MVIDDeciveConnector

Generated by Doxygen 1.8.5

Thu Mar 27 2014 11:02:41

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

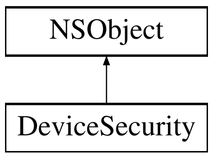
Chapter 3

Class Documentation

3.1 DeviceSecurity Class Reference

```
#import <DeviceSecurity.h>
```

Inheritance diagram for DeviceSecurity:



Instance Methods

- (int) - registerDevice:
- (int) - applicationLogin:
- (BOOL) - doLogin:parentViewController:altSubView:
- (BOOL) - doLogin:parentViewController:
- (void) - setLoginCenterDisplacement:
- (void) - excludeLoginGroup:
- (void) - includeAllLoginGroups
- (void) - releaseDeviceRegistration
- (void) - releaseApplicationUsage:

Protected Types

- enum ServiceResult {
  Success = 1, InvalidMVSessionID = 2, AccessDenied = 4, ApplicationBorrowTimeExpired = 8,
  DeviceBorrowTimeExpired = 16, ServiceFault = 32, ServiceSuccess = 64, NetworkError = 128 }

Protected Attributes

- LoginViewController ∗ **login_vc**

Properties

- NSString ∗ **mv_session_id**

- NSString ∗ **access_identifier**
- NSString ∗ **device_hash**
- NSNumber ∗ **device_borrow_time**
- NSError ∗ **network_error**
- NSString ∗ **service_res_msg**
- int **service_res_code**

3.1.1 Detailed Description

DeviceSecurity is the client side interface towards the corresponding MVID seb service `DeviceSecurity`. It is designed for easy integration into existing iOS apps. Once the interface is properly integrated into an app it will provide:

- MVID User authentication
- Product access services
- MVID Session ID for further requesting into MVID's web services: `MVID Services`

3.1.2 Usage

The primary method of DeviceSecurity is the doLogin: method. In a normal integration this method should be all that is needed.

The basic integration has 3 steps:

- Instantiate DeviceSecurity as a member of the class responsible for the login process.
- Add a notification observer to listen for the final asynchronious login response "MVIDLoginResponseReady"
- Call doLogin:

3.1.2.1 MVIDLoginResponseReady notification object

The MVIDLoginResponseReady notification is posted via the default notification center with a NSObject as userInfo (user data). The posted userInfo defines the following attributes:

First Header	Second Header	Description
NSNumber	service_result	Bitwise combination of ServiceResult flags
NSNumber	has_access	1 if the access_identifier is granted 0 if not
NSNumber	service_res_code	The server side result code
NSString	service_res_msg	The server side result message

3.1.2.2 Coding example

The following is a ver simple example of how a UIViewController with MVID Login responsibility could be coded.

AppLoginViewController.h:

```
// Very basic header
#import <UIKit/UIKit.h>
@class DeviceSecurity;
@interface AppLoginViewController : UIViewController {
    DeviceSecurity *device_security;
    NSString *cur_mv_session_id;
}

@property(copy) NSString *cur_mv_session_id;

@end
```

AppLoginViewController.mm:

```
#import "AppLoginViewController.h"
#import "DeviceSecurity.h"
@interface AppLoginViewController ()
@end

@implementation AppLoginViewController

@synthesize cur_mv_session_id = _cur_mv_session_id;


// Initializing MVID DeviceSecurity
- (void)viewDidLoad
{
    // Custom initialization
    device_security = [[DeviceSecurity alloc] init];
    // Some pre-configurations (only want school logins)
    [device_security excludeLoginGroup:@"private"];
    [device_security excludeLoginGroup:@"company"];
    // Start observing notifications named "MVIDLoginResponseReady"
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(loginResponse:) name:@"MVIDLo
      ginResponseReady" object:nil];
    [super viewDidLoad];
}


// Notification callback method
- (void) loginResponse:(NSNotification *) notification {
    // Extract interesting notification values
    int service_result = [[notification.userInfo valueForKey:@"service_result"] intValue];
    BOOL has_access = [[notification.userInfo valueForKey:@"has_access"] boolValue];
    int service_res_code = [[notification.userInfo valueForKey:@"service_res_code"] intValue];
    NSString *service_res_msg = [notification.userInfo valueForKey:@"service_res_msg"];
    // copy MVID Session ID into a class local property.
    cur_mv_session_id = device_security.mv_session_id;
    // Just log some results
    NSLog(@"service_res_code: %d", service_res_code);
    NSLog(@"service_res_msg: %@", service_res_msg);
    NSLog(@"mv_session_id: %@", cur_mv_session_id);
    NSLog(@"Access granted: %d", has_access);
    NSLog(@"service_result: %d", service_result);
}


// Something triggered the MVID Login process (in this case a button was touched)
- (IBAction)startLogin:(id)sender {
    [device_security doLogin:@"product.ios.ml.myapp"
        parentViewController:self];
}

@end
```

3.1.3 Member Enumeration Documentation

3.1.3.1 - (enum) ServiceResult [protected]

Following values apply to the DeviceSecurity web service. In the descriptions below it is specified which methods in the DeviceSecurity class that can recieve which results directly.

When using doLogin: both registerDevice: and applicationLogin: are candidates for being called, therefore all ServiceResult codes are bitwise candidates of the final login result, which is posted in the MVIDLoginResponse-Ready notification object via the default notification center.

Read more about Apple's notification system here: Notification Programming Topics.

Enumerator

Success Successful operation.

- registerDevice:
  - registerDevice method has been succesfully queried.
- applicationLogin:
  - The registered user has succesfully queried the applicationLogin method

InvalidMVSessionID Invalid MV Session ID.

- registerDevice:
  - The mv_session_id passed to registerDevice is invalid. Obvious reason would be that it is too old and therefore timed out on the server side.

AccessDenied Application access denied.

- applicationLogin:
  - The server was contacted and device hash is good, but the specific user mapped to the device does not have access to the application.

ApplicationBorrowTimeExpired Application borrow-time has expired.

- applicationLogin:
  - Either the device or the MV-ID server is telling you that the time since last applicationLogin has expired the borrow-time allowed for the application.

DeviceBorrowTimeExpired Device borrow-time has expired.

- applicationLogin:
  - Either the device or the MV-ID server is telling you that the time since last applicationLogin has expired the borrow-time allowed for the device registration. This meens that the device is no longer mapped to a user on MV-ID and a login is requiered.

Note

If device has no device hash registered this ServiceResult is also returned.

ServiceFault Fault occured while calling the Device Security service.

- registerDevice:
- applicationLogin:
  - Some fault occured during the invocation of a service method. Info about the fault is available in the response object itself.

ServiceSuccess ServiceSuccess meens that the a service method invocation succeeded with no faults. This does not necessarily meen that a device was registered succesfully or that access was granted to an application successfully.

NetworkError Network error occured.

- registerDevice:
- applicationLogin:
  - A network transport error has occured. Probably no link or other kind of network related problem like missing DNS service. Service requests timeout after 2 seconds, so connection problems should not "hang" the application.

3.1.4 Method Documentation

3.1.4.1 - (int) applicationLogin: (NSString $*$) *access_identifier*

Register the device for application usage. This method will only succeed on a properly registered device (see registerDevice:).

It should not be necessary to call this method from a third-party application since it is automatically invoked from DeviceSecurity:doLogin: if needed.

Parameters

access_identifier	The access_identifier identifying the application being started.

Returns

bitwise combination of DeviceSecurity.ServiceResult

3.1.4.2 - (BOOL) doLogin: (NSString ∗) *access_identifier* parentViewController:(UIViewController ∗) *vc*

Start the login process. This is a convenience method that calls doLogin with alt_sub_view as nil. This means that the parentViewController's root view will be used as parent view for the login UI.

Example usage:

```
// In this example LoginViewController's view are placed on top
// of the parent view controller's root view.
BOOL login_view_displayed =
   [device_security doLogin:current_access_identifier
       parentViewController:self];
```

Parameters

access_identifier	The access_identifier that needs to be checked.
parentView-Controller	The visible part of the login process is controlled by the LoginViewController which should be owned by a UIViewController in the applications domain. Therefore a parent UIViewController is mandatory in order to display the login UI. It is legal to pass nil if you just want to check if the user already has access and need to know that the login UI is not shown.

Returns

(Has the login view been displayed) This value does *NOT* state if the login was a success or not, the actual result of the login is sent via the notificationCenter in the MVIDLoginResponseReady notification. The return value only says whether or not a login view has been displayed. For example:

- If for instance the user calls doLogin with a nil parentViewController, and the device is not registered, then doLogin cannot display the Login UI. This means that FALSE is returned and a MVIDLoginResponse-Ready notification can be posted right away telling the calling function that the login gave no access.
- If a device is not registered (or has not been online for a while) and a parentViewController is passed then the LoginViewController is displayed and TRUE is returned.

3.1.4.3 - (BOOL) doLogin: (NSString ∗) *access_identifier* parentViewController:(UIViewController ∗) *vc* altSubView:(UIView ∗) *alt_sub_view*

Start the login process, communicating with MVID backend servers if online, else if any previous login has been performed and is still within the "borrow time" grace period, this will be used directly off the device. This means that a single login operation can be reused in offline mode some time.

Note

To force device to re-login you must call resetDeviceRegistraion first and be sure to pass a parentView-Controller. To direct the login screen to a subview in your view controller use the altSubView parameter.

Example usage:

```
// In this example the LoginViewController's views are displayed in a
// overlay subview of my LoginController called myLoginView
BOOL login_view_displayed =
   [device_security doLogin:current_access_identifier
       parentViewController:self
                 altSubView:self.myLoginView];
if (login_view_displayed) {
   // display and fade in my overlay view
   self.myLoginView.hidden = NO;
   [UIView animateWithDuration:0.25
                    animations:^{self.myLoginView.alpha = 1.0;}];
}
```

Parameters

access_identifier	The access_identifier that needs to be checked.
parentView-Controller	The visible part of the login process is controlled by the LoginViewController which should be owned by a UIViewController in the applications domain. Therefore a parent UIViewController is mandatory in order to display the login UI. It is legal to pass nil if you just want to check if the user already has access and need to know that the login UI is not shown.
altSubView	If nil is passed the then the LoginViewController's root UIView will be a sub view of the parent-ViewController's root UIView. By passing an alternate UIView you can control where the login UI should be displayed in your UIViewController's view hierarchi.

Returns

(Has the login view been displayed) This value does *NOT* state if the login was a success or not, the actual result of the login is sent via the notificationCenter in the MVIDLoginResponseReady notification. The return value only says whether or not a login view has been displayed. For example:

- If for instance the user calls doLogin with a nil parentViewController, and the device is not registered, then doLogin cannot display the Login UI. This means that FALSE is returned and a MVIDLoginResponse-Ready notification can be posted right away telling the calling function that the login gave no access.
- If a device is not registered (or has not been online for a while) and a parentViewController is passed then the LoginViewController is displayed and TRUE is returned.

3.1.4.4 - (void) excludeLoginGroup: (NSString ∗) *login_group*

Exclude a login group from the login UI.

Example usage:

```
// Exclude the school login group
[device_security excludeLoginGroup:@"company"];
```

Parameters

login_group	The name of the login group. At the time of writing possible groups are: school, company and private

3.1.4.5 - (void) includeAllLoginGroups

Re-include all login groups. Use this if you have already excluded some login groups and want to include them all again without re-instantiating DeviceSecurity (which is discouraged)

Example usage:

```
// Include all login groups
[device_security includeAllLoginGroups];
```

3.1.4.6 - (int) registerDevice: (NSString ∗) *mv_session_id*

Register a device at MVID with a valid mv_session_id using `DeviceSecurity` web service.

It should not be necessary to call this method from a third-party application since it is automatically invoked from doLogin if needed.

Parameters

mv_session_id	The MVID session ID which should be associated with the device.

Returns

bitwise combination of DeviceSecurity.ServiceResult

3.1.4.7 - (void) releaseApplicationUsage: (NSString ∗) *access_identifier*

Release the device-side knowledge about whether or not access has been granted to an application. This will force the application to do a server-side check next time doLogin is used.

This will not throw away the device registration, so re-login by user is not needed next time the device is online.

Example usage:

```
// Forget whether access is granted or not to MyApp
[device_security releaseApplicationUsage:@"products.ios.ml.myapp"]ß ;
```

Parameters

access_identifier	The access_identifier to be released.

3.1.4.8 - (void) releaseDeviceRegistration

Release the device-side knowledge about MVID within the app's sandbox forcing a new login from the user.

Calling this method will not affect other apps using MVIDDeviceConnector.

Example usage:

```
// Release device-side knowlegde of MVID
[device_security releaseDeviceRegistration];
```

3.1.4.9 - (void) setLoginCenterDisplacement: (int) *pixels*

Move the center point of the login UI n pixels to the left or right depending on the numeric sign.

Example usage:

```
// Horizontally displace the login 40 pixels to the left
[device_security setLoginCenterDisplacement:-40];
```

Parameters

pixels	The amount of pixels to displace the center with.

The documentation for this class was generated from the following file:

- dist/lib/include/DeviceSecurity.h

Index