



Developing with MAF Logon for iOS

PDF download from SAP Help Portal:

http://help.sap.com/saphelp_smp305sdk/helpdata/en/7c/05845970061014a4fba37e8d86dfd/content.htm

Created on April 14, 2015

The documentation may have changed since you downloaded the PDF. You can always find the latest information on SAP Help Portal.

Note

This PDF document contains the selected topic and its subtopics (max. 150) in the selected structure. Subtopics from other structures are not included. The selected structure has more than 150 subtopics. This download contains only the first 150 subtopics. You can manually download the missing subtopics.

© 2015 SAP SE or an SAP affiliate company. All rights reserved. No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE. The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary. These materials are provided by SAP SE and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty. SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE in Germany and other countries. Please see www.sap.com/corporate-en/legal/copyright/index.epx#trademark for additional trademark information and notices.

Table of content

- 1 Developing with MAF Logon for iOS
 - 1.1 Architectural Overview
 - 1.2 MAF Logon Task Flow
 - 1.2.1 Create Your Logon Handler
 - 1.2.2 Integrate Logon Handler in Your App
 - 1.2.3 Present the Logon Screen
 - 1.3 Adding SAP Discovery Service
 - 1.4 Executing Logon Operations
 - 1.5 Data Requests
 - 1.6 Handling Delegate Calls
 - 1.7 Error Handling
 - 1.8 Onboarding with SAP Mobile Place
 - 1.8.1 SAP Mobile Place Workflow
 - 1.8.2 HttpConversation Workflow
 - 1.9 Developing Logon with Certificate Authentication
 - 1.9.1 Developing Logon with Afaria
 - 1.9.2 Developing Logon with Third-Party Certificate Provider
 - 1.9.3 Mutual Certificate Handling
 - 1.9.4 API Reference
 - 1.9.4.1 CertificateProvider protocol
 - 1.9.4.1.1 getCertificate:delegate method
 - 1.9.4.1.2 getStoredCertificate:error method
 - 1.9.4.1.3 deleteStoredCertificateWithError:error method
 - 1.9.4.2 CertificateProviderDelegate protocol
 - 1.9.4.2.1 currentViewController method
 - 1.9.4.2.2 onGetCertificateSuccess method
 - 1.9.4.2.3 onGetCertificateFailure:error method
 - 1.10 SAML Authentication
 - 1.11 Customizing the Logon UI
 - 1.12 Logon Screen Configuration Options
 - 1.12.1 Data Vault Life Cycle
 - 1.13 MAF Onboarding Scenarios
 - 1.14 MAF Configuration Provisioning
 - 1.15 Logon APIs for iOS
 - 1.15.1 HttpConversation API
 - 1.15.1.1 SAP Mobile Platform 3.0 OData SDK for iOS
 - 1.15.1.1.1 ChallengeFilterProtocol protocol
 - 1.15.1.1.1.1 handleChallenge:conversationManager:completionBlock: method
 - 1.15.1.1.2 ChallengeHandler class
 - 1.15.1.1.2.1 URLSession.task:didReceiveChallenge:completionHandler: method
 - 1.15.1.1.3 ChangeSet class
 - 1.15.1.1.4 ClientCertObserverProtocol protocol
 - 1.15.1.1.4.1 observeClientCertificate: method
 - 1.15.1.1.5 HttpConversationManager class
 - 1.15.1.1.5.1 addChallengeFilter: method
 - 1.15.1.1.5.2 addObserver: method
 - 1.15.1.1.5.3 addRequestFilter: method
 - 1.15.1.1.5.4 addResponseFilter: method
 - 1.15.1.1.5.5 allChallengeFilters method
 - 1.15.1.1.5.6 allObservers method
 - 1.15.1.1.5.7 allRequestFilters method
 - 1.15.1.1.5.8 allResponseFilters method
 - 1.15.1.1.5.9 copy method
 - 1.15.1.1.5.10 currentTimeInMillis method
 - 1.15.1.1.5.11 executeRequest:completionHandler: method
 - 1.15.1.1.5.12 executeRequest_Private:completionHandler: method
 - 1.15.1.1.5.13 generateTraceEntriesFromRequest: method
 - 1.15.1.1.5.14 generateTraceEntriesFromResponse:receivedData:method: method
 - 1.15.1.1.5.15 init method
 - 1.15.1.1.5.16 setResponseData:intoResponse:error: method

- 1.15.1.15.17 URLSession.taskDidReceiveChallenge:completionHandler: method
- 1.15.1.1.6 HttpConversationObserverProtocol protocol
- 1.15.1.1.7 ManagerConfiguratorProtocol protocol
- 1.15.1.1.7.1 configureManager: method
- 1.15.1.1.8 PluginProviderProtocol protocol
- 1.15.1.1.9 RequestFilterProtocol protocol
- 1.15.1.1.9.1 prepareRequest:conversationManager:completionBlock: method
- 1.15.1.1.10 ResponseFilterProtocol protocol
- 1.15.1.1.10.1 processResponse:responseData:conversationManager:completionBlock: method
- 1.15.1.1.11 SAPPProvider class
- 1.15.1.1.12 SupportabilityUploader class
- 1.15.1.1.12.1 initWithHttpConversationManager:urlRequest: method
- 1.15.1.1.12.2 sendWithContentType:headers:payloadType:payload:completion: method
- 1.15.1.1.12.3 statusCodeToErrorMessage: method
- 1.15.1.1.13 batchElements_C method
- 1.15.1.1.14 dataTaskWithMutableRequest_Block method
- 1.15.1.1.15 sessionConfiguration_C method
- 1.15.1.1.16 setBatchElements_C method
- 1.15.1.1.17 setSessionConfiguration_C method
- 1.15.1.1.18 setupRequestWithXCSRFToken method
- 1.15.1.1.19 setXCSRFToken_Private method
- 1.15.1.1.20 start_C method
- 1.15.1.1.21 XCSRFToken_Private method
- 1.15.1.1.22 XCSRFTokenFromResponse method
- 1.15.1.2 Deprecated API List
- 1.15.2 HttpConvAuthFlows API
- 1.15.2.1 SAP Mobile Platform 3.0 OData SDK for iOS
- 1.15.2.1.1 AuthenticationUIViewController class
- 1.15.2.1.1.1 addContentViews method
- 1.15.2.1.1.2 authenticationCanceled method
- 1.15.2.1.1.3 closeActivityIndicator method
- 1.15.2.1.1.4 initWithParentViewController: method
- 1.15.2.1.1.5 privateCloseActivityIndicator method
- 1.15.2.1.1.6 privateShowActivityIndicatorWithText: method
- 1.15.2.1.1.7 showActivityIndicatorWithText: method
- 1.15.2.1.1.8 viewDidLoad method
- 1.15.2.1.1.9 webView.shouldStartLoadWithRequest:navigationType: method
- 1.15.2.1.1.10 webViewDidFinishLoad: method
- 1.15.2.1.2 AuthenticationUIViewControllerDelegate protocol
- 1.15.2.1.2.1 authenticationCanceled method
- 1.15.2.1.3 BaseChallengeFilter class
- 1.15.2.1.4 BaseRequestFilter class
- 1.15.2.1.5 BaseResponseFilter class
- 1.15.2.1.6 ClientCertChallengeFilter class
- 1.15.2.1.6.1 callNextProviderOfEnumerator:completionBlock: method
- 1.15.2.1.6.2 handleChallenge:conversationManager:completionBlock: method
- 1.15.2.1.7 ClientCertProviderProtocol protocol
- 1.15.2.1.7.1 provideClientCertForAuthChallenge:completionBlock: method
- 1.15.2.1.8 CommonAuthenticationConfigurator class
- 1.15.2.1.8.1 addClientCertProvider: method
- 1.15.2.1.8.2 addOAuth2ConfigProvider: method
- 1.15.2.1.8.3 addSAML2ConfigProvider: method
- 1.15.2.1.8.4 addUsernamePasswordProvider: method
- 1.15.2.1.8.5 allClientCertProvider method
- 1.15.2.1.8.6 allOAuth2ConfigProvider method
- 1.15.2.1.8.7 allSAML2ConfigProvider method
- 1.15.2.1.8.8 allUsernamePasswordProvider method
- 1.15.2.1.8.9 configureManager: method
- 1.15.2.1.9 HttpConvAuthFlowsLocalizationManager class
- 1.15.2.1.9.1 localizationBundle method
- 1.15.2.1.9.2 localizedStringForKey: method
- 1.15.2.1.10 OAuth2ConfigProviderProtocol protocol

1.15.2.1.10.1 isAcceptedResponse:forProvidedToken:completionBlock: method
1.15.2.1.10.2 provideOAuth2ConfigurationForURL:completionBlock: method
1.15.2.1.10.3 provideOAuth2TokenForURL:completionBlock: method
1.15.2.1.11 OAuth2RequestFilter class
1.15.2.1.11.1 addTokenToAuthorizationHeader method
1.15.2.1.11.2 authenticationCanceled method
1.15.2.1.11.3 callCompletionHandler method
1.15.2.1.11.4 callNextConfigProviderOfEnumerator:completionBlock: method
1.15.2.1.11.5 callNextTokenProviderOfEnumerator:completionBlock: method
1.15.2.1.11.6 closeWebView method
1.15.2.1.11.7 displayLoginWebView: method
1.15.2.1.11.8 prepareRequest:conversationManager:completionBlock: method
1.15.2.1.11.9 retrieveTokenForAuthorizationCode method
1.15.2.1.11.10 retrieveTokenFromResponseData: method
1.15.2.1.11.11 startOAuthFlow method
1.15.2.1.11.12 webView:shouldStartLoadWithRequest:navigationType: method
1.15.2.1.11.13 webViewDidFinishLoad: method
1.15.2.1.12 OAuth2TokenObserverProtocol protocol
1.15.2.1.12.1 observeOAuth2Token: method
1.15.2.1.13 SAML2ConfigProviderProtocol protocol
1.15.2.1.13.1 provideSAML2ConfigurationForURL:completionBlock: method
1.15.2.1.14 SAML2ResponseFilter class
1.15.2.1.14.1 authenticationCanceled method
1.15.2.1.14.2 callNextConfigProviderOfEnumerator:completionBlock: method
1.15.2.1.14.3 checkIfSAMLNeeded method
1.15.2.1.14.4 closeWebView method
1.15.2.1.14.5 displayLoginWebView method
1.15.2.1.14.6 processResponse:responseData:conversationManager:completionBlock: method

1 Developing with MAF Logon for iOS

MAF logon components consist of MAF Logon UI and MAF Logon Manager, and support user onboarding.

MAF logon includes:

- User-initiated registration requests to a specified security infrastructure
- Locking and unlocking the secure store to protect sensitive information
- Changing secure store and back-end passwords to enhance security in an enterprise environment
- Various customization options

MAF Logon UI presents logon screens on the device, providing easy integration for applications that use default logon UI behavior. While MAF Logon UI constructs a dynamic UI from context elements, MAF Logon Manager executes operations and constructs UI contexts to be presented by MAF Logon UI. Thus, MAF Logon Manager handles all UI-related configurations.

In this section:

- [Architectural Overview](#)
- [MAF Logon Task Flow](#)
- [Adding SAP Discovery Service](#)
- [Executing Logon Operations](#)
- [Data Requests](#)
- [Handling Delegate Calls](#)
- [Error Handling](#)
- [Onboarding with SAP Mobile Place](#)
- [Developing Logon with Certificate Authentication](#)
- [SAML Authentication](#)
- [Customizing the Logon UI](#)
- [Logon Screen Configuration Options](#)
- [MAF Onboarding Scenarios](#)
- [MAF Configuration Provisioning](#)
- [Logon APIs for iOS](#)

Parent topic: [iOS Applications](#)

1.1 Architectural Overview

MAF includes a configurable, multipurpose onboarding component. It consists of a core component and a UI component, but using the core with a custom UI is also possible.

The Logon Core layer contains code to execute logon operations. The component builds on SAP Mobile Platform libraries, such as OData CoreServices, Request, and Client Hub.

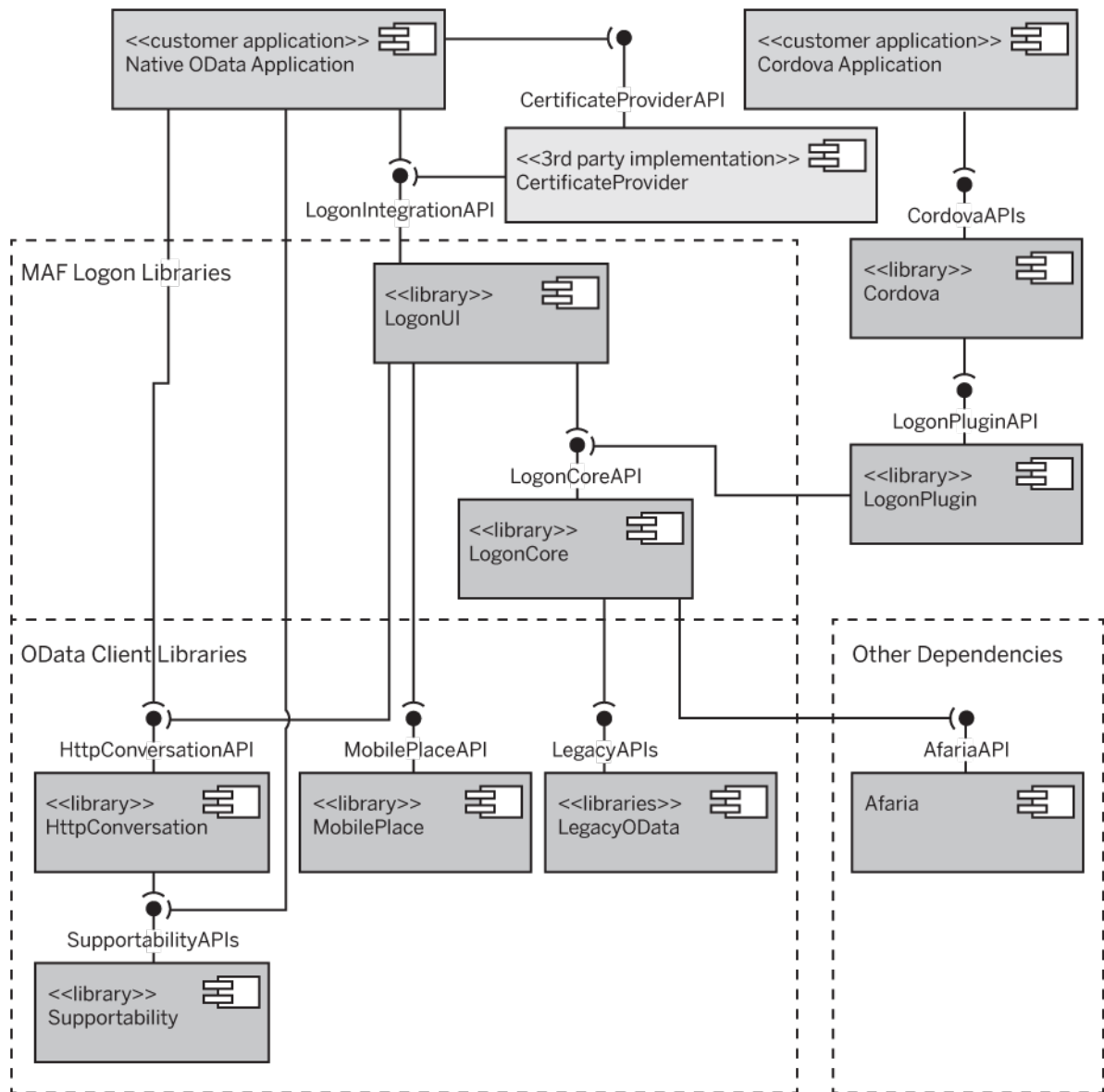
The Logon Component supports Afaria, the SAP-provided MDM solution to:

- Provision application configuration
- Provision X.509 certificate for registration

The Logon UI component provides native logon screens. This component collects information from Logon Core and runtime configuration options. Based on the collected information it decides whether:

- a particular screen needs to be presented to get input from the end user
- a third-party client certificate provider needs to be called
- a configuration provider needs to be called to provide configuration

Logon UI exposes the `CertificateProvider` and `CertificateProviderListener` APIs to integrate third-party Certificate Provider. This interface enables application developers to implement custom components to fetch the X.509 certificate from a third-party MDM infrastructure.



SAML2 based authentication can be achieved by using the HttpConversation library. The configuration that the SAML2 protocol needs can be acquired via the following methods:

- Mobile Place Discovery Service
- Runtime configuration APIs

If no Afaria or Mobile Place Discovery Service is configured to provide authentication data then the user has to input information through the Logon UI. Based on user input, the Logon Core determines which type of registration to execute. After registration, the Logon Core:

1. Checks if any scenario is forced via configuration.
2. Triggers a test HTTP(S) request to the host and port specified by the user.
3. Analyzes the response based on:
 - HTTP(S) response code
 - Cookies
 - Response headers

The Logon Core next determines which communicator to use. If the Logon Core cannot determine which communicator to use, it falls back to HTTP Rest. If the decision flow can find a suitable communication setup, it reports a success, otherwise the logon process stops and issues a failure message.

The default Logon UI supports customization of headers and footers, and includes a fully skinnable UI. This component builds on the MAF Logon Core, and provides an Integration API, which supports these logon operations:

- **Logon** – register or unlock the secure store so that the app has access to server information and credentials to initiate requests.
- **Change back-end password** – update the back-end password stored in the secure store of the client.
- **Change secure store password (App Passcode)** – change the password of the secure store.
- **Lock secure store** – force the secure store to lock itself while the application is still in the foreground.
- **Delete user** – unregister and delete all locally stored content from the secure store.
- **Registration information** – present information provided by the user during logon.
- **Update application settings** – get server settings from SAP Mobile Platform.
- **Registration data** – used by the application to get all registration data from MAF logon.

Parent topic: [Developing with MAF Logon for iOS](#)

1.2 MAF Logon Task Flow

To complete this process, you need to get the iOS Native Client SDK libraries. The libraries are available as part of the SAP Mobile Platform Client SDK Installer.

Context

MAFLogon is shipped in the form of separate artifacts such as:

- statically linked libraries
- resource bundle folders
- public header files

Before you can use MAFLogon, you must add associated artifacts. You must also add dependent frameworks and configure your build settings. For the rest of the procedure, we assume that you are starting a new project for iOS. If you are working with an existing project, the steps may vary.

MAFLogon is distributed in separated libraries, resource bundles, and header files

- Libraries
 - libMAFLogger.a - logging helper classes
 - libMAFLogonManagerNG.a - Logon Core components that implement registration logic
 - libMAFUIHelper.a - utility UI components
 - libMAFZipHelper.a - ZIP utilities
 - libMAFLogonUING.a - Logon integration API with default Logon UI
 - libMAFUIComponents.a - skinnable controls library
- Bundles
 - MAFLogonManagerNG.bundle - configuration of Logon Core functionalities and localization
 - MAFLogonUING - default Logon UI configuration
 - MAFUIComponents.bundle - skinning configuration and default resources
- Various header folders

Use this procedure to integrate MAF Logon into your application.

Procedure

1. Create a new project (or open an existing one) in Xcode.
2. Add MAFLogon artifacts into your project:
 - libMAFLogger.a
 - libMAFLogonManagerNG.a
 - libMAFUIHelper.a
 - libMAFZipHelper.a
 - libMAFLogonUING.a
 - libMAFUIComponents.a
 - MAFLogonManagerNG.bundle
 - MAFLogonUING.bundle
 - MAFUIComponents.bundle
 - MobilePlace.bundle
 - HttpConvAuthFlows.bundle
3. Add other required dependencies into your project:
 - libAfarisSLL.a
 - libConnectivity.a
 - libCoreServices.a
 - libDatavault.a
 - libPerformanceLib.a
 - libRequest.a
 - libE2ETrace.a
 - libParser.a
 - libLogger.a
 - libCache.a
 - libsqlcipher.a
 - libClientHubSLL.a
 - libHttpConversation.a
 - libHttpConvAuthFlows.a
 - libClientLog.a
 - libE2ETrace2.a
 - libSupportability.a
 - libPerformanceLib.a
4. Select your project from the **Project Navigator**, and choose your application's target.
5. Open the **Build Phases** tab and, within Link Binary with Libraries, add the following frameworks:
 - CFNetwork.framework
 - CoreData.framework
 - MessageUI.framework
 - Security.framework
 - QuartzCore.framework
 - Libstdc++.dylib
 - libz.dylib
 - MobileCoreServices.framework
 - SystemConfiguration.framework
 - UIKit.framework
 - Foundation.framework
 - CoreGraphics.framework

- `libsqlite3.dylib`
- Choose your project, rather than a specific target, and open the **Build Settings** tab.
 - In the **Other Linker Flags** section, add `-ObjC` and `-all_load`. If these settings are not visible, change the filter in the **Build Settings** bar from **Basic** to **All**.
 - In the header search path section, adjust the path to look for public header files. Depending on where you copied the header file, the path string should look something like this:


```
$(PROJECT_DIR)/includes/public/**
```
 - In the library search path section, adjust the path to look for the binary files. Depending on where you copied the library files, the path string should look something like this:


```
$(PROJECT_DIR)/libraries/$(BUILD_STYLE)-universal/**
```
 - The MobilePlace UI ships with a Storyboard file. To use this storyboard file, you must import `MobilePlace.bundle` into your Xcode project. Once that is present:
 - Open the associated group.
 - Drag the `MobilePlace.storyboard` file into a group in your project, anywhere outside of the bundles group.
 - Make sure that **Copy items into destination group's folder (if needed)** is selected.
 This enables you to present the Mobile Place UI.
 - Open `MobilePlace.storyboard` in the Xcode **Storyboard Editor**.
 - In the **File Inspector** tab of the right Xcode pane, select the **Localization** section.
 - Ensure that **Base** and **English** are selected.
 - Ensure that **Localizable Strings** is selected from the drop-down list next to **English**.
 - To import the translation of the Storyboard UI definition, you must specify all the supported localizations in the **Project Settings** > **Info** > **Localizations** section.
 - Add all required languages, and select MobilePlace storyboard. Also select Localizable Strings as the type of the localization. Each localization you add creates a copy of the technical language, and should appear under the Storyboard in the project navigator pane. Open the files and copy the content from the corresponding translation file shipped in the `MobilePlace.bundle` group. You need to override the generated content with the one shipped by SAP.
 - Add the `HttpConvAuthFlows.bundle` to your project.
 - [Create your logon handler.](#)
Create a central class for logon-related work, so your code remains clean.
 - [Integrate logon handler into your app.](#)
After implementing your logon handler, use a property to hold a reference to it in your `AppDelegate` singleton.
 - [Present the logon screen.](#)
To present the MAF Logon UI, adjust your root view controller. Do not change your root view controller interface.

In this section:

- [Create Your Logon Handler](#)
- [Integrate Logon Handler in Your App](#)
- [Present the Logon Screen](#)

Parent topic: [Developing with MAF Logon for iOS](#)

1.2.1 Create Your Logon Handler

Create a central class for logon-related work, so your code remains clean.

Context

This example is a simple interface of the logon handler:

```
#import <Foundation/Foundation.h>
#import "MAFLogonNGPublicAPI.h"
#import "MAFLogonManagerNG.h"
#import "MAFLogonUIViewModel.h"
#import "MAFLogonNGDelegate.h"

@interface MyLogonHandler : NSObject <MAFLogonNGDelegate>

@property (strong, nonatomic) MAFLogonUIViewModel *logonUIViewModel;
@property (strong, nonatomic) NSObject<MAFLogonNGPublicAPI> *logonManager;

@end
```

This class implements `MAFLogonNGDelegate` and exposes an instance of `MAFLogonUIViewModel`.

Example

This is an example implementation (`MyLogonHandler.m`) of the logon handler:

```
#import "MyLogonHandler.h"

@implementation MyLogonHandler

-(id) init{
    self = [super init];
    if(self){
        self.logonUIViewModel = [[[MAFLogonUIViewModel alloc] init] autorelease];
        // save reference to LogonManager for code readability
        self.logonManager = self.logonUIViewModel.logonManager;
#warning You must set your own application id. The application id can be specified in the info.plist file.
        // set up the logon delegate
        [self.logonManager setLogonDelegate:self];
    }
}
```



```

    return self;
}

@end

```

In the info.plist file of your Xcode project, add these lines:

```

<?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>keyMAFLogonApplicationId</key>
    <string>com.example.ios.app</string>
</dict>
</plist>

```

You must implement all MAFLogonNGDelegate methods in MyLogonHandler:

```

#pragma mark - MAFLogonNGDelegate implementation

-(void) logonFinishedWithError:(NSError*)anError {
    NSLog(@"logonFinishedWithError:%@", anError);
}

-(void) lockSecureStoreFinishedWithError:(NSError*)anError {
    NSLog(@"lockSecureStoreFinishedWithError:%@", anError);
}

-(void) updateApplicationSettingsFinishedWithError:(NSError*)anError { NSLog(@"updateApplicationSettingsFinishedWi
}

-(void) uploadTraceFinishedWithError:(NSError *)anError {
    NSLog(@"uploadTraceFinishedWithError:%@", anError);
}

-(void) changeBackendPasswordFinishedWithError:(NSError*)anError {
    NSLog(@"Password change with error:%@", anError);
}

-(void) deleteUserFinishedWithError:(NSError*)anError {
    NSLog(@"deleteUserFinishedWithError:%@", anError);
}

-(void) changeSecureStorePasswordFinishedWithError:(NSError*)anError {
    NSLog(@"changeSecureStorePasswordFinishedWithError:%@", anError);
}

-(void) registrationInfoFinishedWithError:(NSError*)anError {
    NSLog(@"registrationInfoFinishedWithError:%@", anError);
}

-(void) startDemoMode {
    NSLog(@"startDemoMode");
}
}

```

Parent topic: [MAF Logon Task Flow](#)

1.2.2 Integrate Logon Handler in Your App

After implementing your logon handler, use a property to hold a reference to it in your AppDelegate singleton.

Context

```

#import <UIKit/UIKit.h>
#import "MyLogonHandler.h"
#import "MAFUIStyleParser.h"

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;
@property (strong, nonatomic) MyLogonHandler *logonHandler;

@end

```



Example

For example, you can implement the AppDelegate.m file using code similar to:

```

#import "AppDelegate.h"
#import "MAFUIStyleParser.h"

@implementation AppDelegate

- (void)dealloc
{
    [_window release];
    [_logonHandler release];
    [super dealloc];
}

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

```

```

{
    // Load the styles for the customizable controls, should be called before any MAF* control and MAFLogonUI created
    [MAFUIStyleParser loadSAPDefaultStyle];
    self.logonHandler = [[MyLogonHandler alloc] init];
    return YES;
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    // Execute logon operation, which will automatically present logon screen if needed.
    [self.logonHandler.logonManager logon];
}
...
@end

```

Parent topic: [MAF Logon Task Flow](#)

1.2.3 Present the Logon Screen

To present the MAF Logon UI, adjust your root view controller; however, do not change your root view controller interface.

Context

This example shows how to implement the view controller (ViewController.m). It first gets the logonHandler from the AppDelegate singleton, and then sets the parentViewController property of the MAFLogonUIViewModel class to point to the root view controller (self):

```

#import "ViewController.h"
#import "MyLogonHandler.h"
#import "AppDelegate.h"

@interface ViewController ()
@property (nonatomic, retain) MyLogonHandler *logonHandler;
@end;

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    AppDelegate* appDelegate = [[UIApplication sharedApplication] delegate];
    self.logonHandler = appDelegate.logonHandler;
    self.logonHandler.logonUIViewModel.parentViewController = self;
}

@end

```

The view controller (MainViewController.m) is only responsible for setting the self reference as the parentViewController property of the MAFLogonUIViewModel. The Logon UI is presented using the parentViewController reference.

You can now run the project to see the default MAF logon UI. The Logon UI appears if you call the logon selector on the Logon Manager from the appDelegate when the application becomes active.

Parent topic: [MAF Logon Task Flow](#)

1.3 Adding SAP Discovery Service

Use the SAP Discovery Service cloud solution to publish app connection settings for end users on your e-mail domains or sub-domains using only their e-mail address.

Context

The Discovery Service user interface is included in the Logon component and flow, so with you can include it in your project without much effort. Discovery Service features in the SAP Mobile Platform SDK are usually noted as `Mobile Place`, also known as SAP Mobile Secure.

Procedure

1. Add the Logon component and required bundles to your project:
 - MAFUIComponents.bundle
 - HttpConvAuthFlows.bundle
 - Logger.bundle
 - MAFLogonManagerNG.bundle
 - MAFLogonUI.bundle
 - Settings.bundle
 - MobilePlace.bundle
2. Drag the `MobilePlace.storyboard` file into a group in your project, anywhere outside of the bundles group.
3. Verify that the key value in `MAFLogonUING.bundle/MAFLogonUIOptions.plist` is set to `keyMAFLogonUseMobilePlace = YES`.

The MAFLogonUNING shows the Discovery Service user interface.

4. (Optional) Disable the configuration check by MAFLogonManagerNG to SAP Afaria MDM client.

If applications get configurations from Afaria and fail to find a configuration from Afaria, it will not fall back to try the Discovery Service. If you distribute your application through a public store, you should:

- Advise customers using SAP Afaria to continue to distribute configurations with SAP Afaria.
- SAP recommends that you disable the SAP Afaria feature on MAFLogonManagerNG by setting key value `keyMAFUseAfaria = NO` in `MAFLogonManagerNG.bundle/MAFLogonManagerOptions.plist`. Your customers should then upload their application connection configurations to Discovery Service.

If your customers also use SAP Afaria to distribute client certificates, you can implement the `<CertificateProvider>` protocol in your application so that the provider invokes the Afaria APIs to obtain the client certificate. The Discovery Service is then used to obtain the configurations, Logon calls the `ClientCertificate` protocol, and Afaria supplies the client certificate.

5. Build and run your project. If it is successful, the **Acquire Settings** screen appears, and you see the request and response from the Discovery Service in the console:

```
2014-08-15 TravelAgency_RKT[] URL: https://discovery.sapmobilesecure.com/config-api.svc/ApplicationConfigurati
2014-08-15 TravelAgency_RKT[] received response: { "host" : "smpqa12-01.sybase.com", "port" : 80, "protocol" :
|||
```

Next Steps

Upsert configurations to Discovery Service from SAP HANA Cloud Platform mobile services.

Note

SAP HANA Cloud Platform mobile services is currently planned but not available. SAP has no obligation to develop or release this product. All future development is subject to change and may be changed by SAP at any time for any reason without notice.

Parent topic: [Developing with MAF Logon for iOS](#)

1.4 Executing Logon Operations

After you initialize `MAFLogonUIViewModel`, present the MAF logon UI screens. The View Manager maintains a reference to the `LogonMediator` (which, in the code examples, may also be called the `logonManager`). To present a particular screen, call the corresponding operation on the `LogonMediator`, which notifies the View Manager of `MAFLogonUNING` that the action requires input. The View Manager generates and presents the input screen. The View Manager then calls back the `LogonMediator` in `MAFLogonManagerNG` and executes the next step of the requested operation.

Logon

To present the logon screen, call the `MAFViewManager` logon method:

```
// call this method from any place in your application code
- (void)doLogon {
    [logonUIViewModel.logonManager logon];
}
```

This presents the default logon screen.

Enter the input that corresponds to your onboarding scenario. Tap the Log In button to trigger the registration process, which has these possible outcomes :

- The request fails with an error message.
- Registration succeeds and the success delegate is called on `MAFLogonNGDelegate`.
- Registration succeeds and the App Passcode screen is presented.

The `MAFLogonManagerNG` provides a build-time configuration of the structure and content of the MAF logon UI. You can:

- Hide the advanced registration section
- Hide each field individually
- Set a default value for each field individually

Users cannot change the default values of hidden fields, but the onboarding process uses these values.

App Passcode

Upon successful registration, the device user sees the Set Passcode screen. This is the second screen of the logon process, unless the Afaria configuration prohibits the usage of app passcodes, or if the password policy is disabled on SAP Mobile Platform. The password policy is set for the application connection template on the SAP Mobile Platform server by the IT administrator. The confirmation must match the app passcode, and the app passcode must match the password policy. This screen appears only if the password policy is enabled. The `MAFLogonUNING` checks the password policy and presents the screen as appropriate.

Delete Registration

Call the `deleteUser` operation to delete all registration information from the device, so the user can start using the application anew.

```
// call this method from any place in your application code
- (void)doDelete {
    [logonUIViewModel.logonManager deleteUser];
}
```

When you call the logon manager's `deleteUser` method, the Delete Registration Confirmation `UIAlertView` is presented.

Lock Secure Store

To close the secure store and present the App Passcode Unlock screen, call `lockSecureStore` from your application:

```
// call this method from any place in your application code
- (void)doLockSecureStore {
    [logonUIViewManager.logonManager lockSecureStore];
}
```

Change App Passcode

To update the secure store passcode, present the Change App Passcode screen:

```
// call this method from any place in your application code
- (void)doChangeAppPasscode {
    [logonUIViewManager.logonManager changeSecureStorePassword];
}
```

Update Back-End Password

When device users change their back-end passwords, MAF Logon Manager must get the new passwords:

```
// call this method from any place in your application code
- (void)doChangeBackendPassword {
    [logonUIViewManager.logonManager changeBackendPassword];
}
```

This method call displays the Update Backend Password screen.

Reviewing Login Data

To review registration information, present the Login Data screen by executing the Login Data operation:

```
// call this method from any place in your application code
- (void)doPresentRegInfo {
    [logonUIViewManager.logonManager registrationInfo];
}
```

This method call displays the Registration Information Screen. This screen displays, in read-only mode, every input for the login screen during registration.

Storing the Application Connection ID

You can store the application connection ID after logon with this code:

```
MAFLogonRegistrationData* registrationData = [self.logonUIViewManager.logonManagerregistrationDataWithError:&error];
appid = registrationData.applicationConnectionId;
```

Parent topic: [Developing with MAF Logon for iOS](#)

1.5 Data Requests

After you complete the registration and initialize the secure store, you can request data from the back end.

Application developers can get information about the registration data with this MAF Logon Integration API call:

```
NSError* localError = nil;
MAFLogonRegistrationData* data = [self.logonManager registrationDataWithError:&localError];

if (localError) {
    //handle error
} else {
    //access registration data, like:
    self.appEndpoint = data.applicationEndpointURL;
    self.communicatorId = data.communicatorId;
}
```

Note

As a best practice, network requests should be managed in a central place. It is not required, but there is usually a central `HttpConversationManager` instance. You can configure this instance with any `Configurator` implementation that implements `ManagerConfiguratorProtocol`. In order for the `HttpConverstionManager` to preset UIs, you must keep a reference to the currently presented `UIViewController` instance.

Initialize the `HttpConversationManager` instance.

```
-(void) initializeConversationManager{
    self.manager = [[[HttpConversationManager alloc] init] autorelease];
}
```

Get the reference to the `Configurator` instance initialized by the MAF Logon upon a successful registration. Set the currently presented `UIViewController` instance as a parent view controller. Use the configurator to configure the `HttpConversationManager` instance with the `Configurator`:

```
/**
 * Update Conversation Manager with the currently persented UIViewController
 * @param: viewController the reference to the currently presented UIViewController
 */
-(void) configureConversationManagerWithParentViewController:(UIViewController *) viewController{
    //get Logon provided Conversation Manager Configurator
    id<ManagerConfiguratorProtocol> logonConfigurator = [logonUIViewManager.logonManager logonConfigurator];
    //update the reference to the provided UIViewController
    [logonConfigurator setParentView:viewController];
    //configure manager with the updated configurator
}
```

```
[logonConfigurator configureManager:self.manager];
}
```

Create an instance of `NSMutableURLRequest` class. Depending on the registration type used by MAF Logon, you might need to construct the base URL.

```
NSString *fullEndpoint = self.applicationEndpoint;
NSString *serviceDocumentFormat = @"";
if([self.communicatorId isEqualToString:idMAFLogonCommunicator_GatewayOnly]){
    NSString* serviceDocumentFormat = @"";
    if([fullEndpoint hasSuffix:@"/"]){
        serviceDocumentFormat = @"sap/opu/odata/GBHCM/LEAVEREREQUEST/";
    }
    else{
        serviceDocumentFormat = @"/sap/opu/odata/GBHCM/LEAVEREREQUEST/";
    }

    fullEndpoint = [fullEndpoint stringByAppendingString:serviceDocumentFormat];
    self.applicationEndpoint = fullEndpoint;
}

fullEndpoint = [fullEndpoint stringByAppendingString:serviceDocumentFormat];
self.applicationEndpoint = fullEndpoint;
NSMutableURLRequest* request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:fullEndpoint]];
```

If the registration is , you need to provide additional headers.

```
if([self.communicatorId isEqualToString:idMAFLogonCommunicator_SMPHTTPREST]){
    if(self.applicationConnectionId){
        [request setValue:self.applicationConnectionId forHTTPHeaderField:@"X-SMP-APPCID"];
        [request setValue:self.applicationConnectionId forHTTPHeaderField:@"X-SUP-APPCID"];
    }
    if(self.securityConfig){
        [request setValue:self.securityConfig forHTTPHeaderField:@"X-SUP-SC"];
    }
    if(self.domain){
        [request setValue:self.domain forHTTPHeaderField:@"X-SUP-Domain"];
    }
}
}
```

Execute the request with `HttpConversationManager`:

```
[_conversationManager executeRequest:request completionHandler:^(NSData *data, NSURLResponse *response, NSError *e:
    /* Handle response, error and data! */
});
```

Parent topic: [Developing with MAF Logon for iOS](#)

1.6 Handling Delegate Calls

The `MAFLogonUING` notifies applications about the result of each `MAFLogonManagerNG` operation it executes. The `MAFLogonNGDelegate` protocol must be implemented by your application's View Controller.

Use these methods to implement your application logic. This code calls a local method call to present a `UIAlertView`:

```
#pragma mark - MAFLogonNGDelegate implementation
/**
    Called when the logon finished either with success or error.
    */
-(void) logonFinishedWithError:(NSError*)anError {
    [self showAlertView:@"Logon" withError:anError];
}
/**
    Called when the lockSecureStore method call finished either with success or error.
    */
-(void) lockSecureStoreFinishedWithError:(NSError*)anError {
    [self showAlertView:@"DataVault lock" withError:anError];
}
/**
    Called when the updateApplicationSettings method call finished either with success or error.
    */
-(void) updateApplicationSettingsFinishedWithError:(NSError*)anError {
    [self showAlertView:@"Update app. settings" withError:anError];
}
/**
    Called when the changeBackendPassword method call finished either with success or error.
    */
-(void) changeBackendPasswordFinishedWithError:(NSError*)anError {
    [self showAlertView:@"Password change" withError:anError];
}
/**
    Called when the deleteUser method call finished either with success or error.
    */
-(void) deleteUserFinishedWithError:(NSError*)anError {
    [self showAlertView:@"Delete user" withError:anError];
}
/**
```

```

    Called when the changeSecureStorePassword method call finished either with success or error.
    */
    -(void) changeSecureStorePasswordFinishedWithError:(NSError*)anError {
        [self showAlertView:@"DataVault password change" withError:anError];
    }
    /**
    Called when the registrationInfo method call finished either with success or error.
    */
    -(void) registrationInfoFinishedWithError:(NSError*)anError {
        [self showAlertView:@"Display registration info" withError:anError];
    }
}

```

Parent topic: [Developing with MAF Logon for iOS](#)

1.7 Error Handling

MAFLogonManagerNG communicates errors using iOS-standard NSError objects. If the passed NSError object is not nil, an error is generated. MAF provides an error-handling method for these: the NSError instance contains the localized error description, the error domain, and the error code.

For a description of the error messages that MAFLogonManagerNG can send, see these header files in the API Reference:

- [MAFSecureStoreManagerErrors.h](#)
- [MAFLogonManagerNGErrors.h](#)

Parent topic: [Developing with MAF Logon for iOS](#)

1.8 Onboarding with SAP Mobile Place

Mobile Place provides a single known endpoint to get the connection configurations for applications for all customers.

This table shows the information that must be provided to use this type of connection.

Parameter Name	Example Value
authType	saml2.web.post, oauth2, or BLANK
EmailAddress	UTF-8 encoded base64 string
ApplicationConfigurationID	UTF-8 encoded string

See also SAP HANA Cloud Platform Mobile Services > Application Administration > Enabling Applications to Discover Configurations.

Note

SAP HANA Cloud Platform mobile services is currently planned but not available. SAP has no obligation to develop or release this product. All future development is subject to change and may be changed by SAP at any time for any reason without notice.

In this section:

- [SAP Mobile Place Workflow](#)
- [HttpConversation Workflow](#)

Parent topic: [Developing with MAF Logon for iOS](#)

1.8.1 SAP Mobile Place Workflow

Add Mobile Place to your iOS project.

Context

You must use the iOS Native Client SDK libraries, which are available as part of the SAP Mobile Platform Client SDK Installer. Mobile Place ships in the form of separate artifacts such as:

- statically linked libraries
- resource bundle folders
- public header files

Before you can use Mobile Place, you must add associated artifacts. You also need to add dependent frameworks and configure your build settings. For the rest of the procedure, we assume that you are starting a new project for iOS. If you are working with an existing project, the steps may vary.

Procedure

1. Create a new project or open an existing one in Xcode.
2. Add `MobilePlace` artifacts into your project:
 - `libHttpConversation.a`
 - `libMobilePlace.a`
 - `libHttpConvAuthFlows.a`
 - `HttpConvAuthFlows.bundle`
 - `MobilePlace.bundle`
 - Header files for the libraries
3. Add artifacts that the `MobilePlace` library depends on:
 - `libClientLog.a`
 - `libE2ETrace2.a`
 - `libSupportability.a`

- libPerformanceLib.a
 - Header files for the libraries
4. Select your project from the Project Navigator, and choose your application's target.
 5. Open the Build Phases tab, and within Link Binary with Libraries, add these frameworks:
 - Security.framework
 - libsqlite3.dylib
 6. Choose your project, rather than a specific target, and open the Build Settings tab.
 1. In the Other Linker Flags section, add `-ObjC` and `-all_load`. If these settings are not visible, change the filter in the Build Settings bar from Basic to All.
 2. In the header search path section, adjust the path to look for public header files. Depending on where you copied the header files, the path string should look something like this:


```
$(PROJECT_DIR)/includes/public/**
```
 3. In the library search path section, adjust the path to look for the binary files. Depending on where you copied the library files, the path string should look something like this:


```
$(PROJECT_DIR)/libraries/$(BUILD_STYLE)-universal/**
```
 7. MobilePlace UI is shipped with a Storyboard file. To use this storyboard file, you must have `MobilePlace.bundle` file imported into your Xcode project. Once that is present:
 1. Open the associated group.
 2. Drag the `MobilePlace.storyboard` file into a group in your project anywhere outside of the bundles group.
 3. Be sure that "Copy items into destination group's folder (if needed)" is selected.
 This enables you to present the MobilePlace UI.
 8. Open the `MobilePlace.storyboard` in the Storyboard Editor of the Xcode. On the File Inspector tab of the Right Xcode pane, check the Localization section.
 - Ensure that both Base and English are selected.
 - Ensure that the type Localizable Strings is selected from the drop-down list next to English.
 9. To import the translation of the Storyboard UI definition you will need to specify all the supported localizations in the **Project Settings** > **Info** > **Localizations**.
 10. Add all required languages and select MobilePlace storyboard. Also select Localizable Strings as the type of localization. Each localization you add creates a copy of the technical language, and should appear under the Storyboard in the project navigator pane. Open these files and copy the content from the corresponding translation file shipped in the `MobilePlace.bundle` group. This involves manually overriding the generated content with the one shipped by SAP.
 11. Add the `HttpConvAuthFlows.bundle` to your project.
 12. Open the main `UIViewController` and add the following code to connect an `IBAction` from the storyboard's `UIButton` and implement mobile place call:

```
-(IBAction)getMobilePlaceConfiguration:(id)sender {
    //get shared instance of the Mobile Place Controller
    MobilePlaceController *mobilePlaceController = [[MobilePlaceController alloc] init];

    //get configuration properties for the application
    //helper method to show activity indicator
    [self showActivityIndicator];
    //present mobile place UI and use it to get configuration
    //if an email parameter is provided then the UI will not be presented
    [mobilePlaceController configurationParametersWithEmail:nil appID:@"myApplicationId" versionNumber:@"1.0"
    [self closeActivityIndicator];
    if (error) {
        NSLog(@"Error: %@",\nConfigurationParameters: %@", error, configurationParameters);
        NSString *errorDesc = [NSString stringWithFormat:@"Error: %@",\nConfigurationParameters: %@", error,
        [self displayText:errorDesc];
    }
    else {
        //set the configuration for the rest of the application logic
        _configurationParameters = configurationParameters;
        //log the actual content of the configurationProperties dictionary to the console
        NSLog(@"Configuration parameters received: %@", [configurationParameters description]);
        //set the content of the dictionary to be visible on the UI
        [self displayText:[configurationParameters description]];
    }
}
};
```

Parent topic: [Onboarding with SAP Mobile Place](#)

1.8.2 HttpConversation Workflow

Add `HttpConversation` to your iOS project.

Context

To complete this flow, you need the iOS Native Client SDK libraries. The libraries are available as part of the SMP 3.0 SP05 Client SDK Installer.

`HttpConversation` is shipped in form of separate artifacts such as:

- statically linked libraries
- resource bundle folders
- public header files

Before you can use `HttpConversation`, you must add associated artifacts. You also need to add dependent frameworks and configure your build settings. For the rest of the procedure, we assume that you are starting a new project for iOS. If you are working with an existing project, the steps may vary.

To trigger a network request with the `HttpConversation` libraries, you can use this simple logic. In your project, you would centralize the network handling

and separate it from the UI handling. This procedure shows how to code the simplest GET request to a public endpoint on the Web. The endpoint is not protected, so there is no authentication challenge.

Procedure

1. Create a new project or open an existing one in Xcode.
2. Add `HttpConversation` artifacts to your project:
 - `libHttpConversation.a`
 - `libHttpConvAuthFlows.a`
 - `HttpConvAuthFlows.bundle`
 - Header files for the libraries
3. Add artifacts that the `HttpConversation` library depends on:
 - `libClientLog.a`
 - `libE2ETrace2.a`
 - `libSupportability.a`
 - `libPerformanceLib.a`
 - Header files for the libraries
4. Select your project from the Project Navigator, and choose your application's target.
5. Open the Build Phases tab, and within Link Binary with Libraries, add these frameworks:
 - `Security.framework`
 - `libsqlite3.dylib`
6. Choose your project, rather than a specific target, and open the Build Settings tab.
 1. In the Other Linker Flags section, add `-ObjC` and `-all_load`. If these settings are not visible, change the filter in the Build Settings bar from Basic to All.
 2. In the header search path section, adjust the path to look for public header files. Depending on where you copied the header files, the path string should look something like this:

```
$(PROJECT_DIR)/includes/public/**
```
 3. In the library search path section, adjust the path to look for the binary files. Depending on where you copied the library files, the path string should look something like this:

```
$(PROJECT_DIR)/libraries/$(BUILD_STYLE)-universal/**
```
7. Add the `HttpConvAuthFlows.bundle` to your project.
8. Open the main `UIViewController` and add code.

1. Import statements, constants, and IBOutlets:

```
#import "ViewController.h"
#import "HttpConversationManager.h"
#import "CommonAuthenticationConfigurator.h"

#define URL @"http://services.odata.org/V3/Northwind/Northwind.svc/"

@interface ViewController ()
    @property (weak, nonatomic) IBOutlet UITextView *result;
@end

@implementation ViewController {
    HttpConversationManager *conversationManager;
}
```

2. Initialize an instance of the `HttpConversationManager` in the `viewDidLoad` method of the `UIViewController`:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    //create instance of default configurator
    CommonAuthenticationConfigurator *configurator = [[CommonAuthenticationConfigurator alloc] init];
    //set the parentView property to the currently presented ViewController
    configurator.parentView = self;
    //init an instance of the HttpConversationManager
    conversationManager = [[HttpConversationManager alloc] init];
    //configure the HttpConversationManager with the default configurator
    [configurator configureManager:conversationManager];
}
```

3. Connect an `IBAction` from the storyboards `UIButton` and implement the Request sending in it:

```
- (IBAction)doConversation:(id)sender {
    //Create a new NSMutableURLRequest
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL alloc] initWithString:URL]
    //execute the request with the conversation manager
    [conversationManager executeRequest:request completionHandler:^(NSData *data, NSURLResponse *response,
    //process the response data
    NSString *responseString = [NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
    NSLog(@"got response: %@", responseString);
    dispatch_async(dispatch_get_main_queue(), ^{
        [_result setText:responseString];
    });
}
}
```

Parent topic: [Onboarding with SAP Mobile Place](#)

1.9 Developing Logon with Certificate Authentication

Use MAF to develop a logon with Afaria or third-party X.509 certificate provider.

Certificates are one way to authenticate user access to over-the-air distribution of applications, data, and configuration settings for all types of mobile devices, for example, mobile phones, smartphones, tablet computers, ruggedized mobile computers, mobile printers, and mobile POS devices.

By controlling and protecting the data and configuration settings for all mobile devices in the network, developers can reduce support costs and business risks. The intent is to optimize the functionality and security of a mobile communications network while minimizing costs and downtime.

SAP Mobile Platform provides support full support for the SAP-provided solution, Afaria. As of 3.0 SP03, application developers can use the custom extension point to integrate non-Afaria certificate provider options, for example, MobileIron or AirWatch, or file-system installation.

In this section:

- [Developing Logon with Afaria](#)
- [Developing Logon with Third-Party Certificate Provider](#)
- [Mutual Certificate Handling](#)
- [API Reference](#)

Parent topic: [Developing with MAF Logon for iOS](#)

1.9.1 Developing Logon with Afaria

MAFLogonManagerNG fully supports Afaria-based configuration management, which is built into the library.

To enable full Afaria integration, forward openURL requests from AppDelegate to MAFLogonManager:

```
/**
 * Needed because of Afaria initialization.
 */
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication
{
    [self.logonUIViewManager.logonManager setUrl:url];
    return YES;
}
```

See [Creating an MAF Logon Provisioning File in Application Administration](#).

Parent topic: [Developing Logon with Certificate Authentication](#)

1.9.2 Developing Logon with Third-Party Certificate Provider

SAP Mobile Platform SDK includes a Provider API, which enables third-party platforms to provide a way for the application developer to download certificates from their infrastructure. The API allows the third-party developers to focus on their own processes without knowing in detail how the logon processes work.

Parent topic: [Developing Logon with Certificate Authentication](#)

Implementing Content from Third-Party Certificate Providers

The `CertificateProvider` API implements a Logon extension for integrating non-Afaria certificate provider options, for example, MobileIron or AirWatch, or file-system installation.

Prerequisites

A Client Hub application is installed on the client device and SSO pincode is enabled.

Creating an Xcode Library Project

To implement the certificate from the your third-party provider, you need to create a new static linked library project in Xcode.

You can obtain the required SAP Mobile Platform SDK dependencies from Service Marketplace. The libraries come bundled with the SAP Mobile Platform SDK installer. Unzip the installer on your system, then add the following dependency to your project:

```
maflonguining.a min version: 1.203.0
```

Creating the Certificate Provider Implementation

Context

The Provider class implements the `CertificateProvider` protocol:

```
@interface CertificateProviderSample : NSObject <CertificateProvider>
```

Procedure

1. Implement the `getCertificate` method:

```
-(void) getCertificate:(id<CertificateProviderDelegate>) aProviderDelegate
```

In this method, if the provider implementation requires a UI, the current view controller can be retrieved from the provider delegate instance:

```
[aPluginDelegate currentViewController];
```

2. When the `SecIdentityRef` is created, call the provider delegate instance:

```
[pluginDelegate onGetCertificateSuccess:clientId];
```

If any error prevents the return of a valid `SecIdentityRef`, call this method with an `NSError` instance:

```
[pluginDelegate onGetCertificateFailure: anError];
```

After a successful registration, when the application has stopped and restarted, the `LogonManager` needs the `SecIdentityRef` again because it is stored only in the provider. Use the `getStoredCertificate` method:

```
-(BOOL) getStoredCertificate: (SecIdentityRef *) secIdentityRef error: (NSError **) anError
```

When you call this method, return the `SecIdentityRef` that was selected during registration. This is a sync method; therefore, do not show any UI here.

If users inadvertently delete the registration or forget the passcode, `LogonManager` invalidates the registration and calls this method:

```
-(BOOL) deleteStoredCertificateWithError: (NSError **) anError
```

If the provider can successfully remove the stored certificate, `deleteStoredCertificateWithError` returns yes. In case of any error, it returns no and the error description.

Note

This method is called in the beginning of the registration process to ensure that no client certificate exists, for example, from a previous registration.

Setting the CertificateProvider

Procedure

- You can set the `CertificateProvider` on the `MAFLogonUIViewModeler` instance:

```
CertificateProviderSample *certificateProviderSample = [[[CertificateProviderSample alloc] init] autorelease];  
[logonUIViewModeler setCertificateProvider:certificateProviderSample];
```

- If your application does not require a `CertificateProvider`, you can remove it by setting a nil:

```
[logonUIViewModeler setCertificateProvider:nil];
```

Refreshing a Certificate

The certificate used for registration and communicating with the server might become invalid at some point, for example, if the validity period ends.

When a used certificate becomes invalid and you want to use a different, valid one, call:

```
-(void) refreshCertificate;
```

`refreshCertificate`:

1. Calls the `deleteStoredCertificate` method, so `CertificateProvider` can delete the invalid certificate.
2. Calls the `getCertificate` method to set a new, valid certificate. This method is only called if the `deleteStoredCertificate` returns yes.

1.9.3 Mutual Certificate Handling

To execute requests with certificates, set an identity in the SAP Mobile Platform RequestManager.

You can get the identity from MAF Logon after the registration has been finished:

```
SecIdentityRef identity = nil;  
[logonUIViewModeler.getClientIdentity:&identity error:&localError];
```

Once you receive an identity from MAF Logon Manager, set the identity for each request before starting it:

```
if (identity!=nil) {  
    [request setClientCertificateIdentity:identity];  
}
```

The request executes and authenticates with the certificate if these conditions are met:

- The user creation policy is set to certificate.
- SAP Mobile Platform Server security configuration used during registration is set to support mutual certificate handling.

Parent topic: [Developing Logon with Certificate Authentication](#)

1.9.4 API Reference

This API reference describes the certificate provider interface and associated methods.

In this section:

- [CertificateProvider protocol](#)
- [CertificateProviderDelegate protocol](#)

Parent topic: [Developing Logon with Certificate Authentication](#)

1.9.4.1 CertificateProvider protocol

Certificate providers need to implement this interface. Describes methods that will be called by the Logon component during registration.

Syntax

```
@protocol CertificateProvider <NSObject>
```

Members

All members of `CertificateProvider`, including inherited members.

Method	Description
<code>-(void) getCertificate:(id<CertificateProviderDelegate>) delegate;</code>	Invoked during the registration, when MAFLogonManager needs a client certificate and a CertificateProvider instance is set.
<code>-(BOOL) getStoredCertificate:(SecIdentityRef*) secIdentityRef error:(NSError**) anError;</code>	Sync method for getting the stored client certificate. Do not present a UI here. Call this method only when registration is complete: <ul style="list-style-type: none">• After locking secure store• On application start, if secure store is not locked
<code>-(NSError*) deleteStoredCertificate;</code>	CertificateProvider implementation deletes the stored certificate when called. Invoked in these cases: <ul style="list-style-type: none">• During application unregistration.• During application registration: if any successful registration occurred in the past, this call ensures that the previously used certificate will be deleted.• When a DataVault is deleted due to too many failed attempts.• The certificate becomes invalid, and the application developer calls <code>refreshCertificate</code> method on the Public API of MAFLogonManager. Returns yes if the certificate was successfully deleted, or there is no certificate. Otherwise, returns no and an error message.

In this section:

- [getCertificate:delegate method](#)
- [getStoredCertificate:error method](#)
- [deleteStoredCertificateWithError:error method](#)

Parent topic: [API Reference](#)

1.9.4.1.1 getCertificate:delegate method

Invoked during the registration, when MAFLogonManager needs a client certificate and a CertificateProvider instance is set.

Syntax

```
-(void) getCertificate:(id<CertificateProviderDelegate>) delegate;
```

Parameters

- **CertificateProviderDelegate** – a CertificateProviderDelegate instance.

Parent topic: [CertificateProvider protocol](#)

1.9.4.1.2 getStoredCertificate:error method

Synchronous method for getting the stored client certificate. Do not present a UI here.

Call this method only when registration is complete:

- After unlocking secure store
- On application start, if secure store is not locked

If `getStoredCertificate()` returns null or error, the application developer should make a call to the [Logon API](#) method for `refreshCertificate()`.

Syntax

```
-(BOOL) getStoredCertificate:(SecIdentityRef*) secIdentityRef error:(NSError**) anError;
```

Parameters

- **secIdentityRef** – address of a SecIdentityRef variable that stores the returned value on success.
- **anError** – the address of an NSError*. Returns the address of an error, or nil on success.

Returns

Boolean that indicates whether secIdentityRef returns successfully.

Parent topic: [CertificateProvider protocol](#)

1.9.4.1.3 deleteStoredCertificateWithError:error method

CertificateProvider implementation deletes the stored certificate when called.

Invoked in these cases:

- During application unregistration.
- During application registration: if any successful registration occurred in the past, this call ensures that the previously used certificate will be deleted.
- When a DataVault is deleted due to too many failed attempts.
- The certificate becomes invalid, and the application developer calls `refreshCertificate` method on the Public API of MAFLogonManager.

In the first three cases, LogonPlugin and the Mediator call `deleteStoredCertificate()` on the CertificateProvider.

Returns yes if the certificate was successfully deleted, or there is no certificate. Otherwise, returns no and an error message.

Syntax

```
-(NSError*) deleteStoredCertificate;
```

Parameters

- **anError** – the address of an NSError*. Returns the address of an error, or nil on success.

Returns

Boolean that indicates whether the certificate was successfully deleted.

Parent topic: [CertificateProvider protocol](#)

1.9.4.2 CertificateProviderDelegate protocol

This interface describes methods that the certificate providers can invoke to respond to the methods called by the Logon component.

Syntax

```
@protocol CertificateProviderDelegate <NSObject>
```

Members

All members of `CertificateProviderDelegate`, including inherited members.

Method	Description
<code>-(UIViewController*) currentViewController;</code>	Returns the current UIViewController, which application developers can use to present their own screen(s).
<code>-(void) onGetCertificateSuccess:(SecIdentityRef) aCertObject;</code>	Delegate method that notifies the CertificateProviderDelegate implementation when SecIdentityRef has been successfully loaded.
<code>-(void) onGetCertificateFailure:(NSError*) error;</code>	Delegate method that notifies the CertificateProviderDelegate implementation when SecIdentityRef cannot be loaded.

In this section:

- [currentViewController method](#)
- [onGetCertificateSuccess method](#)
- [onGetCertificateFailure:error method](#)

Parent topic: [API Reference](#)

1.9.4.2.1 currentViewController method

Returns the current UIViewController, which application developers can use to present their own screen(s).

Syntax

```
-(UIViewController*) currentViewController;
```

Returns

rootUIViewController

Parent topic: [CertificateProviderDelegate protocol](#)

1.9.4.2.2 onGetCertificateSuccess method

Delegate method that notifies the CertificateProviderDelegate implementation when SecIdentityRef has been successfully loaded.

Syntax

```
-(void) onGetCertificateSuccess:(SecIdentityRef) aCertObject;
```

Parameters

- **secIdentityRef**

Parent topic: [CertificateProviderDelegate protocol](#)

1.9.4.2.3 onGetCertificateFailure:error method

Delegate method that notifies the CertificateProviderDelegate implementation when SecIdentityRef cannot be loaded.

Syntax

```
-(void) onGetCertificateFailure:(NSError*) error;
```

Parameters

- **error** – NSError* instance, which contains the error description.

Parent topic: [CertificateProviderDelegate protocol](#)

1.10 SAML Authentication

Develop SAML-enabled registration for iOS.

To implement SAML-enabled registration, you:

- Set the runtime settings
- Integrate with SAP Mobile Place

You can use these approaches with MAF Logon or the Apple iOS UserDefaults API.

Note

SAML Authentication works only with SAP HANA Cloud Platform mobile services. SAP HANA Cloud Platform mobile services is currently planned but not available. SAP has no obligation to develop or release this product. All future development is subject to change and may be changed by SAP at any time for any reason without notice.

MAF Logon

Using MAF Logon is the simplest and recommended way to use SAML-protected resources. By default, MAF Logon uses Mobile Place and gets the configuration based in the end user's e-mail address. Once you integrate MAF Logon into your application, you have completed SAML enablement on the client side. If you want to use a different setup, you can use the MAF Logon runtime configuration to enable the SAML flow. SAML support is integrated with the `HttpConversation` library via the `IManagerConfigurator` object exposed by the `LogonUIFacade` class. Refer to the documentation of these classes and to that of the `HttpConversationManager` class.

The structure goes inside the `MAFLogonConfigurationContextDefaultValues.plist`. For example:

```
//Get the defaultValues object form UserDefaults
NSMutableDictionary* defaultValues = [self.logonUIWebViewManager.logonManager defaultValues];

//Get config dictionary from defaultValues dictionary
NSMutableDictionary* config = [defaultValues objectForKey:@"keyMAFLogonRegistrationContextConfig"];

//Get auth array from config dictionary
NSMutableArray* auth = [defaultValues objectForKey:@"keyMAFLogonRegistrationContextAuth"];

//Set new content
NSMutableDictionary* samlAuth = [[NSMutableDictionary alloc] init];
[samlAuth setObject:@"saml2.web.post" forKey:@"type"];
NSMutableDictionary* samlConfig = [[NSMutableDictionary alloc] init];
[samlConfig setObject:@"com.sap.cloud.security.login" forKey:@"saml2.web.post.authchallengeheader.name"];
[samlConfig setObject:@"finishEndpointParam" forKey:@"saml2.web.post.finish.endpoint.redirectparam"];
[samlConfig setObject:@"saml2.web.post.finish.endpoint.uri" forKey:@"SAMLAuthLauncher"];
[samlAuth setObject:samlConfig forKey:@"config"];
[config setObject:samlAuth forKey:@"keyMAFLogonRegistrationContextAuth"];
```

Apple iOS UserDefaults API

See [Logon Screen Configuration Options](#).

Parent topic: [Developing with MAF Logon for iOS](#)

1.11 Customizing the Logon UI

To be able to use the customization options that MAF Logon UI provides, implement `MAFLogonUICustomizationDelegate` in one of your classes.

To implement it, use:

```
#import <UIKit/UIKit.h>
#import "MAFLogonNGDelegate.h"
#import "MAFLogonUICustomizationDelegate.h"

@interface MAFLogonNGSampleViewController : UIViewController <MAFLogonNGDelegate, MAFLogonUICustomizationDelegate>

@end
```

You can decide whether to provide your own `UIView` subclass for the header and the footer views. This is the most flexible way to customize the screens. The `UIView` subclass can contain any layout and elements you wish. To provide a custom `UIView`, use code similar to this:

```
- (void) willRenderHeaderFooterFromCustomContext: (MAFLogonUICustomizationContext*) aCustomContext {

    // the current screen can be determined using aCustomContext.screenType property

    if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPhone) {
        if (UIDeviceOrientationIsLandscape([[UIApplication sharedApplication] statusBarOrientation])){
            aCustomContext.headerView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPhone_landscape_...]]];
            aCustomContext.footerView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPhone_landscape_...]]];
        }
        else{
            aCustomContext.headerView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPhone_portrait_...]]];
            aCustomContext.footerView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPhone_portrait_...]]];
        }
    }
    else{

        if (UIDeviceOrientationIsLandscape([[UIApplication sharedApplication] statusBarOrientation])){
```

```

        aCustomContext.headerView = [[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPad_landscape_header.png"]] autorelease];
        aCustomContext.footerView = [[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPad_landscape_footer.png"]] autorelease];
    }
    else{
        aCustomContext.headerView = [[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPad_portrait_header.png"]] autorelease];
        aCustomContext.footerView = [[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"iPad_portrait_footer.png"]] autorelease];
    }
}
}
}

```

MAF provides a more basic customization option with the `willRenderUIFromContext` delegate. Use this delegate to provide localized titles and descriptions for any of the logon UI screens:

```

-(void)willRenderUIFromContext:(MAFLogonUIContext *)aUIContext{
    aUIContext.title=[NSString stringWithFormat:@"Custom Title"];
    aUIContext.headerTitle=[NSString stringWithFormat:@"Custom Header Title"];
    aUIContext.headerDescription=[NSString stringWithFormat:@"Custom Header Description"];
    aUIContext.footerDescription=[NSString stringWithFormat:@"Custom Footer Description"];
}

```

You can also check which screen is presented, and perform the customization per screen.

Parent topic: [Developing with MAF Logon for iOS](#)

1.12 Logon Screen Configuration Options

To change the behavior of `MAFLogonManagerNG`, adjust the feature and default values configuration files at build time. The plist format configuration files are located in the `MAFLogonManagerNG.bundle` folder. plist is an XML format that can be adjusted in Xcode or in any text editor.

Feature Configuration

To change the behavior of the component, adjust the `MAFLogonManagerOptions.plist` file, which is in the `MAFLogonManagerNG` bundle folder in your project.

Key	Value Type	Description	Default Value
keyMAFURLSchemeForAfaria	String	Changes the default URL Scheme of the Afaria application.	EMPTY
keyMAFUseAfaria	BOOL	Determines whether to use Afaria. NO - the Afaria client application is not called, even if it is present on the iOS device YES - Afaria client application is always called.	YES

You can also declare resources with the `MAFLogonUIOptions.plist` feature configuration, which is in the `MAFLogonUI` bundle.

Key	Value Type	Description	Default Value
keyMAFLogonUseFlowMode	BOOL	YES - Logon Core tries to find a suitable communicator implementation for the infrastructure used. It analyses the input provided by the user and triggers test requests to test the various onboarding scenarios.	YES
keyMAFLogonAllowAutomaticSecureStoreCreation	BOOL	Whether to allow the creation of the secure store automatically. NO - the secure store should be provided/created before the logon component is initialized.	YES
keyMAFLogonUseSplashScreen	BOOL	Determines whether to present splash screen when logon operation is called for the first time.	YES
keyMAFLogonOperationContextHideRegistrationFields	BOOL	Defines which fields to show on the registration screen.	Multiple values

Default Values Configuration

The `MAFLogonManagerNG.bundle` folder contains the `MAFLogonOperationContextDefaultValues.plist` file, which declares the configuration structure. The plist defines the keys that are used to set default values for the logon scenarios. `keyMAFLogonOperationContextDefaultValues` is the main key in the default values dictionary. The only screen that supports default values is the registration screen. You can set the default values for the registration screen can be set in the `idMAFLogonOperationContextTypeRegister` key. Use these keys to set default values for the individual input fields:

Key	Value Type	Description	Default Value
keyMAFLogonOperationContextServerPortNonSecure	Number	Non secure port	8080
keyMAFLogonOperationContextServerPortSecure	Number	Secure port	8081

erPortSecure			
keyMAFLogonOperationContextServerHost	String	Server host name without protocol or port or suffixes	EMPTY
keyMAFLogonOperationContextServerPort	Number	Server port	EMPTY
keyMAFLogonOperationContextIsHttps	BOOL	Whether to use secure channel	YES
keyMAFLogonOperationContextServerDomain	String	The default domain to use. Only used when an SAP Mobile Platform Server is present. The domain separation should be supported by the used SAP Mobile Platform Server.	default
keyMAFLogonOperationContextFarmId	String	The default farm ID name to be used if a Relay server is included.	EMPTY
keyMAFLogonOperationContextSecurityConfig	String	The default security configuration name to use.	EMPTY
keyMAFLogonOperationContextGatewayClient	String	The default sap-client to be used when communicating with the GW.	EMPTY
keyMAFLogonOperationContextGatewayPingPath	String	The default GW ping path to be used.	sap/bc/ping
keyMAFLogonOperationContextResourcePath	String	The default path to be used for routing through the reverse proxy if any exists.	EMPTY
keyMAFLogonOperationContextBackendUserName	String	The default GW username to be used.	EMPTY
keyMAFLogonOperationContextPasswordPolicy	Dictionary	The default password policy to apply if no server side policy was provided.	Default password policy

To set these values, use code similar to this:

```
//replace this with your way of accessing MAFLogonUIViewModeler->NSObject<MAFLogonNGPublicAPI> (the logon manager)
NSMutableDictionary* defaultValues = [logonUIViewModeler.logonManager defaultValues];

NSMutableDictionary* contextDefaultValues = [defaultValues objectForKey:@"keyMAFLogonOperationContextDefaultValues"];
NSMutableDictionary* registerContextType = [contextDefaultValues objectForKey:@"idMAFLogonOperationContextTypeRegister"];

[registerContextType setObject:@"300" forKey:@"keyMAFLogonOperationContextGatewayClient"];

[defaultValues setObject:contextDefaultValues forKey:@"keyMAFLogonOperationContextDefaultValues"];
```

These values are available for SAML registration.

Key	Value Type	Description	Default Value
keyMAFLogonRegistrationContextConfig	Dictionary		
keyMAFLogonRegistrationContextAuthConfigs	Array	.	
saml2.web.redirect.finish.endpoint.param	String		
saml2.web.redirect.finish.endpoint	String		
saml2.web.redirect.identifyingHeader.name	String		
oauth2.authorizationEndpoint	String		
oauth2.tokenEndpoint	String		
oauth2.clientID	String		
oauth2.scopes	String		
keyMAFLogonRegistrationContextAuthType	String Array		

When the values are set, the SAML registration will be executed. For example:

```
<key>keyMAFLogonRegistrationContextConfig</key>
<dict>
  <key>keyMAFLogonRegistrationContextAuthConfigs</key>
  <array>
    <dict>
      <key>saml2.web.redirect.finish.endpoint.param</key>
      <string>finishEndpointParam</string>
      <key>saml2.web.redirect.finish.endpoint</key>
      <string>https://samltesthmsdktenant.neo.ondemand.com/samltest/SAMLAuthLauncher</string>
      <key>saml2.web.redirect.identifyingHeader.name</key>
      <string>com.sap.cloud.security.login</string>
    </dict>
  <!-- example <dict>
    <key>oauth2.authorizationEndpoint</key>
    <string>https://samltesthmsdktenant.neo.ondemand.com/samltest/Authorization</string>
```

```

<key>oauth2.tokenEndpoint</key>
<string>https://samltesthmsdktenant.neo.ondemand.com/samltest/Token</string>
<key>oauth2.clientID</key>
<string>65465465465465465465465465465465</string>
<key>oauth2.scopes</key>
<string>profile|feed</string>
</dict> -->
</array>
<key>keyMAFLogonRegistrationContextAuthType</key>
<array>
  <string>saml.web.post</string>
  <!-- example <string> oauth2 </string> -->
</array>
</dict>

```

Enforcing Gateway-Only Registration

To enforce Gateway-only registration, use the `keyMAFLogonOperationContextCommunicatorId` field in the `keyMAFLogonOperationContextHideRegistrationFields` part of the `MAFLogonOperationsDefaultValues.plist` file. The default value is `yes`, which hides the field in the LogonUI.

To change the behavior, you can set the value to `no` or enter code, for example:

```

// get current default values from LogonManager
NSMutableDictionary* defaultValues = [logonUIManager.logonManager defaultValues];

// get field visibility section
NSMutableDictionary* contextDefaultValues = [defaultValues objectForKey:@"keyMAFLogonOperationContextHideRegistrationFields"];

// set communicator id field visibility to NO (hide == NO)
[contextDefaultValues setObject:[NSNumber numberWithInt:NO] forKey:@"keyMAFLogonOperationContextCommunicatorId"];

// set back the default value to LogonManager
[logonUIManager.logonManager setDefaultValues:defaultValues];

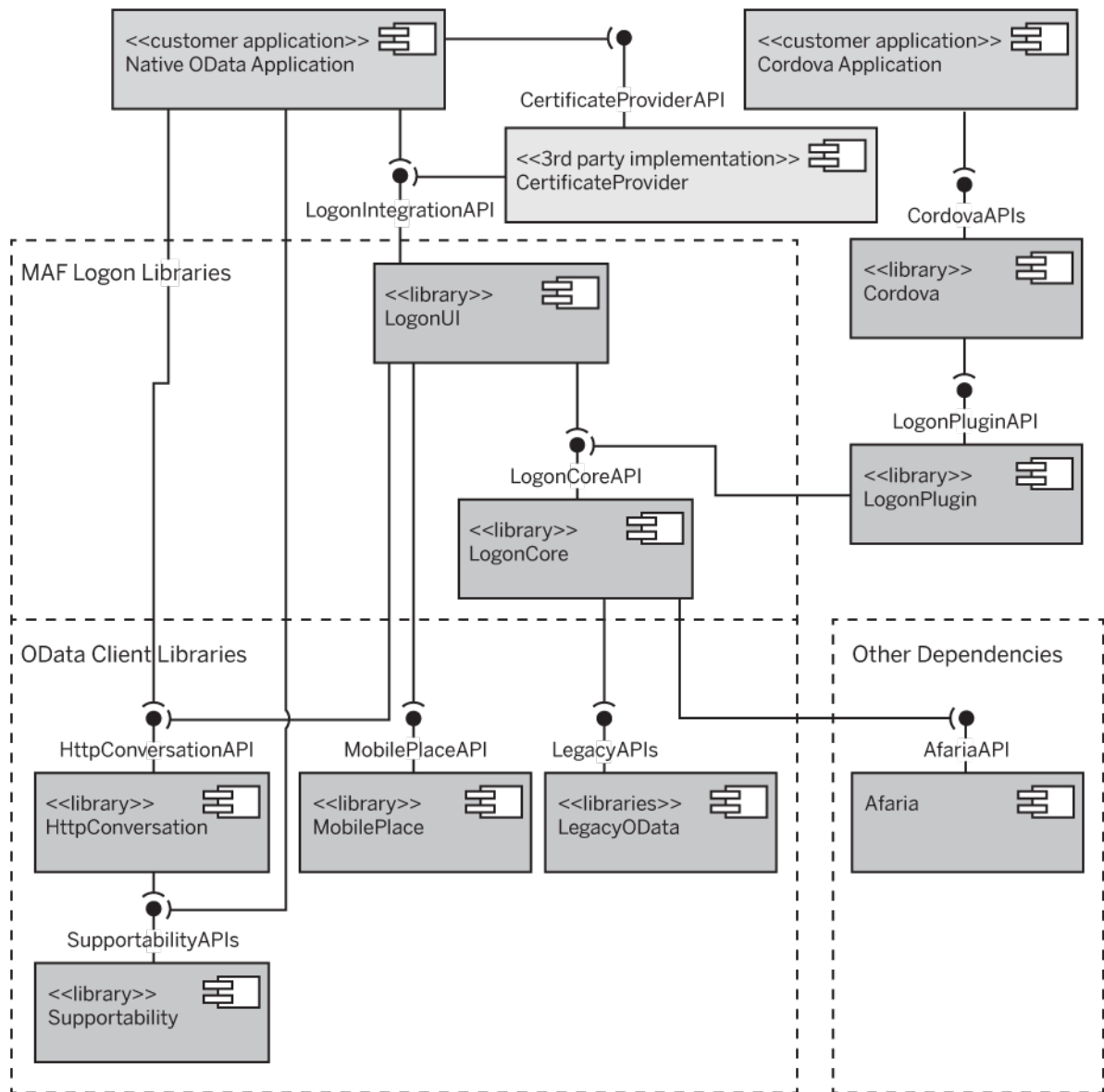
```

When you add the **Gateway Only** field to the LogonUI, it defaults to `off`; the server detection logic does not change. To enforce Gateway registration, turn it on. When it is on, LogonManager only registers to the Gateway server.

Integrating Directly with Logon Manager

`MAFLogonManagerNG` is an Objective-C library that you can integrate with any UI implementation. MAF includes a default native Logon UI, but you can also build your own UI.

This image shows a possible architecture of an application integrating directly with `MAFLogonManagerNG` without using `MAFLogonUI`:



You can integrate your own UI solution through the MAFLogonMediator by implementing your own LogonUIViewManager. This is similar to using the default MAFLogonUING implementation, because MAFLogonManagerNG uses context objects to construct the UI:

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

@protocol MAFLogonNGPublicAPI;
@protocol MAFLogonUICustomizationDelegate;

/**
 * This class handles the navigation between the screens
 */
@interface MAFLogonUIViewManager : NSObject <MAFLogonUIViewManaging, MAFLogonUIViewManagingDelegate>

/**
 * The logonMediator where the operations can be initiated. One has to set up the @see logonDelegate to be notified
 */
@property (readonly) NSObject<MAFLogonNGPublicAPI> *logonMediator;

/**
 * The actually presented viewController of the application. This will be used to present the logon related screens.
 */
@property (nonatomic, retain) UIViewController* parentViewController;

@end

The LogonUIViewManager must store a reference to the LogonMediator and the presenting ViewController. The second property presents or dismisses a modal
view. To initialize LogonUIViewManager, use:

- (id)init {
    self = [super init];
    if (self) {
        self.logonMediator = [[MAFLogonMediator alloc] init];
        self.logonMediator.logonUIViewManager = self;
    }
    return self;
}
```

```
}
```

To execute a logon operation, call:

```
- (IBAction)logonButtonTapped:(id)sender {  
    [logonUIViewManager.logonManager logon];  
}
```

MAFLogonManagerNG responds to the request through the MAFLogonUIViewManaging interface:

```
#pragma mark - MAFLogonUIViewManaging methods
```

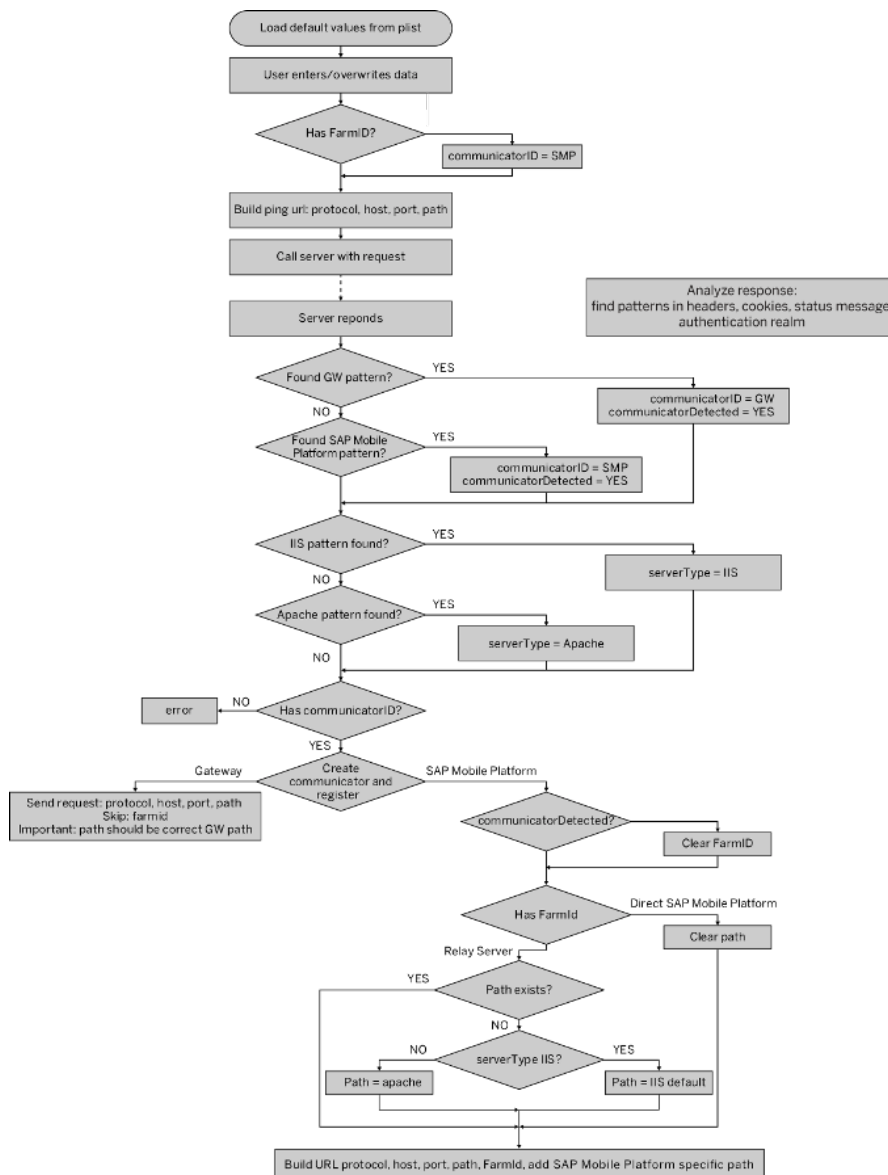
```
-(void) showViewWithUIContext:(NSMutableDictionary *)aUIContext{  
    if([[NSThread currentThread] isMainThread]){  
        [self doShowViewWithUIContext:aUIContext];  
    }  
    else{  
        dispatch_sync(dispatch_get_main_queue(), ^{  
            [self doShowViewWithUIContext:aUIContext]; //do your custom UI building here!  
        });  
    }  
}  
  
-(void) operationRespondsWithOperationContext:(NSMutableDictionary *)anOperationContext error:(NSError *)anError  
    [currentViewManager operationRespondsWithOperationContext:anOperationContext error:anError];  
}  
  
-(void) closeUI {  
    [currentViewManager release];  
    currentViewManager = nil;  
    currentOperationContextType = @"";  
  
    [[self baseViewController] popToRootViewControllerAnimated:NO];  
    [self.parentViewController dismissViewControllerAnimated:YES completion:^(void){  
    }];  
}  
  
#pragma mark - MAFLogonUIViewManaging  
-(void) operationRespondsWithOperationContext:(NSMutableDictionary *)anOperationContext error:(NSError *)anError  
    [self hideActivityIndicator];  
    if (anError != nil && anError.code!=0) {  
        [self presentError:anError];  
    }  
}  
}
```

Supported Onboarding Scenarios

The MAFLogonManagerNG supports multiple onboarding scenarios, based on the most common infrastructure setups supported by enterprise mobile applications. The company IT of the customer can have SAP Mobile Platform installed and use relay server to route Internet requests to intranet-hosted Gateway systems. Another possibility is to have SAP Mobile Platform, but replace the relay server with IIS or Apache and use it as a reverse proxy.

Logon Method Selection

MAFLogonManagerNG determines the target infrastructure, which can be SAP Mobile Platform or Gateway. It uses this logic:



The operation context object, which contains all information entered by the end user, passes the information to MAFLogonManagerNG. Based on the information, MAFLogonManagerNG decides whether the device user wants to use SAP Mobile Platform Server. If the user provides a farm ID, the logon manager constructs and sends an SAP Mobile Platform Server communicator-based test request.

Based on the test request response headers, code and cookies, the logon manager decides to use one of these scenarios:

- Direct Gateway
- Direct SAP Mobile Platform Server
- SAP Mobile Platform Server with relay server
- SAP Mobile Platform Server with third-party reverse proxy

If none of these scenarios are true, MAFLogonManagerNG cannot identify the communicator, and generates an error message.

Additionally, MAFLogonManagerNG searches for any information in the context and selects a method based on all of this information. The context can have default values that are configured by the application developer.

Success and Failure Handling

Onboarding consists of two steps. The first is registration, where MAFLogonManagerNG uses input from the device user to contact the server and register. If this step is successful, your application can send requests to the GW server and obtain data. The second step is to create a data vault on the user device, in which to securely store user credentials and other sensitive information. The data vault must be secured with a passcode and satisfy the requirements of the password policy.

Both of these steps can generate success and failure messages. Error messages are identified by an error code and an error domain, which fits into the iOS error message handling method. You can find definitions for error messages and error codes in the `MAFLogonManagerNGErrors.h` header file.

```

#define kErrorDomainLogonManager @"MAFLogonManagerNGErrorDomain"

#define errMAFLogonManagerErrorOperationCancelled 0
#define errMAFLogonManagerErrorInvalidParameter 1
#define errMAFLogonManagerErrorInvalidOperationCode 2
#define errMAFLogonManagerErrorCommunicationManagerError 3
#define errMAFLogonManagerErrorUserAlreadyLoggedIn 4
#define errMAFLogonManagerErrorOperationExecutionInProgress 5
#define errMAFLogonManagerErrorInvalidOperationContext 6
  
```

```
#define errMAFLogonManagerErrorInvalidCommunicatorId 7
#define errMAFLogonManagerErrorInvalidActionItemId 8
#define errMAFLogonManagerErrorCouldNotDecideCommunicator 9
#define errMAFLogonManagerErrorPasswordsNotEqual 10
#define errMAFLogonManagerErrorOperationNotAllowed 11
#define errMAFLogonManagerErrorInvalidServerHost 12
#define errMAFLogonManagerErrorInvalidBackendPassword 13
#define errMAFLogonManagerErrorPasscodeChangeNotEnabled 14
#define errMAFLogonManagerErrorUploadTraceFailed 15
```

In this section:

- [Data Vault Life Cycle](#)

Parent topic: [Developing with MAF Logon for iOS](#)

1.12.1 Data Vault Life Cycle

If SAP Mobile Platform Server is used, an SAP Mobile Platform administrator can enable the password policy. If enabled, the user must satisfy each rule. If "Allow Default Password" is enabled in the password policy, the Set Passcode screen does not appear, but the user can enable the app passcode using the Change Passcode option.

Logon Manager Default Password Policy

If the password policy is disabled, or if communication is not through the SAP Mobile Platform, MAFLogonManagerNG applies the default password policy. You can configure the default password policy from the MAFLogonOperationsDefaultValues.plist located in the MAFLogonManagerNG.bundle folder of your project.

Table 1: Default Password Policy Rules

Rule Name	Description	Default Value
keyMAFLogonPasswordPolicyIsEnabled	Whether a password policy is provided	True
keyMAFLogonPasswordPolicyExpirationDays	Number of days after which the password policy expires	0 ~ infinite
keyMAFLogonPasswordPolicyHasDigits	Whether the password requires one or more numeric digit	NO
keyMAFLogonPasswordPolicyHasLowerCaseLetters	Whether the password requires one or more lowercase letters	NO
keyMAFLogonPasswordPolicyHasSpecialLetters	Whether the password requires one or more special characters	NO
keyMAFLogonPasswordPolicyHasUpperCaseLetters	Whether the password requires one or more uppercase letters	NO
keyMAFLogonPasswordPolicyIsDefaultPassword Allowed	Allows the device user to switch off custom app passcode	YES
keyMAFLogonPasswordPolicyIsDefaultPassword UsageOnByDefault	If a default password is allowed, whether the switch on the device is, by default, on	NO
keyMAFLogonPasswordPolicyLockTimeout	Length of time, in seconds, after which the secure store is locked	0 ~ infinite
keyMAFLogonPasswordPolicyMinLength	Minimum length of the password	8
keyMAFLogonPasswordPolicyMinUniqueChars	Minimum number of unique characters in password	0 ~ none
keyMAFLogonPasswordPolicyRetryLimit	Maximum number of retries before the secure store is erased	0 ~ infinite

Creating the Data Vault

The data vault is created after a successful registration. If device users are allowed to do so, they can switch off the data vault passcode. Even if the user does not specify a passcode, the data is secured with a default datavault passcode.

Protecting the Data Vault

If the data vault is locked with a custom passcode, the logon UI is presented with an Unlock screen. Users can unlock the data vault with the passcode they have set. If the user provides a wrong passcode multiple times, the data vault deletes itself. You can determine the maximum number of tries by a data vault property, which you can set in the password policy.

Setting a Data Vault Timeout

You can set the data vault timeout in the password policy. MAFLogonManagerNG downloads it from the SAP Mobile Platform Server during registration.

Forgotten Passcode

When device users forget their passcodes, they can tap the Forgot passcode button on the Unlock screen, which erases all client-side data.

If there is an SAP Mobile Platform Server in the landscape, tapping the Forgot passcode button deletes both the client- and server-side registrations. This works only when the device is connected to a network where the SAP Mobile Platform Server is reachable. Otherwise, an administrator must manually remove the server-side registration.

Updating Data Vault Properties

MAFLogonManagerNG does not currently support policy changes. That is, if the password policy is changed on the SAP Mobile Platform Server after a device user registers, MAFLogonManagerNG does not download and apply the new policy.

Error Messages

Data vault creation has a set of error codes and error messages that the MAFLogonManagerNG can report. You can find the error codes and messages in the `MAFSecureStoreManagerErrors.h` file.

```
#define kErrorDomainMAFSecureStoreManager    @"MAFSecureStoreManagerErrorDomain"

#define keyMAFSecureStoreManagerErrorDescriptionParameters    @"errorDescriptionParameters"
#define keyMAFSecureStoreManagerErrorExceptionName            @"exceptionName"
#define keyMAFSecureStoreManagerErrorExceptionReason          @"exceptionReason"

#define errMAFSecureStoreManagerErrorUnknown                    0
#define errMAFSecureStoreManagerErrorAlreadyExists              1
#define errMAFSecureStoreManagerErrorDataTypeError             2
#define errMAFSecureStoreManagerErrorDoesNotExist              3
#define errMAFSecureStoreManagerErrorInvalidArg                4
#define errMAFSecureStoreManagerErrorInvalidPassword           5
#define errMAFSecureStoreManagerErrorLocked                    6
#define errMAFSecureStoreManagerErrorOutOfMemory               7
#define errMAFSecureStoreManagerErrorPasswordExpired           8
#define errMAFSecureStoreManagerErrorPasswordRequired          9
#define errMAFSecureStoreManagerErrorPasswordRequiresDigit    10
#define errMAFSecureStoreManagerErrorPasswordRequiresLower    11
#define errMAFSecureStoreManagerErrorPasswordRequiresSpecial   12
#define errMAFSecureStoreManagerErrorPasswordRequiresUpper    13
#define errMAFSecureStoreManagerErrorPasswordUnderMinLength    14
#define errMAFSecureStoreManagerErrorPasswordUnderMinUniqueChars 15
```

Handling Timeout with MAF Logon Components

MAF manages Secure Store lock timeout in the following ways:

- It keeps the Secure Store open until the application is in the foreground. This way you can avoid data request failures happening because the application cannot read the necessary sensitive information (such as username and password) from the Secure Store.
- When the application is sent to the background, MAF sets the Secure Store lock timeout to a specified value. When the application is brought to the foreground again, the Secure Store can be either open or closed, depending on the amount of time the application spent in the background.

When the Secure Store is closed and the application calls the `logonManager login` API, MAF presents the MAF Logon UI and the user can unlock the Secure Store with the app passcode provided at registration time.

The system administrator can have the user specify a custom app passcode, or can let the user have the option to switch off the passcode functionality. If the user switches off this functionality, the Secure Store is protected with the default password provided by the underlying component. The system administrator can specify this behavior per application or per connection type by adjusting the SAP Mobile Platform Server-side password policy.

If the password policy is not set on the server, the default password policy is enforced:

- The password must be minimum 8 characters, regardless of the type of characters.
- The user can switch off the app passcode.
- Immediate timeout when the application is sent into background.

Parent topic: [Logon Screen Configuration Options](#)

1.13 MAF Onboarding Scenarios

The MAF Logon Manager supports multiple onboarding scenarios, based on the most common infrastructure setups supported by enterprise mobile applications.

All these scenarios require the user to enter different parameters during the onboarding process (see the tables below). However, using Afaria, Mobile Place, or the Client Hub can provide some or all of the parameters.

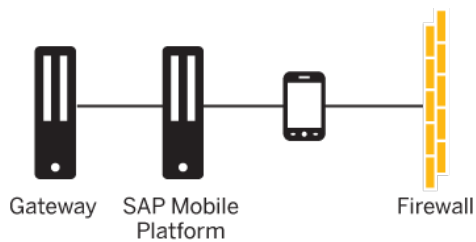
For example, an enterprise might have SAP Mobile Platform installed and use reverse proxy to route Internet requests to intranet-hosted Gateway systems. An alternate scenario also uses SAP Mobile Platform, but does not use reverse proxy, and only allows intranet access to the Gateway.

Note

You can test HTTPS connections only by using actual iOS devices. Additionally, the root CA certificate and all intermediate CA certificates that are used to sign the SAP Mobile Platform or reverse proxy certificates must be installed on the device. The iOS simulator does not contain the full implementation of the iOS Security Framework, and the server's SSL certificate cannot be validated by the iOS CFNetwork Framework.

Direct SAP Mobile Platform

The direct SAP Mobile Platform landscape contains an SAP NetWeaver Gateway (subsequently referred to as Gateway) server. SAP Mobile Platform Server is installed and configured to connect to the Gateway server. Connections are accepted only from the company intranet.



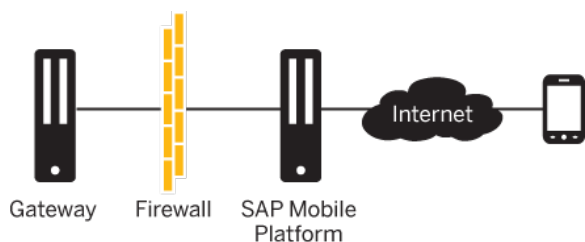
This table shows the information that users must provide, in the MAF login UI, to use this type of connection. The parameter values depend on the configuration the administrators set.

Parameter Name	Example Value
Server Host	<SMP_HOST_NAME>
User Name	<USERNAME>
Password	<PASSWORD>
Server Port	<SMP_SERVER_HTTP(S)_PORT>
Secured Connection	ON - DEFAULT
Security Configuration	EMPTY - DEFAULT <SMP_SECURITY_CONFIG_NAME>

By default, the secured connection (HTTPS) is ON, so also by default, the connection type is HTTPS.

Cloud-Based SAP Mobile Platform

The cloud-based SAP Mobile Platform landscape is similar to the direct SAP Mobile Platform scenario; however, the SAP Mobile Platform Server is installed in the cloud.



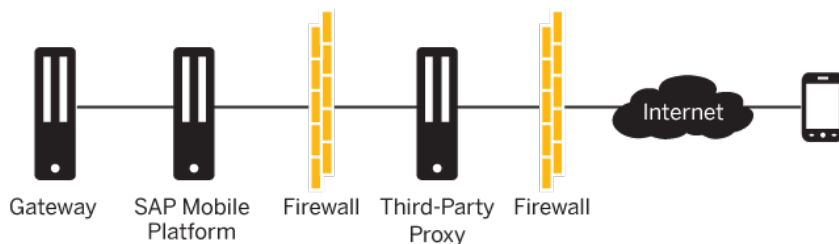
This table shows the information that users must provide, in the MAF login UI, to use this type of connection. The parameter values depend on the configuration the administrators set.

Parameter Name	Example Value
Server Host	< SMP_SERVER_HOST_NAME>
User Name	<USERNAME>
Password	<PASSWORD>
Server Port	< SMP_SERVER_HTTPS_PORT>
Secured Connection	ON - DEFAULT
Security Configuration	EMPTY - DEFAULT <SMP_SERVER_SECURITY_CONFIG_NAME>

By default, the secured connection (HTTPS) is ON, so also by default, the connection type is HTTPS.

SAP Mobile Platform with Third-Party Proxy

The SAP Mobile Platform with third-party proxy landscape is similar to the direct SAP Mobile Platform scenario, but you can access the SAP Mobile Platform Server from both the Internet and intranet. The reverse proxy enables the mobile device to access the landscape from the Internet.



This table shows the information that users must provide, in the MAF login UI, to use this type of connection. The parameter values depend on the configuration the administrators set.

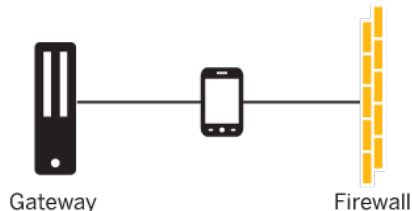
Parameter Name	Example Value
Server Host	<PROXY_HOST_NAME>
User Name	<USERNAME>

Password	<PASSWORD>
Server Port	<PROXY_HTTP(S)_PORT>
Secured Connection	ON
Security Configuration	EMPTY - DEFAULT <SMP_SERVER_SECURITY_CONFIG_NAME>

By default, the secured connection (HTTPS) is ON, and is set to port 443. When the secured connection is OFF, the default port used is 80.

Direct Gateway

The direct Gateway landscape contains only the Gateway system, and restricts access to the intranet. The mobile device must connect to the intranet via internal Wi-Fi or through VPN.



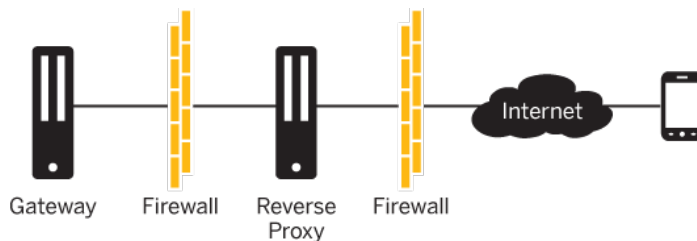
This table shows the information that users must provide, in the MAF login UI, to use this type of connection. The parameter values depend on the configuration the administrators set.

Parameter Name	Example Value
Server Host	<GATEWAY_HOST_NAME>
User Name	<GATEWAY_USERNAME>
Password	<GATEWAY_PASSWORD>
Path	<PATH_TO_GATEWAY_CONTENT>
Server Port	<GATEWAY_HTTP(S)_PORT_NUMBER>
Secured Connection	ON - DEFAULT

The default value of the path is the ping URL of the Gateway server (<https://gwhost:gwport/sap/bc/ping>). You can customize the path by modifying the MAFLogonManagerNG.bundle/MAFLogonOperationsDefaultValues.plist default value of keyMAFLogonOperationContextResourcePath. If you modify this file in the application's IPA file, you must resign the file with a valid signing key before you can distribute it.

Gateway with Third-Party Proxy

The Gateway with third-party proxy landscape contains a Gateway server that resides inside the company intranet, and a third-party reverse proxy that enables the mobile device to access the Gateway from the Internet.



This table shows the information that users must provide, in the MAF login UI, to use this type of connection. The parameter values depend on the configuration the administrators set.

Parameter Name	Example Value
Server Host	<GATEWAY_HOST_NAME>
User Name	<GATEWAY_USERNAME>
Password	<GATEWAY_PASSWORD>
Server Port	<GATEWAY_HTTPS_PORT_NUMBER>
Secured Connection	ON - DEFAULT

The default value of the path is the ping URL of the Gateway server (<https://gwhost:gwport/sap/bc/ping>). You can customize the path by modifying the MAFLogonManagerNG.bundle/MAFLogonOperationsDefaultValues.plist default value of keyMAFLogonOperationContextResourcePath. If you modify this file in the application's IPA file, you must resign the file with a valid signing key before you can distribute it.

SAP HANA Cloud Platform Mobile Services

Mobile services are deployed on the SAP HANA Cloud Platform, which also runs a HANA instance, and the services are typically protected by SAML authentication. The parameters that this authentication method requires must be provided by Mobile Place, Afaria, or the runtime configuration; the Login UI is not able to acquire these entries interactively from the user.

Note

SAP HANA Cloud Platform mobile services is currently planned but not available. SAP has no obligation to develop or release this product. All future development is subject to change and may be changed by SAP at any time for any reason without notice.



HANA Cloud Platform

Other Configuration Prerequisites for Onboarding Scenarios

You must properly configure the Gateway system so the MAF logon UI component can detect the direct Gateway setup.

- Back-end users must have authorization to access the Gateway system's ping URL.
- The `is/HTTP/show_server_header` property must be enabled in SMICM transaction (ICM@SAPNetweaver). See http://help.sap.com/saphelp_nw73ehp1/helpdata/en/48/49e4dd2e3131c3e10000000a42189d/frameset.htm.

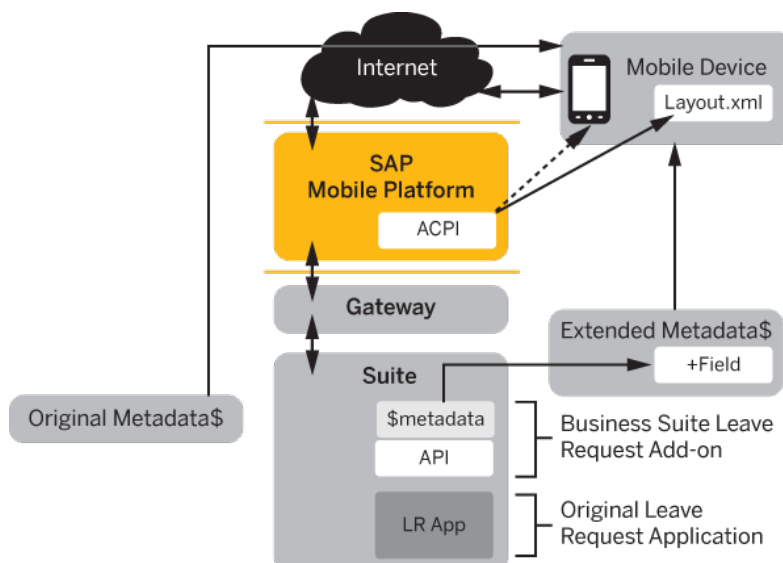
All requests in all onboarding scenarios go through the default sap-client (client 100). To change that, adjust your ICF configuration for the server (SICF transaction in Gateway System). See http://help.sap.com/saphelp_nw73ehp1/helpdata/en/48/cae5cc356c3254e10000000a42189b/frameset.htm.

Parent topic: [Developing with MAF Logon for iOS](#)

1.14 MAF Configuration Provisioning

SAP mobile applications are delivered with a default skin. You can change skinning parameters at runtime without recompiling the mobile application from the source. The skinning is based on metadata in an XML file. You must upload the styling content to the SAP Mobile Platform Server that manages the application. Mobile applications also ship with a default structural configuration. As well as skinning, you can also structurally extend applications based on a metadata description.

Configure applications using an Application Configuration Profile (ACP), which is a set of files that include layout, style, localized files, and images. Each mobile application has one ACP. The SAP mobile application workbench can generate the ACP package.



For more information on the contents of each of the files, see the MAF XSD documentation on the SAP Help Portal and the Mobile Application Workbench User Guide.

On-Premise SAP Mobile Platform Server

If you are using an on-premise version of SAP Mobile Platform Server, use the cockpit to upload and assign ACP bundles to your application. For more information, see the SAP Mobile Platform documentation: <http://help.sap.com/mobile-platform/>. Navigate to ► **Administer** ► **Applications** ► **Defining Applications** ► **Application Customization Resource Bundles** .

Cloud SAP Mobile Platform Server

You can provision ACP bundles using the cloud version of SAP Mobile Platform.

Parent topic: [Developing with MAF Logon for iOS](#)

1.15 Logon APIs for iOS

APIs for SAML and Mobile Place.

In this section:

- [HttpConversation API](#)
- [HttpConvAuthFlows API](#)
- [Mobile Place API](#)

Parent topic: [Developing with MAF Logon for iOS](#)

1.15.1 HttpConversation API

API that supports SAML-enabled registration.

In this section:

- [SAP Mobile Platform 3.0 OData SDK for iOS](#)
- [Deprecated API List](#)

Parent topic: [Logon APIs for iOS](#)

1.15.2.1 SAP Mobile Platform 3.0 OData SDK for iOS

Use the OData SDK API reference as the primary source for API usage and error code information.

Refer to the OData SDK API reference for each available package.


In this section:

- [ChallengeFilterProtocol protocol](#)
- [ChallengeHandler class](#)
- [ChangeSet class](#)
- [ClientCertObserverProtocol protocol](#)
- [HttpConversationManager class](#)
- [HttpConversationObserverProtocol protocol](#)
- [ManagerConfiguratorProtocol protocol](#)
- [PluginProviderProtocol protocol](#)
- [RequestFilterProtocol protocol](#)
- [ResponseFilterProtocol protocol](#)
- [SAPProvider class](#)
- [SupportabilityUploader class](#)

Parent topic: [HttpConversation API](#)

1.15.1.1.1 ChallengeFilterProtocol protocol

Defines the ChallengeFilter interface.

 Syntax

@protocol ChallengeFilterProtocol

Members

All members of ChallengeFilterProtocol, including inherited members.

Methods

Method	Description
<code>-(void)handleChallenge:(NSURLAuthenticationChallenge *) challenge conversationManager:(HttpConversationManager *) conversationManager completionBlock:(void (^)(BOOL useCredential, NSURLCredential *credential)) completionBlock</code>	Delegate method called when authentication challenge occurs.


Remarks

ChallengeFilter is an object, which will be called when the request execution requires some authentication to be successful.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.6.2 handleChallenge:conversationManager:completionBlock: method

Delegate method called when authentication challenge occurs.

 Syntax

```
- (void) handleChallenge : ( NSURLAuthenticationChallenge * ) challenge conversationManager : ( HttpConversationManager * )
conversationManager completionBlock : ( void^( BOOL useCredential, NSURLCredential *credential ) ) completionBlock
```

Parameters

challenge

NSURLAuthenticationChallenge

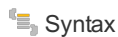
conversationManager

copy of HttpConversationManager instance, can be used for starting additional requests

completionBlock

call the block when the filter finished its job. Return YES, and a NSURLCredential object, if the challenge is handled, or return NO and nil, if the challenge is not handled. If return YES and NSURLCredential object is nil, it will be handled as no credential provided.

1.15.1.1.2 ChallengeHandler class



Syntax

```
@interface ChallengeHandler : <NSObject, NSURLSessionTaskDelegate>
```

Members

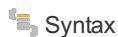
All members of ChallengeHandler, including inherited members.

Methods

Method	Description
<code>-(void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition disposition, NSURLCredential *credential))completionHandler</code>	

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.5.17 URLSession:task:didReceiveChallenge:completion Handler: method



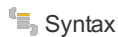
Syntax

```
-(void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition disposition, NSURLCredential *credential))completionHandler
```

Parent topic: [ChallengeHandler class](#)

1.15.1.1.3 ChangeSet class

ChangeSet definition.



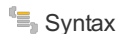
Syntax

```
@interface ChangeSet : <NSObject>
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.4 ClientCertObserverProtocol protocol

Defines ClientCertObserver interface.



Syntax

```
@protocol ClientCertObserverProtocol
```

Members

All members of ClientCertObserverProtocol, including inherited members.

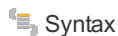
Methods

Method	Description
<code>-(void)observeClientCertificate:(SecIdentityRef)secdRef</code>	Called when an authentication challenge is to be handled using a client certificate.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.4.1 observeClientCertificate: method

Called when an authentication challenge is to be handled using a client certificate.



Syntax

```
-(void)observeClientCertificate:(SecIdentityRef)secdRef
```

Parameters

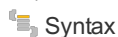
secdRef

the certificate used to try to handle the challenge. Important: it is not guaranteed, that the certificate solves the challenge.

Parent topic: [ClientCertObserverProtocol protocol](#)

1.15.1.1.5 HttpConversationManager class

Request execution manager class, supports request, response and challenge filtering.



Syntax

```
@interface HttpConversationManager : <NSObject, NSURLSessionDelegate, NSURLSessionTaskDelegate>
```

Members

All members of `HttpConversationManager`, including inherited members.

Methods

Method	Description
<code>-(void)addChallengeFilter: (id< ChallengeFilterProtocol >) challengeFilter</code>	Add <code>ChallengeFilterProtocol</code> implementation to the <code>HttpConversationManager</code> instance.
<code>-(void)addObserver: (id< HttpConversationObserverProtocol >) observer</code>	Add <code>HttpConversationObserverProtocol</code> implementations to the <code>HttpConversationManager</code> instance.
<code>-(void)addRequestFilter: (id< RequestFilterProtocol >) requestFilter</code>	Add <code>RequestFilterProtocol</code> implementation to the <code>HttpConversationManager</code> instance.
<code>-(void)addResponseFilter: (id< ResponseFilterProtocol >) responseFilter</code>	Add <code>ResponseFilterProtocol</code> implementation to the <code>HttpConversationManager</code> instance.
<code>-(NSArray *)allChallengeFilters</code>	Returns all the <code>ChallengeFilterProtocol</code> implementation added to the <code>HttpConversationManager</code> instance.
<code>-(NSArray *)allObservers</code>	Returns all the observers added to the <code>HttpConversationManager</code> instance.
<code>-(NSArray *)allRequestFilters</code>	Returns all the <code>RequestFilterProtocol</code> implementation added to the <code>HttpConversationManager</code> instance.
<code>-(NSArray *)allResponseFilters</code>	Returns all the <code>ResponseFilterProtocol</code> implementation added to the <code>HttpConversationManager</code> instance.
<code>-(id) copy</code>	
<code>-(long) currentTimelnMilis</code>	
<code>-(void)executeRequest: (NSMutableURLRequest *) urlRequest completionHandler: (void(^)(NSData *data, NSURLResponse *response, NSError *error)) completionHandler</code>	Execute <code>NSMutableURLRequest</code> instance using <code>HttpConversationManager</code> .
<code>-(void)executeRequest_Private: (NSMutableURLRequest *) urlRequest completionHandler: (void(^)(NSData *data, NSURLResponse *response, NSError *error)) completionHandlerOriginal</code>	
<code>-(void)generateTraceEntriesFromRequest: (NSMutableURLRequest *) request</code>	
<code>-(void)generateTraceEntriesFromResponse: (NSHTTPURLResponse *) response receivedData: (NSData *) receivedData method: (NSString *) method</code>	
<code>-(id) init</code>	
<code>-(void)setResponseData: (NSData *) data intoResponse: (NSURLResponse *) response error: (NSError *) error</code>	
<code>-(void)URLSession: (NSURLSession *) session task: (NSURLSessionTask *) task didReceiveChallenge: (NSURLAuthenticationChallenge *) challenge completionHandler: (void(^)(NSURLSessionAuthChallengeDisposition disposition, NSURLCredential *credential)) completionHandler</code>	

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.5.1 addChallengeFilter: method

Add `ChallengeFilterProtocol` implementation to the `HttpConversationManager` instance.

Syntax

```
-( void ) addChallengeFilter : ( id< ChallengeFilterProtocol > ) challengeFilter
```

Parameters

`challengeFilter`

`ChallengeFilterProtocol` implementation

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.2 addObserver: method

Add `HttpConversationObserverProtocol` implementations to the `HttpConversationManager` instance.

Syntax

```
-( void ) addObserver : ( id< HttpConversationObserverProtocol > ) observer
```

Remarks

For more information, check `HttpConversationObserverProtocol`

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.3 addRequestFilter: method

Add `RequestFilterProtocol` implementation to the `HttpConversationManager` instance.

Syntax

```
-( void ) addRequestFilter : ( id< RequestFilterProtocol > ) requestFilter
```

Parameters

requestFilter

RequestFilterProtocol implementation

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.4 addResponseFilter: method

Add ResponseFilterProtocol implementation to the HttpConversationManager instance.

Syntax

```
- (void) addResponseFilter : ( id< ResponseFilterProtocol > ) responseFilter
```

Parameters

responseFilter

ResponseFilterProtocol implementation

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.5 allChallengeFilters method

Returns all the ChallengeFilterProtocol implementation added to the HttpConversationManager instance.

Syntax

```
- ( NSArray * ) allChallengeFilters
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.6 allObservers method

Returns all the observers added to the HttpConversationManager instance.

Syntax

```
- ( NSArray * ) allObservers
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.7 allRequestFilters method

Returns all the RequestFilterProtocol implementation added to the HttpConversationManager instance.

Syntax

```
- ( NSArray * ) allRequestFilters
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.8 allResponseFilters method

Returns all the ResponseFilterProtocol implementation added to the HttpConversationManager instance.

Syntax

```
- ( NSArray * ) allResponseFilters
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.9 copy method

Syntax

```
- ( id ) copy
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.10 currentTimeInMilis method

Syntax

```
- ( long ) currentTimeInMilis
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.11 executeRequest:completionHandler: method

Execute NSMutableURLRequest instance using HttpConversationManager.

Syntax

```
- ( void ) executeRequest : ( NSMutableURLRequest * ) urlRequest completionHandler : ( void(^)( NSData *data, NSURLResponse *response, NSError *error) ) completionHandler
```

Parameters

urlRequest

NSMutableURLRequest instance

completionHandler

will be called when the request is finished. Contains the response, the received data, and in case of any error the root cause.

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.12 executeRequest_Private:completionHandler: method

Syntax

```
- (void) executeRequest_Private : ( NSMutableURLRequest * ) urlRequest completionHandler : ( void(^)(NSData *data, NSURLResponse *response, NSError *error) ) completionHandlerOriginal
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.13 generateTraceEntriesFromRequest: method

Syntax

```
- (void) generateTraceEntriesFromRequest : ( NSMutableURLRequest * ) request
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.14 generateTraceEntriesFromResponse:receivedData: method

Syntax

```
- (void) generateTraceEntriesFromResponse : ( NSHTTPURLResponse * ) response receivedData : ( NSData * ) receivedData  
method : ( NSString * ) method
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.15 init method

Syntax

```
- (id) init
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.16 setResponseData:intoResponse:error: method

Syntax

```
- (void) setResponseData : ( NSData * ) data intoResponse : ( NSURLResponse * ) response error : ( NSError * ) error
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.5.17 URLSession:task:didReceiveChallenge:completion Handler: method

Syntax

```
- (void) URLSession : ( NSURLSession * ) session task : ( NSURLSessionTask * ) task didReceiveChallenge : ( NSURLAuthenticationChallenge * ) challenge completionHandler : ( void(^)(NSURLSessionAuthChallengeDisposition disposition, NSURLCredential *credential) ) completionHandler
```

Parent topic: [HttpConversationManager class](#)

1.15.1.1.6 HttpConversationObserverProtocol protocol

Defines HttpConversationObserver interface.

Syntax

```
@protocol HttpConversationObserverProtocol
```

Remarks

Observers will be notified when a specific event occurs.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.7 ManagerConfiguratorProtocol protocol

Defines ManagerConfigurator interface.

Syntax

```
@protocol ManagerConfiguratorProtocol
```

Members

All members of ManagerConfiguratorProtocol, including inherited members.

Methods

Method	Description
<code>-(void) configureManager: (HttpConversationManager *) manager</code>	Call this method to configure.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.8.9 configureManager: method

Call this method to configure.

Syntax

```
-(void) configureManager: (HttpConversationManager *) manager
```

Parameters

manager

HttpConversationManager instance to be configured

Parent topic: [ManagerConfiguratorProtocol protocol](#)

Related Information

[HttpConversationManager class](#)

[RequestFilterProtocol protocol](#)

[ResponseFilterProtocol protocol](#)

[ChallengeFilterProtocol protocol](#)

1.15.1.1.8 PluginProviderProtocol protocol

Syntax

```
@protocol PluginProviderProtocol
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.9 RequestFilterProtocol protocol

Defines the RequestFilter interface.

Syntax

```
@protocol RequestFilterProtocol
```

Members

All members of RequestFilterProtocol, including inherited members.

Methods

Method	Description
<code>-(void) prepareRequest: (NSMutableURLRequest *) mutableRequest conversationManager: (HttpConversationManager *) conversationManager completionBlock: (void (^)()) completionBlock</code>	Delegate method called before executing the request.

Remarks

RequestFilter is an object, which will be called before the request is triggered, and it can modify the `mutableRequest` instance before execution, e.g. adding request header, set POST body, etc.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.11.8 prepareRequest:conversationManager:completionBlock: method

Delegate method called before executing the request.

Syntax

```
-(void) prepareRequest: (NSMutableURLRequest *) mutableRequest  
conversationManager: (HttpConversationManager *) conversationManager  
completionBlock: (void (^)( )) completionBlock
```

Parameters

mutableRequest

the request instance which will be executed

conversationManager

copy of HttpConversationManager instance, can be used for starting additional requests


completionBlock

call the block when the filter finished the modification of `mutableRequest` object

Parent topic: [RequestFilterProtocol protocol](#)

1.15.1.1.10 ResponseFilterProtocol protocol

Defines the ResponseFilter interface.

 Syntax

@protocol ResponseFilterProtocol

Members

All members of ResponseFilterProtocol, including inherited members.

Methods

Method	Description
<code>-(void) processResponse: (NSURLResponse *) urlResponse responseData: (NSData *) responseData conversationManager: (HttpConversationManager *) conversationManager completionBlock: (void (^)(BOOL shouldRestartRequest)) completionBlock</code>	Delegate method called after the request is executed.


Remarks

ResponseFilter is an object, which will be called when the request is finished, and it can modify the `urlResponse` and the `responseData` instances before the request executor gets the result.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.14.6 processResponse:responseData:conversationManager:completionBlock: method

Delegate method called after the request is executed.

 Syntax

```
-(void) processResponse: (NSURLResponse *) urlResponse responseData: (NSData *) responseData conversationManager: (HttpConversationManager *) conversationManager completionBlock: (void (^)(BOOL shouldRestartRequest)) completionBlock
```

Parameters

`urlResponse`

contains the overview result of the finished request

`responseData`

contains the data returned for the finished request

`conversationManager`


copy of `HttpConversationManager` instance, can be used for starting additional requests

`completionBlock`

call the block when the filter finished its job. Return YES, if the original request needs to be restarted. This can happen if the ResponseFilter started a new request for some additional authentication, which is needed for the original request, like SAML.

Parent topic: [ResponseFilterProtocol protocol](#)

1.15.1.1.11 SAPProvider class


 Syntax

@interface SAPProvider : <NSObject>

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.12 SupportabilityUploader class

SupportabilityUploader implementation, used for E2ETrace and ClientLog uploading.

 Syntax

@interface SupportabilityUploader : <NSObject, SupportabilityUploading>

Members

All members of SupportabilityUploader, including inherited members.

Methods

Method	Description
<code>-(id) initWithHttpConversationManager: (HttpConversationManager *) conversationManager urlRequest: (NSMutableURLRequest *) mutableURLRequest</code>	Initialize SupportabilityUploader class.
<code>-(void) sendWithContentType: (NSString *) contentType headers: (NSDictionary *) headers payloadType: (PAYLOAD_TYPE) payloadType payload: (NSString *) payload completion: (void (^)(NSError *)) completion</code>	
<code>-(NSString *) statusCodeToErrorMessage: (NSInteger) statusCode</code>	

1.15.1.1.12.1 initWithHttpConversationManager:urlRequest: method

Initialize SupportabilityUploader class.

Syntax

```
-( id ) initWithHttpConversationManager : ( HttpConversationManager * ) conversationManager urlRequest : ( NSMutableURLRequest * ) mutableURLRequest
```

Parameters

conversationManager

mutableURLRequest

request instance which is prepared for log uploading

Parent topic: [SupportabilityUploader class](#)

1.15.1.1.12.2 sendWithContentType:headers:payloadType:payload:completion: method

Syntax

```
-( void ) sendWithContentType : ( NSString * ) contentType headers : ( NSDictionary * ) headers payloadType : ( PAYLOAD_TYPE ) payloadType payload : ( NSString * ) payload completion : ( void(^)(NSError *) ) completion
```

Parent topic: [SupportabilityUploader class](#)

1.15.1.1.12.3 statusCodeToErrorMessage: method

Syntax

```
-( NSString * ) statusCodeToErrorMessage : ( NSInteger ) statusCode
```

Parent topic: [SupportabilityUploader class](#)

1.15.1.1.13 batchElements_C method

Syntax

```
-( NSMutableArray * ) batchElements_C : ( id ) self : ( SEL ) _cmd
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.14 dataTaskWithMutableRequest_Block method

Syntax

```
-( NSURLSessionDataTask * ) dataTaskWithMutableRequest_Block : ( id ) self : ( SEL ) _cmd : ( NSMutableURLRequest * ) mutableRequest : ( void(^)(NSData *o_data, NSURLResponse *o_response, NSError *o_error) ) originalCompletionHandler
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.15 sessionConfiguration_C method

Syntax

```
-( NSString * ) sessionConfiguration_C : ( id ) self : ( SEL ) _cmd
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.16 setBatchElements_C method

Syntax

```
-( void ) setBatchElements_C : ( id ) self : ( SEL ) _cmd : ( NSMutableArray * ) batchElements
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.17 setSessionConfiguration_C method

Syntax

```
-( void ) setSessionConfiguration_C : ( id ) self : ( SEL ) _cmd : ( NSURLSessionConfiguration * ) sessionConfig
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.18 setupRequestWithXCSRFToken method

Syntax

```
-( void ) setupRequestWithXCSRFToken : ( id ) self : ( NSMutableURLRequest * ) mutableRequest
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.19 setXCSRFToken_Private method

Syntax

```
-(void) setXCSRFToken_Private : (id) self : (SEL) _cmd : (NSString *) tokenValue
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.20 start_C method

Syntax

```
-(void) start_C : (id) self : (SEL) _cmd
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.21 XCSRFToken_Private method

Syntax

```
-(NSString *) XCSRFToken_Private : (id) self : (SEL) _cmd
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.1.22 XCSRFTokenFromResponse method

Syntax

```
-(void) XCSRFTokenFromResponse : (id) self : (NSHTTPURLResponse *) response
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.1.2 Deprecated API List

Starting with SAP Mobile Platform 3.0, the following protocols, classes, and methods are officially deprecated. SAP recommends you update the APIs to the newly refactored APIs listed in the section [SAP Mobile Platform SDK > Developer > Migration > Migrating Native OData Applications](#). The deprecated SDM APIs are supported for backward compatibility in SAP Mobile Platform 3.0.

Parent topic: [HttpConversation API](#)

1.15.2 HttpConvAuthFlows API

API that supports the authorization flows of SAML-enabled registration.

In this section:

- [SAP Mobile Platform 3.0 OData SDK for iOS](#)
- [Deprecated API List](#)

Parent topic: [Logon APIs for iOS](#)

1.15.2.1 SAP Mobile Platform 3.0 OData SDK for iOS

Use the OData SDK API reference as the primary source for API usage and error code information.

Refer to the OData SDK API reference for each available package.

In this section:

- [AuthenticationUIViewController class](#)
- [AuthenticationUIViewControllerDelegate protocol](#)
- [BaseChallengeFilter class](#)
- [BaseRequestFilter class](#)
- [BaseResponseFilter class](#)
- [ClientCertChallengeFilter class](#)
- [ClientCertProviderProtocol protocol](#)
- [CommonAuthenticationConfigurator class](#)
- [HttpConvAuthFlowsLocalizationManager class](#)
- [OAuth2ConfigProviderProtocol protocol](#)
- [OAuth2RequestFilter class](#)
- [OAuth2TokenObserverProtocol protocol](#)
- [SAML2ConfigProviderProtocol protocol](#)
- [SAML2ResponseFilter class](#)
- [UsernamePasswordChallengeFilter class](#)
- [UsernamePasswordProvider class](#)
- [UsernamePasswordProviderProtocol protocol](#)

Parent topic: [HttpConvAuthFlows API](#)

1.15.2.1.1 AuthenticationUIViewController class

UIViewController which contains a UIWebView for authentication methods like SAML and OAuth.

Syntax

```
@interface AuthenticationUIViewController : UIViewController < UIWebViewDelegate >
```

Members

All members of AuthenticationUIViewController, including inherited members.

Methods

Method	Description
- (void) addContentViews	
- (void) authenticationCanceled	
- (void) closeActivityIndicator	
- (id) initWithParentViewController: (UIViewController *) parentViewController	Initializes the AuthenticationUIViewController.
- (void) privateCloseActivityIndicator	
- (void) privateShowActivityIndicatorWithText: (NSString *) anIndicatorText	
- (void) showActivityIndicatorWithText: (NSString *) anIndicatorText	
- (void) viewDidLoad	
- (BOOL) webView: (UIWebView *) webView shouldStartLoadWithRequest: (NSURLRequest *) request navigationType: (UIWebViewNavigationType) navigationType	
- (void) webViewDidFinishLoad: (UIWebView *) webView	

Remarks

Displays an activity indicator until the content is loaded.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.1.1 addContentViews method

Syntax

```
- (void) addContentViews
```

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.14.1 authenticationCanceled method

Syntax

```
- (void) authenticationCanceled
```

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.1.3 closeActivityIndicator method

Syntax

```
- (void) closeActivityIndicator
```

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.1.4 initWithParentViewController: method

Initializes the AuthenticationUIViewController.

Syntax

```
- (id) initWithParentViewController: (UIViewController *) parentViewController
```

Parameters

parentViewController

: AuthenticationUIViewController will be presented on top of this view controller.

Returns

AuthenticationUIViewController instance

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.1.5 privateCloseActivityIndicator method

Syntax

```
- (void) privateCloseActivityIndicator
```

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.1.6 privateShowActivityIndicatorWithText: method

Syntax

```
- (void) privateShowActivityIndicatorWithText : (NSString *) anIndicatorText
```

Parent topic: [AuthenticationUIViewController class](#)


1.15.2.1.1.7 showActivityIndicatorWithText: method

 Syntax

```
- (void) showActivityIndicatorWithText : (NSString *) anIndicatorText
```

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.1.8 viewDidLoad method

 Syntax

```
- (void) viewDidLoad
```

Parent topic: [AuthenticationUIViewController class](#)


1.15.2.1.11.12 webView:shouldStartLoadWithRequest:navigationType: method

 Syntax

```
- (BOOL) webView : (UIWebView *) webView shouldStartLoadWithRequest : (NSURLRequest *) request navigationType : (UIWebViewNavigationType) navigationType
```

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.11.13 webViewDidFinishLoad: method


 Syntax

```
- (void) webViewDidFinishLoad : (UIWebView *) webView
```

Parent topic: [AuthenticationUIViewController class](#)

1.15.2.1.2 AuthenticationUIViewControllerDelegate protocol

Defines delegate interface for AuthenticationUIViewController.

 Syntax

```
@protocol AuthenticationUIViewControllerDelegate
```

Members

All members of AuthenticationUIViewControllerDelegate, including inherited members.


Methods

Method	Description
-(void) authenticationCanceled	Called when Cancel button pressed.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.14.1 authenticationCanceled method

Called when Cancel button pressed.

 Syntax

```
- (void) authenticationCanceled
```

Parent topic: [AuthenticationUIViewControllerDelegate protocol](#)

1.15.2.1.3 BaseChallengeFilter class

Base ChallengeFilter implementation, defines properties which are used in ChallengeFilters.

 Syntax

```
@interface BaseChallengeFilter : <NSObject>
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.4 BaseRequestFilter class

Base RequestFilter implementation, defines properties which are used in RequestFilters.

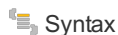
 Syntax

```
@interface BaseRequestFilter : <NSObject>
```

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.5 BaseResponseFilter class

Base ResponseFilter implementation, defines properties which are used in ResponseFilters.

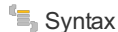


`@interface BaseResponseFilter : <NSObject>`

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.6 ClientCertChallengeFilter class

ChallengeFilter which handles client certificate authentication challenge.



`@interface ClientCertChallengeFilter : BaseChallengeFilter <ChallengeFilterProtocol>`

Members

All members of ClientCertChallengeFilter, including inherited members.

Methods

Method	Description
<code>-(void) callNextProviderOfEnumerator: (NSEnumerator *) enumerator completionBlock: (void (^)(NSURLCredential *)) completionBlock</code>	
<code>-(void) handleChallenge: (NSURLAuthenticationChallenge *) challenge conversationManager: (id) conversationManager completionBlock: (void (^)(BOOL, NSURLCredential *)) completionBlock</code>	

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.6.1 callNextProviderOfEnumerator:completionBlock: method



`-(void) callNextProviderOfEnumerator: (NSEnumerator *) enumerator completionBlock: (void (^)(NSURLCredential *)) completionBlock`

Parent topic: [ClientCertChallengeFilter class](#)

1.15.2.1.6.2 handleChallenge:conversationManager:completionBlock: method

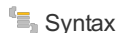


`-(void) handleChallenge: (NSURLAuthenticationChallenge *) challenge conversationManager: (id) conversationManager completionBlock: (void (^)(BOOL, NSURLCredential *)) completionBlock`

Parent topic: [ClientCertChallengeFilter class](#)

1.15.2.1.7 ClientCertProviderProtocol protocol

Defines ClientCertProvider interface.



`@protocol ClientCertProviderProtocol`

Members

All members of ClientCertProviderProtocol, including inherited members.

Methods

Method	Description
<code>-(void) provideClientCertForAuthChallenge: (NSURLAuthenticationChallenge *) authChallenge completionBlock: (void (^)(NSURLCredential *, NSError *)) completionBlock</code>	Called when client certificate authentication challenge occurs during request execution.

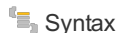
Remarks

Implementations will be called when request execution requires client certificate authentication.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.7.1 provideClientCertForAuthChallenge:completionBlock: method

Called when client certificate authentication challenge occurs during request execution.



`-(void) provideClientCertForAuthChallenge: (NSURLAuthenticationChallenge *) authChallenge completionBlock: (void (^)(NSURLCredential *, NSError *)) completionBlock`

Parameters

authChallenge

: NSURLAuthenticationChallenge

completionBlock

: call the completionBlock when provider finishes its tasks. Return NSURLCredential instance, if challenge is handled, or return an NSError instance, if not.

Parent topic: [ClientCertProviderProtocol protocol](#)

1.15.2.1.8 CommonAuthenticationConfigurator class

Configurator which contains the common authentication methods: basic, client certificate, OAuth and SAML.

Syntax

```
@interface CommonAuthenticationConfigurator : <NSObject, ManagerConfiguratorProtocol>
```

Members

All members of CommonAuthenticationConfigurator, including inherited members.

Methods

Method	Description
<code>-(void)addClientCertProvider:(id< ClientCertProviderProtocol >) clientCertProvider</code>	Adds a ClientCertProviderProtocol implementation.
<code>-(void)addOAuth2ConfigProvider:(id< OAuth2ConfigProviderProtocol >) oauth2ConfigProvider</code>	Adds a OAuthConfigProviderProtocol implementation.
<code>-(void)addSAML2ConfigProvider:(id< SAML2ConfigProviderProtocol >) samlConfigProvider</code>	Adds a SAMLConfigProviderProtocol implementation.
<code>-(void)addUsernamePasswordProvider:(id< UsernamePasswordProviderProtocol >) usernamePasswordProvider</code>	Adds a UsernamePasswordProtocol implementation.
<code>-(NSArray *)allClientCertProvider</code>	Returns all the added ClientCertProviderProtocol implementations.
<code>-(NSArray *)allOAuth2ConfigProvider</code>	Returns all the added OAuthConfigProviderProtocol implementations.
<code>-(NSArray *)allSAML2ConfigProvider</code>	Returns all the added SAMLConfigProviderProtocol implementations.
<code>-(NSArray *)allUsernamePasswordProvider</code>	Returns all the added UsernamePasswordProtocol implementations.
<code>-(void)configureManager:(HttpConversationManager *) manager</code>	

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.8.1 addClientCertProvider: method

Adds a ClientCertProviderProtocol implementation.

Syntax

```
-(void) addClientCertProvider : ( id< ClientCertProviderProtocol > ) clientCertProvider
```

Parameters

clientCertProvider

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.2 addOAuth2ConfigProvider: method

Adds a OAuthConfigProviderProtocol implementation.

Syntax

```
-(void) addOAuth2ConfigProvider : ( id< OAuth2ConfigProviderProtocol > ) oauth2ConfigProvider
```

Parameters

oauthConfigProvider

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.3 addSAML2ConfigProvider: method

Adds a SAMLConfigProviderProtocol implementation.

Syntax

```
-(void) addSAML2ConfigProvider : ( id< SAML2ConfigProviderProtocol > ) samlConfigProvider
```

Parameters

samlConfigProvider

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.4 addUsernamePasswordProvider: method

Adds a UsernamePasswordProtocol implementation.

Syntax

```
- (void) addUsernamePasswordProvider : (id<UsernamePasswordProviderProtocol>) usernamePasswordProvider
```

Parameters

usernamePasswordProvider

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.5 allClientCertProvider method

Returns all the added ClientCertProviderProtocol implementations.

Syntax

```
- (NSArray *) allClientCertProvider
```

Returns

ClientCertProviderProtocol implementations

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.6 allOAuth2ConfigProvider method

Returns all the added OAuthConfigProviderProtocol implementations.

Syntax

```
- (NSArray *) allOAuth2ConfigProvider
```

Returns

OAuthConfigProviderProtocol implementations

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.7 allSAML2ConfigProvider method

Returns all the added SAMLConfigProviderProtocol implementations.

Syntax

```
- (NSArray *) allSAML2ConfigProvider
```

Returns

SAMLConfigProviderProtocol implementations

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.8 allUsernamePasswordProvider method

Returns all the added UsernamePasswordProtocol implementations.

Syntax

```
- (NSArray *) allUsernamePasswordProvider
```

Returns

UsernamePasswordProtocol implementations

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.8.9 configureManager: method

Syntax

```
- (void) configureManager : (HttpConversationManager *) manager
```

Parent topic: [CommonAuthenticationConfigurator class](#)

1.15.2.1.9 HttpConvAuthFlowsLocalizationManager class

Localization Manager.

Syntax

```
@interface HttpConvAuthFlowsLocalizationManager : <NSObject>
```

Members

All members of `HttpConvAuthFlowsLocalizationManager`, including inherited members.

Methods

Method	Description
+ (NSBundle *) <code>localizationBundle</code>	
+ (NSString *) <code>localizedStringForKey:(NSString *)aKey</code>	Returns the localized text for given key.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.9.1 `localizationBundle` method

Syntax

```
+ (NSBundle *) localizationBundle
```

Parent topic: [HttpConvAuthFlowsLocalizationManager class](#)

1.15.2.1.9.2 `localizedStringForKey:` method

Returns the localized text for given key.

Syntax

```
+ (NSString *) localizedStringForKey : (NSString *) aKey
```

Parameters

`aKey`

key for which the localized text will be returned

Returns

localized text

Parent topic: [HttpConvAuthFlowsLocalizationManager class](#)

1.15.2.1.10 `OAuth2ConfigProviderProtocol` protocol

Defines `OAuth2ConfigProvider` interface.

Syntax

```
@protocol OAuth2ConfigProviderProtocol
```

Members

All members of `OAuth2ConfigProviderProtocol`, including inherited members.

Methods

Method	Description
- (void) <code>isAcceptedResponse:(NSURLResponse *) response forProvidedToken:(NSString *) token completionBlock:(void (^)(BOOL cancelRequest, BOOL restartOAuth2Flow)) completionBlock</code>	Called when a request with OAuth2 authentication finished.
- (void) <code>provideOAuth2ConfigurationForURL:(NSURL *) url completionBlock:(void (^)(NSURL * authorizationEndpoint, NSURL * tokenEndpoint, NSString * scope, NSString * clientId, NSString * clientSecret)) completionBlock</code>	Called when OAuth authentication is needed for request execution.
- (void) <code>provideOAuth2TokenForURL:(NSURL *) url completionBlock:(void (^)(NSString * token)) completionBlock</code>	Called when OAuth authentication is needed for request execution.

Remarks

Implementations will be called when request execution requires SAML authentication.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.10.1 `isAcceptedResponse:forProvidedToken:completionBlock:` method

Called when a request with OAuth2 authentication finished.

Syntax

```
- (void) isAcceptedResponse : (NSURLResponse *) response forProvidedToken : (NSString *) token completionBlock : (void (^)(BOOL cancelRequest, BOOL restartOAuth2Flow)) completionBlock
```

Parameters

`response`

response received from the server

token
token sent with the request
completionBlock
call the completionBlock when provider finishes its tasks. If cancelRequest is YES, the request will be canceled, and the response will be returned. If restartOAuth2Flow is Yes, the request will be restarted and OAuth2 authentication will take place again. Important: if cancelRequest is YES, the restartOAuth2Flow parameter is omitted.
Parent topic: [OAuth2ConfigProviderProtocol protocol](#)

1.15.2.1.10.2 provideOAuth2ConfigurationForURL:completionBlock: method

Called when OAuth authentication is needed for request execution.

Syntax

```
- (void) provideOAuth2ConfigurationForURL : (NSURL *) url completionBlock : (void (^)(NSURL *authorizationEndpoint, NSURL *tokenEndpoint, NSString *scope, NSString *clientId, NSString *clientSecret)) completionBlock
```

Parameters

url
: url which secured with OAuth
completionBlock
: call the completionBlock when provider finishes its tasks. All the parameters except `clientSecret` are mandatory.
Parent topic: [OAuth2ConfigProviderProtocol protocol](#)

1.15.2.1.10.3 provideOAuth2TokenForURL:completionBlock: method

Called when OAuth authentication is needed for request execution.

Syntax

```
- (void) provideOAuth2TokenForURL : (NSURL *) url completionBlock : (void (^)(NSString *token)) completionBlock
```

Parameters

url
: url which secured with OAuth
completionBlock
: call the completionBlock when provider finishes its tasks. Return OAuth token if authentication already take place.
Parent topic: [OAuth2ConfigProviderProtocol protocol](#)

1.15.2.1.11 OAuth2RequestFilter class

RequestFilter which handles OAuth authentication flow.

Syntax

```
@interface OAuth2RequestFilter : BaseRequestFilter <RequestFilterProtocol, UIWebViewDelegate, AuthenticationUIViewControllerDelegate>
```

Members

All members of OAuth2RequestFilter, including inherited members.

Methods

Method	Description
- (void) addTokenToAuthorizationHeader	
- (void) authenticationCanceled	Called when Cancel button pressed.
- (void) callCompletionHandler	
- (void) callNextConfigProviderOfEnumerator: (NSEnumerator *) enumerator completionBlock: (void (^)(BOOL)) completionBlock	
- (void) callNextTokenProviderOfEnumerator: (NSEnumerator *) enumerator completionBlock: (void (^)(NSString *)) completionBlock	
- (void) closeWebView	
- (void) displayLoginWebView: (NSURL *) url	
- (void) prepareRequest: (NSMutableURLRequest *) mutableRequest conversationManager: (HttpConversationManager *) conversationManager completionBlock: (void (^)(X)) completionHandler	
- (void) retrieveTokenForAuthorizationCode	
- (void) retrieveTokenFromResponseData: (NSData *) responseData	
- (void) startOAuthFlow	
- (BOOL) webView: (UIWebView *) webView shouldStartLoadWithRequest:	

(NSURLRequest *) request navigationType: (UIWebViewNavigationType) navigationType	
-(void) webViewDidFinishLoad: (UIWebView *) webView	

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.11.1 addTokenToAuthorizationHeader method

Syntax

```
-(void) addTokenToAuthorizationHeader
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.14.1 authenticationCanceled method

Called when Cancel button pressed.

Syntax

```
-(void) authenticationCanceled
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.11.3 callCompletionHandler method

Syntax

```
-(void) callCompletionHandler
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.14.2 callNextConfigProviderOfEnumerator:completionBlock: method

Syntax

```
-(void) callNextConfigProviderOfEnumerator : (NSEnumerator *) enumerator completionBlock : (void(^)(BOOL)) completionBlock
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.11.5 callNextTokenProviderOfEnumerator:completionBlock: method

Syntax

```
-(void) callNextTokenProviderOfEnumerator : (NSEnumerator *) enumerator completionBlock : (void(^)(NSString *)) completionBlock
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.14.4 closeWebView method

Syntax

```
-(void) closeWebView
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.11.7 displayLoginWebView: method

Syntax

```
-(void) displayLoginWebView : (NSURL *) url
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.11.8 prepareRequest:conversationManager:completionBlock: method

Syntax

```
-(void) prepareRequest : (NSMutableURLRequest *) mutableRequest conversationManager : (HttpConversationManager *) conversationManager completionBlock : (void(^)( )) completionHandler
```

Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.11.9 retrieveTokenForAuthorizationCode method

Syntax

```
-(void) retrieveTokenForAuthorizationCode
```

Parent topic: [OAuth2RequestFilter class](#)


1.15.2.1.11.10 retrieveTokenFromResponseData: method

 Syntax
- (void) retrieveTokenFromResponseData : (NSData *) responseData
Parent topic: [OAuth2RequestFilter class](#)


1.15.2.1.11.11 startOAuthFlow method

 Syntax
- (void) startOAuthFlow
Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.11.12 webView:shouldStartLoadWithRequest:navigationType: method

 Syntax
- (BOOL) webView : (UIView *) webView shouldStartLoadWithRequest : (NSURLRequest *) request navigationType : (UIViewNavigationType) navigationType
Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.11.13 webViewDidFinishLoad: method

 Syntax
- (void) webViewDidFinishLoad : (UIView *) webView
Parent topic: [OAuth2RequestFilter class](#)

1.15.2.1.12 OAuth2TokenObserverProtocol protocol

Defines OAuth2TokenObserver interface.

 Syntax
`@protocol OAuth2TokenObserverProtocol`

Members

All members of OAuth2TokenObserverProtocol, including inherited members.

Methods

Method	Description
- (void) observeOAuth2Token: (NSString *) token	Called when OAuth2 token is ready for usage.


Remarks

Implementations will be called when OAuth authentication flow is completed, and token is present.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.12.1 observeOAuth2Token: method

Called when OAuth2 token is ready for usage.

 Syntax
- (void) observeOAuth2Token : (NSString *) token

Parameters

token

OAuth2 token

Parent topic: [OAuth2TokenObserverProtocol protocol](#)

1.15.2.1.13 SAML2ConfigProviderProtocol protocol

Defines SAML2ConfigProvider interface.

 Syntax
`@protocol SAML2ConfigProviderProtocol`

Members

All members of SAML2ConfigProviderProtocol, including inherited members.

Methods

Method	Description
- (void) provideSAML2ConfigurationForURL: (NSURL *) url completionBlock:	Called when SAML2 authentication is needed for request execution.

```
(void(^)(NSString *responseHeader, NSString *finishEndPoint, NSString *finishParameters)) completionBlock
```

Remarks

Implementations will be called when request execution requires SAML authentication.

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.13.1 provideSAML2ConfigurationForURL:completionBlock: method

Called when SAML2 authentication is needed for request execution.

Syntax

```
-(void) provideSAML2ConfigurationForURL: (NSURL *) url completionBlock: (void(^)(NSString *responseHeader, NSString *finishEndPoint, NSString *finishParameters)) completionBlock
```

Parameters

url

: url which secured with SAML

completionBlock

: call the completionBlock when provider finishes its tasks. All the parameters are mandatory.

Parent topic: [SAML2ConfigProviderProtocol protocol](#)

1.15.2.1.14 SAML2ResponseFilter class

ResponseFilter which handles SAML2 authentication flow.

Syntax

```
@interface SAML2ResponseFilter : BaseResponseFilter <ResponseFilterProtocol, UIWebViewDelegate, AuthenticationUIViewControllerDelegate>
```

Members

All members of SAML2ResponseFilter, including inherited members.

Methods

Method	Description
-(void) authenticationCanceled	Called when Cancel button pressed.
-(void) callNextConfigProviderOfEnumerator: (NSEnumerator *) enumerator completionBlock: (void(^)(BOOL)) completionBlock	
-(void) checkIfSAMLNeeded	
-(void) closeWebView	
-(void) displayLoginWebView	
-(void) processResponse: (NSURLResponse *) urlResponse responseData: (NSData *) responseData conversationManager: (HttpConversationManager *) conversationManager completionBlock: (void(^)(BOOL)) completionBlock	
-(void) startSAMLAuth	
-(BOOL) webView: (UIWebView *) webView shouldStartLoadWithRequest: (NSURLRequest *) request navigationType: (UIWebViewNavigationType) navigationType	
-(void) webViewDidFinishLoad: (UIWebView *) webView	

Parent topic: [SAP Mobile Platform 3.0 OData SDK for iOS](#)

1.15.2.1.14.1 authenticationCanceled method

Called when Cancel button pressed.

Syntax

```
-(void) authenticationCanceled
```

Parent topic: [SAML2ResponseFilter class](#)

1.15.2.1.14.2 callNextConfigProviderOfEnumerator:completionBlock: method

Syntax

```
-(void) callNextConfigProviderOfEnumerator: (NSEnumerator *) enumerator completionBlock: (void(^)(BOOL)) completionBlock
```

Parent topic: [SAML2ResponseFilter class](#)

1.15.2.1.14.3 checkIfSAMLNeeded method



Syntax

```
- (void) checkIfSAMLNeeded
```

Parent topic: [SAML2ResponseFilter class](#)

1.15.2.1.14.4 closeWebView method



Syntax

```
- (void) closeWebView
```

Parent topic: [SAML2ResponseFilter class](#)

1.15.2.1.14.5 displayLoginWebView method



Syntax

```
- (void) displayLoginWebView
```

Parent topic: [SAML2ResponseFilter class](#)

1.15.2.1.14.6 processResponse:responseData:conversationManager:completionBlock: method



Syntax

```
- (void) processResponse : (NSURLResponse *) urlResponse responseData : (NSData *) responseData conversationManager : (HttpConversationManager *) conversationManager completionBlock : (void (^)(BOOL)) completionBlock
```

Parent topic: [SAML2ResponseFilter class](#)