Acceptto FIDO iOS Toolkit / Sample App Documentation

INDEX

1. Introduction

1.1 Purpose

This document describes how to integrate all of the framework components of the Acceptto FIDO Toolkit into a target mobile application. The entire installation process is described with examples and all of the exposed methods of each framework are detailed.

This guide is intended for developers to have a general overview of the Acceptto FIDO Toolkit.

1.2 Scope

This document covers all relevant integration guidelines for all the components of the Acceptto FIDO Toolkit for iOS.

1. Overview

1.1 Components

The Acceptto FIDO Toolkit is comprised of three frameworks and a demo app to exemplify all of the main functionalities. These components are:

1.1.1 Acceptto FIDO Core Framework

- Communicates with the Acceptto FIDO server

- Performs register, authenticate and deregister operations

- Conforms to FIDO protocol

- Automatically manages keyID and login details using the device keychain

1.1.2 Acceptto FIDO Authenticators Framework

- Provides full authenticator functionality for two generic iOS authenticators:
    - PIN authentication
    - Biometric authentication (Touch ID or Face ID)

1.1.3 Acceptto FIDO Manager Framework

- Provides a wizard-like, fully automated interface for enrolling and authenticating in Acceptto FIDO server via iOS

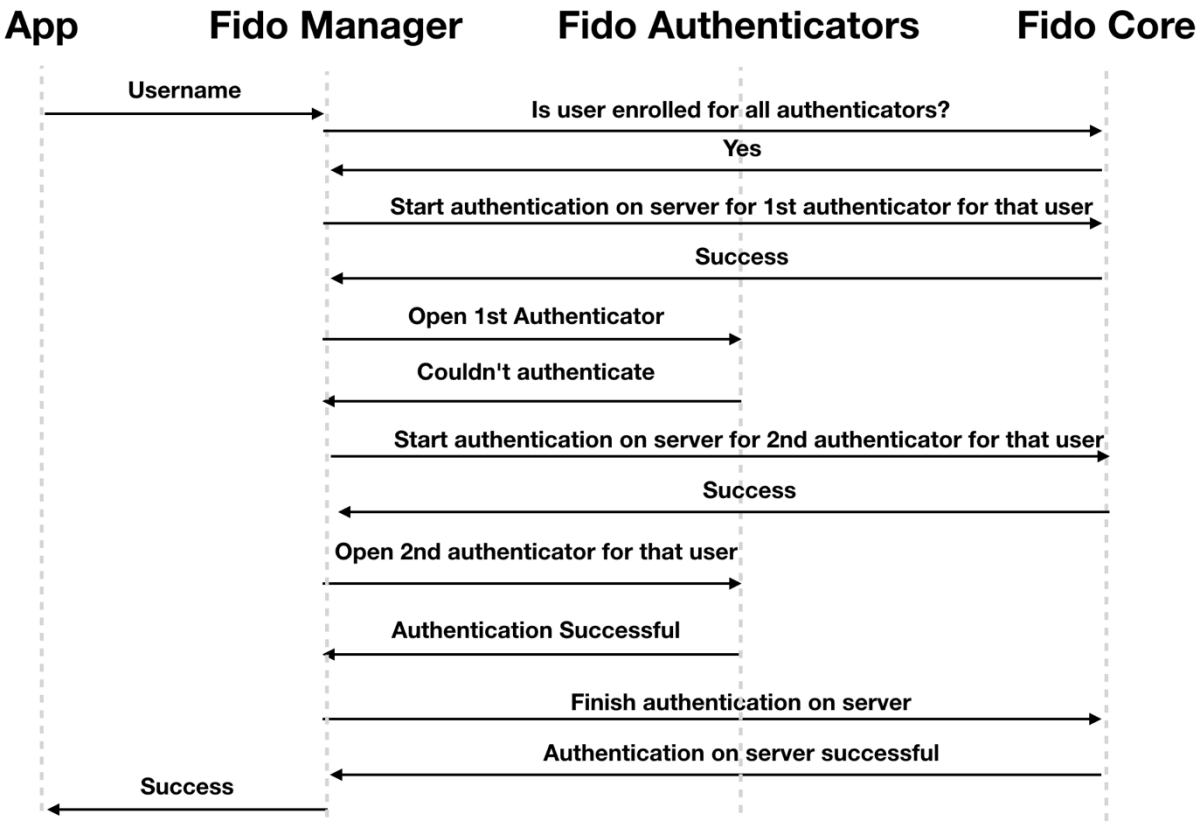
- Manages the authenticators that the user chooses to use, as well as the order in which they should be presented (also fully automated)

1.1.4 Acceptto FIDO Sample App

- Demo app for full functionality of all frameworks above

1.2 Example authentication workflow

This schematic tries to describe a simple authentication workflow, emphasizing each of the toolkit's components roles in the process:



2. Setup and iOS-Specific Implementation Details

This section covers installation and configuration of the selected framework(s) from the toolkit. It also describes all the public methods and properties of each framework, and their purpose and functionality.

2.1 Setup

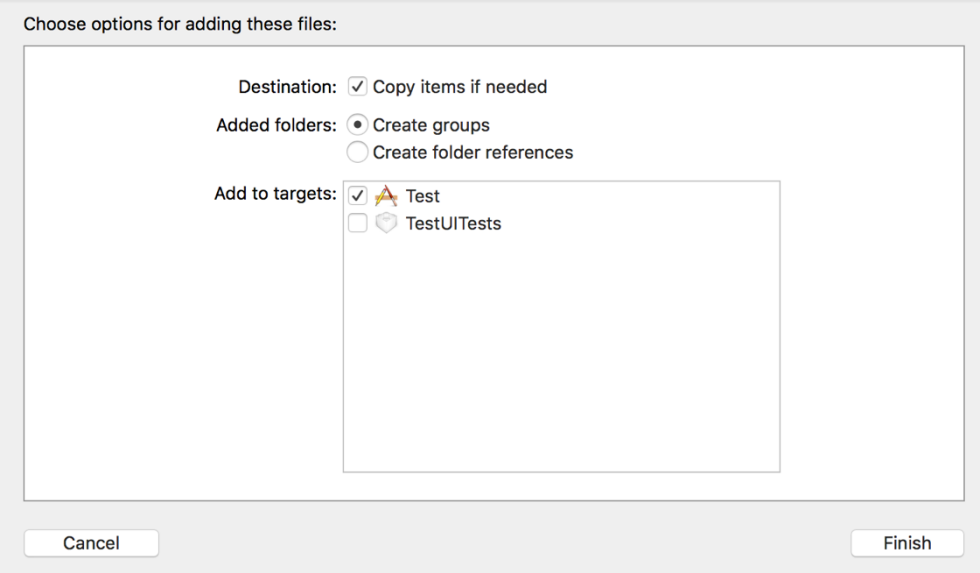
2.1.1 Framework dependency installation checklist

Please note the steps needed to install each framework, according to the following table:

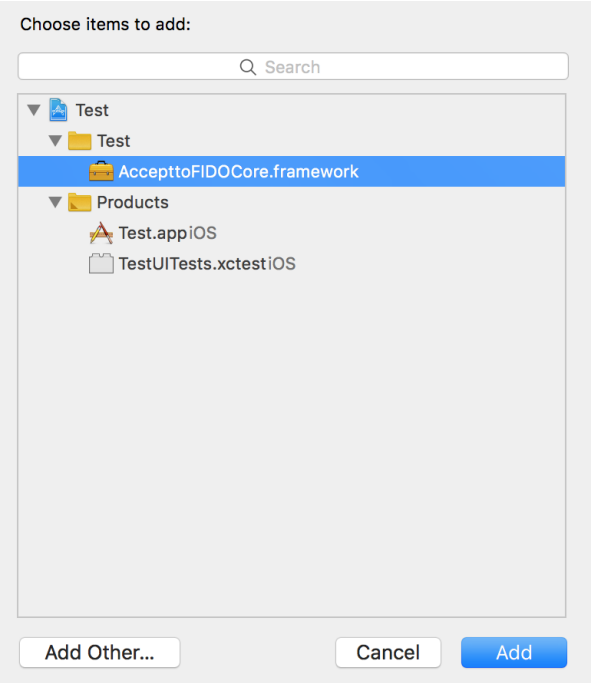
To install… ➔	Acceptto Fido Core Framework	Acceptto Fido Authenticators Framework	Acceptto Fido Manager Framework
Copy AccepttoFidoCore.framework to the host app	✓		✓
Copy AccepttoAuthenticatorsFramework.framework to the host app		✓	✓
Copy AccepttoFidoManagerFramework.framework to the host app			✓
Install AFNetworking via cocoapods	✓		✓
Install tinycbor via cocoapods	✓		✓

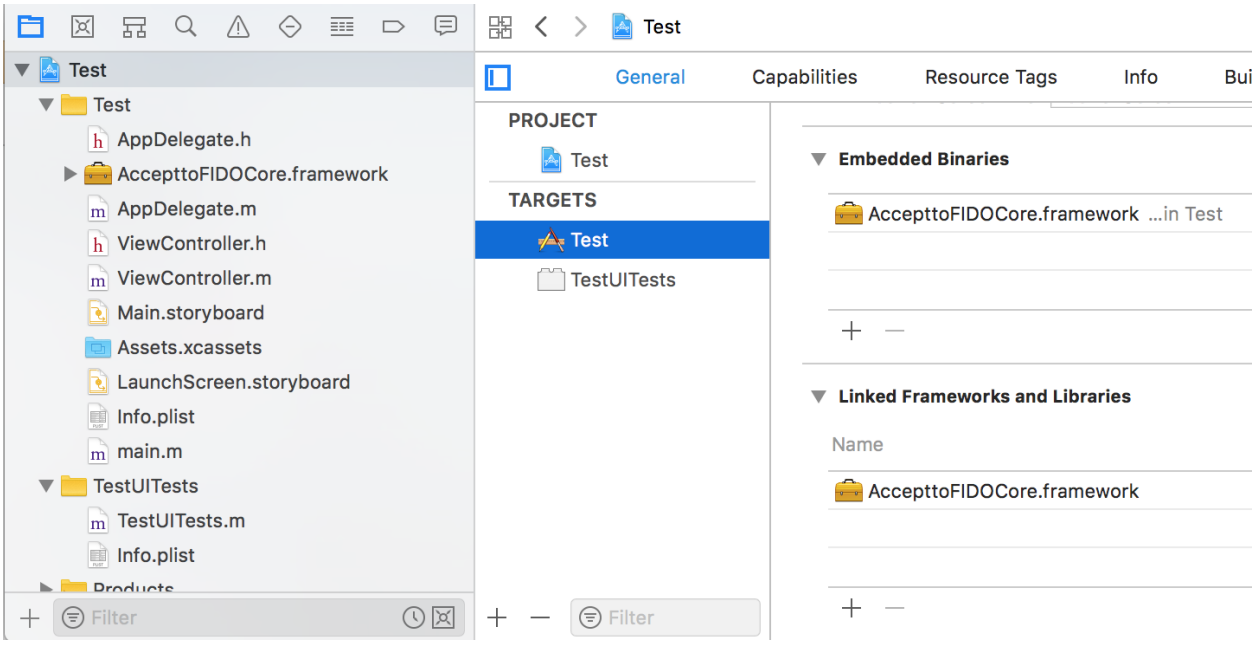
2.1.2 How to copy a framework file to the host app

1. Drag the file from the finder to your XCode project structure. Make sure the **Copy items if needed** checkbox is ticked, and press **Finish** to copy the file to your project.



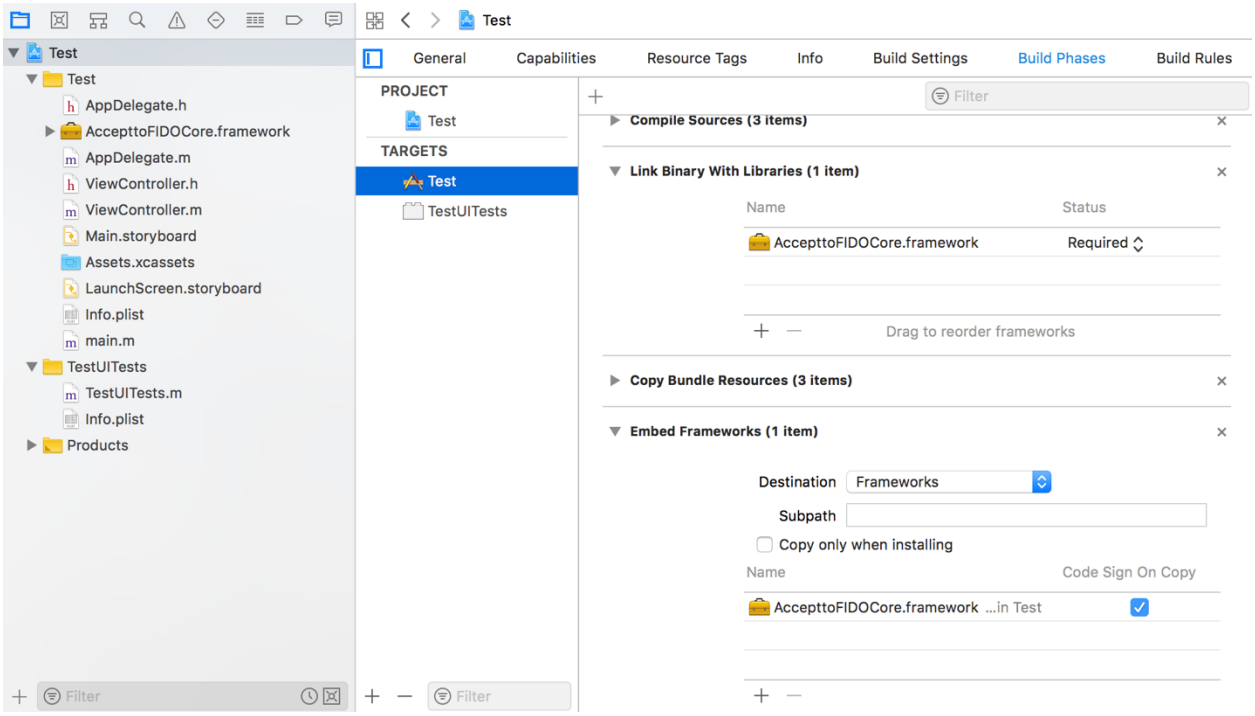
2. Go to the host app's targets settings, **General** tab. Scroll down to **Embedded Binaries**, press the "**+**", and select the framework you just added. Then, press the **Add** button.





3. Check that the framework you added is already in the **Linked Frameworks and Libraries** list. If not, add it in a similar way as in step 2.

4. Go to the host app's targets settings, **Build Phases** tab. Check that the framework you added is already in the **Link Binary With Libraries** and **Embed Frameworks** lists. If it's not, add it to both of them in a similar way as step 2.



2.1.3 How to install AFNetworking and tinycbor via CocoaPods

Add Cocoapods to your project and configure it to download and install the **AFNetworking** and **tinycbor** (version 0.5.3-alpha3) pods. Please refer to the Cocoapods documentation at https://guides.cocoapods.org/using/using-cocoapods.html for more information about this procedure.

The podfile you create should be similar to the following example:

```
platform :ios, '10.0'

use_frameworks!
```

```
target 'TargetName' do

    pod 'AFNetworking'
    pod 'tinycbor', '0.5.3-alpha3'

end
```

2.2 iOS-Specific Implementation Details

This section covers the methods available for use in each framework, as well as some general information about how these methods should be set up and called. In the sample app it's possible to check every method here.

2.2.1 Acceptto FIDO Core Framework Implementation Details

The Acceptto FIDO Core Framework is an easy gateway to the otherwise complex FIDO protocol procedures and communication with a FIDO server. It manages all users, authenticators and registrations.

For most procedures, the only input parameters you will need are:

- Username (usually the user's email address)
- The authenticator's AAID. This is an authenticator identification string, provided by the FIDO alliance and/or vendor. This should be a 9-character string value, in the format "V#M", where:
    - "#" is a separator
    - "V" indicates the authenticator Vendor Code. This code consists of 4 hexadecimal digits.
    - "M" indicates the authenticator Model Code. This code consists of 4 hexadecimal digits.

  In Acceptto FIDO SDK, the AAID for the biometric authenticator is 0023#0001 and the AAID for the pin authenticator is 0023#0002.

The Acceptto FIDO Core Framework follows the FIDO UAF 1.0 protocol specification. For more information about the process, please refer to the documentation on the FIDO alliance website: https://fidoalliance.org

2.2.1.1 Importing the framework headers

To use the Acceptto FIDO Core Framework's methods, please import AccepttoFidoCore.h:

```
#import <AccepttoFIDOCore/AccepttoFIDOCore.h>
```

2.2.1.2 Managing the URL for the FIDO Server

2.2.1.2.1 Setting the URL for the FIDO Server

Before trying to do any FIDO operation, you must set the URL for the FIDO server you want to use. For that, you should use the following method:

```
+ (NSError *)setFidoServer:(fidoServer)server;
```

This will set the base url according to the selected server. fidoServer is an enum value that consists of the following:

fidoUAFServerAcceptto – connects to the Acceptto server at https://uaf.acceptto.com/v1/public/

fidoUAFServerITU – connects to the ITU server at https://afido.itu.int/v1/public/

This method returns an NSError object, which will point to nil if the operation has been successful. If the server URL wasn't set, refer to this document's **error list** section for the possible returned errors.

This operation needs to occur only once in the application's lifetime. It's advisable to call it in the didFinishLaunchingWithOptions method of the application's delegate.

Example usage from the sample app code:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    [AccepttoFIDOCore setFidoServer:fidoUAFServerITU];
    return YES;
}
```

2.2.1.3 Authenticator registration

Your application should execute three steps to register an authenticator under a specific username:

1. Tell the FIDO server to start a registration process

2. Present the authenticator to the user for authentication
3. If the authentication is successful, tell the FIDO server to finish the registration process

The Acceptto FIDO Core framework performs steps 1. and 3. in an easy and automated way

2.2.1.3.1 Tell the FIDO server to start a registration process

To start a registration process, use the following method:

```
+ (void)requireRegistrationForUsername:(NSString *)userName aaid:(NSString *)aaid withSuccess:(void
(^)(void))success fail:(void(^)(NSError *error)) fail;
```

This will start the registration process in the server (step 1 above), for the specified user and authenticator pair. Success and fail blocks are provided for easy asynchronous operation.

The fail block returns an error that describes what went wrong with the process. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
[AccepttoFIDOCore requireRegistrationForUsername:userName aaid:aaid withSuccess:^{

        [self actionCheckUser:nil]; //if a user was registered previously and starts to register again,
he will be unregistered before requiring registration, so an update of the interface is required

        [self addMessageToLogWithTitle:@"INFO" message:@"Success requiring registration to server"
color:[UIColor whiteColor]];

        self.currentOperationType=@"register";

        self.currentOperationUsername=userName;

        self.currentOperationAaid=aaid;

        [self presentAuthenticatorOfType:authenticatorType];

    } fail:^(NSError *error) {

        [self actionCheckUser:nil]; //see comment above

        [self enableAllControls:YES];

        [self addMessageToLogWithTitle:@"ERROR" message:[NSString stringWithFormat:@"Error requiring
registration from server: %@",error.localizedDescription] color:[UIColor redColor]];

    }];
```

2.2.1.3.2 Present the authenticator to the user for authentication

This is a required step on the normal FIDO workflow. The FIDO iOS Toolkit provides a framework that includes methods for this functionality with biometric and/or pin authenticators. Please check the documentation for Acceptto Authenticators Framework below.

2.2.1.3.3 Tell the FIDO server to finish a registration process

To finish a registration process, use the following method:

```
+ (void)finishRegistrationForUsername:(NSString *)userName aaid:(NSString *)aaid withSuccess:(void
(^)(void))success fail:(void(^)(NSError *error))fail;
```

This will finish the registration process in the server (step 3 above), for the specified user and authenticator pair. Please note this step should only be called after the user successfully authenticates with the specified authenticator.

Success and fail blocks are provided for easy asynchronous operation.

The fail block returns an error that describes what went wrong with the process. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
[AccepttoFIDOCore finishRegistrationForUsername:_currentOperationUsername aaid:_currentOperationAaid
withSuccess:^{

                [self addMessageToLogWithTitle:@"INFO" message:@"Success completing registration on
server" color:[UIColor whiteColor]];

                [self actionCheckUser:nil];

                [self enableAllControls:YES];

            } fail:^(NSError *error) {

                [self addMessageToLogWithTitle:@"ERROR" message:[NSString stringWithFormat:@"Error
completing registration on server: %@",error.localizedDescription] color:[UIColor redColor]];

                [self enableAllControls:YES];

            }];
```

2.2.1.4 User Authentication

Your application should execute three steps to authenticate a user that has already registered an authenticator under his username:

1. Tell the FIDO server to start an authentication process
2. Present the authenticator to the user for authentication
3. If the authentication is successful, ask the FIDO server to authorize this user/aaid pair

The Acceptto FIDO Core Framework performs steps 1 and 3 in an easy and automated way.

2.2.1.4.1 Tell the FIDO server to start an authentication process

To start an authentication process, use the following method:

```
+ (void)requireAuthenticationForUsername:(NSString *)userName aaid:(NSString *)aaid withSuccess:(void
(^)(void))success fail:(void (^)(NSError *))fail;
```

This will start the authentication process in the server (step 1 above), for the specified user and authenticator pair.

Success and fail blocks are provided for easy, asynchronous operation.

The fail block returns an error that describes what went wrong with the process. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
    [AccepttoFIDOCore requireAuthenticationForUsername:userName aaid:aaid withSuccess:^{

        [self addMessageToLogWithTitle:@"INFO" message:@"Success requiring authentication to server"
color:[UIColor whiteColor]];

        self.currentOperationType=@"auth";

        self.currentOperationUsername=userName;

        self.currentOperationAaid=aaid;

        [self presentAuthenticatorOfType:authenticatorType];
```

```
    } fail:^(NSError *error) {

        [self enableAllControls:YES];

        [self addMessageToLogWithTitle:@"ERROR" message:[NSString stringWithFormat:@"Error requiring
authentication from server: %@",error.localizedDescription] color:[UIColor redColor]];

    }];
```

2.2.1.4.2 Present the authenticator to the user for authentication

This is a required step on the normal FIDO workflow. The FIDO iOS Toolkit provides a framework that includes methods for this functionality with biometric and/or pin authenticators. Please check the documentation for Acceptto Authenticators Framework below.

2.2.1.4.3 Ask the FIDO server to finish an authentication process (authorize a user/aaid pair)

To finish an authentication process and validate the user/aaid pair, use the following method:

```
+ (void)finishAuthenticationForUsername:(NSString *)userName aaid:(NSString *)aaid withSuccess:(void
(^)(void))success fail:(void(^)(NSError *error)) fail;
```

This will finish the authentication process in the server (step 3 above), for the specified user and authenticator pair.

Success and fail blocks are provided for easy, asynchronous operation.

The fail block returns an error that describes what went wrong with the process. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
[AccepttoFIDOCore finishAuthenticationForUsername:_currentOperationUsername aaid:_currentOperationAaid
withSuccess:^{

                [self addMessageToLogWithTitle:@"INFO" message:@"Success completing authentication on
server" color:[UIColor whiteColor]];
```

```
                [self actionCheckUser:nil];

                [self enableAllControls:YES];

            } fail:^(NSError *error) {

                [self addMessageToLogWithTitle:@"ERROR" message:[NSString stringWithFormat:@"Error
completing authentication on server: %@",error.localizedDescription] color:[UIColor redColor]];

                [self enableAllControls:YES];

            }];
```

2.2.1.5 User Deregistration

The deregistration process is executed in one step, with no needed user intervention. The Acceptto FIDO Core Framework can perform this step in an easy and automated way.

2.2.1.5.1 Tell the FIDO server to deregister all authenticators that were registered for a specific user

To deregister a user and all authenticators that he previously registered, use the following method:

```
+ (void)performDeregistrationForUsername:(NSString *)userName withSuccess:(void (^)(void))success
fail:(void(^)(NSError *error))fail;
```

Success and fail blocks are provided for easy, asynchronous operation.

The fail block returns an error that describes what went wrong with the process. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
[AccepttoFIDOCore performDeregistrationForUsername:userName withSuccess:^{

        [self addMessageToLogWithTitle:@"INFO" message:@"Success deregistering on server" color:[UIColor
whiteColor]];

        [self actionCheckUser:nil];
```

```
        [self enableAllControls:YES];

    } fail:^(NSError *error) {

        [self enableAllControls:YES];

        [self addMessageToLogWithTitle:@"ERROR" message:[NSString stringWithFormat:@"Error performing
deregistration on server: %@",error.localizedDescription] color:[UIColor redColor]];

    }];
```

2.2.1.6 Utility methods

2.2.1.6.1 Check if a username is valid

The Acceptto FIDO Core Framework provides an easy method to check if an input string is a valid username, according to the FIDO protocol specification.

This check may be performed using the following function:

```
+ (BOOL)isValidUsername:(NSString *)userName;
```

The function returns a boolean value that indicates if the candidate username is valid.

2.2.1.6.2 Check if an AAID is valid

The Acceptto FIDO Core Framework provides an easy method to check if an input string is a valid authenticator AAID, according to the FIDO protocol specification.

This check may be performed using the following function:

```
+ (BOOL)isValidAAID:(NSString *)authenticator_aaid;
```

The function returns a boolean value that indicates if the candidate AAID is valid.

2.2.1.6.3 Check if a user is registered with a specific authenticator

The Acceptto FIDO Core Framework provides an easy method to verify if a specific username/aaid pair is registered on the server.

This check may be performed using the following function:

```
+ (BOOL)isUserEnrolled:(NSString *)userName withAaid:(NSString *)aaid;
```

The function returns a boolean value that indicates if the specified authenticator is already registered for the specified username.

Example usage from the sample app code:

```
BOOL isEnrolledTouch=[AccepttoFIDOCore isUserEnrolled:userToVerify withAaid:@"0023#0001"];

BOOL isEnrolledPIN=[AccepttoFIDOCore isUserEnrolled:userToVerify withAaid:@"0023#0002"];
```

2.2.2 Acceptto Authenticators Framework Implementation Details

The Acceptto Authenticators Framework provides an easy and streamlined way to authenticate a user via two iOS widely used methods:

- PIN Authentication
    - A complete Pin authenticator is provided, with both 'set new pin' and 'ask for pin' operation modes, and with automatic save of the user's pin

- Biometric Authentication
    - A complete Biometric authenticator is provided. It acts as an easy interface to the device's TouchID or FaceID technology, complete with other features such as asking for biometric permissions if not given previously

2.2.2.1 Importing the framework headers

To use the Acceptto Authenticators Framework's methods, please import AccepttoAuthenticatorsFramework.h:

```
#import <AccepttoAuthenticatorsFramework/AccepttoAuthenticatorsFramework.h>
```

2.2.2.2 Invoking and using the Acceptto PIN authenticator

The Acceptto PIN authenticator supports multiple users, each with their own pin securely stored.

2.2.2.2.1 Operation mode enumerated values

The operation modes of the Acceptto PIN authenticator are described in an enum as follows:

```
typedef enum {
    accepttoPinAuthenticatorOperationModeGetPin,
    accepttoPinAuthenticatorOperationModeSetNewPin
} accepttoPinAuthenticatorOperationMode;
```

- accepttoPinAuthenticatorOperationModeSetNewPin
    - Describes the operation mode in which the user sets a new pin for the authenticator
    - The user will have to repeat the desired pin to provide confirmation
    - The authenticator may be configured to only accept pins of a given minimum length.
- accepttoPinAuthenticatorOperationModeGetPin
    - Describes the operation mode in which the user authenticates by typing his personal PIN number
    - The authenticator may be configured to accept a limited number of retries before failing

2.2.2.2.2 Invoking the Acceptto PIN authenticator

The AccepttoPinAuthenticatorViewController is provided as a subclass of UIViewController, with an associated NIB. The host app will have control over its presence on screen. Since it's an independent view controller, it can be presented modally or in a navigation sequence.

To instantiate the AccepttoPinAuthenticatorViewController, two methods are provided:

Method 1:

```
- (id)initWithUsername:(NSString *)userName
andOperationMode:(accepttoPinAuthenticatorOperationMode)operationMode;
```

Instantiates an AccepttoPinAuthenticatorViewController for the specified username, operating in the specified mode (please refer to the section above for the available operation modes)

Method 2:

```
- (id)initWithUsername:(NSString *)userName;
```

Instantiates an AccepttoPinAuthenticatorViewController for the specified username, operating in the mode determined by the following rules:

- If the specified user has already set a pin, the authenticator will start in the accepttoPinAuthenticatorOperationModeGetPin operation mode
- If the specified user didn't set a pin yet, the authenticator will start in the accepttoPinAuthenticatorOperationModeSetNewPin operation mode

Please refer to the section above for more information on these operation modes.

Example usage from the sample app code:

```
acceptoPinAuthenticatorOperationMode mode=[_currentOperationType
isEqualToString:@"register"]?accepttoPinAuthenticatorOperationModeSetNewPin:accepttoPinAuthenticatorOper
ationModeGetPin;

            AccepttoPinAuthenticatorViewController *vc=[[AccepttoPinAuthenticatorViewController alloc]
initWithUsername:_currentOperationUsername andOperationMode:mode];

            vc.delegate=self;

            [self presentViewController:vc animated:YES completion:nil];
```

2.2.2.2.3 Changing settings for the Acceptto PIN authenticator

The Acceptto PIN authenticator provides two customizable settings for its operation.

2.2.2.2.3.1 Setting the maximum number of retries

You can set the maximum number of retries the user can make while trying to authenticate. For that, just set the **pinMaxRetries** property.

```
@property (nonatomic, assign) NSInteger pinMaxRetries;
```

The default value for this property is 3.

If you set this property to 0 or less, retries will be unlimited.

2.2.2.2.3.2 Setting the minimum pin length

You can set the minimum digits that the PIN number can have, for a customizable security level. For that, just set the **minimumPinLength** property.

```
@property (nonatomic, assign) NSInteger minimumPinLength;
```

The default value for this property is 4.

If you set this property to 0 or less, the minimum pin length will revert to the default, which is 4.

If you set this property to a value that is incompatible with the maximumPinLength property value, both properties will revert to default.

2.2.2.2.3.3 Setting the maximum pin length

You can set the maximum digits that the PIN number can have, for a customizable security level. For that, just set the **maximumPinLength** property.

```
@property (nonatomic, assign) NSInteger maximumPinLength;
```

The default value for this property is 0.

If you set this property to 0 or less, there will be no limit to the pin length.

If you set this property to a value that is incompatible with the minimumPinLength property value, both properties will revert to default.

2.2.2.2.4 The Acceptto Pin Authenticator delegate protocol and callbacks

For proper asynchronous operation, the Acceptto Pin Authenticator defines a delegation protocol (**AccepttoPinAuthenticatorViewControllerDelegate**) with two callbacks for the host app to act upon the end of operation for each operation mode:

```
@protocol AccepttoPinAuthenticatorViewControllerDelegate <NSObject>
@optional
- (void)finishedPinSetup:(AccepttoPinAuthenticatorViewController *)controller completed:(BOOL)completed
error:(NSError *)error;
- (void)finishedPinAuthentication:(AccepttoPinAuthenticatorViewController *)controller
accessPermitted:(BOOL)permitted error:(NSError *)error;
@end
```

2.2.2.2.4.1 Acting upon the completion of the setting a new pin operation

Implement the following method in the delegate class:

```
- (void)finishedPinSetup:(AccepttoPinAuthenticatorViewController *)controller completed:(BOOL)completed
error:(NSError *)error;
```

This method will be called when the "set new pin" operation is finished by the user. Two parameters will be received:

- A boolean value **completed** which defines if the user completed the operation

- An NSError object **error** which describes what went wrong with the process, if not completed. If there was no error, this object will be nil. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
- (void)finishedPinSetup:(AccepttoPinAuthenticatorViewController *)controller completed:(BOOL)completed
error:(NSError *)error {

    [self doProceduresAfterPresentingAuthenticatorWithResultsPermitted:completed error:error];

}
```

2.2.2.2.4.2 Acting upon the completion of the pin authentication operation

Implement the following method in the delegate class:

```
- (void)finishedPinAuthentication:(AccepttoPinAuthenticatorViewController *)controller
accessPermitted:(BOOL)permitted error:(NSError *)error;
```

This method will be called when the pin authentication operation is finished by the user. Two parameters will be received:

- A boolean value **permitted** which defines if the user successfully introduced the correct pin, and thus his access is permitted.
- An NSError object **error** which describes what went wrong with the process, if not permitted. If there was no error, this object will be nil. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
- (void) finishedPinAuthentication:(AccepttoPinAuthenticatorViewController *)controller
accessPermitted:(BOOL)permitted error:(NSError *)error {

    [self doProceduresAfterPresentingAuthenticatorWithResultsPermitted:permitted error:error];

}
```

2.2.2.3 Invoking and using the Acceptto Biometric authenticator

The Acceptto Biometric authenticator will use the biometric sensor of the device for user authentication. It will use TouchID or FaceID, whichever is available on the iOS device.

It will also manage user permission for the app to use the device's sensor. If the user doesn't provide it, the authenticator will ask for permission in a dialog box that will open the settings app directly.

2.2.2.3.1 Invoking the Acceptto Biometric authenticator

The AccepttoTouchAuthenticatorViewController is provided as a subclass of UIViewController, with an associated nib. The host app will have control over its presence on screen. Since it's an independent view controller, it can be presented modally or in a navigation sequence.

You can instantiate the AccepttoTouchAuthenticatorViewController via the usual **init** method:

```
- (id)init;
```

Instantiates an AccepttoTouchAuthenticatorViewController for the iOS device's user.

Example usage from the sample app code:

```
            AccepttoTouchAuthenticatorViewController *vc=[[AccepttoTouchAuthenticatorViewController
alloc] initWithAutoStartIdentification:NO];

            vc.delegate=self;

            [self presentViewController:vc animated:YES completion:nil];
```

2.2.2.3.2 The Acceptto Biometric Authenticator delegate protocol and callbacks

For proper asynchronous operation, the Acceptto Biometric Authenticator defines a delegation protocol (**AccepttoTouchAuthenticatorViewControllerDelegate**) with a callback for the host app to act upon the end of operation by the user:

```
@protocol AccepttoTouchAuthenticatorViewControllerDelegate <NSObject>
- (void)finishedTouchAuthentication:(AccepttoTouchAuthenticatorViewController *)controller
accessPermitted:(BOOL)permitted error:(NSError *)error;
@end
```

2.2.2.3.2.1 Acting upon the completion of the biometric authentication operation

Implement the following method in the delegate class:

```
- (void)finishedTouchAuthentication:(AccepttoTouchAuthenticatorViewController *)controller
accessPermitted:(BOOL)permitted error:(NSError *)error;
```

This method will be called when the biometric authentication operation is finished by the user. Two parameters will be received:

- A boolean value **permitted** which defines if the user successfully authenticated, and thus his access is permitted.
- An NSError object **error** which describes what went wrong with the process, if not permitted. If there was no error, this object will be nil. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
- (void)finishedTouchAuthentication:(AccepttoTouchAuthenticatorViewController *)controller
accessPermitted:(BOOL)permitted error:(NSError *)error {

    [self doProceduresAfterPresentingAuthenticatorWithResultsPermitted:permitted error:error];

}
```

2.2.2.4 Utility Methods

The Acceptto Authenticators Framework provides some utility methods to analyse the iOS device in which the app is running. These static methods are part of the **AccepttoAuthenticatorSystemInfo** class.

2.2.2.4.1 Check if the device has capable hardware for biometric authentication

The Acceptto Authenticators Framework provides an easy method to check if a device is capable of biometric authentication (e.g. is capable of TouchID or FaceID).

This check may be performed using the following function:

```
+ (BOOL) isBiometricIDAvailableOnDevice;
```

The function returns a boolean value that indicates if the device has a hardware biometric identification sensor available.

2.2.2.4.2 Check if the device has its biometric authentication active and configured, and permission is given to the host app

The Acceptto Authenticators Framework provides an easy method to check if the host device has an active and configured biometric authentication process, and if the host app can use it.

This check may be performed using the following function:

```
+ (BOOL) isBiometricIDActiveAndEnrolledOnDevice;
```

The function returns a boolean value that indicates if the host app can perform biometric authentication on the device.

2.2.2.4.3 Get the host iOS device model name

The Acceptto Authenticators Framework provides an easy method to check the model name of the app's host iOS device.

The model's name is returned by the following function:

```
+ (NSString *)getDeviceName;
```

The function returns a ready-to-display string (e.g. "iPad Air 2" or "iPhone X")

2.2.2.4.4 Get the host iOS operating system version

The Acceptto Authenticators Framework provides an easy method to get the version of iOS running on the host device.

The version's string is returned by the following function:

```
+ (NSString *)getiOSVersion;
```

The function returns a ready-to-display string (e.g. "11.3.2")

2.2.2.4.5 Get the host iOS app version

The Acceptto Authenticators Framework provides an easy method to get the version and build of its running host app.

The version's string is returned by the following function:

```
+ (NSString *)getAppVersion;
```

The function returns a ready-to-display string, in the format "V build B", in which V is the version of the app, and B is the build number (e.g. "1.2.2 build 7")

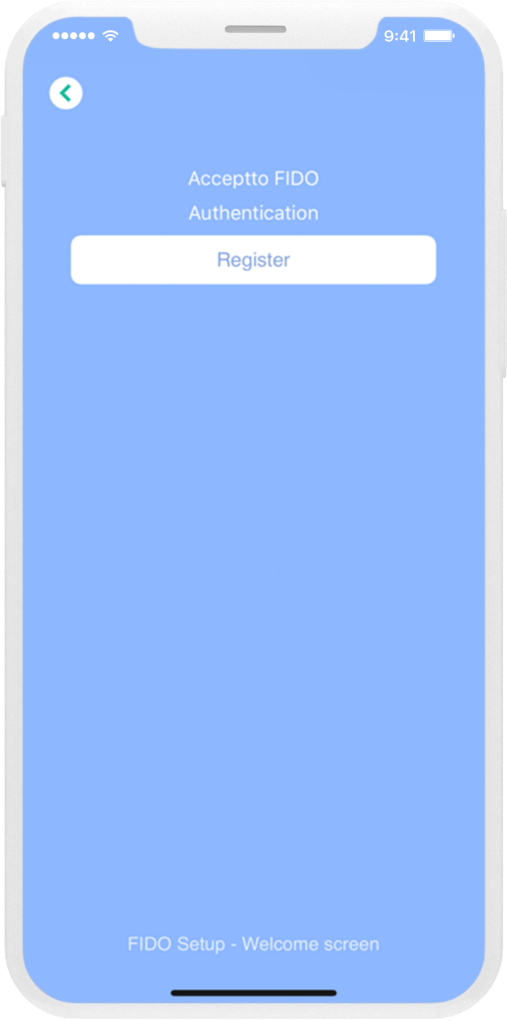
2.2.3 Acceptto FIDO Manager Framework Implementation Details

The Acceptto FIDO Manager Framework provides an easy and streamlined way to present the user a complete passwordless authentication system. It includes authentication setup, authenticator selection and fido server communication.

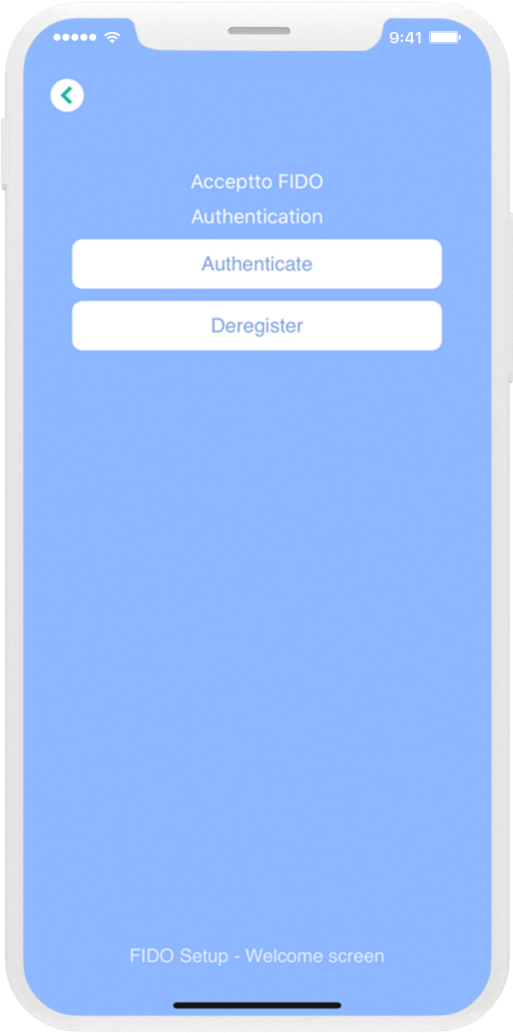
2.2.3.1 Operation

The Acceptto FIDO Manager Framework, when invoked, will operate in the following sequence:

1. **The initial screen**
    a. If the user isn't registered, there will be a "Register" button on screen, which will start the Register process.

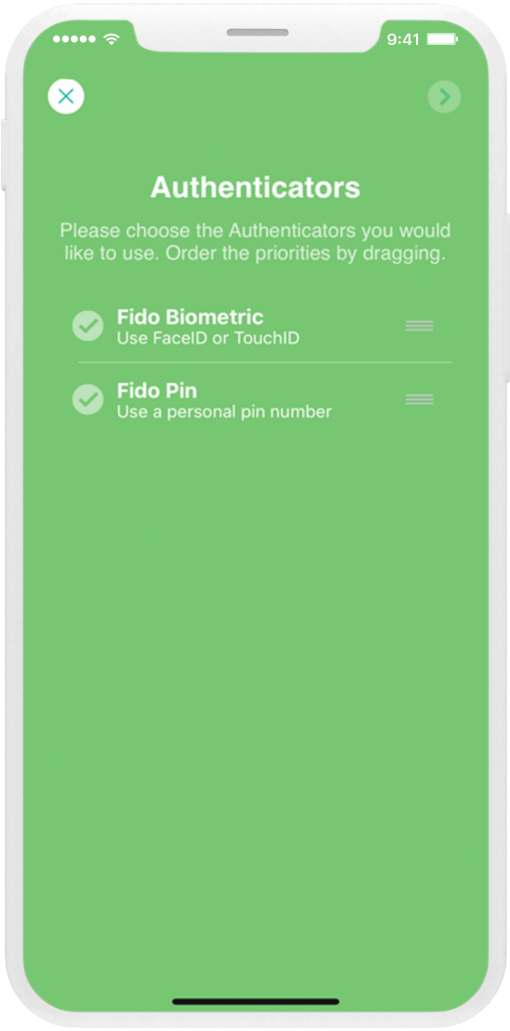


    b. If the user is already registered, there will be a "Authenticate" button on screen, which will start the authentication process. There may or may not be a "Deregister" button in this screen (please check below for more information on this)



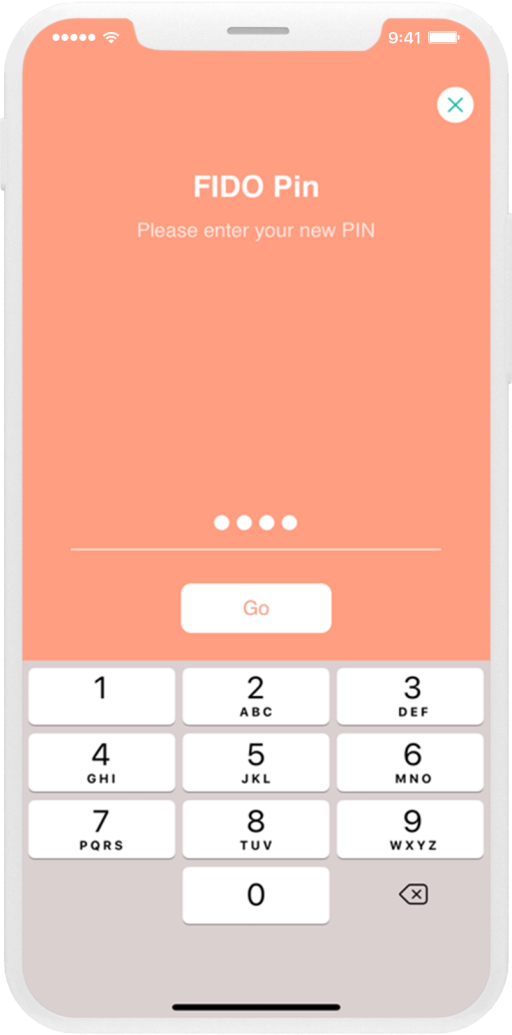
2. The Register process

a. When the register process starts, the user will be presented with a list of all authenticators his iOS device supports. At the moment, two authenticators are supported by Acceptto FIDO Manager: Pin and Biometric (TouchID or FaceID).



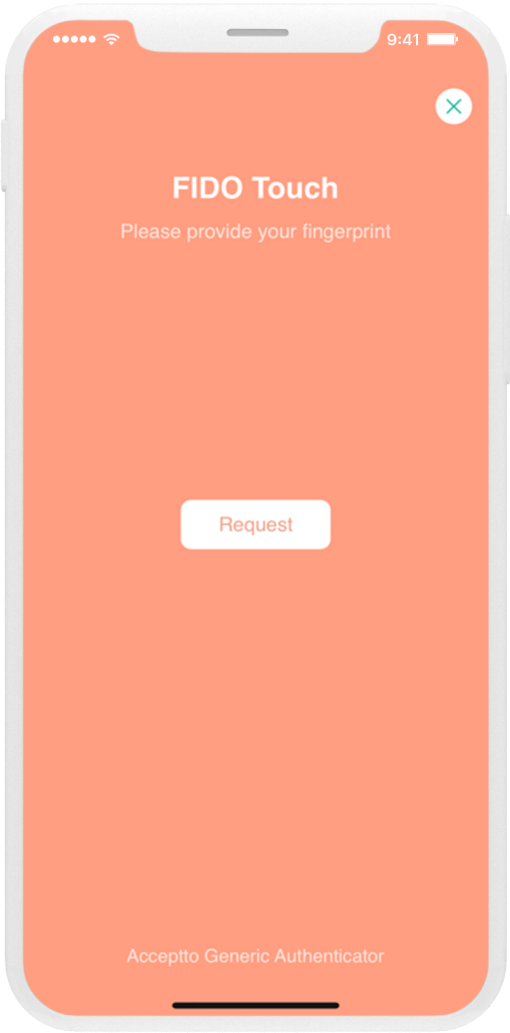
The user can then select which authenticators he wishes to register, and the order they will be presented when authenticating.

b. To register the Pin authenticator, the user will be prompted to choose his desired pin, and repeat it correctly:



The user's pin will then be securely stored.

c. To register the Biometric authenticator, the user will be prompted to authenticate in the iOS device with his fingerprint or face:



3. **The Authentication Process**
    a. The user will be prompted to authenticate with the authenticators he selected, in the order he selected when registering. Please note that, to complete the authentication process, he will only have to correctly authenticate in one of the authenticators. This means that if the user successfully authenticates in the first authenticator that is presented to him, authentication with other authenticators will not be required.

b. To authenticate with the Pin authenticator, the user be prompted to type the correct pin he chose when registering.
c. To authenticate with the Biometric authenticator, the user will be prompted to authenticate in the iOS device with his fingerprint or face.

4. **The Deregister Process**
   a. In the Deregister process, all registered authenticators from the user will be deregistered.

2.2.3.2 Importing the framework headers

To use the Acceptto FIDO Manager Framework's methods, please import AccepttoFidoManagerFramework.h:

```
#import <AccepttoFidoManagerFramework/AccepttoFidoManagerFramework.h>
```

2.2.3.3 Invoking and using the Acceptto FIDO Manager View Controller

To start the FIDO Manager user interface, you just instantiate and invoke the Acceptto FIDO Manager View Controller.

2.2.3.3.1 Invoking the Acceptto FIDO Manager View Controller

The AccepttoFidoManagerViewController is provided as a subclass of UIViewController. The host app will have control over its presence on screen.

To instantiate the AccepttoFidoManagerViewController, use the following method:

```
- (id)initWithUsername:(NSString *)userName showUnenrollButton:(BOOL)shouldShowUnenroll;
```

This instantiates an AccepttoFidoManagerViewController for the specified username

- If the user didn't register yet, a screen will appear prompting the user to start the registration process
- If the user already registered one or more authenticators, a screen will appear with an authenticate button to start the authentication process

- If you'd like a Deregister button to be present in this screen (allowing the user to deregister the already registered authenticators), set shouldShowUnenroll to YES. If you'd like the user to only have the option to start authentication in this screen, set shouldShowUnenroll to NO.

If you try to instantiate an AccepttoFidoManagerViewController with an invalid username according to the FIDO protocol specification, this method will return nil.

Please note that the usual **init** method with no parameters is unavailable for this class.

Example usage from the sample app code:

```
    AccepttoFidoManagerViewController *vc=[[AccepttoFidoManagerViewController alloc]
initWithUsername:_txtUsername.text showUnenrollButton:YES];

    if (vc) {

        //Documentation cross-reference --> DocRef#024

        vc.delegate=self;

        [self showViewController:vc sender:self];

    }

    else {

        [self showAlertWithTitle:@"Error" andMessage:@"Invalid Username"];

    }
```

2.2.3.3.2 The Acceptto FIDO Manager delegate protocol and callbacks

For proper asynchronous operation, the Acceptto FIDO Manager defines a delegation protocol (**AccepttoFidoManagerViewControllerDelegate**) with callbacks for the host app to act upon the end of operation for each operation mode:

```
@protocol AccepttoFidoManagerViewControllerDelegate <NSObject>
@optional
- (void)finishedEnrolling:(AccepttoFidoManagerViewController *)controller success:(BOOL)success
errors:(NSArray *)errors;
- (void)finishedAuthenticating:(AccepttoFidoManagerViewController *)controller
accessPermitted:(BOOL)permitted errors:(NSArray *)errors;
- (void)finishedUnenrolling:(AccepttoFidoManagerViewController *)controller success:(BOOL)success
error:(NSError *)error;
@end
```

2.2.3.3.2.1 The Fido Manager Framework Delegate's NSDictionary structure for errors

To describe the produced errors, a NSDictionary structure is used, with the following key/value pairs:

- Key: @"authenticatorName"
  - Contains an NSString with the internal name of the authenticator which produced the error
- Key: @"authenticatorFriendlyName"
  - Contains an NSString with the friendly name (ready to be shown to the user) of the authenticator which produced the error.
- Key: @"authenticatorError"
  - Contains an NSError object with the specific produced error

2.2.3.3.2.2 Acting upon the completion of the Register operation

Implement the following method in the delegate class:

```
- (void)finishedEnrolling:(AccepttoFidoManagerViewController *)controller success:(BOOL)success
errors:(NSArray *)errors;
```

This method will be called when the Register operation is finished by the user. Two parameters will be received:

- A boolean value **success** which defines if the user successfully chose at least one authenticator and registered with it
- An NSArray object **errors** which contains all the errors that occurred during the process. If there was no error, this object will be nil. Each error will be contained in an Acceptto Fido Manager NSDictionary structure for errors. For more information about

this structure, please refer to point 1 above. Refer to this document's **error list** section for the possible errors.

- Example usage from the sample app code:

```
- (void)finishedEnrolling:(AccepttoFidoManagerViewController *)controller success:(BOOL)success
errors:(NSArray *)errors {

    [self dismissViewControllerAnimated:YES completion:^{

        if (success) {

            [self showAlertWithTitle:@"Register was successful" andMessage:nil];

        }

        else {

            NSString *friendlyReport=[self getFriendlyReportFromErrorsArray:errors];

            [self showAlertWithTitle:@"Register failed" andMessage:friendlyReport];

        }

    }];

}
```

2.2.3.3.2.3 Acting upon the completion of the Authentication operation

Implement the following method in the delegate class:

```
- (void)finishedAuthenticating:(AccepttoFidoManagerViewController *)controller
accessPermitted:(BOOL)permitted errors:(NSArray *)errors;
```

This method will be called when the Authentication operation is completed. Two parameters will be received:

- A boolean value **permitted** which defines if the user successfully authenticated with one of the registered authenticators
- An NSArray object **errors** which contains all the errors that occurred during the process. If there was no error, this object will be nil. Each error will be contained in an

Acceptto Fido Manager NSDictionary structure for errors. For more information about this structure, please refer to point 1 above. Refer to this document's **error list** section for the possible errors.

- Example usage from the sample app code:

```
- (void)finishedAuthenticating:(AccepttoFidoManagerViewController *)controller
accessPermitted:(BOOL)permitted errors:(NSArray *)errors{

    [self dismissViewControllerAnimated:YES completion:^{

        if (permitted) {

            [self showAlertWithTitle:@"Access Granted" andMessage:nil];

        }

        else {

            NSString *friendlyReport=[self getFriendlyReportFromErrorsArray:errors];

            [self showAlertWithTitle:@"Access Refused" andMessage:friendlyReport];

        }

    }];

}
```

2.2.3.3.2.4 Acting upon the completion of the Deregister operation

Implement the following method in the delegate class:

```
- (void)finishedUnenrolling:(AccepttoFidoManagerViewController *)controller success:(BOOL)success
error:(NSError *)error;
```

This method will be called when the Deregister operation is completed. Two parameters will be received:

- A boolean value **success** which defines if all of the user's previously registered authenticators were successfully deregistered in the server

- An NSError object **error** which describes what went wrong with the process, if not completed. If there was no error, this object will be nil. Refer to this document's **error list** section for the possible errors.

Example usage from the sample app code:

```
- (void)finishedUnenrolling:(AccepttoFidoManagerViewController *)controller success:(BOOL)success
error:(NSError *)error {

    [self dismissViewControllerAnimated:YES completion:^{

        if (success) {

            [self showAlertWithTitle:@"Deregister was successful" andMessage:nil];

        }

        else {

            [self showAlertWithTitle:@"Deregister failed" andMessage:error.localizedDescription];

        }

    }];

}
```

2.2.3.4 Deregistering all authenticators for a user

The FIDO Manager Framework provides an easy way to deregister all authenticators that are registered on the server for a particular user, without instantiating the view controller. This procedure may be useful when a user closes his account on the host app, resets his preferences, or even logs out, depending on the intended behavior for the processes.

To deregister the authenticators, use the following static method in AccepttoFidoManagerViewController:

```
+ (void)performDeregistrationForUsername:(NSString *)userName withSuccess:(void (^)(void))success
fail:(void(^)(NSError *error))fail;
```

Success and fail blocks are provided for easy asynchronous operation.

Example usage from the sample app code:

```
[AccepttoFIDOCore performDeregistrationForUsername:userName withSuccess:^{

        [self addMessageToLogWithTitle:@"INFO" message:@"Success deregistering on server" color:[UIColor
whiteColor]];

        [self actionCheckUser:nil];

        [self enableAllControls:YES];

    } fail:^(NSError *error) {

        [self enableAllControls:YES];

        [self addMessageToLogWithTitle:@"ERROR" message:[NSString stringWithFormat:@"Error performing
deregistration on server: %@",error.localizedDescription] color:[UIColor redColor]];

    }];
```

2.2.4 Acceptto FIDO iOS Toolkit Error List

2.2.4.1 Local errors generated by the Acceptto FIDO Core Framework

Please refer to the following table for information on the local errors generated by the Acceptto FIDO Core Framework:

Error Code	Error Description	Observations
1001	Invalid Username	Tried to make a FIDO request with an invalid username specified. The username must conform to the rules according to the FIDO protocol specification.
1002	Empty request from server	The FIDO server did not return a valid request upon requiring the start of a registration or authentication operation. You should check that the FIDO server is fully conform to the FIDO protocol specification.
1003	Empty response from server	The FIDO server did not return a valid response upon requiring the completion of a registration, authentication or deregistration operation. You should check that the FIDO server is fully conform to the FIDO protocol specification.
1004	Invalid AAID	Tried to make a FIDO request with an invalid authenticator AAID specified. The authenticator AAID string must conform to the rules according to the FIDO protocol specification.
1005	Data not found for current operation	No data was found for starting the current operation, though it was saved previously. This may mean that the some data is no

		longer on the device, and registration should be performed again.

2.2.4.2 Local errors generated by the Acceptto Authenticators Framework

Please refer to the following table for information on the local errors generated by the Acceptto Authenticators Framework:

Error Code	Error Description	Observations
2001	User canceled authentication	The user closed the authenticator's screen, thus canceling authentication
2002	Incorrect pin after maximum number of retries	Pin authentication failed because the user typed an incorrect pin the maximum number of times allowed.
2003	Biometric ID permission unavailable	Biometric authentication couldn't be performed because the user didn't give the app permission to access the biometric sensor features of the device.

2.2.4.3 Remote errors generated by the FIDO server

When the FIDO server returns an error, it will be returned by the Fido iOS Toolkit Frameworks, untouched.

3. Understanding the Sample App

The Acceptto Fido iOS Toolkit Sample App is comprised of two 'separate apps' in one:

3.1 The Acceptto Fido Manager Framework Wizard

To access this section of the app, use the "Fido Wizard" button on the main screen, after typing the username in the box above. This will use the Acceptto FIDO Manager framework to start the wizard and provide the user with enroll or authenticate/unenroll buttons.

A host app that uses the Acceptto FIDO iOS Toolkit will usually only need this button to invoke the FIDO authentication.

3.2 The Acceptto FIDO Core test board

To access this section of the app, use the "Fido Testboard" button on the main screen. This can be used to test all functionality of direct communication with the Acceptto FIDO server.

This section of the app doesn't use the Acceptto FIDO Manager, e.g. doesn't manage which authenticators each user has chosen. For this reason, it can be used to test the results produced by the wizard, but not the other way around.

A host app that uses this toolkit isn't supposed to have a 'fido testboard' button. This is only for testing and example purposes.

4. Best Practices

Here is a short list of best practices that the host app should follow when implementing the Acceptto FIDO iOS Toolkit Frameworks:

- The toolkit can deal with any kind of values, but it's recommended to have validations on the host app, such as username validation. Please note that the frameworks provide easy to use functions to validate input data.
- Verify internet connectivity/reachability before every call to the Core framework

5. Revision History

Date	Version	Revision	Revised By
6/15/2018	1.0	Initial Release	Jorge Coelho
6/20/2018	1.1	Revision of frameworks functionality and documentation	Jorge Coelho
7/9/2019	1.2	Revision of frameworks functionality and documentation	Jorge Coelho