

NMAAuthLib

MicroFocus/NetIQ NMAAuthLib is an iOS library which helps developer easily achieve single sign-on utilizing OAuth 2 spec. It implements the [native application profile](<http://tools.ietf.org/html/rfc6749#page-52>) and supports the end user authorization endpoint via an internal and external user agent.

Furthermore, it also supports the user credentials flow by prompting the end user for their username and password and uses that information directly to obtain an access token. For more information about how to choose the authentication flow, see the description of the delegate.

Getting Started

To get started, add the `NMAAuthLib.framework` file in your **Xcode** project's Frameworks section. Ensure that you have copy file & appropriate build Target selected. Go to your target and drag the framework file to the **Embedded binaries** section.

There are two different versions of the framework files, one for the simulator and the other for the devices. Ensure that you choose the appropriate one.

Using the NMAAuthLib

Configuring Your Client

One of the places to configure your client is `+[UIApplicationDelegate initialize]`. There you can call `-[NMAAuthLibManager setClientID:secret:authorizationURL:tokenURL:redirectURL:forAccountName:]` on the shared Account Manager for each service you want to have access to from your application. Services use the account name string, as an identifier for a certain service.

```
+ (void)initialize
{
    [[NMAAuthLibManager sharedInstance] setClientID:@"myClientID"
```

```

secret:@"mySecret"
authorizationURL:[NSURL URLWithString:@"https://your authz URL..."]
tokenURL:[NSURL URLWithString:@"https://your token URL..."]
redirectURL:[NSURL URLWithString:@"https://your redirect URL..."]
forAccountName:@"myOAuth2Service"];
}

```

Content-Type

Unless otherwise specified, the token request will be called with a HTTP Header 'Content-Type' set to 'multipart/form-data'. You must modify the custom header, if you wish that header to be set to 'application/x-www-form-urlencoded'.

```

NSMutableDictionary *configuration = [NSMutableDictionary
dictionaryWithDictionary:[[NMAAuthLibManager sharedInstance]
configurationForAccountName:kOAuth2accountName]];

```

```

NSDictionary *customHeaderFields = [NSDictionary
dictionaryWithObject:@"application/x-www-form-urlencoded" forKey:@"Content-
Type"];

```

```

[configuration setObject:customHeaderFields
forKey:kNMAAuthAccountManagerConfigurationCustomHeaderFields];

```

```

[[NMAAuthLibManager sharedInstance] setConfiguration:configuration
dforAccountName:kOAuth2accountName];

```

Requesting Access to a Service

After you have configured your client, you are ready to request access to one of those services. The NMAAuthLibManager provides three different methods for this:

- Username and Password

```

[[NMAAuthLibManager sharedInstance]
signInToAccountWithName:@"myOAuth2Service"

```

```

username:userName

```

```
password:password];
```

- External Browser

```
[[NMAAuthLibManager sharedInstance]
signInToAccountWithName:@"myOAuth2Service"];
```

If you are using an external browser, your application needs to handle the URL you have registered as an redirect URL for the account name (e.g. a custom URL scheme like `myappscheme://oauth`). The service will redirect to that URL after the authentication process.

pass the redirect URL to

```
[[NMAAuthLibManager sharedInstance] handleRedirectURL:url];
```

- Provide an Authorization URL Handler

```
[[NMAAuthLibManager sharedInstance]
signInToAccountWithName:@"myOAuth2Service"
withPreparedAuthorizationURLHandler:^(NSURL *preparedURL) {
    // Open a web view or similar
}];
```

Using an authorization URL handler gives you the ability to open the URL in an own web view or do some custom things for authentication. Therefore, you pass a block to the `NMAAuthLibManager` while requesting access.

One method for receiving a code and exchanging it for an auth token requires the following:

1) Load the `preparedURL` into an existing `UIWebView` as part of the block code above. Ensure that you have set the delegate for the `UIWebView`.

```
[_webView loadRequest:[NSURLRequest requestWithURL:preparedURL]];
```

2) In the `webViewDidFinishLoad:` delegate method, you will need to parse the URL for your callback URL. If there is a match, pass that URL to `handleRedirectURL:`

```
if ([webView.request.URL.absoluteString
    rangeOfString:kOAuth2RedirectURL
    options:NSCaseInsensitiveSearch].location != NSNotFound)
{
```

```

[[NMAAuthLibManager sharedInstance] handleRedirectURL:[NSURL
    URLWithString:webView.request.URL.absoluteString]];
}

```

This is a very basic example. In the above, it is assumed that the `code` being returned from the OAuth2 provider is in the query parameter of the `webView.request.URL` (i.e. `http://myredirecturl.com?code=<code>`). This is not always the case and you might have to look elsewhere for the code (e.g. in the page content, web page title). You must prepare a URL in the format described above to pass to `[[NMAAuthLibManager handleRedirectURL:]`.

On Success

After a successful authentication, a new `NMAAuthAccount` object is in the list of accounts of `NMAAuthAccountManager`. You will receive a notification of type `NMAAuthAccountManagerAccountsDidChangeNotification`, e.g., for updating your UI.

```

[[NSNotificationCenter defaultCenter]
 addObserverForName:NMAAuthAccountManagerAccountsDidChangeNotification
    object:[NMAAuthLibManager sharedInstance]
    queue:nil
    usingBlock:^(NSNotification *aNotification){
        // Update your UI
    }];

```

If an account was added, the `userInfo` dictionary of the notification will contain the new account at the `NMAAuthAccountManagerNewAccountUserInfoKey`. Note, this notification can be triggered on other events (e.g. account removal). In that case, this key will not be set.

On Failure

If the authentication did not succeed, a notification of type `NMAAuthAccountManagerDidFailToRequestAccessNotification` containing an `NSError` will be sent.

```

[[NSNotificationCenter defaultCenter]
 addObserverForName:NMAAuthAccountManagerDidFailToRequestAccessNotifica
tion
    object:[NMAAuthLibManager sharedInstance]
    queue:nil
    usingBlock:^(NSNotification *aNotification){

```

```
        NSError *error = [aNotification.userInfo
                        objectForKey:NMAAuthAccountManagerErrorKey];
        // Do something with the error
    }];
```

Getting a List of all Accounts

The authenticated accounts can be accessed via the `NMAAuthAccountManager``. Either the complete list, only a list of accounts for a specific service, or an account with an identifier may be cached in the user settings.

```
for (NMAAuthAccount *account in [[NMAAuthLibManager sharedInstance]
accounts]) {
    // Do something with the account
};
```

```
for (NMAAuthAccount *account in [[NMAAuthLibManager sharedInstance]
accountsWithName:@"myOAuth2Service"]) {
    // Do something with the account
};
```

```
NMAAuthAccount *account = [[NMAAuthLibManager sharedInstance]
accountWithIdentifier:@"cached account id..."];
```

Each `NMAAuthAccount`` has a property `userData`` which can be used to store some related information for that account.

```
NMAAuthAccount *account = // ... get an account
NSDictionary *userData = // ...
account.userData = userData;
```

This payload will be stored together with the accounts in the device Keychain. It should not be too big.

Removing Accounts

To remove an account and its tokens from the store:

```
[[NMAAuthLibManager sharedInstance] removeAccount:account];
```

Note, that if you used a `UIWebView` to request access to a service as described above, it's likely that the token has been cached in `[[NSHTTPCookieStorage sharedHTTPCookieStorage]`

You can remove the auth token from the cookie cache using:

```
for(NSHTTPCookie *cookie in [[NSHTTPCookieStorage
sharedHTTPCookieStorage] cookies]) {
    if([[cookie domain] isEqualToString:@"myoathapp.com"]) {
        [[NSHTTPCookieStorage sharedHTTPCookieStorage]
deleteCookie:cookie];
    }
}
[[NSUserDefaults standardUserDefaults] synchronize];
```

Where `myoathapp.com` is the domain from which you received the auth token -- likely the `authorizationURL` assigned to the store at the time of request. As a convenience, you may want to store the domain of the token in the `account.userData` so that it is readily available if you need to delete the cookies.

Invoking a Request

There are a couple of ways to invoke the request to Protected Resources. A request using the authentication for a service can be invoked via `NMAAuthRequest`. The preferred way to do this is to pre-authorize the request and set the access token.

Prepare a request

```
(NMAAuthRequest *) theRequest = [ [NMAAuthRequest alloc]
initWithResource:pr1-url
                                method:@"GET"
                                parameters:params];
theRequest.account = [[NMAAuthManager sharedInstance]
accountWithAccountName:acctName]; //signedIn provider account

NSURLRequest *authorizedRequest = [theRequest
authorizeRequestWithError:&error];
//use the authorizedRequest for e.g. in web view etc.
```

Another convenient method (see below) is a class method which you pass the method, a resource, and some parameters (or nil) for the request and to handlers (both optional). One handler is for a progress and the other is for the response. The account is used for authentication and can be nil. After this process, a normal request without authentication will be invoked.

```

[NMAAuthRequest performMethod:@"GET"
                 onResource:[NSURL URLWithString:@"https://your
service URL..."]
                 usingParameters:nil
                 withAccount:anAccount
                 sendProgressHandler:^(unsigned long long bytesSend,
unsigned long long bytesTotal) { // e.g., update a progress indicator}
                 responseHandler:^(NSURLResponse *response, NSData
                                *responseData, NSError *error){
                                // Process the response
                                }
                ]];

```

Refreshing a Token

If you are using `NSURLConnection` in the app, the `NMAAuthRequest` gives you the possibility to get an `NSURLRequest` containing the additional information to authenticate that request.

Note: This is a synchronous call and is made on the calling method thread.

```

NMAAuthRequest *theRequest = [[NMAAuthRequest alloc]
                               initWithResource:[NSURL URLWithString:@"https://your service
URL..."]
                               method:@"GET"
                               parameters:nil];

theRequest.account = // ... an account
NSURLRequest *authorizedRequest = [theRequest
authorizeURLRequestWithError];

[theRequest release];
// Invoke the request with you preferred method

```