

### SimpleAnimator\_Color

Methods:

SimpleAnimator\_Color( **uint** start,

**uint** end,

**float** duration,

**AnimationUpdate** update,

**AnimationComplete** complete)

Constructor used to initialize the animator.

void Start( )

Called to begin the animator. If delegates were provided to the constructor, they will be called as the animation progresses.

**Note:** Color is expected in the **uint** format 0xRRGGBBAA. (The format used by Rock.Mobile.UI). It is important to use this animator when animating color, as it handles each channel correctly without overflow issues.

## 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.

## Graphics

Provides a utility class for common functions related to graphics.

Usage:

Namespace: Rock.Mobile.Graphics

Classes:

### Util

Methods:

**uint** ScaleRGBAColor( **uint** color,

**uint** scale,

**bool** scaleAlpha )

Given a color in the **uint** format 0xRRGGBBAA, the color will be “scaled” based on the scale provided. If **scaleAlpha** is true, the alpha will be scaled as well. If false, it won’t be.

**float** UnitToPx( **float** unit )

Given a point in unit-space, this will return the value as pixels. Useful for Android devices that have varying pixel densities.

## IO

Provides classes for managing IO on devices.

Usage:

Namespace: Rock.Mobile.IO

Classes:

### FileCache

Manages saving files of any type to the device, with an optional expiration date for cleanup purposes.

Methods:

**FileCache** Instance( )

Returns a static singleton FileCache object.

**void** SaveCacheMap( )

Saves the database that tracks what files are cached. Call before your app exits.

**void** CleanUp( **bool** forceEraseAll = false )

Runs a cleanup, removing any expired files. Called at startup by the FileCache, and can be optionally called by your app.

Arguments:

**bool** forceEraseAll – If true, all files, regardless of age, will be removed.

**void** SaveFile( **object** buffer,

**string** filename,

**TimeSpan?** expirationTime = null )

Stores a file in the FileCache.

Arguments:

**object** buffer – Reference to the data to be cached.

**string** filename – The name to give the file being cached.

**TimeSpan** expirationTime – The time **delta time** from **DateTime.Now** to keep the file.

Ex: If the file should last for one day, this would be new TimeSpan( 0, 1, 0 )

**bool** FileExists( string filename )

Tests whether a particular file exists in the cache.

**void** RemoveFile( **string** filename )

Removes a file with the given filename.

**void** DownloadFileToCache( **string** downloadUrl,

**string** cachedFileName,

**ImageDownload** callback )

Downloads and caches a file to the FileCache, and calls **callback** on completion.

Arguments:

**string** downloadUrl – The fully qualified URL of the file to download.

**string** cachedFileName – The name to be given the cached file.

**ImageDownload** callback – The completion callback to invoke when the file is finished

downloading.

### AssetConvert

Takes a bundled asset (like a .png file), opens it and converts it to a c# memory stream.

Usage:

Rock.Mobile.IO

Methods:

**MemoryStream** AssetToStream( **string** assetPath )

Returns a **MemoryStream** representing the asset loaded from **assetPath.**

## Math

Math provides helper functions for basic algebraic calculations.

Classes:

### Util

Provides helper functions for various math calculations.

Methods:

**float** DegToRad – Converts degrees to radians.

**float** DotProduct( **PointF** v1,

**PointF** v2 )

Performs a 2D dot product.

**float** MagnitudeSquared( **PointF** v )

Returns the squared magnitude of the given 2D vector.

**float** Magnitude( **PointF** v )

Returns the magnitude of the given 2D vector.

## PlatformMedia

Provides platform abstracted access to various media components.

Classes:

### 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:

**object** 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.

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

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

#### 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)

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

Arguments:

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

controller.

**ImagePickEvent** 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.

## Network

Network provides platform abstracted reusable network components.

Usage:

Rock.Mobile.Network

Classes:

### HttpRequest

This provides quick access for retrieving data from http sockets.

Methods:

**void** ExecuteAsync<TModel>( **string** requestUrl,

**RestRequest** request,

**RequestResult**<TModel> resultHandler )

Asynchronously makes a REST request, invoking **resultHandler** on completion. Expects TModel to be the argument passed to or received from the REST call.

Arguments:

**string** requestUrl - The URL to request.

**RestRequest** request – The **RestRequest** defining the type of request.

**RequestResult** resultHandler – The handler called when the request is complete.

**void** ExecuteAsync( **string** requestUrl,

**RestRequest** request,

**RequestResult** resultHandler )

Asynchronously makes a REST request, invoking **resultHandler** on completion and returning no body.

Arguments:

**string** requestUrl - The URL to request.

**RestRequest** request – The **RestRequest** defining the type of request.

**RequestResult** resultHandler – The handler called when the request is complete.

**void** ExecuteAsync( **string** requestUrl,

**RestRequest** request,

**RequestResult**<byte[]> resultHandler )

Asynchronously makes a REST request, invoking **resultHandler** on completion and returning the raw byte-stream body (no parsing into a known type).

Arguments:

**string** requestUrl - The URL to request.

**RestRequest** request – The **RestRequest** defining the type of request.

**RequestResult** resultHandler – The handler called when the request is complete.

**Note:** Use this when needing to make a “normal” or non-REST style call. You can then cast the resulting byte-stream to whatever type necessary.

### Util

Provides convenience methods related to network operations.

Methods:

**bool** StatusInSuccessRange( **HttpStatusCode** code )

Returns true if the **HttpStatusCode** falls within a known success range. (200 – 299)

## PlatformSpecific

Provides various functionality for specific platforms. This code is not cross-platform compatible.

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

## Android

### Util

Classes:

### AsyncLoader

Provides functionality for loading files on a background thread.

Usage:

Namespace: Rock.Mobile.PlatformSpecific.Android.Util

Methods:

**void** LoadImage(**string** imageName, **bool** bundled, **bool** scaleForDpi, **OnLoaded** onLoaded)

Loads an image on a worker thread, and then calls **onLoaded** on the UI/Main thread.

### Graphics

Classes:

### Util

Usage:

Namespace: Rock.Mobile.PlatformSpecific.Android.Graphics

Methods:

**Bitmap** ApplyMaskToBitmap( **Bitmap** image,

**Bitmap** mask,

int x,

int y )

Returns a new **Bitmap** that contains **image** masked by **mask**.

Arguments:

**Bitmap** image – The source image to place the mask over.

**Bitmap** mask – The mask to apply to **image**.

**int** x – The x offset for the mask.

**int** y – The y offset for the mask.

### FontManager

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.PlatformSpecific.Graphics

Singleton: FontManager.Instance

Methods:

**Typeface** GetFont( **string** fontName )

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

Arguments:

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

### MaskLayer

Implements a view that acts as a mask, allowing a square region to be masked and the outside area to be darkened by opacity. (Like an image cropper with a dark area for what will be cropped.)

Usage:

Rock.Mobile.PlatformSpecific.Android.Graphics

Methods:

MaskLayer( **int** layerWidth,

**int** layerHeight,

**int** maskWidth,

**int** maskHeight,

**Context** context )

Constructor that creates a Mask Layer.

Arguments:

**int** layerWidth **–** Thefull width of the layer. (So if the mask should cover a full image, it would be the image width.)

**int** layerHeight – The full height of the layer.

**int** maskWidth – The width of the area to mask. (So if you were displaying an image cropper, this might be the width of the area that won’t be cropped.)

**int** maskHeight – The heght of the area to mask.

**PointF** Position – The position of the mask within the layer.

**int** Opacity – The opacity of the unmasked region.

### CircleView

Implements an Android view like any other, except as a circle.

Usage:

Rock.Mobile.PlatformSpecific.Android.Graphics

Methods:

**float** StrokeWidth – The width of the circle outline. (If its center should be clear.)

**Color** Color – The color of the circle (including the outline.)

**Paint.Style** – The **Android** paint style.

CircleView( **Context** c )

Constructor for the CircleView.

### AspectScaledImageView

An image view that can be resized according to its width and will maintain the correct height aspect ratio.

Usage:

Rock.Mobile.PlatformSpecific.Android.Graphics

Methods:

AspectScaledImageView( **Context** c )

Constructor for creating the view.

### Core

Provides common ‘core’ functionality for Android.

Usage:

Rock.Mobile.PlatformSpecific.Android

Methods:

**Android.Content.Context** Context – A reference to the application context. Set this as soon as your application starts up.

### ListAdapter

Because list item references generally aren't stored by a list adapter, properties of a list item can store references to objects like bitmaps that need to be

freed.

This ListAdapter tracks all added items and exposes methods for calling Destroy andallowing each list item to release any references its holding.

**Bottom line, when implementing a list in Android, derive the adapter from**

**this.**

Usage:

Rock.Mobile.PlatformSpecific.Android.UI

Methods:

ListAdapter( )

Constructor for the ListAdapter

**View** AddView( **ListItemView** newView )

Adds a **ListItemView** to the internal list of items that will need **Destroy** called.

**void** Destroy( )

Calls **Destroy** on all items added to the list. Call this method when your List will be unloaded.

### public abstract class ListItemView

All items that will go in ListAdapter must derive from this class. They should implement override **Destroy,** and clean up resources there.

**void** Destroy( )

Called by the ListAdapter, this is where list items should free resources.

### WebLayout

Implements a reusable embedded browser. Useful for in-app webpages.

Usage:

Rock.Mobile.PlatformSpecific.Android.UI

Methods:

WebLayout( **Context** c )

Constructor for the WebLayout

**void** ResetCookies( )

Removes cookies stored during this app’s web sessions.

**void** LoadUrl( **string** url,

**PageLoaded** loadedHandler )

Causes the webLayout to load the page provided by **url**.

Arguments:

**string** url – The URL to load.

**PageLoaded** loadedHandler – The callback to invoke when the page finishes loading. **This must not be null.**

### NotificationBillboard

A specialized view designed to slide out like a “toast” notification. Is tappable and can contain an icon and label.

Usage:

Rock.Mobile.PlatformSpecific.Android.UI

Methods:

NotificationBillboard( **float** deviceWidth,

**Context** context )

Constructor for creating a NotificationBillboard.

Arguments:

**float** deviceWidth – The width of the device. Important so the NotificationBillboard knows the ‘edge’ of the screen.

**Context** context – The application context.

**void** Reveal( )

Causes the **NotificationBillboard** to animate into visible range.

**void** Hide( )

Causes the **NotificationBillboard** to animate out of visible range.

**void** SetLabel( **string** iconStr,

**string** iconFont,

**float** iconSize,

**string** labelStr,

**string** labelFont,

**string** labelSize,

**uint** textColor,

**uint** bgColor,

**EventHandler** onClick )

Sets the appearance of the Notification Billboard.

### Util

Provides common UI related functions for Android.

Usage:

Rock.Mobile.PlatformSpecific.Android.UI

Methods:

**void** AnimateViewColor( **uint** currColor,

**uint** targetColor,

**View** uiView,

**SimpleAnimator.AnimationComplete** complete )

Animates a UIView’s background color from curr to target color. Calls **complete** when finished.

**void** FadeView( **View** view,

**bool** fadeIn,

**SimpleAnimator.AnimationComplete** complete )

Animates a UIView’s alpha either in or out. Calls **complete** when finished.

## iOS

Usage:

Rock.Mobile.PlatformSpecific.iOS.Graphics

Classes:

### FontManager

Loads and caches fonts for iOS. Important for efficiency so fonts can be cached rather than loaded multiple times.

Methods:

**UIFont** LoadFontDynamic( **string** name,

**float** fontSize )

Loads a font from the application’s asset bundle.

Arguments:

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

**Float f**ontSize – The point size to load for the font.

### Util

Provides common UI utility functions for iOS.

Usage:

Rock.Mobile.PlatformSpecific.iOS.UI

Methods:

**void** AnimateViewColor( **uint** targetColor,

**UIView** uiView )

Animates a **UIView**’s background color from its current color to **targetColor**.

Arguments:

**uint** targetColor – The color to animate **uiView** to.

**UIView** uiView – The UIView to animate.

### WebLayout

Provides an embedded webView with progress indication and the ability to cancel.

Usage:

Rock.Mobile.PlatformSpecific.UI

Members:

WebLayout( **CGRect** frame )

Constructor for webLayout, accepts a frame with the dimensions it should use.

**void** LayoutChanged( **CGRect** containerBounds )

Notifies **WebLayout** that its parent dimensions have changed.

Allows it to re-layout itself to fit the parent appropriately.

Arguments:

**CGRect** containerBounds – The bounds of the **parent** to weblayout.

**void** LoadUrl( **string** url,

**LoadResult** resultHandler )

Causes webLayout to load a page given by **url**.

Arguments:

**string** url – The url of the content to load.

**LoadResult** resultHandler – The callback to invoke when the page is finished loading or errors. **resultHandler** may be null.

**void** DeleteCacheAndCookies( )

Deletes cookies and cache stored during the web session.

**void** SetCancelButtonColor( **uint** color )

Sets the color of the cancel button text.

### NotificationBillboard

Provides a billboard that is revealed like a notification ‘toast’ from the screen edge.

Allows a user to tap it for navigation, and offers an icon and text.

Usage:

Rock.Mobile.PlatformSpecific.iOS.UI

Methods:

NotificationBillboard( **nfloat** displayWidth,

**nfloat** displayHeight )

Constructor. Accepts the dimensions that NotificationBillboard should be.

**void** SetLabel( **string** iconStr,

**string** iconFont,

**float** iconSize,

**string** labelStr,

**string** labelFont,

**float** labelSize,

**uint** textColor,

**uint** bgColor,

**EventHandler** onClick )

Sets the appearance of the Notification Billboard. Additionally sets the callback to use when tapped.

Arguments:

**string** iconStr – The icon to use.

**string** iconFont – The name of the font to use for the **icon**.

**float** iconSize – The size in points of the **icon**.

**string** labelStr – The label to use.

**string** labelFont – The name of the font to use for the **label**.

**float** labelSize – The size in points of the **label**.

**uint** textColor – The color to use for both the **label** and **icon**.

**uint** bgColor – The color to use for the background of the Notification Billboard.

**EventHandler** onClick – The callback to invoke when tapped.

### PhoneNumberFormatterDelegate

When attached as the delegate of a **UITextField**, will parse input text to match appropriate **US** phone numbers.

Usage:

Rock.Mobile.PlatformSpecific.iOS.UI

**Note**: Simply attach as the delegate to any **UITextField**.

Methods:

No methods are necessary to use. All public methods are the result of necessary overrides.

### PickerAdjustManager

A helper class that manages adjusting the screen to allow a **UIPicker** to be visible.

Provides a header label and animation.

Usage:

Rock.Mobile.PlatformSpecific.iOS.UI

**Note:** Instantiate and pass in the parent view and child scroll view

(Hierarchy must be View->UIScrollView->Whatever)  
Additionally, create a label and set its styling and text ONLY.

This is used as the "header text" for the picker.

Lastly pass in the target view that should be in focus, and the model for your picker.  
To reveal, call **TogglePicker**

Methods:

PickerAdjustManager( **UIView** parentView,

**UIScrollView** parentScrollView,

**UILabel** pickerLabel,

**UIView** targetView )

Constructor.

Arguments:

**UIView** parentView – The parent view.

**UIScrollView** parentScrollView – The parent scroll view that the PickerAdjustManager should attach itself to.

**UILabel** pickerLabel – The label to display as the header for the picker.

**UIView** targetView – The **UIView** that should be visible while the Picker is open.

**void** SetPicker( **UIView** picker )

Provides a reference to the picker that should be displayed. (The PickerAdjustManager does not create one itself.)

**void** LayoutChanged( )

Should be called when the layout of the parent has changed.

**void** TogglePicker( **bool** enabled,

**bool** animate = true )

Hides or unhides the picker, either with or without animation.

### KeyboardAdjustManager

Utility class that makes sure a text field being edited is in view and not obstructed by the software keyboard.

Usage:

Rock.Mobile.PlatformSpecific.iOS.UI

**Note:** Instantiate and pass in the parent view and child scroll view

(Hierarchy must be View->UIScrollView->Whatever)

The KeyboardAdjustManager is event driven. This means that it needs to know

when text is being changed in the **UITextView.**

The **UITextView** of interest should have its **delegate** set to

**KeyboardAdjustManager.TextViewDelegate**. (You may override this and call the base if needed.)

The **KeyboardAdjustManager** will now send four notifications, and expects four equivalent methods to be called on itself. It cannot handle this internally, because if **you** need to perform code on these events, you wouldn’t be able to.

The basic code is as follows:

NSNotificationCenter.DefaultCenter.AddObserver (

KeyboardAdjustManager.TextFieldDidBeginEditingNotification,

KeyboardAdjustManager.OnTextFieldDidBeginEditing);  
        
NSNotificationCenter.DefaultCenter.AddObserver (

KeyboardAdjustManager.TextFieldChangedNotification,

KeyboardAdjustManager.OnTextFieldChanged );

NSNotificationCenter.DefaultCenter.AddObserver (

UIKeyboard.WillHideNotification,

KeyboardAdjustManager.OnKeyboardNotification);  
  
NSNotificationCenter.DefaultCenter.AddObserver (

UIKeyboard.WillShowNotification,

KeyboardAdjustManager.OnKeyboardNotification);

If you need to perform functionality, simply replace the Notification method call with your own, and in that, call the KeyboardAdjustManager’s corresponding method.

## UIScrollViewWrapper

Provides a wrapper class to **UIScrollView** that forwards touch input to a parent, rather than suppressing it.

Usage:

Rock.Mobile.PlatformSpecific.iOS

Methods:

**UIViewController** Parent

The parent that input should be forwarded to.

## Util

Provides utility methods specific to iOS.

### DateTimeExtensions

Extensions to **DateTime** and **NSDate** to allow conversion back and forth.

Usage:

Rock.Mobile.PlatformSpecific.iOS.Util

Methods:

**DateTime** NSDateToDateTime( )

Converts an **NSDate** to **DateTime**

**NSDate** DateTimeToNSDate( )

Converts a **DateTime** to **NSDate**

### CGObjectExtensions

Provides conversion methods for CGObjects to System.Drawing objects.

Usage:

Rock.Mobile.PlatformSpecific.iOS.Util

Methods:

**System.Drawing.PointF** ToPointF( )

Converts a **CGPoint** to a **PointF**.

**System.Drawing.SizeF** ToSizeF( )

Converts a **CGSize** to a **SizeF**.

**System.Drawing.RectangleF** ToRectF( )

Converts a **CGRect** to a **RectangleF**.

## Profiler

Provides basic cpu time profiling.

Usage:

Namespace: Rock.Mobile

Singleton: Profiler.Instance

Methods:

**void** Start(**string** name)

Begins a timing operation.

Arguments:

**string** name: The name of the sample. (Completely arbitrary)

**float** Stop(**string** name,

**bool** printResult = true)

Stops the **sample** with ‘**name**’ provided in Start. Returns the **time** of the **sample** in **milliseconds**.

Arguments:

**string** name: The name of the sample provided when **Start**() was called.

**bool** 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.

Classes:

### Util

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

Usage:

Namespace: Rock.Mobile.Threading

Methods:

**void** PerformOnUIThread( **ThreadTask** task )

Executes “task” on the main thread.

Arguments:

ThreadTask: A delegate that contains work.

**void** PerformOnWorkerThread( **System.Threading.ThreadStart** task )

Executes “task” on a background worker thread.

Arguments:

ThreadTask: A delegate that contains work.

## Strings

Provides extension methods and utility functions for the **C# String** class.

### Parsers

Provides utility functions for various forms of string parsing.

Usage:

Rock.Mobile.Util.Strings

Methods:

**void** ParseAddress( **string** address,

**ref string** street,

**ref string** city,

**ref string** state,

**ref string** zip )

Given an address formatted “street, city, state, zip” The broken components will be returned.

**string** ParseURLToFileName( **string** url )

Takes a URL assumed to reference a filename (e.g. [www.contoso.com/file.txt)](http://www.contoso.com/file.txt)) and returns the filename portion. (e.g. file.txt)

### StringExtensions

Provides extension methods to **C# String**

Usage:

Rock.Mobile.Util.Strings

Methods:

**string** AsNumeric( )

Returns a new string stripped of the source’s non-numeric characters.

(e.g. “(111)-222-3333” would return as “1112223333”

**bool** IsEmailFormat( )

Returns true if the string passes **basic** email validation checks. ([x@x.com)](mailto:x@x.com)) would be acceptable, tho probably not valid.

**string** AsPhoneNumber( )

Returns a new string formatted to a proper US phone number format.

(e.g. “1112222” would be returned as “111-2222” and “1112223333” would return as “(111) 222-3333”