# ID.me Verify SDK (iOS)

# A. General

## Release Information

- **SDK Version:** 1.1.1 (November 22, 2013)
- **Documentation Version:** 1.1.1 (v1) (November 22, 2013)
- **Maintained By:** Arthur Sabintsev

For more information please email us at mobile@id.me or visit us at http://developer.id.me.

## Changelog

- Renamed `CredentialType` typedef to `IDmeVerifyCredentialType`
- Renamed `CredentialType` typedef elements as well
- Addressed potential issue with dismisisng scanner

## General Information

The ID.me Verify SDK for iOS is a library that adds the following functionality to your mobile iOS application:

- Verify your user's group affiliation
    - Military Verification with TroopID
    - Student Verification with StudentID
    - First Responder Verification with FirstResponderID
- Scan your user's physical credential
    - All U.S. State Licenses (incl. Washington, D.C.)
    - Common Access Cards (a.k.a. CACs)
    - Uniformed Services Cards
        - Cards held by Active Military Personnel
        - Cards held by Retired Military Personnel
        - Cards held by Family Members and Dependents of Military Personnel

## Sample Project

A sample project has been added to the SDK for your convenience. A few *#warning* macros have been added to notify you where you should edit the code before running the project.

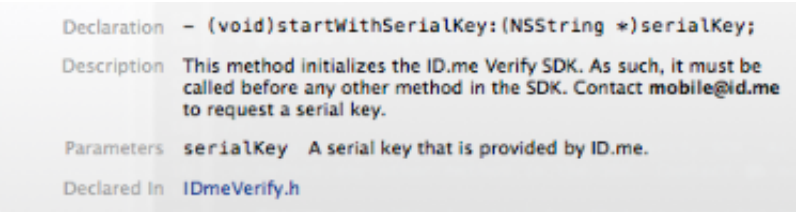
## iOS Version and iDevice Compatibility

- **iOS Compativility:** iOS 6, iOS 7
- **Device Compatibility**
    - *iPhone*: 4, 4S, 5, 5C, 5S
    - *iPad*: 3, 4, Mini
    - *iPod* Touch: 5th Gen

Even though iOS 6 supports the iphone 3GS and the iPad 2, the SDK does not support these devices. This is due to the lack of an auto-focusing camera lens, which is needed for the credential scanner.

## Referencing SDK Documentation within Xcode

For your convenience, *Doxygen* style comments have been added into the public headers of the SDK. If you're using Xcode 5+, simply `ALT + Left Click` on any method or variable found in the SDK to get a tooltip describing said method. An example of one of the tooltips can be found in the image below:




# B. Installation

Before you begin, please consider using utilizing this SDK as a submodule in your application.

1. Drag the **ID.me Verify SDK** file into your project. This includes the following files
    - `libIDmeVerify.a`
    - `IDmeVerify.bundle`
    - `IDmeVerify.h`
    - The `Classes` folder, which contains a some public headers for reference during development.
2. Make the following changes in the Xcode ***Project***
    - Go to your Xcode Project's Build Settings tab
    - Search for the following setting: *Architecture*
        - Change it to the following option: **Standard Architectures (armv7, armv7s)**
    - Search for the following setting: *Other Linker Flags*
        - Add two flags: **-ObjC** and **-all_load**

3. Make the following changes in the Xcode ***Target***
    - Go to your Xcode Target's *Build Settings* tab
    - Search for the following setting: *Architecture*
        - Change it to the following option: **Standard Architectures (armv7, armv7s)**
    - Search for the following setting: *Other Linker Flags*
        - Add two flags: **-ObjC** and **-all_load**
4. Import `IDmeVerify.h` in the classes that you plan
5. Added the following Apple Frameworks to your project:
    - `AVFoundation`
    - `CFNetwork`
    - `CoreMedia`
    - `CoreVideo`
    - `Foundation`
    - `UIKit`

Please make sure to include them in your project.

# C. Activiation

The SDK must be activated using a serial-key provided by ID.me. As the `IDmeVerify` class is a singleton, activation can occur at any point in time. It is our recommendation that activation occur in the `AppDelegate` during the application's startup.

To activate the SDK, simply run the following command with the provided serial-key:

```
[[IDmeVerify sharedInstance] startWithSerialKey:@"YOUR_SERIAL_KEY"];
```

If activation was successful, you'll see the following message in your console:

```
SDK Activation Succeeded!
```

If activation failed, you'll see the following message in your console:

```
SDK Activation Failed! The provided serial number is incorrect. Please make sure
you entered it in correctly and try again. If you need a new serial number,
please contact mobile@id.me.
```

If you have a problem activating the SDK, please contact us at mobile@id.me.

# D. Group Affiliation Verification

## Background

The **Group Affiliation Verification** aspect of the SDK occurs through a modal view controller. The modal view controller is a navigation controller initialized with a web-view. The entire OAuth flow occurs through the web-view. Upon successful completion, the modal will automatically be dismissed, and a JSON object in the form of an NSDictionary object containing your user's verificaiton information will be returned to you.

## Important Note on Modal Orientation

The modal navigation controller is a subclass of UINavigationController, named `IDmeWebVerificationNavigationController`. This navigation controller has only two methods in its implementation, allowing for a portrait-only orientation:

```
- (NSUInteger)supportedInterfaceOrientations
{
    return UIInterfaceOrientationMaskPortrait;
}

- (BOOL)shouldAutorotate
{
    return YES;
}
```

The public header for this class has been provided in the *Classes* folder of the SDK, in case your project has some various rules regarding orientation in your project's AppDelegate, specifically the following method: `application:supportedInterfaceOrientationsForWindow:`.

## Execution

To launch the modal, the following method must be called in the view controller class that will be presenting the modal:

```
- (void)verifyUserInViewController:(UIViewController *)externalViewController
                      withClientID:(NSString *)clientID
                       redirectURI:(NSString *)redirectURI
                  affiliationType:(IDmeVerifyAffiliationType)affiliationType
                    inSandboxMode:(BOOL)sandboxMode
                      withResults:
(IDmeVerifyVerificationResults)verificationResults;
```

The params in that method are as follows:

- `externalViewController`: The viewController which will present the modal navigationController.
- `clientID`: The clientID provided by ID.me when registering the app at http://developer.sandbox.id.me or http://developer.www.id.me.
- `redirectURI`: The redirectURI provided to ID.me when registering your app at http://developer.sandbox.id.me or http://developer.id.me
- `affiliationType`: The type of group verficiation that should be presented. Check the `IDmeVerifyAffiliationType` typedef for more details.
- `sandboxMode`: While developing the app, set this value to YES. The test data values provided by ID.me will be accessible via the sandbox API routes. Before pushing your application live and setting this value to NO, make sure your app is first registered with ID.me's live/production server by filling out a request at http://developer.id.me.
- `verificationResults`: A block that returns an NSDictionary object and an NSError object. The verified user's profile is stored in an NSDictionary object as JSON data. If no data was returned, or an error occured, NSDictionary is nil and NSError returns an error code and localized description of the specific error that occured.

In your code, the implementation of this method should yield an expanded form of the `verificationResults` block. It is our recommendation that the full implementation of this method look as follows:

```
[[IDmeVerify sharedInstance] verifyUserInViewController:
<your_presenting_view_controller>
                                          withClientID:<your_clientID>
                                           redirectURI:<your_redirectURI>
                                      affiliationType:<your_affiliationType>
                                        inSandboxMode:<BOOLEAN>
                                          withResults:^(NSDictionary
*userProfile, NSError *error) {

                                              if (error) { // Error


                                              } else { // Verification was
successful

                                              }

                                          }];
```

For the sake of convenience, the constants below, which are keys for the JSON (NSDictionary)

object, have been made avaialable:

- `kIDmeVerifyKeyAffiliation`, references the *affiliation* key in the JSON object.
- `kIDmeVerifyKeyID`, references the *id* key in the JSON object.
- `kIDmeVerifyKeyVerified`, references the *verified* key in the JSON object.

**NOTE:** Other attributes (e.g., email, first name, last name, etc…) can be returned in the JSON results upon special request. Please email mobile@id.me if your app needs to gain access to more attributes.

All potential errors that could occur are explained in the next section.

## Error Handling

There are four potential outcomes during the group affiliation verification process, three of which are errors. All of the errors are returned in the `IDmeVerifyVerificationResults` block, which is the last parameter in verification method described above. Each error will return a non-nil NSError object, and a nil NSDictionary object. The three verification related errors can be found in the `IDmeVerifyErrorCode` typedef, which deals with all errors in the SDK. The three verification related errors are as follows:

- `IDmeVerifyErrorCodeVerificationDidFailToFetchUserProfile`
  - Error occurs if user succesfully verified their group affiliation, but there was a problem with the user's profile being returned.
  - This should never occur, but this error was added to handle a rare situation involving the inability to reach ID.me's server.
- `IDmeVerifyErrorCodeVerificationWasDeniedByUser`
  - Erorr occurs if user succesfully verified their group affiliation, but decided to deny access to your app at the end of the OAuth flow.
- `IDmeVerifyErrorCodeVerificationWasCanceledByUser`
  - Error occurs if user exits modal navigation controller before OAuth flow could complete.

The following properties of the NSError object should be referenced by your app if you're looking to employ error-specific methods:

- `code`: The error code of the specific issue. The value is defined in the `IDmeVerifyErrorCode` typedef, and should be in the 100s.
- `localizedDescription`: A detailed description of the error.

## Internet Connectivity

Internet connectivity is required, as the verificaiton occurs through a webView.

# E. Credential Scanning

## Background

The second aspect of the ID.me Verify SDK is the **Credential Scanner and Parser**. The scanner works with the following credentials:

- All U.S. State Licenses (incl. Washington, D.C.)
- Common Access Cards (a.k.a. CACs)
- Uniformed Services Cards
    - Cards held by Active Military Personnel
    - Cards held by Retired Military Personnel
    - Cards held by Family Members and Dependents of Military Personnel

The following barcode is found the above-mentioned credentials. The scanner is fine-tuned to scan this specific barcode:

The user is taught to scan this type of barcode on the first launch of the credential scanner. A help button is added to the main screen of the scanner, allowing the user to reference this information on subsequent launches of the scanner.

## Important Note on Modal Orientation

The modal view controller is a subclass of UIViewController, named `IDmeCredentialScannerViewController`. This view controller has only two methods in its implementation, allowing for landscape-only orientation:

```
- (NSUInteger)supportedInterfaceOrientations
{
    return UIInterfaceOrientationMaskLandscape;
}

- (BOOL)shouldAutorotate
{
    return YES;
}
```

The public header for this class has been provided in the *Classes* folder of the SDK, in case your project has some various rules regarding orientation in your project's AppDelegate, specifically the following method: `application:supportedInterfaceOrientationsForWindow:`.

## Execution

The credential scanner is presented as a modal view controller. Once a credential is scanned, it is parsed in the background and data is returned in an Objective-C block.

The scanner is activated with the following method:

```
- (void)scanCredentialInViewController:(UIViewController
*)externalViewController
                    forCredentialTypes:
(IDmeVerifyDetectCredentialTypes)credentialTypes
                           withResults:(IDmeVerifyScanResults)scanResults;
```

The params in this method are as follows:

- `externalViewController` The viewController which will present the credential scanner model viewController.
- `*credentialTypes` One or more credential types defined in the IDmeVerifyDetectCredentialTypes options enumerator.
    - For example: If you want to only scan US State Licenses and Common Access Cards, use the following flags: `IDmeVerifyDetectCredentialTypeStateLicense | IDmeVerifyDetectCredentialTypeCommonAccessCard`
- `scanResults` An Objective-C block that returns the parsed results of the scanned credential and an NSError object.

The scanner should be implemented in the following manner:

```
    [[IDmeVerify sharedInstance] scanCredentialInViewController:self
                                                forCredentialTypes:
<IDmeVerifyDetectCredentialType Option>
                                                        withResults:^(IDmeCredential
*credential, NSError *error) {

                                                             if (error) { // Error

                                                             } else { // Credential
was scanned and parsed successfully

                                                             }
    }];
```

If there are no errors, the following snippet of code may be helpful in your `else` statement:

```
 switch ([credential credentialType]) {

     case IDmeVerifyCredentialTypeStateLicense:{
         IDmeStateLicense *stateLicense = (IDmeStateLicense *)credential;
         // Extract data from stateLicense
         } break;

     case IDmeVerifyCredentialTypeCommonAccessCard:{
         IDmeCommonAccessCard *commonAccessCard = (IDmeCommonAccessCard
*)credential;
         // Extract data from commonAccessCard
         } break;

     case IDmeVerifyCredentialTypeUniformedServicesCard:{
         IDmeUniformedServicesCard *uniformedServicesCard =
(IDmeUniformedServicesCard *)credential;
         // Extract data from uniformedServicesCard
         } break;

     case IDmeVerifyCredentialTypeUniformedServicesDependentCard:{
         IDmeUniformedServicesDependentCard *uniformedServicesDependentCard =
(IDmeUniformedServicesDependentCard *)credential;
         // Extract data from uniformedServicesDependentCard
         } break;

     case IDmeVerifyCredentialTypeUnknown:{
         // Occurs when scanned credential is unknown
         // An error is also thrown, so this conditional should never be entered
if you use the if-else block in the aformenetioned code snippet.
     } break;
 }
```

All potential errors that could occur are explained at a later step in this section.

## Credentials

### Hierarchy

The data returned in the Objective-C block in the aforementioned method are all instances or subclassed instances of the `IDmeCredential` class. The headers for this class, and its subclasses are made public in this library. For your reference, the class hierarchy is as follows:

- `IDmeCredential`
    - `IDmeStateLicense` subclasses `IDmeCredential`
    - `IDmeDefenseCredential` subclasses `IDmeCredential`

- `IDmeCommonAccessCard` subclasses `IDmeDefenseCredential`
- `IDmeUniformedServicesCard` subclasses `IDmeDefenseCredential`
    - `IDmeUniformedServicesDependentCard` subclasses `IDmeUniformedServicesCard`

### Parsed Data

For easy reference, each IDmeCredential instance (and subclassed instance) is made to respond to an overwritten instance of the `description` method, which returns back all the stored data in the `IDmeCredential` class of interest.

For example, let's assume we have an object named `stateLicense`, (e.g., `IDmeStateLicense *stateLicense;`). When calling `[stateLicense description]`, the following ***key : value*** result wil be outputted:

```
==[PARSED DATA]==
Credential Type : US State License
firstName : JOHN
lastName : SMITH
address : 123 MAIN ST
city : WILMINGTON
state : DE
zipCode : 19801
birthdate : 02/01/1970
birthMonth : 01
birthDay : 02
birthYear : 1970
age : 43 years old
```

Each key in the list above is also the the name of a property on the `IDmeStateLicense` object that can be used to extract the data/values on the right side of the colon. To reiterate, each `IDmeCredential` subclass instance has their own set of properties. The easiest way of finding our what those properties are, is by accessing the `description` method. Another way of finding out this informaiton is by checking the header files in the *Classes* folder of the SDK for each `IDmeCredential` subclass and traversing the class hierarchy for the properties that are inherited.

## Error Handling

There are fives potential outcomes during the credential scanning and parsing process. Four of these outcomes are errors. All of the errors are returned in the `IDmeVerifyScanResults` block, which is the last parameter in credential scanner method described above. Each error will return a non-nil NSError object, and a nil Credential object. The four verification related errors can be found in the `IDmeVerifyErrorCode` typedef, which deals with all errors in the SDK. The four

scanning and parsing related errors are as follows:

- `IDmeVerifyErrorCodeScannerAccessForbidden`
  - Error occurs if developer attempts to access scanner, but is not given access to it. To gain access to scanner, send an email to mobile@id.me.
- `IDmeVerifyErrorCodeNoScanPerformed`
  - Error occurs if user exits scanner without scanning a credential.
- `IDmeVerifyErrorCodeCouldNotParseCredential`
  - Error occurs if scanned credential could not be parsed.
- `IDmeVerifyErrorCodeCredentialOwnerIsUnderLegalAge`
  - Error occurs if owner of scanned credential is under age (e.g., under 13).

## Known Exceptions

- **Illinois**: Licenses from this state only encode the first-name, last-name and birthdate on the barcode.
- **Iowa**: License may not scan properly. We are working to address this issue.
- **North Dakota**: License may not scan properly. We are working to address this issue.
- **South Dakota**: License may not scan properly. We are working to address this issue.
- **Vermont**: License may not scan properly. We are working to address this issue.

## Analytics

The ID.me SDK will call home only after a credential scan is performed. The following information will be sent back for analytics purposes:

- ID.me Verify SDK Version
- iOS Version
- iDevice
- Scanned Credential Type (**only** the type, and not the information on the credential).
- Your App's Name
- Your App's Version

## User Acquisiton

Certain portions of the scanned credential will be sent home in the analytics API request, ***IF AND ONLY IF*** you are a partner who has entered into a user-acquisiton agreement with ID.me. For more information, please contact us at mobile@id.me.

## Internet Connectivity

As credential scanning occurs client-side (e.g. on the iDevice), it is understood that internet connectivity may not be available. However, internet connectivity is required for the analytics

API route to be reached. As analytics comes secondary to scanning and parsing, the SDK will handle the lack-of-connectivity situation gracefully without affecting the scanning and parsing of the credential.

# F. Convenience Macros and Constants

The following macros and constants were made public in the SDK for your convenience:

- `IDmeLog(fmt, …)`:This is an NSLog that prefixes every statement with **[ID.me Verify]:**, which can be used for debugging SDK related errors when you implement the SDK into your project.
- `IDME_VERIFY_ERROR_DOMAIN`: All NSError objects in this SDK utilize this Error Domain.
- `kIDmeVerifySDKVersion`: An NSString constant that yields the current version of the SDK.

# G. Terms of Use

By using this SDK, you agree to ID.me's Terms of Use.