RedLaser SDK for iOS
version 3.5.3

Welcome to version 3.5.3 of the RedLaser SDK for iOS. This version includes support for querying user video camera permissions introduced in iOS 7 and required in iOS 8.

We will be also be maintaining a copy of these documents on the redlaser.com website, so be sure to check there for new updates.

Table Of Contents

Adding the RedLaser SDK to your Project

The Developer SDK for RedLaser includes a Sample app that shows how the SDK can be used. It is intended to be a simple demonstration of the SDK's capabilities, and shows a somewhat simple way to use the SDK from an app.

The source code in the Sample app itself should not be confused with the SDK. You are free to copy the code in the Sample app if you wish, but you're also free to use the SDK in a way different than what the Sample app does.

Required Files

To add the SDK to your app's project, you need to add 2 files to your target:

* RedLaserSDK.h
* libRedLaserSDK.a

If you are a licensee, and don't want to use the SDK in evaluation mode, you'll also need to add:

* RedLaser_License.xml

The "RedLaser_License.xml" file needs to be generated for your app from the redlaser.com website. See the section on licensing for details.

Optional Files

If you want to build a 64-bit binary, you'll need to use:

* RedLaserSDK_arm64.a

This file includes armv7, armv7s, arm64, and i386 slices. The Deployment Target for this build is iOS 5.1, the minimum OS version that is (as of Xcode 5.02) compatible with apps that contain a 64 bit slice. Note that this doesn't involve using the 64 bit slice, as no devices with 64 bit processors can run on OS versions before iOS 7.

The SDK package contains a set of image and sound assets that your application may use as part of the app-supplied overlay. These files should make it easier to create an overlay similar to the one used by the RedLaser app. The SDK itself does not use these files.

As a separate issue, showing the "powered_by_redlaser.png" image to the user when scanning is generally a requirement of the SDK license; from a technical standpoint, you can rename the image file, or make it a .jpg, or incorporate it into a larger overlay graphic. The SDK doesn't attempt to access it.

Frameworks

The RedLaser library itself depends on the following Apple frameworks:

- UIKit.framework
- Foundation.framework
- AdSupport.framework
- CoreGraphics.framework
- CoreMedia.framework
- CoreVideo.framework
- AVFoundation.framework
- CoreLocation.framework
- MobileCoreServices.framework
- OpenGLES.framework
- Security.framework
- libstdc++.dylib
- libiconv.dylib
- libz.dylib

Depending on how you set up your project, the standard c++ library may or not be included by default.

Very Important Note for Developers Writing iOS 7 Only apps

If you set your app's "Deployment Target" to iOS 7, Xcode will helpfully link your app against C++11's libc ++.dylib, instead of libstdc++.dylib. This will cause your app to fail to link because the RedLaser SDK needs to link against libstdc++.dylib, due to ABI incompatibilities between library versions. The solution is to add in the path to libstdc++.dylib manually as an "Other Linker Flags" value. The line below will do the trick:

OTHER_LDFLAGS = /usr/lib/libstdc++.dylib

The good news is that (in our case, anyway) having both libs in the same app is OK. Since the SDK's usage of C++ library calls is entirely internal and our API is Objective-C, we can operate alongside code that uses the new libc++.

Linking

The minimum OS version for the RedLaser SDK is 4.3. Earlier versions of the SDK had a lower minimum OS version; the minimum has been raised with SDK 3.3.2 to match the minimums imposed with Xcode 4.5 and iOS SDK 6.0.

Applications that wish to run on iOS versions prior to 4.0 and still use RedLaser features on 4.0 and above can still do so. Developers should be able to weak link against the appropriate OS frameworks, and have their apps load on iOS 3.x. We've provided a

convenience function, RL_CheckReadyStatus(), that checks whether the frameworks required by RedLaser are available. If this function returns RLState_MissingOSLibraries, your app should not attempt to instantiate an instance of BarcodePickerController. It will almost certainly crash your app.

Deprecation Notice: In the near future, the SDK's minimum OS requirement will be increased, probably to 5.0. When that happens, RL_CheckReadyStatus will return RLState_MissingOSLibraries on pre-5.0 devices.

Hardware Considerations

The RL_CheckReadyStatus function also checks for the existence of a camera on the device. If no camera is found, this function returns RLState_NoCamera, and you shouldn't use RedLaser (although it won't crash if you attempt it, it doesn't do anything useful).

The SDK can be used in the iPhone Simulator, but cannot scan actual barcodes. Even if your computer has a webcam, it isn't hooked up to the simulator's AVFoundation framework. When running in the Simulator, the SDK's view will show a UIButton that 'fakes' recognizing a barcode when clicked.

As of SDK version 3.3.2, the iPhone 3G is no longer supported. The SDK no longer ships with a armv6 binary.

Licensing

The RedLaser SDK can be used with or without a license file. With no license file, the SDK runs in evaluation mode, and limits the number of scans per device. The limitations imposed when a license file is present depend on your licensing terms.

When you signed up as a developer, you generated a license file named "RedLaser_License.xml". You will need add this file to your project. The license file needs to be in the application's root bundle, such that it can be loaded with:

```
[[NSBundle mainBundle] pathForResource:@"RedLaser_License" ofType:@"xml"];
```

The license file is digitally signed. Do not modify it. This includes modifying line endings, which some source control systems like to do. The files are currently being generated with Unix-style LF line endings, although it's possible that could change; the correct line ending style is 'what it was signed with'.

Your license file has a list of all the application bundle IDs you've registered with RedLaser.com. The SDK will only work correctly in licensed mode if it's being used in one of the applications you've registered.

Using the RedLaser SDK in your App

This section goes through the "RedLaserSDK.h" header file one section at a time, explaining each section as we go.

Before You Use the SDK

The RedLaser SDK requires an iOS device with a camera. It requires iOS version 4.3 to run, but must be built with iOS SDK 6.0 or above. It has certain licensing requirements, and will refuse to run if these aren't met.

Before using the SDK, your app should call RL_CheckReadyStatus(). If this function returns a negative value, your app should not attempt to use the SDK (most likely, your app should disable the button that opens the SDK's view controller).

The SDK will show alerts to the user if it's called but cannot scan--for example, if it's opened on a device that has no camera. These alerts, however, are a fallback position--they're not localized, and may not provide the user with the best experience. Therefore, it's better for your app to use this function and take appropriate action beforehand.

The value returned from this function could change within an application run if the underlying state changes--for example, if you're using the SDK in evaluation mode and you hit the maximum scan count for a device.

The MissingOSLibraries result should only occur if you've weak-linked iOS libraries that RedLaser needs to function, and those libraries aren't available at run time. To use the RedLaser SDK in apps with a minimum OS version below 6.0, you should weak link the iOS 6 AdSupport.framework--the SDK will work fine on devices where it isn't available. Note that the SDK does not display ads, ever. We use the AdSupportFramework to collect the advertisingIdentifier value, which is sort of a replacement for iOS's deprecated deviceIdentifier. The purpose of our collecting this information is for verifying licensing of the SDK.

The RLState_NoVideoAuthorization result implies that the user has not yet opted in to enable video camera permissions for your application. Using the SDK scanner at this point leads to undefined behavior. The recommended action for RLState_NoVideoAuthorization is to call the new RL_RequestVideoAuthorization function. This function returns immediately and fires a completion handler block with the state of the user permissions. This completion handler fires on an arbitrary queue that is not necessarily the main queue. The SDK will call your completion handler with RL_VideoAuthorizationStatusAuthorized to indicate that the user has granted permission to access the video camera. See RLSampleViewControllerRequestVideoAuthorization from the RLSample project for an example of this function in action.

Finally, RL_CheckReadyStatus() can return RLState_NoKeychainAccess. The RedLaser SDK generates a UUID and stores it in the keychain for licensing purposes. If for some reason your application cannot access its own default keychain this result will be returned. Generally this either means there is an issue with the entitlements file (which sometimes happens due to code re-signing), or the keychain may be corrupt.

Supported Barcode Types

```
#define kBarcodeTypeEAN13                   0x1
#define kBarcodeTypeUPCE                    0x2
#define kBarcodeTypeEAN8                    0x4
#define kBarcodeTypeQRCODE                          0x10
#define kBarcodeTypeCODE128                         0x20
#define kBarcodeTypeCODE39                  0x40
#define kBarcodeTypeDATAMATRIX              0x80
#define kBarcodeTypeITF                             0x100
#define kBarcodeTypeEAN5                    0x200
#define kBarcodeTypeEAN2                    0x400
#define kBarcodeTypeCodabar                 0x800
#define kBarcodeTypeCODE93                  0x1000
#define kBarcodeTypePDF417                  0x2000
#define kBarcodeTypeGS1Databar                      0x4000
#define kBarcodeTypeGS1DatabarExpanded              0x8000
```

If you're new to working with barcodes, there's a good chance you're looking to scan UPC codes, and are about to choose UPCE. You probably want to look at EAN13 instead. EAN13 is a newer standard that is a strict superset of the UPC-A standard. You might want to scan UPC-E barcodes as well, just realize that UPC-E barcodes are the 6-digit shortened barcodes found on some products, not the 12 digit codes which are more common.

Although DATAMATRIX is in the list, it doesn't work very well (yet) and is not officially supported at this time.

EAN5 and EAN2 are codes that can appear next to an EAN13 code, and specify pricing information or issue numbers for certain products. These barcodes are 'associated codes' that are only searched for when a EAN13 is found. If you enable EAN13 or EAN8, you can enable EAN5 and/or EAN2 as well, and information about the associated codes will be returned to you. Enabling only EAN5 or EAN2 will not produce any results.

BarcodeResult class

```
/**********************************************************************************
   BarcodeResult
```

```
    The return type of the recognizer is a NSSet of BarcodeResult objects.
*/

@interface BarcodeResult : NSObject { }

@property (readonly) int barcodeType;
@property (readonly) NSString *barcodeString;
@property (readonly, copy) NSString *extendedBarcodeString;
@property (readonly) BarcodeResult *associatedBarcode;

@property (readonly, retain) NSDate *firstScanTime;
@property (readonly, retain) NSDate *mostRecentScanTime;
@property (readonly, retain) NSMutableArray *barcodeLocation;
@end
```

The SDK delivers a NSSet of these objects to your application, one for each barcode that was recognized during the scanning session. Why does the SDK return sets of codes? To handle situations like this:

The barcodeType property will be one of the values from the list above, and barcodeString will be the value read from the barcode. ExtendedBarcodeString, if non-nil, contains extra information about the barcode. The “Notes on Specific Barcode Types” section of this document details what this field contains for various barcode types.

In the case where the SDK produced a EAN with an associated EAN5 or EAN2 code, both codes are included as separate BarcodeResults in the returned result set, and each of them will have their associatedBarcode set to the other.

Each barcode in the set will have 2 NSDates: the first time it was recognized during the session, and the last time it was recognized during the session.

Finally, each barcode will have a NSArray of NSValues, where each NSValue is a CGPoint, indicating where we located the barcode. The coordinates of the points will be in the same coordinate system as the BarcodePickerController's bounds. The first point in the array will be the top left of the barcode, and the second will be the top right of the barcode. Note that if a barcode is recognized 'upside down', these points will be in the lower right and lower left when viewed onscreen. Also, because the preview is mirrored when using a device's front camera for recognizing, the points aren't necessarily in clockwise winding order either. The array will usually contain 4 points, but it could contain more or fewer.

The path produced from these points may not cover the entire barcode, and may be only one pixel high or wide. The barcode location is only updated on frames where the barcode is actually recognized, so the longer it's been since mostRecentScanTime, the less likely it is that the barcode is still at that position in the camera preview. Barcodes recognized by the partial recognition method (used for some long barcodes, allowing the user to point the camera at each part of the barcode and piece the full code together) will only have recognition information on the most recent part of the barcode to be scanned.

FindBarcodesInUIImage

This method analyses a given image and returns information on any barcodes discovered in the image. It is intended to be used in cases where the user already has a picture of a barcode (in their photos library, for example) that they want to decode. This method performs a thorough check for all barcode symbologies we support, and is not intended for real-time use.

When scanning barcodes using this method, you cannot (and need not) specify a scan orientation or active scan region; the entire image is scanned in all orientations. Nor can you restrict the scan to particular symbol types. If such a feature is absolutely necessary, you can implement it post-scan by filtering the result set.

FindBarcodesInUIImage operates synchronously, but can be placed in a thread. Depending on image size and processor speed, it can take several seconds to process an image.

BarcodePickerControllerDelegate

```
@protocol BarcodePickerControllerDelegate <NSObject>
@optional
- (void) barcodePickerController:(BarcodePickerController *) picker returnResults:(NSSet
*)results;
@end
```

This is the delegate object of the SDK that receives the results of a scan session after the scan session completes. Setting a delegate isn't absolutely required--your BarcodePickerController2 subclass (or your overlay controller if using the original BarcodePickerController) receives live information on barcodes while scanning. The delegate is a convenient method for getting results back to the caller of the scan session after it completes.

Your application should always check for multiple barcodes returned from a scan session. It is possible for multiple codes to get recognized at the same time if multiple barcodes are visible in the preview frame. If your app only wants to operate on a single barcode at a time and you receive two in this call, you should use meta-information about the barcodes such as the position within the frame or the initial scan time for each barcode to determine which of the barcodes is the one the user is most likely intending to scan.

If implemented in the delegate, the SDK will always call barcodePickerController:returnResults: exactly once per scan session. This will happen after doneScanning is called, or if the session encounters an error. This means that the delegate may be called with an empty set of barcodes.

Using the New BarcodePickerController2

The new BarcodePickerController2 class, introduced in version 3.5 of the RL SDK, is intended to be subclassed by applications, as opposed to the legacy BarcodePickerController class which acted as a container controller for an overlay view controller. This makes the design easier to understand, is easier to use with Storyboards, and streamlines the work of integrating RedLaser with your application.

This class works a bit differently than the legacy BarcodePickerController. The new class does not have settings to enable/disable individual barcode types. All barcodes are always on. Instead, the BarcodePickerController2 class has a new method whereby

the application can tell the SDK that a barcode the SDK has found is unwanted, or not the type of barcode the application is expecting.

The new unwanted barcode indicator

Marking a found barcode as “unwanted” by calling reportUnwantedBarcode: on it will cause the SDK to show a visual indicator that a given barcode isn’t the type your application is expecting. This provides an easy method for applications that don’t want to write a bunch of custom layer code to provide feedback to users that the barcode they’re trying to scan isn’t the the type the application is expecting.

Providing feedback to the user in this instance is incredibly important--users don’t know what different barcode types are and what they look like, and in our experience users would often point their device at a barcode type the application had disabled. After about a minute spent attempting to scan a barcode that the application specifically asked to not scan, the user gives up in frustration, cursing the barcode reader’s poor quality.

Using the new unwanted barcode functionality is completely optional. Application developers that want a more customized look are still able to implement their own augmented-reality style view, and your application’s BarcodePickerController2 subclass receives real time updates as to the onscreen location of found barcodes.

What’s Missing

Some of the properties that could be set on the original BarcodePickerController class have been removed for the BarcodePickerController2. We’ve already discussed the removal of the individual enable/disables for barcode types. The activeRegion and orientation properties have been removed as well. These were speed optimizations that were required to achieve good performance in the past, but devices getting faster there’s less reason to have them. Instead, the BarcodePickerController2 scans the entire camera view in all orientations. This may lead to slow scanning on older devices.

If the users of your app are skewed towards older devices you may want to consider (continuing) to use the legacy BarcodePickerController and using these properties to increase speed by constraining the work that must be done per scan cycle. At some point in the future, those older devices are going to become unsupported, and the legacy BarcodePickerController will likely be going away when that happens.

Creating your BarcodePickerController2 Subclass

The BarcodePickerController2 will insert an AVCaptureVideoPreviewLayer in front of the root view of your view hierarchy, and will size that layer to match the bounds of the root view. It will also insert a UIView as a child of the root view that it uses for displaying the unwanted barcode indicators.

This means that your root view will be occluded and won't be seen. Remember that any full-screen subview that has an opaque background will hide the preview layer. It also means that if you're using your own CALayers you might want to consider making a transparent full-size subview, and add your layers to that instead of the root view. Or, just be careful with layer order, as some of the layers of the root view aren't your own.

Controlling the Camera

The useFrontCamera property acts as a recommendation if TRUE; devices that do not have a front facing camera will still use the default camera.

TurnTorch: is used to turn the LED torch on or off. As a convenience, you can use hasTorch to check if a torch is available.

The exposureLock property can be used to lock the exposure level on devices that support it; this prevents the camera's autoexposure mechanism from operating.

CanFocus will be TRUE if the device is capable of focusing its camera. Similarly, isFocusing will be TRUE when the camera is in the middle of a focusing operation and FALSE otherwise. IsFocusing is designed to be observed with KVO.

The torch, exposure, and focus methods are designed to work with the selected camera--if you set useFrontFacingCamera to TRUE, hasTorch may start returning FALSE (but not if Apple releases a device with a front-facing flash). Also, while a session is running, the status methods also take into account whether we were able to obtain the device lock. Camera properties that modify the actual hardware can only be manipulated by one entity at a time, and if the SDK can't acquire the device lock the SDK will report FALSE for hasTorch even if there is a torch, since the SDK wouldn't be able to turn the torch on or off.

PrepareToScan is an optional method you can call to 'warm up' the device's camera before scanning. Calling this method several seconds before initiating a scan session should reduce the time it takes to start capturing frames. Since calling this method

causes the device's camera to be powered up, it does drain the device's battery somewhat. Because of this, if a scan session isn't started within a reasonable time period, the camera will be powered down. If you don't call this method at all, the camera will simply be powered up when you start your scanning session.

PauseScanning and resumeScanning may be used to temporarily halt image processing.

DoneScanning is used to end a scan session. Scan sessions do not end by themselves when a barcode is recognized; if you want this behavior you need to call doneScanning when statusUpdated: indicates that a code has been recognized.

Location Data

The startCollectingLocationData call has been created to support future server-side analytics as a service to SDK clients. By calling startCollectingLocationData, the SDK will attempt to use CoreLocation and add location information to its reported barcode scan packets. In addition to calling startCollectingLocationData in your application, the application must also ask the user for Core Location authorization. Otherwise, no location information will be passed.

NOTE: This feature is entirely opt-in. If you do not call this method, location data will not be collected or transmitted. The SDK will not cause the Core Location authorization dialog to appear.

Camera Snapshots

RequestCameraSnapshot: will cause the SDK to capture a UIImage of the camera view and return it in a future call to the overlay status method (see below). If the boolean argument to this method is TRUE, this method will take a full color picture at the camera's picture resolution and return that in the UIImage. If the argument is FALSE, a preview-sized UIImage will be returned. Creating UIImages of the camera view is much too slow to be done on every frame; use this method sparingly. Also, only one snapshot may be requested at a time.

Using the Legacy BarcodePickerController

The legacy BarcodePickerController is a subclass of the new BarcodePickerController2. The legacy BarcodePickerController is not intended to be subclassed; instead, the scan user interface is customized via a separate UIViewController which is a subclass of the SDK's CameraOverlayViewController.

The BarcodePickerController retains a bunch of properties that have been streamlined out of the BarcodePickerController2, such as on/off switches for finding individual barcode types and the activeRegion rectangle.

We recommend that developers writing new applications implement the SDK using the new BarcodePickerController2 instead of the BarcodePickerController.

BarcodePickerController

```
/**************************************************************************
        BarcodePickerController

        This ViewController subclass runs the RedLaser scanner, detects barcodes, and
        notifies its delegate of what it found.
*/
@interface BarcodePickerController : BarcodePickerController2

@property (nonatomic, retain) CameraOverlayViewController *overlay;

@property (nonatomic, assign) BOOL scanUPCE;
@property (nonatomic, assign) BOOL scanEAN8;
@property (nonatomic, assign) BOOL scanEAN13;
@property (nonatomic, assign) BOOL scanQRCODE;
@property (nonatomic, assign) BOOL scanCODE128;
@property (nonatomic, assign) BOOL scanCODE39;
@property (nonatomic, assign) BOOL scanDATAMATRIX;
@property (nonatomic, assign) BOOL scanITF;
@property (nonatomic, assign) BOOL scanEAN5;
@property (nonatomic, assign) BOOL scanEAN2;
@property (nonatomic, assign) BOOL scanCODABAR;
@property (nonatomic, assign) BOOL scanCODE93;
@property (nonatomic, assign) BOOL scanPDF417;
@property (nonatomic, assign) BOOL scanGS1Databar;
@property (nonatomic, assign) BOOL scanGS1DatabarExpanded;

@property (nonatomic, assign) CGRect activeRegion;
@property (nonatomic, assign) UIImageOrientation orientation;
@property (nonatomic, assign) BOOL torchState;

@end
```

In order to scan barcodes, the application should create an instance of BarcodePickerController, configure its properties, and then display it to the user. Here's some example code that does this:

```
- (void) startScan
{
        BarcodePickerController *picker = [[BarcodePickerController alloc] init];
        [picker setOverlay:customOverlay];
        [picker setDelegate:self];
```

```
        // Initialize with portrait mode as default
        picker.orientation = UIImageOrientationUp;

        // hide the status bar
        [[UIApplication sharedApplication] setStatusBarHidden:YES];

        // Show the scanner
        [self presentModalViewController:picker animated:TRUE];
        [picker release];
}
```

Note that your application is not limited to using presentModalViewController:animated: to display the picker view. Your application is, however, generally responsible for including some method of ending a scanning session by adding appropriate UI to the overlay.

The delegate and overlay properties should be set up before initiating scanning or displaying the BarcodePickerController. It *may* work to change them during scanning, but it is not a supported use case. If you really need this functionality, please give us feedback.

The other listed properties may be modified both before and during scanning. The scan* properties may be set to choose what barcode types the application is interested in.

Set the ‘orientation’ property to the *device-relative* orientation that users are most likely to hold barcodes up to the device–that is, which edge of the device the top edge of the barcode should be pointing at. When holding a device in portrait mode (with the home button at the bottom) setting UIImageOrientationLeft means barcodes should be oriented so their top edge points towards the left edge of the device–the edge with the volume controls for iPhone devices. The RedLaser SDK scans for barcodes in all 4 orientations, but it spends more processing power looking in the orientation you specify in this property. For best results with devices with poorer-quality cameras, you want to choose an orientation and guide the user (in your overlay) to hold barcodes up to the device in that orientation.

The activeRegion property can be used to improve scanning speed by reducing the search area to a rectangle you specify in the coordinate space of the BarcodePickerController’s bounds. If specified, the SDK will spend more processing power looking for barcodes in this area than other areas of the camera view. Barcodes may still be recognized outside of this area; don’t try to use this property to select between multiple on-screen barcodes. Instead, you should use the barcodeLocation property of returned barcodes to analyze where in the view they were found.

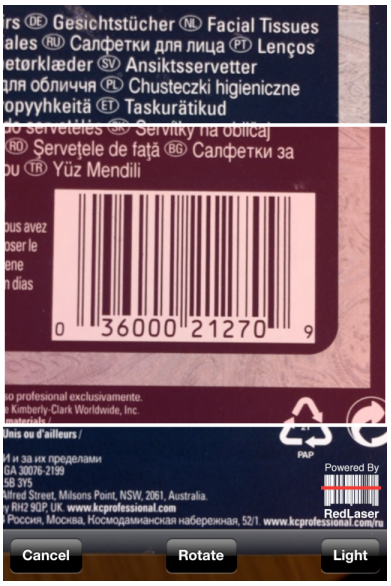
Designing Your Overlay

The CameraOverlayViewController is a subclass of UIViewController. You should design a subclass of CameraOverlayViewController in your app, and set the overlay property of

the BarcodePickerController to the overlay you create before showing the BarcodePickerController.

The CameraOverlayViewController should implement the barcodePickerController:statusUpdated: method, to receive regular status updates from the SDK. You can use these status updates to make your UI responsive to the state of the scan session.

The Sample app (left image, below) has an overlay that contains a toolbar with 3 buttons, and a red shaded area guiding the user where best to place the barcode. The RedLaser app in the app store (middle image) has a different overlay, with arrows that indicate optimal placement, and change color when the barcode is in range (that is when @"InRange" is true). With the information the SDK delivers to your overlay, other user interfaces are possible as well.



CameraOverlayViewController

```
/**************************************************************************************
        CameraOverlayViewController

        An optional overlay view that is placed on top of the camera view.
        This view controller receives status updates about the scanning state, and
        can update the user interface.
*/
@interface CameraOverlayViewController : UIViewController { }

@property (readonly, assign) BarcodePickerController *parentPicker;

- (void)barcodePickerController:(BarcodePickerController*)picker
```

```
                    statusUpdated:(NSDictionary*)status;
@end
```

Your application should create a custom subclass of CameraOverlayViewController, and set the overlay property of the BarcodePickerController to your custom overlay before displaying the BarcodePickerController.

If you don't set a custom overlay, the default overlay is used, which has no user interface and simply exits the scan session as soon as the first barcode is recognized. With no UI, there is no way to cancel, so we really do recommend that you implement an overlay view.

Handling Device Rotations

The overlay is created as a subview of the BarcodePickerController's view; a design that makes handling device orientation inherently difficult for the overlay. iOS 5 includes some support for managing multiple view controllers onscreen at once in this fashion, but that doesn't help iOS 4.

The BarcodePickerController will ask the overlay which orientations it supports, and report those same orientations to the OS in shouldAutorotateToInterfaceOrientation:. Additionally, when the BarcodePickerController is asked to rotate, it will pass through willRotateToInterfaceOrientation:duration:, willAnimateRotationToInterfaceOrientation:duration:, and didRotateFromInterfaceOrientation: to the overlay.

One of the deficiencies of this setup is that the interfaceOrientation property of the overlay should not be trusted, as the OS does not update it correctly.

Implementing the StatusUpdated Method

Other than normal UIViewController operations, your BarcodePickerController2 subclass or overlay view controller receives barcodePickerController:statusUpdated or statusUpdated: messages from the SDK. The status NSDictionary may contain these keys when this method is called:

Key - ValueType	Usage Notes
@”FoundBarcodes” NSSet	NSSet of BarcodeResult objects. Will be the empty set if no barcodes have been found yet.
@”NewFoundBarcodes” NSSet	NSSet of BarcodeResult objects. This key only exists in the NSDictionary if at least one new barcode was recognized in the most recent scan pass.
@”Valid” NSNumber	TRUE if there are valid results.
@”InRange” NSNumber	TRUE if there is a barcode in range in the viewfinder, but the SDK hasn’t decoded it yet. This key may be used to advise the user to hold the phone steady while a barcode is read. Note that barcodes may be decoded without this key ever becoming TRUE. This key does NOT mean “The SDK is about to decode a barcode”, it means, “The application should tell the user to hold the camera steady.”
@”Guidance” NSNumber	This key only exists in the NSDictionary if there is guidance to be given. 1 means that the SDK sees a likely barcode in range, but hasn't been able to decode it for several seconds. The overlay may use this to advise the user to try scanning the barcode in parts by holding the phone close to each part of the barcode. 2 means that the SDK has scanned the first part of a barcode, the contents of which are available in the @"PartialBarcode" property. The overlay may use this to advise the user of the part of the barcode that's been successfully scanned.
@”PartialBarcode” NSString	The part of a barcode that's been scanned when doing partial scanning. The overlay may show the partially scanned barcode text to the user to help guide them on completing the partial scan.
@”CameraSupportsTapToFocus” NSNumber	TRUE if the device supports tap to focus. The overlay should only guide the user to tap on the barcode to focus on it if tap to focus is supported. Note that this guidance isn’t mandatory; recognition will happen without it.

Key - ValueType	Usage Notes
@"CameraSnapshot" UIImage	If present, this key's object will be a UIImage with a capture from the camera. See the method requestCameraSnapshot for more info.

Your code is responsible for determining when to end a scanning session. It can do this by implementing a "Done" or "Cancel" button, or by exiting when a condition is met in the status callback--such as the Valid key being TRUE. Either way, application code should signal the SDK that it is time to end the scan session by calling the BarcodePickerController2's doneScanning method.

The status method will always be called from the main thread.

Notes on the InRange Key

The purpose of the inRange key is to let the overlay know when it should be guiding the user to hold their device steady to get a good read of a barcode. This is important for one of our recognition algorithms. The algorithm tries to ensure that it performs its analysis on a frame that has low motion blur, and it uses inter-frame differencing to find such a frame. In this case, guiding the user to hold the device steady is an important part of the process.

Other recognition algorithms, which run at the same time, don't work in this way. They use a single frame analysis model, and are never in a state where they are confident a barcode is visible in the camera view but don't know what the barcode's value is. Because of this, it is entirely possible to have barcodes discovered and BarcodeResults returned without 'inRange' ever being set.

Your app should not use 'inRange' as a hint that a barcode is about to be found, nor should your overlay rely on entering the 'inRange' state before being informed of a new-found barcode. Your app should use the 'inRange' key as a guide to tell the user to hold the device steady.

ClearResultsSet and reportUnwantedBarcode

These methods are intended to be called from within the statusUpdated: callback (although it's not a requirement). ClearResultsSet tells the SDK to remove all the currently saved BarcodeResults, as if you just started a new session. It will also clear out the set of unwanted barcodes, meaning barcodes that had previously had a marker placed over them will no longer have this.

ReportUnwantedBarcode: does not change the set of barcodes that your status method receives in future updates, nor does it remove barcodes from the barcodePickerController:returnResults: method called when the session completes.

Presenting the BarcodePickerControllers

The Sample app shows how to display the BarcodePickerController2 as a modal view with presentModalViewController:. RedLaser's scan UI can be presented in other ways as well; most presentation methods have some tips developers should be aware of.

Using presentModalViewController:

iOS doesn't handle the case where a modal view controller is asked to disappear while it is in the middle of being animated into place. For many modal views this is not an issue because the view is dismissed via controls in the view, and those controls aren't available until the view is finished animating into place. It is possible for the BarcodePickerController to find a barcode while its view is still being animated into place, and if you call dismissModalViewControllerAnimated: as soon as the first barcode is found, the dismiss may not happen.

iOS 5 works around this issue with the new presentViewController:animated:completion: method. For applications that need iOS 4 compatibility, we recommend delaying dismissing the BarcodePickerController until after the overlay's viewDidAppear: method is called.

Using with a UINavigationController

The BarcodePickerController may be presented on a navigation stack using UINavigationController's pushViewController:animated: method. Depending on your implementation, you'll want to call pauseScanning or doneScanning when the view is popped off the navigation stack.

Using as a Tab in a UITabViewController

The BarcodePickerController may be presented as one of the view controllers of a UITabBarViewController. Again, you should be sure to call pauseScanning or doneScanning, as appropriate, when the user moves to another tab.

Using with a UIPopover

A UIPopoverController will always tell its sub-controller that it is in portrait mode, and does not send rotation events to the sub-controller. For most uses of UIPopoverController, this is not a problem, but the BarcodePickerController needs to show the camera preview and it needs to show it in the same orientation as the real objects behind/in front of the device. In order to get device rotations to work with a UIPopoverController, your root UIViewController will need to handle the following methods and pass the calls through to the BarcodePickerController:

- willRotateToInterfaceOrientation:duration:
- willAnimateRotationToInterfaceOrientation:duration:
- didRotateFromInterfaceOrientation:

That is, the view controller that is creating the popover should have methods that look like this:

```
- (void) willRotateToInterfaceOrientation:(UIInterfaceOrientation) toInterfaceOrientation
        duration:(NSTimeInterval) duration
{
    [popover presentPopoverFromRect:displayButton.frame inView:self.view
            permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];

    [barcodePickerController willRotateToInterfaceOrientation:toInterfaceOrientation duration:duration];
}
```

Capturing Barcodes in Sections

The 3.0 SDK introduces a new method of scanning barcodes, where the user can scan a barcode in parts and the SDK will stitch the parts together to recognize the full barcode. This scan method is useful for capturing long barcodes on older devices, where the camera resolution isn't high enough to resolve the entire barcode at once.

Currently, this new method only works for Code 39 barcodes.

The algorithm to stitch partial barcodes is always enabled in the SDK, however without app-level support in the overlay to guide the user, it is very unlikely a user will ever find it. Most of this section is a discussion of how to design your overlay UI to guide a user through the process of scanning a barcode in parts. Note that all of the cues from the SDK relating to this feature can be safely ignored if you don't want to support partial barcode scanning–you don't **have** to do anything.

Many SDK clients aren't using Code 39 in their apps, and even some of those that are may not need this feature. For developers that need Code 39 support in their app, adding this optional feature can lead to a better user experience, particularly for users with older devices trying to scan Code 39 codes in excess of ~15 characters in length.

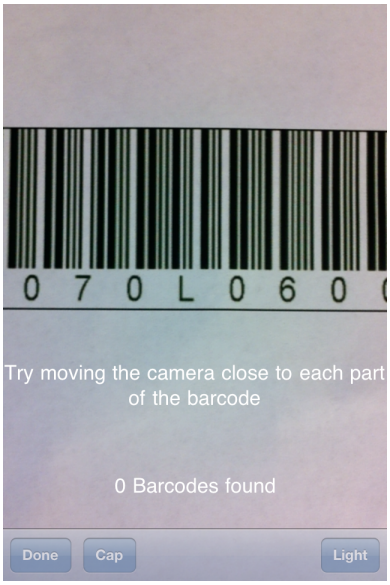
The basic interaction for partial barcode scanning is that the SDK tells the overlay what it sees, and the overlay tells the user what to do to successfully complete the scan. The communication from the SDK to the overlay happens through the @"Guidance" and @"PartialBarcode" keys, which you can read about in the section "Implementing the StatsuUpdated method".

Guidance == 1

After a couple of seconds where the SDK sees that the user is pointing the camera at something that looks like a barcode, if the SDK has not recognized the barcode yet it will start returning 1 in the “Guidance” key of the barcodePickerController:statusUpdated: dictionary.

Note that at this point the SDK isn’t claiming to know for sure if it’s looking at a Code 39 barcode, or for that matter any kind of barcode; it is just looking at something vaguely barcode-like that the SDK can’t fully recognize.

When the Guidance key goes to 1, the overlay should advise the user to try scanning the barcode in parts by holding the phone close to the first part of the barcode, and then scanning across slowly. See the image below.

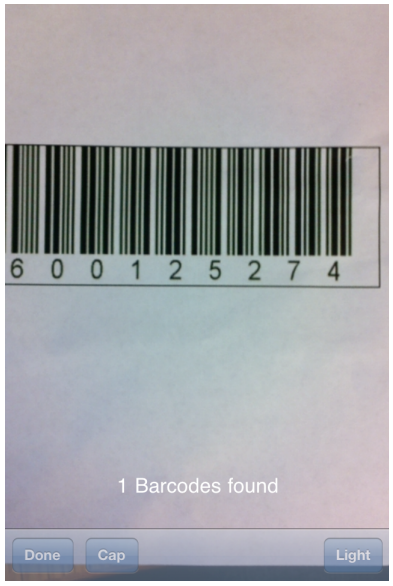


Once the user does this, the first part of the barcode may be scanned, at which time the Guidance key changes to 2.

Guidance == 2

When the guidance changes to 2, an additional key is available, “PartialBarcode”. This key contains the text of the barcode that has so far been decoded. In the example shown below, the partial barcode contains “JT5MAJ07” (the ellipsis is added by the overlay in this example).

As the user scans their device across the barcode, characters will be added to the partial barcode, as shown in the image below left. When they reach the end of the barcode, the completed code is added to the set of found barcodes, and will be sent to the app's CameraOverlayViewController subclass as an element in the "FoundBarcodes" set, just as if the barcode was recognized normally, as shown in the right image below.



Once a barcode has been fully scanned, the Guidance key goes back to zero and the PartialBarcode key will no longer be sent to the overlay during that scan session.

UI Notes

The intent of this design is to allow the app developer significant leeway in how they want to enable this feature for the user. Apps are free to ignore it altogether, or they can provide guidance in the user interface as they see fit. The sample images above show various text strings being shown by the overlay, but what the overlay shows the user is entirely up to the app developer.

Other than the text of the barcode string itself, localization for this feature is up to the app as all the strings come from the app. Or, an app developer could use an icon or a small animation to inform the user as to what to do to scan using this method.

Notes on Specific Barcode Types

This section contains supplemental information about how the SDK returns data for specific barcode types.

UPC-A and EAN-13

The SDK does not have a type for UPC-A, as it has been superseded by EAN-13. When the SDK encounters a UPC-A barcode, it will return the equivalent EAN-13 number, which prepends a 0 to the UPC code. The reasoning for why we do this has a lot to do with the fact that EAN-13 codes whose first number is 0 look exactly the same (to a barcode reader) as UPC-A codes.

UPC-E

UPC-E codes are shortened codes that use the same number system as UPC-A (and EAN-13). There is a specific algorithm for expanding a UPC-E code's digits and recovering the full EAN-13 code. The extendedBarcodeString member of the BarcodeResult object will contain the full EAN-13 code in this case.

QR Codes

When a QR Code is detected, the SDK returns information about the QR Code's location such that the top left and top right corners are oriented as shown in the image below. The QR Code specification does not itself specify a notion of 'top left' for QR Codes, so we chose this as the 'up' orientation.

Code 128

The SDK inserts the string “]C1” when the first symbol in a Code 128 barcode is FNC1, and inserts ASCII code 29 (the Group Separator character) when the FNC1 symbol is found elsewhere in the string.

FNC codes 2 through 4 are ignored and not inserted into the output string.

Codabar

Codabar uses 4 different start/end codes, and the RedLaser SDK will return the start/end codes as part of the barcode string. The start/end codes will always be labeled A, B, C or D. Most Codabar codes seem to just contain numbers, but the characters from the set “-$:/.+” are also allowed.

Including the start/end codes is different behavior than what RedLaser does for other barcode types. This is because the choice of start and end codes for a Codabar barcode has meaning independent of the text of the rest of the barcode string.

PDF417

RedLaser doesn’t support Macro PDF417, nor does it read the optional Compact PDF417 variant. It does decode Extended Channel Interpretations, and outputs a conforming ECI result in the extendedBarcodeString property.

GS1 Databar

RedLaser doesn’t yet handle composite Databar symbols. The SDK may be able to read both the linear and matrix portions of a composite symbol, but it won’t associate them with each other.

For Databar Omnidirectional, Truncated, Stacked, Stacked Omnidirectional, and Limited barcodes, RedLaser returns the raw GTIN-14 value in the barcodeString field, and the full string with symbology identifier and application identifier in the extendedString field.

For Databar Expanded and Expanded Stacked symbols, the barcodeString returned by the SDK will always contain the symbology identifier “]e0” and each data element will be prefixed by an application identifier. This is why the SDK uses a different type constant for Databar Expanded codes.

Aztec

The RedLaser SDK makes use of Apple’s barcode scanner in iOS 7 to find Aztec codes; none of the other scanners we use internally currently decode this barcode type. This

means that finding Aztec codes is dependent on the limitations of Apple's implementation, most notably that it only works on iOS 7.

ZXing License Attribution