OneLogin NAPPS SDK for iOS

www.onelogin.com | twitter.com/onelogin
OneLogin, Inc. | 150 Spear Street, Suite 1400, San Francisco, CA 94105

855.426.7227

OVERVIEW

Welcome to OneLogin's NAPPS software development kit. Contained within, you will find a few elements that will allow you develop and test the functionality of the Token Agent.

In the typical use-case, after successfully authenticating the a user in question, the Token Agent obtains additional security tokens representing that specific user from an identity server for other native applications. Our testing version contains a 'Mock Agent' that doesn't connect to any actual server or outside service, but operates as if one was successfully present, and will allow the testing suite to run successfully.

What you're getting with this testing suite:

- The Mock Token Agent
- The Sample Application
- An SDK Library for the test application to run on.

You'll also be getting the full source code for both the Mock Agent and the Sample Application, with lines of both contained on the next few pages.

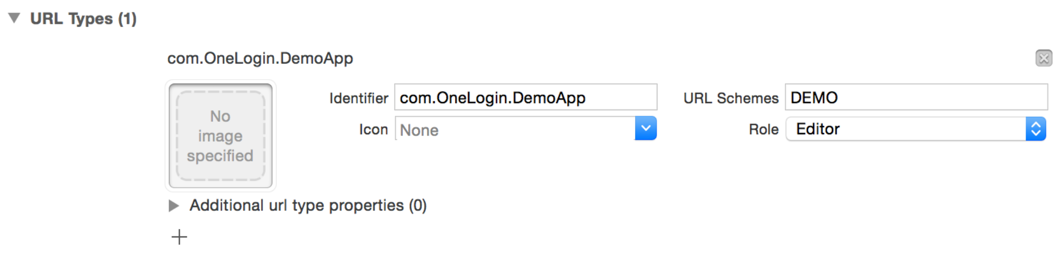
BEST PRACTICES: GETTING STARTED

It is highly recommended that you build out your sample application with the Mock Token Agent to ensure that all binding connectors are properly configured. Once you have established the correct usage flows, remove the mock token agent and install the the launcher on your device which should leverage the pre-configured settings established earlier.

Currently, the mobile NAPPs application connectors are not publicly available, so please contact napps-support@onelogin.com to have your connector configured.

CREATING A CLIENT

Begin by registering a custom URL scheme for your client application. As shown below, this URL scheme will be used in the token agent as part of the callback function to your application. Note that your URL scheme will not match the but will resemble the example below:


URL Types (1)

com.OneLogin.DemoApp

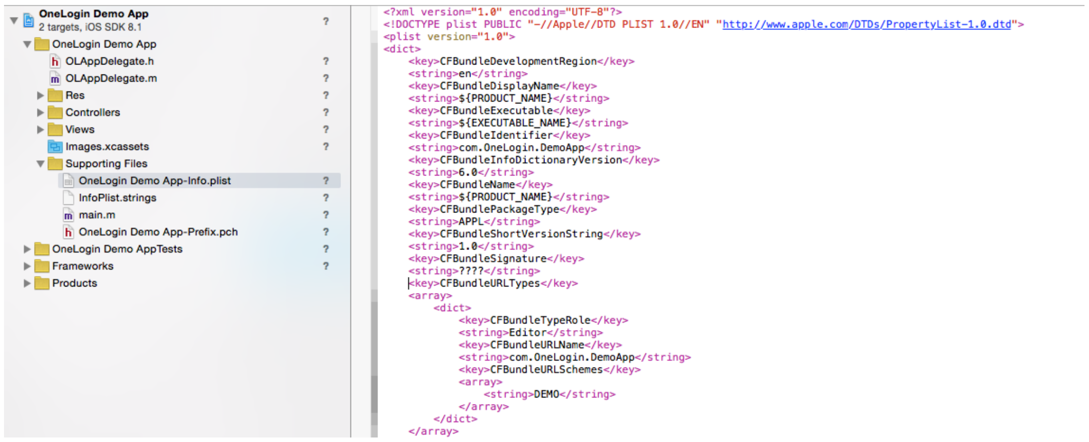
Identifier	com.OneLogin.DemoApp	URL Schemes	DEMO
Icon	None	Role	Editor

Additional url type properties (0)

FINDING THE CUSTOM URL IN XCODE

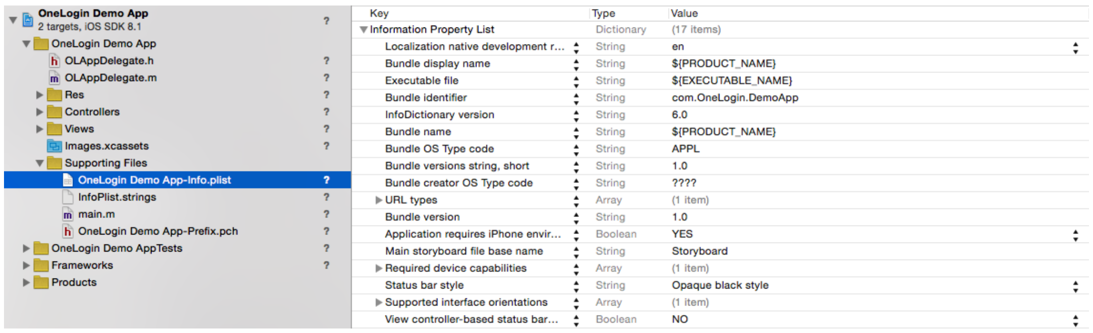
After loading the project into Xcode, finding the required custom URL scheme is fairly straightforward. What you'll be looking for is a filed called **OneLogin Demo App-Info.plist**, found under the directory path of **OneLogin Demo App > Supporting Files**.

Right-clicking on the .plist file and selecting **Open As > Source Code** will reveal the following:



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleDisplayName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundleExecutable</key>
    <string>${EXECUTABLE_NAME}</string>
    <key>CFBundleIdentifier</key>
    <string>com.OneLogin.DemoApp</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleURLTypes</key>
    <array>
        <dict>
            <key>CFBundleTypeRole</key>
            <string>Editor</string>
            <key>CFBundleURLName</key>
            <string>com.OneLogin.DemoApp</string>
            <key>CFBundleURLSchemes</key>
            <array>
                <string>DEMO</string>
            </array>
        </dict>
    </array>
```

Your custom URL will be found under the heading of **CFBundleURLName**, with the URL falling directly below it. Selecting **Show As > Property** List reveals the following options:



Key	Type	Value
Information Property List	Dictionary	(17 items)
Localization native development r...	String	en
Bundle display name	String	${PRODUCT_NAME}
Executable file	String	${EXECUTABLE_NAME}
Bundle identifier	String	com.OneLogin.DemoApp
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
URL types	Array	(1 item)
Bundle version	String	1.0
Application requires iPhone envir...	Boolean	YES
Main storyboard file base name	String	Storyboard
Required device capabilities	Array	(1 item)
Status bar style	String	Opaque black style
Supported interface orientations	Array	(1 item)
View controller-based status bar...	Boolean	NO

With the Demo App and Custom URL Configured, make sure you add the contents of the included Frameworks file, which will allow your sample application to compile and run properly.

MAKING A SECONDARY TOKEN REQUEST

With the client created, you will need to insert the Token Request Code found within the Test Request Endpoint into your own codebase. An example of this code will be found between lines 25 to 35 of the Test Request Endpoint and allows for the client to successfully make the second token request from the Token Agent.

*The **Test Token Agent** works exactly as the real token agent, except that it doesn't call any actual server and the token retrieved is just a random string.*

*On line 4, setTestMode will be set to **YES**. In production, it will be set to **NO** in order to call the actual Token Agent server.*

The result is that within the Mock Token Agent, the developer can trigger the different error codes from the Settings screen. If any error is sent, the Mock Token Agent won't sent the token and will send the error code instead.

The SDK on the third-party app automatically decodes this error and passes it to the developer.

```
// Call to request a token from the Mock Token Agent; also will be inserted into the developer's custom client.
- (IBAction)rightButtonTapped:(id)sender
{
   [[OneLogin SDK] setTestMode:YES]; //In production mode, this will be set to NO, instead of YES.

   NSError *error = [[OneLogin SDK] requestTokenByCheckingIfTokenAgentIsInstalledForAppWithURLScheme:URLScheme];

   if (error) {
      [[[UIAlertView alloc] initWithTitle:[NSString stringWithFormat:@"Error code %tu", error.code] message:[NSString
stringWithFormat:@"%@\n%@", error.localizedDescription, error.localizedRecoverySuggestion] delegate:nil cancelButtonTitle:@"Ok"
otherButtonTitles:nil, nil] show];
   }
}
```

RECEIVING A SECONDARY TOKEN FROM AGENT

When the token agent has successfully retrieved a secondary token or has encountered a failing error, it will use your Identifier (defined previously) as a callback to your application, passing you a token or an error.

If you receive an error, this is the list of error codes you are likely to receive:

OLNAPPSErrorNetworkNotAvailable	= 1000,
OLNAPPSErrorIDPDidNotRespond	= 1100,
OLNAPPSErrorRequestingAppDoesNotHavePermission	= 1201,
OLNAPPSErrorSecondaryTokenRequestRejected	= 1202,
OLNAPPSErrorNoSessionActive	= 1204,
OLNAPPSErrorDeviceCouldNotBeEnrolled	= 1205,
OLNAPPSErrorTokenAgentNotInstalled	= 1300,
OLNAPPSErrorNoSessionActive	= 1500,
OLNAPPSErrorDeviceCouldNotBeEnrolled	= 1600,
OLNAPPSErrorNone	= 0

The errors above are described below in more detail:

Error 1000 - The network is not available
Error 1100 - There was a timeout in the request or the server is down.
Error 1201 - a. The Token Agent gets a request for a second token
b. Verify that the Requesting App is in the current App Info list.
c. If it is, a secondary token for it is acquired.
d. If it is not, the App Info list is refreshed.
e. If after refreshing, the Registering App is still not in the App Info list, return this error.
Error 1202 - If the server responds with an error after requesting an updated App Info List or a Secondary Token
Error 1203 - Couldn't retrieve the App Info List.
Error 1204 - No current active session.
Error 1205 - Device could not be enrolled.
Error 1300 - The Token Agent is not installed on the device.
Error 1500 - The Token Agent does not have the information necessary to begin a session.
Error 1600 - The device could not be enrolled with the server.
Error 0 - Everything is running successfully.

With this information, you are now ready to run the test application.

RUNNING THE TEST APPLICATION

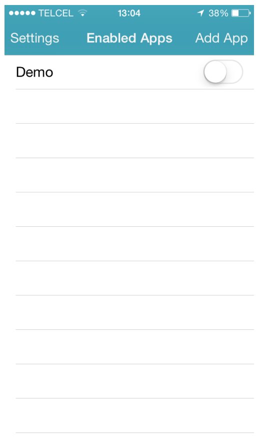
Running the test application is incredibly easy:

1. Load the demo application into your device. Right now iPhone is the current testing environment.
2. Once within the test SDK, select the toggle to enable the **Demo App**.
3. The Mock Token Agent can simulate the following error conditions.
   3.1. By selecting ‘Network not available (1000)’, the resultant errors become:
   - Network Not Available(1000)
   - IdP Did Not Respond(1100)

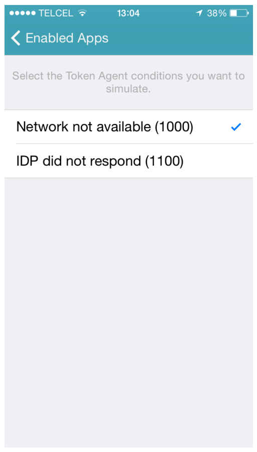
   3.2. By disabling the application in Step 2, or entering an incorrect App URL in step 4, the resultant errors become:
   - Permission Denied(1201)
4. When prompted to enter an **App Name**, enter ‘demo’, and for the **App URL**, enter ‘DEMO’. If you wish, this can be changed on line 13 of the Mock Token Agent source code under ‘URLscheme =‘.
5. With everything complete, the test application should run successfully and prompt you with a response saying that the secondary tokens was successfully generated by the Token Agent.

   When in Production, you will have your secondary token in your application to be used against a secondary resource server.
6. And with that you’re done!

2.



3.

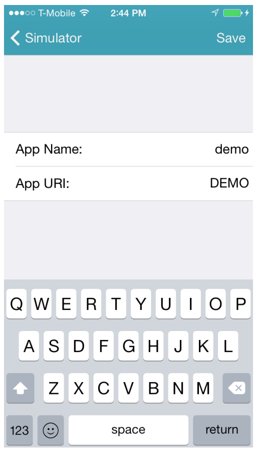


Select the Token Agent conditions you want to simulate.

Network not available (1000) ✓

IDP did not respond (1100)

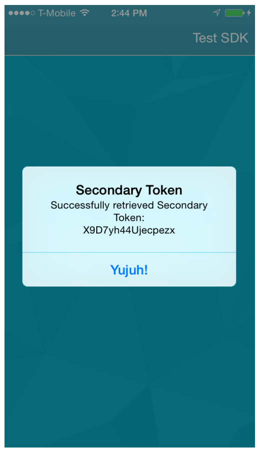
4.



App Name: demo

App URI: DEMO

5.



Secondary Token

Successfully retrieved Secondary Token: X9D7yh44Ujecpezx

The Test Application Source

```
// OLAppDelegate.m
// OneLogin Demo App
// Created by Oscar Swanros on 5/27/14.
// Copyright (c) 2014 OneLogin. All rights reserved.

#import "OLAppDelegate.h"

#import <OneLogin SDK/OneLogin.h>

@implementation OLAppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

    return YES;
}

// Custom URL called by Token Agent when it has data to return.
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
    NSLog(@"%@", url);

    [[OneLogin SDK] handleURL:url success:^(NSString *token) {

        NSLog(@"%@", token);

        [[[UIAlertView alloc] initWithTitle:@"Secondary Token" message:[NSString stringWithFormat:@"Successfully retrieved Secondary Token:\n%@", token] delegate:nil
cancelButtonTitle:@"Yujuh!" otherButtonTitles:nil, nil] show];
    } failure:^(NSError *error) {

        NSLog(@"%@", error);

        [[[UIAlertView alloc] initWithTitle:[NSString stringWithFormat:@"Error code %tu", error.code] message:[NSString stringWithFormat:@"%@\n%@", error.localizedDescription,
error.localizedRecoverySuggestion] delegate:nil cancelButtonTitle:@"Ok" otherButtonTitles:nil, nil] show];
    }];

    return YES;
}

@end
```

Test Application Endpoint for Secondary Token from Token

```
// OLViewController.m
// OneLogin Demo App
// Created by Oscar Swanros on 5/27/14.
// Copyright (c) 2014 OneLogin. All rights reserved.

#import "OLViewController.h"

#import <OneLogin SDK/OneLogin.h>

static NSString *const URLScheme = @"DEMO"; // This app URL Scheme

@implementation OLViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

// Call to request a token from the Mock Token Agent; also will be inserted into the developer's custom client.
- (IBAction)rightButtonTapped:(id)sender
{
    [[OneLogin SDK] setTestMode:YES]; //In production mode, this will be set to NO, instead of YES.

    NSError *error = [[OneLogin SDK] requestTokenByCheckingIfTokenAgentIsInstalledForAppWithURLScheme:URLScheme];

    if (error) {
        [[[UIAlertView alloc] initWithTitle:[NSString stringWithFormat:@"Error code %tu", error.code] message:[NSString stringWithFormat:@"%@\n%@", error.localizedDescription,
error.localizedRecoverySuggestion] delegate:nil cancelButtonTitle:@"Ok" otherButtonTitles:nil, nil] show];
    }
}

@end
```