AIL IOS integration document

AppInfra IOS Integration

Doc id - AIL000008

Author	Team Huma
Approved by	
Email ID	DL_Huma <DL_Huma@philips.com>

1. Table of Contents

1. Introduction:

This document provides an overview of integration procedure for AppInfra library in iOS mobile applications.

AppInfra is an infrastructure library for propositions and platform which contains common utilities such as logging , tagging, appconfig, service discovery, A/B testing, app identity . This library ensures that the usage and behavior of these functionalities standardized across multiple components or propositions

2. Integration:

2.1. Cocoa pod Integration:

The easiest and preferred way to use these components is using Cocoa pods (version 0.36.0 or newer). In your podfile, you have to add a source line for the Philips internal pod spec repository. So a minimal Podfile would look like this:

Source: http://tfsemea1.ta.philips.com:8080/tfs/TPC_Region24/CDP2/_git/ail-ios-appinfra

pod 'AppInfra'

2.2. Library Integration:

Git source path: http://tfsemea1.ta.philips.com:8080/tfs/TPC_Region24/CDP2/_git/ail-ios-appinfra

Check out the code from above path where in you can find DemoAppInfra app which depends on AppInfra.

AppInfra needs other libraries to build which are as below

pod 'NHNetworkTime', '1.7'

pod 'AdobeMobileSDK', '4.13.8'

pod 'CocoaLumberjack', '2.3.0'

pod 'AILAFNetworking', '3.1.0'

3. Initialization:

a) It is must to turn on Key chain sharing capabilities in your project settings to use AppInfra

Steps: Target ->Capabilities->KeyChain sharing ->On

2.

b) Make sure to remove AFNetworking pod dependency if you have any, as we have dependency on forked version of AFNetworking i.e AILAFNetworking in our appinfra

c) Import <AppInfra/AppInfra.h> in AppDelegate.h file

Create object for AIAppInfra in didFinishLaunchingWithOptions method

// create app infra instance and set it to the app infra singleton

AIAppInfra * appInfra = [AIAppInfra buildAppInfraWithBlock:nil];

AppInfra boot up time is 40 – 50 milliseconds.

3.

4. Secure Storage:

Secure Storage is used to store user sensitive data locally with encrypted way using AES And parseData:forType API uses MD5 Algorithm to hash the data

Use cases

we can use this component to store, fetch, delete user data and to encrypt the data and return

-> Store some user sensitive data

ex: [objAppInfra.storageProvider storeValueForKey:@"myPasswordKey" value:@"abc" error:nil];

-> Fetch user sensitive data

ex: [objAppInfra.storageProvider fetchValueForKey: @"myPasswordKey" error:nil];

-> Delete user sensitive data

ex: [objAppInfra.storageProvider removeValueForKey:@"myPasswordKey"];

-> Encrypt user sensitive data

ex:NSError *error;

NSData   *encryptedData = [objAppInfra.storageProvider encryptData:<#input data#> error:&error];

-> Decrypt user sensitive data

ex:

NSError *error;

NSData *decryptedData = (NSData *)[objAppInfra.storageProvider decryptData:<#encryptedData#> error:&error];

-> Hashing user data using MD5 Algorithm

parseData:forType API uses MD5 Algorithm to hash the given data and returns the hashed data

Checking whether device is jailbroken

getDeviceCapability method can be used to check whether device is jailbroken or not

NSString * status = [self.appInfra.storageProvider getDeviceCapability];

It returns String “true” or “false”. If the method returns “true” then device is jailbroken and if it returns “false” then it is not jailbroken

5. Tagging:

AppInfra Tagging Changes

Tagging Status	Old Tagging Logic	New Tagging Logic (From 1805 release)
optunknown	If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in.	If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in.
optedin	Hits are sent immediately	Hits are sent immediately
optedout	Hits are discarded	Hits are discarded. Only App Lifecycle data is tracked and sent to Adobe Server.
optunknown -> optedout	All queued hits are discarded	All queued hits are sent to Adobe Server.

App tagging is used to track pages and button actions of the propositions or common components with page/action name and several other default values such as timestamps, device info, OS info etc. Common components should create separate instance of tagging with instance name (component TLA) and version

The following validations we are having in app tagging.

- Page names should be limited to 100 bytes in length.
- Page name and previous page name should not be equal
- Page name cannot be nil.
- Event name should be limited to 250 bytes in length.

AppInfra will log an error message if these conditions are not met. Tagging will send the request to Adobe SDK even if these conditions are not met except if page name is nil.

Disable or Filterout data sending to server

If you want to disable tagging completely

you can specify in ADBMobileConfig.json file "analytics" group add "privacyDefault" : "optedout"

or you can call the API - (void)setPrivacyConsent:(AIATPrivacyStatus)privacyStatus;

If you want to filter out specific field appear in the tagging packets

add the sensitive data to Appconfig file under appinfra group - "tagging.sensitiveData": ["language"] then **setPrivacyConsentForSensitiveData to YES**

The default value is optedin.

- For optedin, the hits are sent immediately.
- For optedout, the hits are discarded. Only App Lifecycle data is tracked.
- For optunknown, if your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (hits are sent) or opt-out (hits are discarded).

If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in.

This only sets the initial value. If this value is set or changed in code, the new value is used until it is changed again, or when the app is uninstalled and reinstalled.

For enabling Adobe debug logs we need to add "enableAdobeLogs" key in AppConfig.json and value should be set to "true".

Integration

1. Include ADBMobileConfigDev.JSON file into the app for Development

Keep this json file in the app bundle Make sure SSL is “true” for secure HTTPS requests.

Change rsids tag accordingly to dev or release. batchLimit is another tag where one can define the count of requests.

```
{
    "version" : "1.0",
    "acquisition": {
        "server": "c00.adobe.com",
        "appid": ""
    },

    "analytics" : {
        "referrerTimeout": 15,
        "rsids" : "philipsmobileappsdev",
        "server" : "philips.112.2o7.net",
        "charset" : "UTF-8",
        "ssl" : true,
        "offlineEnabled" : true,
        "lifecycleTimeout" : 30,
        "batchLimit" : 10,
        "privacyDefault" : "optunknown",
        "poi" : [
        ]
    },
    "target" : {
        "clientCode" : "amsdk",
        "timeout" : 15
    },
    "audienceManager" : {
        "server" : ""
    }
}
```

2. Include ADBMobileConfigProd.JSON file into the app for Production

Keep this json file in the app bundle Make sure SSL is “false” for secure HTTPS requests.

Change rsids tag accordingly to Production or release. batchLimit is another tag where one can define the count of requests.


```
{
    "version" : "1.0",
    "acquisition": {
        "server": "c00.adobe.com",
        "appid": ""
    },

    "analytics" : {
        "referrerTimeout": 5,
        "rsids" : "philipsmobileappsregistrationProduction",
        "server" : "philips.112.2o7.net",
        "charset" : "UTF-8",
        "ssl" : false,
        "offlineEnabled" : true,
        "lifecycleTimeout" : 30,
        "batchLimit" : 0,
        "privacyDefault" : "optunknown",
        "poi" : [
        ]
    },
    "target" : {
        "clientCode" : "amsdk",
        "timeout" : 5
    },
    "audienceManager" : {
        "server" : ""
    }
}
```


3. Following APIs should be called in app delegate method to initialize App Tagging

configureAnalyticsWithFilePath: To configure analytics by providing the config file (the particular API is deprecated in 1803 release).

setPrivacyConsent: setting the privacy status based on user consent

EX:

In AppDelegate.h

Create object for Tagging Protocol

@property (nonatomic,strong) id<AIAppTaggingProtocol> objAppTaggingForRegistration, objAppTaggingForDemo, objAppTaggingForAppLifeCycle;

In AppDelegate.m

// Set configuration and privacy consent for Tagging on the main Instance

[objAppInfra.tagging configureAnalyticsWithFilePath:[[NSBundle mainBundle] pathForResource:@"ADBMobileConfigDev" ofType:@"json"]]; (following api is depcreated in 1803 release)

USE : [ADBMobile overrideConfigPath:confingFilePath];

[objAppInfra.tagging setPrivacyConsent:AIATPrivacyStatusUnknown];

Dedicated App tagging instances should be created for every component

EX:

// create app tagging Instances for Registration component

self.objAppTaggingForRegistration = [objAppInfra.tagging createInstanceForComponent:@"usr" componentVersion:@"1.0.1"];

// proposition should not create instance of tagging they shpuld use the instance available in appinfra ie appinfra.tagging

and we can include the following methods in app delegate to track app state changes

- (void)applicationDidEnterBackground:(UIApplication *)application {

// Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.

// If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.

[objAppInfra.tagging trackActionWithInfo:@"Application Entered Background" params:nil];

}

- (void)applicationWillEnterForeground:(UIApplication *)application {

// Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.

[objAppInfra.tagging trackActionWithInfo:@"Application Entered foreground" params:nil];

}

- (void)applicationDidBecomeActive:(UIApplication *)application {

```
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.

    [[NSNotificationCenter defaultCenter] postNotificationName:@"appDidBecomeActive" object:nil];


}


- (void)applicationWillTerminate:(UIApplication *)application {

    // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.

    [objAppInfra.tagging trackActionWithInfo:@"Application Killed" params:nil];

}
```

3.Tracking Pages

we can track a page by calling '**trackPageWithInfo**' method in viewDidLoad/viewWillAppear methods of any class

use this method when we have multiple key/values to be tagged against a page

```
- (void)trackPageWithInfo:(NSString*)pageName params:(NSDictionary*)paramDict
```

eg:

```
[objAppInfra.tagging setPreviousPage:@"Registration-WelcomeScreen"];

    // Tag the page with app tagging

    [[AppDelegate sharedAppDelegate].objAppTaggingForDemo trackPageWithInfo:@"Demo app tagging page" params:nil];
```

use this method when we have single key/values to be tagged against a page

```
- (void)trackPageWithInfo:(NSString*)pageName paramKey:(NSString*)key andParamValue:(id)value;
```

eg:

```
[[AppDelegate sharedAppDelegate].objAppTaggingForDemo trackPageWithInfo: :@"Demo app tagging page" paramKey:@" Track Param Key" andParamValue:@"Track Param Value"];
```

4.Tracking Actions

we can track an action by calling '**trackActionWithInfo**' method in event handlers (IBAction in case of buttons)

use this method when we have multiple key/values to be tagged against an action

```
- (void)trackActionWithInfo:(NSString*)actionName params:(NSDictionary*)paramDict
```

eg:

```
[[AppDelegate sharedAppDelegate].objAppTaggingForDemo trackActionWithInfo:@" Demo app tagging page" params:dictParam];
```

use this method when we have single key/values to be tagged against an action

(void)trackActionWithInfo:(NSString*)actionName paramKey:(NSString*)key andParamValue:(id)value

eg:

[[AppDelegate sharedAppDelegate].objAppTaggingForDemo trackActionWithInfo: :@"Demo app tagging page" paramKey:@"Action Param Key" andParamValue:@"Action Param Value"];

5.Getting privacy status: By using getPrivacyConsent method we can get the privacy, which has been set

- (AIATPrivacyStatus)getPrivacyConsent;

Privacy Status enum values placed in AIAppTaggingProtocol

```
// Enum to select privacy status
typedef NS_ENUM(NSUInteger,AIATPrivacyStatus)
{
    AIATPrivacyStatusOptIn = 1,
    AIATPrivacyStatusOptOut = 2,
    AIATPrivacyStatusUnknown = 3
};
```

6. trackVideoStart : This API can be used to track when a video has been started

```
/* Track Video started with a videoName
 @param: videoName: name of the video file
 */
- (void)trackVideoStart:(NSString*)videoName
```

7. trackVideoEnd: This API can be used to track when a video has been stopped/ended

```
/* Track Video started with a videoName
 @param: videoName: name of the video file
 */
- (void) trackVideoEnd:(NSString*)videoName
```

8. trackSocialSharing: This API can be used to track a social share buttons are tapped

```
/* Track social sharing with social media like facebook, twitter, mail etc…
 @param: socialMedia: Type of socail media through user sharing message
 @param: sharedItem: item to be shared string
 */
- (void)trackSocialSharing:(AIATSocialMedia)socialMedia WithItem:(NSString*)sharedItem
```

Possible Values of AIATSocialMedia are

```
    AIATSocialMediaFacebook,
```

AIATSocialMediaTwitter,

AIATSocialMediaMail,

AIATSocialMediaAirDrop

9. trackTimedActionStart

This method is used to track the start of a timed action

(void) trackTimedActionStart:(nullable NSString *)action data:(nullable NSDictionary *)data;

@brief Tracks the start of a timed event

@param action a required NSString value that denotes the action name to track. (**Note**: **Same "action name" must be used in track end API as well**

)

@param data optional dictionary pointer containing context data to track with this timed action.

@note This method does not send a tracking hit

@attention If an action with the same name already exists it will be deleted and a new one will replace it.

10. trackTimedActionEnd

This method is used to track the end of a timed action

- (void) trackTimedActionEnd:(nullable NSString *)action

logic:(nullable BOOL (^)(NSTimeInterval inAppDuration, NSTimeInterval totalDuration, NSMutableDictionary* __nullable data))
block;

@brief Tracks the end of a timed event

@param action a required NSString pointer that denotes the action name to finish tracking. . (**Note**: **Same "action name" must be used which was used in 'trackTimedActionStart' API**

)

@param block optional block to perform logic and update parameters when this timed event ends, this block can cancel the sending of the hit by returning NO.

@note This method will send a tracking hit if the parameter logic is nil or returns YES.

11. setPrivacyConsentForSensitiveData

Using this method we can set the privacy consent, possible values could be Yes or No, and the value set will be stored in secure storage

When **PrivacyConsentForSensitiveData is** set to **Yes** App Tagging will not send the values correspoding to the keys mentioned in the AppConfig file as shown below to the adobe cloud server

And when the **PrivacyConsentForSensitiveData is set to No** App Tagging will send all the data collected to the adobe server irrespective of the sensitive keys mentioned in the app config file

and we should set the keys of the sensitive data to the Appconfig file as shown above for the key tagging.sensitiveData,

we can set an array to this key and array can have any of these following values

language,

bundleId,

timestamp,

UTCTimestamp,

appsId

12. getPrivacyConsentForSensitiveData

This API is used to read the PrivacyConsentForSensitiveData which was previously set, possible values could be YES or NO

13. trackLinkExternal**:** This API can be used to track external link opening

@param: url: external url link that needs to be tracked

-(void)trackLinkExternal:(nullable NSString*)url;

14. trackFileDownload**:** This API can be used to track file downloading

@param: filename: downloading file name

-(void)trackFileDownload:(nullable NSString*)filename;

15. -(NSString*)getTrackingIdentifier: This API Retrieves the analytics tracking identifier. This method can cause a blocking network call and should not be used from a UI thread.

16. kAilTaggingNotification: It's a optional notification added in order to broadcast the tagging parameters to the observer, When ever the track page/action APIs are invoked, It posts the same data to the observer of kAilTaggingNotification what ever is being posted to the adobe server , and It is one to many asynchronous notification, one are more observers can listen to the notification, the data received from the notificationcan be used to post to Crashlytics or Appteligent or to any alternate analytics server

6. Logging:

Logging is used to maintain the logs which user access of the propositions or common components with component details and several other default values such as UTC timestamps, Log type, Component ID, Event and Message.

Important Notes:

Propositions or Common Components needs to remove Cocoa Lumberjack config setup if they have.

Logging update:

Logging property file LoggingConfig.plist is removed and now all the configuration should be in appconfig.json file.

Proposition needs to add below mentioned keys and values under appinfra group. However if this key is not present in appconfig.json, appinfra logging will pick values from LoggingConfig.plist file making it backward compatible.

These two key value needs to be added to appconfig.json under appinfra group.


```
"logging.releaseConfig":{
        "fileName":"AppInfraLog",
        "numberOfFiles":5,
        "fileSizeInBytes":50000,
        "logLevel":"Off",
        "fileLogEnabled":false,
        "consoleLogEnabled":false,
        "componentLevelLogEnabled":false,
        "componentIds": [
                        "ail",
                        "dcc"
                        ]
    },
    "logging.debugConfig":{
        "fileName":"AppInfraLog",
        "numberOfFiles":5,
        "fileSizeInBytes":50000,
        "logLevel":"All",
        "fileLogEnabled":true,
        "consoleLogEnabled":true,
        "componentLevelLogEnabled":false,
        "componentIds": [
                        "DemoAppInfra",
                        "PhilipsRegistration","component1"
                        ]
    }
```


It is recommended to add these new fields in appconfig.json and delete LoggingConfig.plist file

Key Details

- logLevel : String

It is used to filter log messages

Possible Values:

Off – No logs

Error – Error logs only

Warn – Error and warning logs

Info – Error, warning and info logs

Debug - Error, warning, info and debug logs

Verbose - Error, warning, info, debug and verbose logs

All – All type of logs will come

- consoleLogEnabled : Bool

true – console log is enabled

false – console log is disabled

- fileLogEnabled : Bool

true – console log is enabled

false – console log is disabled

- filename : String

Log files will be generated with this file name appended by timestamp of the file creation. File will be saved in folder “Application root directory /Library/Caches/Logs”

- fileSizeInBytes : Integer

It is the maximum size of single file, once a file reached this limit a next new file is generated

E.g. 51200 for 50KB

- numberOfFiles : Integer

Number of log files generated

- componentLevelLogEnabled : Bool

true - then logs will be filtered out based on list of components mentioned in

componentIds

false – no filtering, all the components log will come.

- componentIds : Array

If componentLevelLogEnabled is true, then logs will be enabled only for components present in the list.

Note:It is the proposition’s responsibility to disable logging when releasing to the market. Most certainly the console logging. But also for file as we are not safe guarding the log files as of yet (or if they want to live dangerously ensure that no sensitive data ends up in the file log).

Integration

Create app Logging Instance

Common components need to create separate logging instance with TLA as component name. This instance creation enable proposition to filter out logs from individual component

Propositions should use logging instance available in appinfra.

id<AILoggingProtocol> logging = [self.appInfra.logging createInstanceForComponent:@"ail" componentVersion:@"1.0.1"];

the same logging instance needs to be used for all the logging calls

Whenever we create component automatically log will print as

2016-07-15 15:31:31.961 | INFO | ail:1.0.1 | NA | Logger Created for ail

(UTC Time Stamp | Log Type | Component Name:Version | Event | Message)

Logging messages

This method is used for logging

- (void)log:(AILogLevel)level eventId:(NSString *)eventId message:(NSString *)message;

level : the type of logging

available values:

AILogLevelError

AILogLevelWarning

AILogLevelInfo

AILogLevelDebug

AILogLevelVerbose

eventId : the identifier of the event being logged

message : message that needs to be logged

Example: logging error message

[logging log:AILogLevelError eventId:@"Event ID" message:@"Error message"];

Log Message Format eg:

2016-07-15 15:34:00.310 | ERROR | ail:1.0.1 | Event ID | Error message

7. Service Discovery

Service Discovery reduces the hard dependency between app and cloud services. The main idea is that the list of URLs that are to be used by an application is maintained server side, at the service discovery server. The app only has to download this list from **one single global location**, this list tells the app where all other cloud services can be found. It is the service discovery server's responsibility to ensure that the correct URLs are returned for the country and our language in which that app is being used. If cloud services are relocated, only the list at that service discovery server needs to be updated, no changes on app side are required. App Identity concept(refer App Identity module for more details) is required to execute Service Discovery and the below information must be given in the AppConfig.json file under “appinfra” group.

"appidentity.micrositeId": <proposition micrositeId>,

"appidentity.sector": "<app sector>",

"appidentity.appState": "<app state>",

"appidentity.serviceDiscoveryEnvironment": "<proposition service discovery environment>",

"servicediscovery.platformMicrositeId":<platform micrositeID>,

"servicediscovery.platformEnvironment":"<platform service discovery environment>",

"servicediscovery.propositionEnabled":true/false

Servicediscovery environment and platformEnvironment state should be only staging and production.

Service discovery server supports proposition specific services and platform services. There will be two micrositeId defined for proposition and platform. Service discovery will download from 2 microsites

AppInfra will throw runtime exception if these values are not added correctly. Both micrositeId and environment are required for Service Discovery.

If a URL occurs in both microsites, then URL in the proposition microsite will be returned. Proposition can overwrite platform URL by configuring an empty value ("https://delete.delete") for that serviceId

"servicediscovery.propositionEnabled" configuration in AppInfra used to disable proposition microsite id. Default setting is “true”. If propositions don’t have data in service discovery server, they should disable (give value “false”). So that Service discovery will download urls from platform. If this configuration is not present / value is true , SD will download both platform and proposition urls. If the valus is false it will download only from platform microsite id.

Please refer this table for the URL configurations

Proposition URL (Entry in Server)	Platform URL (Entry in Server)	Returned URL from Service Discovery	Error message
Yes	Yes	Proposition	None
Yes	No	Proposition	None
No	Yes	Platform	None
No	No	Null	Error will be thrown (“Service Discovery cannot find the URL”)
https**://**delete**.**delete	Yes	Null	Error will be thrown (“Service Discovery cannot find the URL”)

Service discovery will persistently cache proposition and platform URLs for better performance. Service discovery methods will return value from the cached data if its available and if not expired. The cached data will become invalid if any of the below conditions are met.

- App state has changed
- Country or primary locale has changed
- Cached data is older than 24 hours.

Use cases

we can use this component to Get Home Country Code, Set Home Country Code, Get ServiceUrl /ServiceUrls With Language Preference, Get ServiceUrl/ServiceUrls With Country Preference, Get ServiceLocale With Language Preference and Get ServiceLocale With Country Preference.

Getting Home Country

When not yet set, the country code is automatically determined from the SIM card's country of registration. If no SIM card is available/accessible; then geo-IP is used to determine the country. Once determined the country is stored persistently and the stored country will be returned.

```
[appInfra.serviceDiscovery getHomeCountry:^(NSString *countryCode, NSString *sourceType, NSError *error) {

    <#code#>

}];
```

Getting Home Country – synchronous API

```
NSString * homeCountry = [appInfra.serviceDiscovery getHomeCountry];
```

Returns the saved home country. Returns nil if home country is not set.

Setting Home Country

Persistently store Home country, overwrites any existing country value.

```
[appInfra.serviceDiscovery setHomeCountry:@"DE"];
```

Observing home country change

We can observe to home country change using this notification

AILServiceDiscoveryHomeCountryChangedNotification

New county code will be sent along with the notification in userInfo dictionary for key “ail.servicediscovery.homeCountry”

Get ServiceURL with Language Preference

This will return the URL for a specific service with a preference for the current language.

```
[appInfra.serviceDiscovery getServiceUrlWithLanguagePreference:@"AppInfra.Testing" withCompletionHandler:^(NSString *serviceURL, NSError *error) {

        <#code#>

    }];
```

Get ServiceURL with Language Preference & replacement

This will return the URL for a specific service with a preference for the current language. This will replace the placeholders in the URL with the values we supply in the replacement parameter.

```
NSDictionary * map = @{@"sector":@"sec1",@"catalog":@"cat1",@"ctn":@"ctn2"};

[appInfra.serviceDiscovery getServiceUrlWithLanguagePreference:@"AppInfra.Testing" withCompletionHandler:^(NSString *serviceURL, NSError *error) {

        <#code#>

    } replacement:map];
```

Get ServiceURL with Country Preference

This will return the URL for a specific service with a preference for the current home country.

```
[appInfra.serviceDiscovery getServiceUrlWithCountryPreference:@"AppInfra.Testing" withCompletionHandler:^(NSString *serviceURL, NSError *error) {

        <#code#>

    }];
```

Get ServiceURL with Country Preference & replacement

This will return the URL for a specific service with a preference for the current home country. This will replace the placeholders in the URL with the values we supply in the replacement parameter.

```
NSDictionary * map = @{@"sector":@"sec1",@"catalog":@"cat1",@"ctn":@"ctn2"};

[appInfra.serviceDiscovery getServiceUrlWithCountryPreference:@"AppInfra.Testing" withCompletionHandler:^(NSString *serviceURL, NSError *error) {

        <#code#>

    } replacement:map];
```

Get Services with Language Preference

You can pass array of service ids and you will get the service urls as collection with key equal to service id and value corresponding to service url.

This will return the URL and locale for a specific services with a preference for the current language.

```
[appInfra.serviceDiscovery getServicesWithLanguagePreference:@[@"AppInfra.Testing1",@"AppInfra.Testing2"] withCompletionHandler:^(NSDictionary<NSString *,AISDService *> *services, NSError *error) {
```

```
    <#code#>

}];
```

Get Services with Language Preference & replacement

This will return the URLs and locales for a set of services with a preference for the current language. This will replace the placeholders in the URL with the values we supply in the replacement parameter.

```
NSDictionary * map = @{@"sector":@"sec1",@"catalog":@"cat1",@"ctn":@"ctn2"};

[appInfra.serviceDiscovery getServicesWithLanguagePreference:@[@"AppInfra.Testing1",@"AppInfra.Testing2"] withCompletionHandler:^
(NSDictionary<NSString *,AISDService *> *services, NSError *error) {

    <#code#>

} replacement:map];
```

Get Services with Country Preference

This will return the URLs and locales for a set of services with a preference for the current home country.

```
[appInfra.serviceDiscovery getServicesWithCountryPreference:@[@"AppInfra.Testing1",@"AppInfra.Testing2"] withCompletionHandler:^
(NSDictionary<NSString *,AISDService *> *services, NSError *error) {

    <#code#>

}];
```

Get Services with Country Preference & replacement

This will return the URLs and locales for a set of services with a preference for the current home country. This will replace the placeholders in the URL with the values we supply in the replacement parameter.

```
NSDictionary * map = @{@"sector":@"sec1",@"catalog":@"cat1",@"ctn":@"ctn2"};

[appInfra.serviceDiscovery getServicesWithCountryPreference:@[@"AppInfra.Testing1",@"AppInfra.Testing2"] withCompletionHandler:^
(NSDictionary<NSString *,AISDService *> *services, NSError *error) {

    <#code#>

} replacement:map];
```

Get Service Locale with Language Preference

This will return the locale to be used for a specific service with a preference for the current language.

```
[appInfra.serviceDiscovery getServiceLocaleWithLanguagePreference:@"AppInfra.Testing" withCompletionHandler:^(NSString *serviceLocale,
NSError *error) {

    <#code#>

}];
```

Get Service Locale with Country Preference

Returns the locale to be used for a specific service with a preference for the current home country.

```
[appInfra.serviceDiscovery getServiceLocaleWithCountryPreference:@"AppInfra.Testing" withCompletionHandler:^(NSString *serviceLocale,
NSError *error) {

    <#code#>
```

```
}];
```

Refreshing

Refresh the list of service for this application. List is based on sector, microsite, home country, and language

```
[appInfra.serviceDiscovery refresh:^(NSError *error) {
    <#code#>
}];
```

Apply URL parameters

Replaces all '%key%' placeholders in the given URL, where the key is the key in the replacement table and the placeholder is replaced with the value of the entry in the replacement table

```
[appInfra.serviceDiscovery applyURLParameters:@"https://dev.philips.com/prx/product/%sector%/en_GB/%catalog%/products/%ctn%.assets" parameters:map];
```

Returning key manager keys along with service discovery urls

Service discovery getServices method returns AISDService object in the completion handler.

AISDService class has the below properties

url – gives service discovery URL value

locale – gives locale from service discovery

error - will return error if url from service discovery is nil.

kMap - this property will return the de obfuscated keys bundled in the app. Key index will be found from service discovery. If key index url is present in service discovery, keys will be set to this property.

kMapError - will return error if not able to find key manager values

So if key indexes are defined for a service id (‘<serviced>.kindex’ is present in service discovery urls) when we request for multiple services using ‘getServices’ method, then keys also will be returned along with urls in ‘kMap’ dictionary.

These are the error code values if key cannot be found

```
///invalid url index from service discovery
    AIKMInvalidIndexUrl = 3400,
    ///data not found in key bag json
    AIKMDataNotFound,
    /// keybag json is invalid
    AIKMInvalidJSON,
    ///kindex is not present in service discovery
    AIKMNoKindexURLFound,
    ///deobfuscation error
    AIKMConversionError,
```

///key bag json is not added in bundle

AIKMNoJSONFound,

Retry Mechanism:

When service discovery fails further API calls will wait unitil the hold backtime (10 seconds) to reduse the bandwidth consumed

AppInfraDevTools:

This is a library which can be used as an extension with the app infra library to to statically test the service URLs with service discovery by reading the URLs from the static CSV file which should be added to the main bundle, find the sample CSV file at https://bitbucket.atlas.philips.com/projects/MAIL/repos/app-infra_ios/browse/Documents/External/77000.CSV

Step1: Intall the pod AppInfraDevTools

pod 'AppInfraDevTools', '1.0.0-rc.1'

Step2: Create app infra instance in app delegate as follows

Import the following header

#import "AppInfraDevTools.h"

Create app infra instance by injecting the service discovery static implementation as alternative implementation

Create instance of the alternate implementation file

AIDTServiceDiscoveryManagerCSV *objAIDTSD = [[AIDTServiceDiscoveryManagerCSV alloc]init];

Create app infra instance by passing the alternate implemenatation to the builder pattern

self.appInfra = [AIAppInfra buildAppInfraWithBlock:^(AIAppInfraBuilder *builder) {

builder.serviceDiscovery = objAIDTSD;

}];

Set the appinfra instance back to the alternate implementation instance as follows to use other features of appinfra

[objAIDTSD setAppInfra:self.appInfra];

Step3: include the csv file **into the main bundle**

Containing the sevicediscovery data same as the excelsheet . The file name should be same as microsite id.

i.e suppose microsite ID is 77000 fine name should be 7700.csv

Please note the csv file validation is not done as of now. So please make sure that it is same as the attached file else may be result in crash

Note: CSV file can be created by copy-pasting the excel sheet data into a text file and saving it with .CSV extension

8. App Identity

An application is identified by various parameters; this item maintains all these parameters. The parameters are used amongst others for tagging and service discovery but are also shown in the UI.

Integration

1. Include AppConfig.json **file into the app**

In AppConfig.json file add following keys

"appidentity.micrositeId",

"appidentity.sector,

"appidentity.appState",

"appidentity.serviceDiscoveryEnvironment"

under group “appinfra”

If static config for Appstate is “production” all the get APIs will read from static config file and dynamic configurations will be completely ignored. If the appstate is anything other than “production” the parameters state and serviceDiscoveryEnvironment can be modified using appconfig component of appinfra. If the appstate is anything other than “production” AppIdentity will give modified value of the parameters

Note, if runtime changes have been made to config settings, then making changes to static configuration file will not be visible.

Only way to see static config changes is:

1.Not make dynamic configuration changes

2.Delete dynamic configuration change

3.Use the new getDefaultPropertyForKey from appconfig

Will through Exceptions if any of the following condition is not met

All the parameters should be present in appconfig file with valid values under the group “appinfra”

Valid appversion in info.plist file

All 4 fields are mandatory.

The parameters are defined as follows:

env: PIL environment to be used. The PIL environment to be used depends on the status of the application itself. Possible values:

tst: test environment, used when application is in **DEVELOPMENT** or **TEST** status;

acc: acceptance environment, used when application is in **STAGING** or **ACCEPTANCE** status;

www: production environment, used when application is in **PRODUCTION** status.

sect: sector for the application, to be configured by app development team in App Identity. Possible values:

B2C

B2B_HC

corporate

B2B_LI

microsite: identity of the app in the PIL environment, to be configured by app development team in App Identity.

locale: locale used for the UI of the app itself, available in Internationalization

tags: status of the app (URL encoded), configured by the app development team in App Identify. Possible values:

apps%2b%2benv%2bdev: used when application is in **DEVELOPMENT** status

apps%2b%2benv%2btest: used when application is in **TEST** status

apps%2b%2benv%2bstage: used when application is in **STAGING** status

apps%2b%2benv%2bacc: used when application is in **ACCEPTANCE** status

apps%2b%2benv%2bprod: used when application is in **PRODUCTION** status

country: ISO-3166-1 2-character uppercase country code. If this parameter is not included, the server will use Geo-IP to automatically determine the country.

2. Include InfoPlist.strings file into the app

Create InfoPlist.strings file and add the language specific files

Eg:

1) InfoPlist.strings (English)

"CFBundleDisplayName" = "App Infra";

"CFBundleName" = "App Infra";

2) InfoPlist.strings (Chinese (Simplified))

"CFBundleDisplayName" = "";

"CFBundleName" = "";

Use cases

we can use this component to Get App Name, Get Localized App Name, Get App Version, Get App State, Get Microsite ID and Get Sector.

-> Get App Name

ex: [objAppInfra.appIdentity getAppName];

-> Get Localized App Name

ex: [objAppInfra.appIdentity getLocalizedAppName];

-> Get App Version

ex: [objAppInfra.appIdentity getAppVersion];

-> Get App State

ex:

```
#define kStateTest @"TEST"
#define kStateDevelopment @"DEVELOPMENT"
#define kStateStaging @"STAGING"
#define kStateAccepteance @"ACCEPTANCE"
#define kStateProduction @"PRODUCTION"
// get coresponding State in the string format
    NSString *appState;
    switch ([objAppInfra.appIdentity getAppState]) {
        case TEST:
            appState = kStateTest;
            break;
        case DEVELOPMENT:
            appState = kStateDevelopment;
            break;
        case STAGING:
            appState = kStateStaging;
            break;
        case ACCEPTANCE:
            appState = kStateAccepteance;
            break;
        case PRODUCTION:
            appState = kStateProduction;
            break;
        default:
            break;
    }
```

-> Get Microsite ID

ex: [objAppInfra.appIdentity getMicrositeId];

-> Get Sector

ex: [objAppInfra.appIdentity getSector];

-> Get Service Discovery Environment

ex:[objAppInfra.appIdentity getServiceDiscoveryEnvironment];

Notes:

If the appsate is “production” in app config file all apis return values from static config file

irre

9. Time

The time module provides a UTC clock that is synchronized with cloud servers independent from the device local time.

Integration:

Create a property for NSDateFormatter

@property(nonatomic,retain) NSDateFormatter *dateFormatter;

// set date format

self.dateFormatter = [[NSDateFormatter alloc] init];

[self.dateFormatter setDateFormat: @"MM-dd-yyyy hh:mm:ss a"];

Get UTC Time:

[objAppInfra.time getUTCTime]

07-18-2016 06:32:56 pm

Refresh UTC Time:

[objAppInfra.time refreshTime];

Check Status:

To check if the time synchronized

[objAppInfra.time isSynchronized];

Custom NTP pools support: Time Module supports NTP pools customization, this could be achieved by adding ntp.hosts file to the main bundle of your application, please find the default ntp.hosts at this path: **Philips_Git_Repos/app-infra_ios/Documents/External/ntp.hosts**

10. Internationalization

The internationalization module facilitates in optimizing UI information to fit to the user’s preference, based on the app locale.

-(NSLocale *)getUILocale

The Locale module shall provide an API to get the current: UI locale (language + country), the country information may be missing. The Internationalization module shall base the default UI locale on OS settings.

-(NSString *)getUILocaleString

This API will give localeIdentifier as string of format xx_XX. . [[NSLocale currentLocale] localeIdentifier] will give locale identifier in different format for some countries and language like HongKong- Simplified Chinese (ie zh-Hans_HK). This API is to solve that issue.

-(NSString *)getBCP47UILocale

Get current locale as string with '-' as seperator between language and country code (HSDP supported)

This API returns complete locale along with language code country code seperated by '-' (ex: returns en-US when 'English' is set as Language), while the getUILocale API returns only 'en'

Ex: [objAppInfra.internationalization getBCP47UILocale];

[objAppInfra.internationalization getUILocaleString];

[objAppInfra.internationalization getUILocale];

Current UI Locale: en_US

11. App Configuration

The app configuration module maintains configuration settings of the app and its included common components, in the form of key value pairs organised in groups. The app configuration module will read a configuration file “AppConfig.json” containing a default configuration at the first start of the application after clean install. The configuration file will be two levels first levels will be groups which will be the component name, the second level will be key value pairs for a specific group. All the values will be stored securely in the device and persists across app launches. This configuration will be saved in memory and updating the configuration is only possible through AppConfig setPropertyForKey method.

Key –Value DataTypes:

1. This key value pairs should be single level
2. Key should be string
3. Key and group is case insensitive
4. Value can be any of the following
    1. String
    2. NSNumber
    3. Array of Strings
    4. Array of NSNumber
    5. Dictionary – Dictionary should contain only strings and NSNumbers

Future Plan:

configuration data migration

Use Cases:

Common components and vertical app can use this module to save and retrieve configurations securely and persistently in the device

Integration:

Include “AppConfig.json” in main Bundle to include default values of the configuration

sample of AppConfig.json as follows

```
{
    "groupName": {

        "key1": “value1”
   },
    "AI": {
        "MicrositeID": 77000,
        "RegistrationEnvironment": "Staging",
        "NL": ["googleplus", "facebook"  ],
```

```
        "US": ["facebook","googleplus" ]
    }
}
```

getting values

```
 NSError * error ;
 id value =   [objAppInfra.appConfig getPropertyForKey:@“key” group:@“group” error:&error];
```

Description :Gets property for key

Parameters:

key: the key

group: the group name

error: the configError as OUT parameter; errorCode is null in case of success, or contains error value on failure

Throws: IllegalArgumentException if group or key are null, empty string, or contain other characters than [a-zA-Z0-9_.-]

Returns: the value for key mapped by name, or null if no such mapping exists if value is literal then 'String' Object is returned if value is number then 'Integer' Object is returned if value is array of literal then 'array of String' Object is returned if value is array of number then 'array of Integer' Object is returned. This API will return the value if it is modified dynamically (through set api). If it is not there it will return from cloud provided cloud refresh was success. If it is not there in cloud then it will return from static json file. If there is no key static file you will get appropriate error.

```
  id value =   [objAppInfra.appConfig getDefaultPropertyForKey:@“key” group:@“group” error:&error];
```

for this APi the parameters same as above but this api will return the value that is specified in the staic AppConfig.json file

setting values

```
NSError *error;

    [objAppInfra.appConfig setPropertyForKey:@“key” group:@“group” value:@“value” error:&error];
```

Description :Sets property for key

Parameters:

key: the key

group: the group name

value: value new value that needs to be set for the key. Value can be String, Number, Array of Strings, Array of Numbers and Dictionary

error: the configError as OUT parameter; errorCode is null in case of success, or contains error value on failure

Throws: IllegalArgumentException if group or key or value are null, empty string, or contain other characters than [a-zA-Z0-9_.-]

Returns: Boolean indicating whether successfully updated or not, If returns NO, need to check Error object.

You can also set nil to key tis will effectively remove that key from dynamic config. It may still return value if any exists in cloud or static

resetConfig API should be called as follow

```
        NSError *error;
        Bool success = [appInfraInstance.appConfig resetSonfig:&error];
```

12. REST Client

This Module is a extended version of AFNetworking library which is used to perform HTTP operations such as GET/PUT/DELETE/Patch and the other methods of NSURLSessions like upload/download tasks and resume/cancel task (Apple docs has the list of all the APIs)

Instance of this RESTClient can be created with the following APIs

createInstanceWithBaseURL

createInstanceWithBaseURL:sessionConfiguration

createInstanceWithSessionConfiguration

createInstanceWithBaseURL:sessionConfiguration:withCachePolicy

a.Configuration:

And It has a feature of caching and encrypting the responses

Cache size and limit can be set in the AppConfig.json file against the key

“restclient.cacheSizeInKB” as shown below


```
{
    "appinfra": {
        "appidentity.micrositeId": 77001,
        "appidentity.sector": "b2c",
        "appidentity.appState": "production",
        "appidentity.serviceDiscoveryEnvironment": "PRODUCTION",
        "restclient.cacheSizeInKB": 51200,
        "tagging.sensitiveData": ["language"],
        "abtest.precache": ["testStart", "testUpdate", "philipsmobileappabtest1content","DOT-ReceiveMarketingOptIn"],
        "contentLoader.limitSize": 100,
        "servicediscovery.platformMicrositeId": 70000,
        "servicediscovery.platformEnvironment": "production",
        "appconfig.cloudServiceId": "appinfra.appconfigdownload",
        "languagePack.serviceId": "appinfra.languagepack",
        "servicediscovery.propositionEnabled": true,
        "servicediscovery.countryMapping":{"LU":"BE","MO":"HK"},
       "appUpdate.serviceId.test": "appinfra.appupdate",
        "appUpdate.serviceId": "appinfra.testing.version",
        "appUpdate.autoRefresh": true,
        "abtest.mapping":{
        "usr.login":"DOT-ReceiveMarketingOptIn"
        }
    }
}
```


API Calls Ex:

restClient = [objAppInfra.RESTClient createInstanceWithBaseURL:nil sessionConfiguration:[NSURLSessionConfiguration defaultSessionConfiguration] withCachePolicy: AIRESTURLRequestUseProtocolCachePolicy];

b.Cache policy could be any one of these policies:

AIRESTURLRequestUseProtocolCachePolicy = 0,

AIRESTURLRequestReloadIgnoringLocalCacheData = 1,

AIRESTURLRequestReturnCacheDataElseLoad = 2,

c.HTTPRequestSerializer:

AFHTTPRequestSerializer class is used to build the Http request by passing the formdata/headers and we can use any methods of this class

Ex:

NSMutableURLRequest * getRequest = [[AFHTTPRequestSerializer serializer] requestWithMethod:@"GET" URLString:URLString parameters: dictParamas error:nil];

d.HTTPResponseSerializer:

AIAFHTTPResponseSerializer class is used to build httpResponse serealizer

Ex:

restClient = [objAppInfra.RESTClient createInstanceWithBaseURL: sessionConfiguration: withCachePolicy:];

restClient.responseSerializer = [AIAFHTTPResponseSerializer serializer];

e.Authentication:

RESTClient provides 2 delegate method to set the authentication as follow

-(AIRestClientTokenType)getTokenType;

-(nullable NSString *)getTokenValue;

And propositions should confirm to these protocol methods just by adopting to that protocol

As shown in the below ex, and the auth token is set to http Header with the key "Authorization"

Ex:

```
// Interface
@interface testViewController ()<AIRESTClientDelegate>{
}
@end

// Implementation
@implementation testViewController
-(AIRestClientTokenType)getTokenType{
    return AIRestClientTokenTypeOAUTH2;
}
-(nullable NSString *)getTokenValue{

    return self.authToken;
}
@end
```

4. Internet reachability check

REST Client provides following 2 APIs to check internet availability. It also post notification if there is a change in network status

1) **getNetworkReachabilityStatus:** Used to check the internet availability which returns the enum

possible return values are

AIRESTClientReachabilityStatusNotReachable ,

AIRESTClientReachabilityStatusReachableViaWWAN,

AIRESTClientReachabilityStatusReachableViaWiFi,

2) **isInternetReachable :** Used to check the internet availability Which returns the bool value

3)to get network status change observe for notification “kAILReachabilityChangedNotification”

objective c

1. Get status

```
AIRESTClientReachabilityStatus status =   [appInfra.RESTClient getNetworkReachabilityStatus];
```

2. Status changes

1. ```
   [[NSNotificationCenter defaultCenter]addObserver:self selector:@selector(aireachabilityChanged:) name:
   kAILReachabilityChangedNotification object:nil];
   ```
   ```

```
-(void)aireachabilityChanged:(NSNotification*)notification{

    id<AIRESTClientProtocol> restClient = [notification object];
    [self displayStatus:[restClient getNetworkReachabilityStatus]];

}

- (void)displayStatus:(AIRESTClientReachabilityStatus)status {
    NSString *statusMessage = @"not available";
    UIColor *color = [UIColor orangeColor];
    switch (status) {
        case AIRESTClientReachabilityStatusNotReachable:
            statusMessage = @"NotReachable";
            color = [UIColor redColor];
            break;
        case AIRESTClientReachabilityStatusReachableViaWiFi:
            statusMessage = @"ReachableViaWiFi";
            color = [UIColor greenColor];
            break;
        case AIRESTClientReachabilityStatusReachableViaWWAN:
            statusMessage = @"ReachableViaWWAN";
           color = [UIColor greenColor];
            break;
        default:
            break;
    }
}
```

swift

1. Get network status

```
let status =  appInfra.restClient.getNetworkReachabilityStatus()

        switch status {
        case .notReachable:
            print("appinfra : not reachable")
        case .reachableViaWWAN:
```

```
        print("appinfra :  reachableViaWWAN")
    case .reachableViaWiFi:
        print("appinfra :  reachableViaWiFi")
    }
```

2. Observe for status changes

```
NotificationCenter.default.addObserver(self, selector: #selector(self.reachabilityChanged), name: NSNotification.Name(rawValue:
kAILReachabilityChangedNotification), object: nil)

@objc func reachabilityChanged(notification:NSNotification){
    if let reachability = notification.object as? AIRESTClientProtocol     {
      //let connectionRequired = reachability.connectionRequired()
    let status = reachability.getNetworkReachabilityStatus()

    switch status {
    case .notReachable:
        print("ail:   network notreachable")
    case .reachableViaWiFi :
        print("ail:   reachable via wifi.")
    case .reachableViaWWAN :
        print("ail:   reachable via WWAN.")

    }

    }
  }
```

13. A/B Test

A/B (Alpha/Beta) testing feature is to facilitate the other micro apps or vertical application to make a choice of their Application flow to be considered for execution. It could be an alternate UI flow, Theme settings etc. The definition of the different flows itself is out of the scope from this feature. The A/B library will use ADOBE SDK which in turn will talk to Test and Target infrastructure to retrieve the flow ID's for a given flow keys. These keys will come from the other micro apps via the interface exposed by the A/B library component.

A/B will expose an API to receive the Key. So the A/B will hand over the key to the API exposed by Adobe and expect a value in return. This value is then handed over to the calling application.

Integration

Specify the keys in AppConfig

Specify the keys in AppConfig.json

All the testnames / mbox name should be added to abtest.precache key. If you didnt specify you test name in appconfig file you will always get default value only.

"abtest.mapping" key is a dictionary which contain the mapping for coco testname and the actual test name in adobe server. cocos we suggest to prepend coco tla to the coco test name eg "usr.loginFlowTest" for readability.

Include adobe config json file and give target configuration

```
"appinfra": {
        "abtest.precache":["testStart","DOT-ReceiveMarketingOptIn","philipsmobileappabtest1content"],
      "abtest.mapping":{
                    "usr.login":"DOT-ReceiveMarketingOptIn"
                        }
            }
```

Include adobe config json file and give target configuration

```
  "target" : {
      "clientCode" : "philipselectronicsne",
      "timeout" : 15
  },
```

Path can set through apptagging if needed

Update cache api to prefetch the value for tests specified in AppConfig file

Tests not specified in AppConfig will return default value only

Test Names can be any of two types

AIABTestUpdateTypeAppStart : These tests will be refreshes if you update cache after apprestart. The test value will not change till the next app start

AIABTestUpdateTypeAppUpdate: These tests will be refreshes if you update cache after app update. The test value will not change till the next app updadate

API's:

-(void)updateCacheWithSuccess:(nullable void(^)())successBlock

error:(nullable void(^)(NSError* _Nullable error))errorBlock;

This API downloads and cache the experience values from the server. Test names should be mentioned in the App configuration.

Refresh will not happen if there are no tests defined in configuration or when experiences are already updated.

The method will give success callback if all the test Names that need to be refreshed

This method will give error callback in case of error error code can be any of the following

```
typedef NS_ENUM(NSUInteger,AIABTestErrorCode)
{   /**
     *this error will happen if try to update cache without internet
     */
    AIABTestErrorCodeNoNetwork = 0,
    AIABTestErrorCodeExperiencesPartiallyDownloaded,
};
```

2) -(AIABTestCacheStatus)getCacheStatus;

Returns the status of the cached experiences for the configured list of tests.At initialization of the module, the status is either NoTestsDefined, NoCachedExperiences, or ExperiencesNotUpdated.An updateCache may change the state to ExperiencesPartiallyUpdated or ExperiencesUpdated.

@return status of the experience cache.

Cache Status can be any one of these:

```
typedef NS_ENUM(NSUInteger,AIABTestCacheStatus)
{
    /**
     *No test names are defined in app config file
     */
    AIABTestCacheStatusNoTestsDefined = 0,
    /**
```

```
    *tests are defined but not cached
    */
   AIABTestCacheStatusNoCachedExperiences,
   AIABTestCacheStatusExperiencesNotUpdated,
   AIABTestCacheStatusExperiencesPartiallyUpdated,
   /**
    *all the experiences are updated in cache
    *need not be from server, it can be default value also
    */
   AIABTestCacheStatusExperiencesUpdated,

};
```

```
-(nullable NSString*)getTestValue:(nullable NSString*)testName
                   defaultContent:(nullable NSString*)defaultValue
                      updateType:(AIABTestUpdateType)updateType
                      parameters:(nullable NSDictionary *)parameters;
```

This method returns the value for the given test either from the cache.

Default value will be returned if there is no value present in the cache.

This method will give test value from server only after update cache and the value will be locked based on the update type. The value will not change till the next app update or app restart

Parameters:

testName: TestName for which the testValue is needed. or The mapped key specified in abtest.mapping config

defaultValue: default value to be returned .

updateTypes: updateType can be App restart and App Update .

parameters : parameters that needs to be passed to adobe server

14. Securing SQLite and CoreData databases in iOS

Provides encryption to the SQLite storage in iOS apps.

File Protection Classes

Using Data Protection Classes:

When a new file is created on an iOS device, it’s assigned a class by the app that creates it. Each class uses different policies to determine when the data is accessible. The basic classes and policies are described in the following sections.

Complete Protection

(NSFileProtectionComplete): The file is stored in an encrypted format on disk and cannot be read from or written to while the device is locked or booting.

completeUnlessOpen

(NSFileProtectionCompleteUnlessOpen): The file is stored in an encrypted format on disk. Files can be created while the device is locked, but once closed, cannot be opened again until the device is unlocked. If the file is opened when unlocked, you may continue to access the file normally, even if the user locks the device.

completeUntilFirstUserAuthentication

(NSFileProtectioncompleteUntilFirstUserAuthentication): The file is stored in an encrypted format on disk and cannot be accessed until after the device has booted. After the user unlocks the device for the first time, your app can access the file and continue to access it even if the user subsequently locks the device.

none

(NSFileProtectionNone): The file has no special protections associated with it. It can be read from or written to at any time.

Implementation

Core Data

While creating the persistent store coordinator set the file protection key as bellow

```
NSDictionary *storeOptions = @{NSPersistentStoreFileProtectionKey  : NSFileProtectionComplete};

[_persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL options:storeOptions error:&error]
```

If the file needs to be accessed when device is in the background use NSFileProtectionCompleteUnlessOpen or NSFileProtectioncompleteUntilFirstUserAuthentication

Normal Files

While creating new files we can set the file protection type

```
[[NSFileManager defaultManager] createFileAtPath:[self filePath] contents:[@"super secret file contents" dataUsingEncoding:
NSUTF8StringEncoding] attributes:[NSDictionary dictionaryWithObject:
NSFileProtectionComplete                                    forKey:NSFileProtectionKey]];
```

We can change the file protection type for a file

```
[[NSFileManager defaultManager] setAttributes:[NSDictionary dictionaryWithObject:NSFileProtectionComplete

                                                forKey:NSFileProtectionKey] ofItemAtPath:[self filePath] error:NULL];
```

How to verify

Create a sample file and set the file protection key as NSFileProtectionComplete.

```
[[NSFileManager defaultManager] createFileAtPath:[self filePath] contents:[@"super secret file contents" dataUsingEncoding:
NSUTF8StringEncoding] attributes:[NSDictionary dictionaryWithObject:
NSFileProtectionComplete                                    forKey:NSFileProtectionKey]];
```

```
NSDictionary<NSFileAttributeKey,id> *fileProtectionValue = [[[NSFileManager defaultManager] attributesOfItemAtPath:[self filePath] error:NULL]
valueForKey:NSFileProtectionKey];


NSLog(@"file protection value: %@", fileProtectionValue);
```

It will log

file protection value: NSFileProtectionComplete

In applicationDidEnterBackground try to access the file in background after a delay.

```
    UIApplication *app = [UIApplication sharedApplication];
    UIBackgroundTaskIdentifier bgTask;
    bgTask = [app beginBackgroundTaskWithExpirationHandler:^{
        [app endBackgroundTask:bgTask];
    }];
    [self performSelector:@selector(doReload) withObject:nil afterDelay:20];
```

reload method

```
- (void)doReload {

    NSLog(@"protected data available: %@",[[UIApplication sharedApplication] isProtectedDataAvailable] ? @"yes" : @"no");

    NSError *error;

    NSString * fileContents = [NSString stringWithContentsOfFile:[self filePath]
                                          encoding:NSUTF8StringEncoding
                                             error:&error];

    NSLog(@"file contents: %@\nerror: %@", fileContents, error);

}
```

Now run the app and lock the device. After 20 min it will log in console

protected data available: no

file contents: (null)

error: Error Domain=NSCocoaErrorDomain Code=257 "The file “myfile.txt” couldn’t be opened because you don’t have permission to view it." UserInfo={NSFilePath=/var/mobile/Containers/Data/Application/4B6BEAED-6BDE-42BC-8328-9B798BA97AED/Documents /myfile.txt, NSUnderlyingError=0x170046630 {Error Domain=NSPOSIXErrorDomain Code=1 "Operation not permitted"}}

This means if we set complete protection, file will be encrypted and cannot be read from the app if device is locked or booting. However, if there is some background task that tries to access the file when device is locked, it will give error. If we need to access the file in background use NSFileProtectionCompleteUntilFirstUserAuthentication protection.

Checking whether device has set passcode

Since file protection works only when user has enabled passcode lock to device, we may need to alert the user to set passcode lock to his device if it is not set. Below method can be used to check whether device has enabled passcode lock. This works from iOS 8 and above devices

```
-(BOOL) deviceHasPasscode {

    NSData* secret = [@"Device has passcode set?" dataUsingEncoding:NSUTF8StringEncoding];

    NSDictionary *attributes = @{ (__bridge id)kSecClass: (__bridge id)kSecClassGenericPassword, (__bridge id)kSecAttrService: @"
LocalDeviceServices",  (__bridge id)kSecAttrAccount: @"NoAccount", (__bridge id)kSecValueData: secret, (__bridge id)kSecAttrAccessible:
(__bridge id)kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly };


    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)attributes, NULL);

    if (status == errSecSuccess) { // item added okay, passcode has been set

        SecItemDelete((__bridge CFDictionaryRef)attributes);


        return true;

    }


    return false;

}
```

15. API Signing

Some HSDP services use API signing (and not an oAuth token) to prove the caller is a known app. App Infra is providing an API which can create a signature for a given data blob. The signature is created using an algorithm provided by Philips Security Technologies plus a key. As multiple services may require different signatures, the key may differ per signature created.

Instance of APISigning should be created using the following constructor by passing the shared key and a hex key

```
-   (id)initApiSigner:(NSString *)sharedKey andhexKey:(NSString *)hexKey;
```

EX:

```
id <AIAPISigningProtocol> APISigning = [[AIClonableClient alloc]initApiSigner:@"c62362a0-f02c-11e5-9ce9-5e5517507c66" andhexKey:@"
d9efcaeb7077f16729c1568bde56eed25635030f688990d3fc9281cb809d4666db0057e8b902382f9de16fed325889a46e7c22e31a143ee60b33c1a
c22bc8b28"];
```

and createSignature API should be used to create the signature

Method:

```
/**
    * Create an API signer instance according to HSDP specification
    * @param requestMethod Type of method(POST, GET)
    * @param dhpUrl Request URL
    * @param queryString Request URL query. applicationName=uGrow
    * @param headers Request http header fields. Current date is passed as http header value.
 * @param requestBody Request body
 */
- (NSString *)createSignature:(NSString *)requestMethod dhpUrl:(NSString *)url queryString:(NSString *)requestUrlQuery headers:(NSDictionary *)allHttpHeaderFields requestBody:(NSData *)httpBody;
```

EX:

```
NSString *authorizationHeader = [APISigning createSignature:@"POST" dhpUrl:@"/authentication/login/social" queryString:@"applicationName=uGrow" headers:headerDict requestBody:nil];
```

16. Component Version Info

- AppInfra.ComponentVersionInfo protocol has following APIs

– getComponentId()
returns component three letter acronym as defined at developer portal

– getVersion()
returns component version, which may include snapshot indication

- At creation use the injected AppInfra instance to create its own logging instance and use that instance throughout the component to perform all logging operations.

– This logging instance is to be created while providing the documented component TLA and version number (identical to ComponentVersionInfo defined above)

- For components that include tagging functionality, they shall create their own tagging instance using the injected AppInfra instance similar like for logging, see above.
- Strong recommendation: retrieve version info directly from the build environment rather than trying to keep it up to date manually.

Each coco should implement ComponentVersionInfo Protocol and should return the valid component version and Id

17. Language Pack

Configuration

1.setup service id

add new key in appconfig “languagePack.serviceId” and specify the service id. This service id should be mapped to the overview file url

2.upload overview file

upload overview file to your server and add that url to service discovery as explained in previous step. Sample overview file format is available in external folder. Specify language pack url and version for all supported locale in this file.

3.upload language pack

upload language pack for the locale in your server . The sample json format is available in external folder.

note: after every app update propositions should update overview file and language pack file

update Language pack

In order to update the language pack. Upload the new language pack to the server and increase the version number of that language pack in overview file.

All apps contain text which is visualized to the user in some way, mainly this text is shown directly in the UI. A part of this text is more or less static and fundamental to the operation of the app. For that reason, this text is embedded according to the App UI internationalization guidelines. The text is shown in the locale as selected by the user on his device. This module enable to change these texts dynamically from cloud

Service Discovery is best used for managing the URL for a language pack overview file using the 'match-by-country' functionality, this overview file in turn contains the actual language pack URLs for all available languages. This enables the app the select the best language pack from the available language packs. And by using the 'match-by-country' feature of Service Discovery a proposition can limit the available language packs or optimize the content of a language pack for specific countries (for example supplying language packs which don't contain wordings that are considered offensive in some countries and using other language packs for other countries).

Downloaded language pack will be deleted if there is locale change in os settings. Once the language pack is deleted it will fall back to the bundled strings until the cloud language pack is downloaded and activated.

```
/**

 @brief refreshes the overview file containing the list of languages available in the server and also downloads the language pack based on the user preferred ui locale

 @param : completion handler with refresh status and error if refresh failed

 */

-(void)refresh:(nullable void(^)(AILPRefreshStatus refreshResult,NSError * _Nullable error))completionHandler;
```

It provides the list of available language packs based on device locale.

Language packs contains list of supported locales with URLs and version.

Refresh API finds best matching language pack and stores into internal memory

```
/**

 @brief activates the Language Pack downloaded for the preferred ui locale

 @param : completion handler with activate status and error if activate failed

 */

-(void)activate:(nullable void(^)(AILPActivateStatus activatedStatus,NSError * _Nullable error))completionHandler;
```

Activate API activates stored language pack and returns status through completion block

Note: Refresh and Activate is suggested to be invoked during application launch

```
/**

 * @brief returns the localized string from cloud bundle

 * if it not there in cloud will look in main bundle

 * if not found will return the key
```

```
 */

- (nullable NSString *)localizedStringForKey:(NSString *_Nonnull)key;
```

18. AppUpdate

What is App update information :

AppUpdate checks a user's currently installed version of your app against the version that is currently available in the App Store (as specified in the appupdate json file).

If a new version is available, an alert can be presented to the user informing them of the newer version, and giving them the option to update the application. AppInfra is not providing any userinterface or alert but it provides necessary apis to check the if the appversion is no longer supported or of new update available and necessary messages specified in the cloud file. Since appupdate info is downloaded from service discovery url , it can be country specific . See the api details for more info .

note : all the url links (appupdate info json file)should be https://* . Appinfra RESt client support only https server

Step-by-step guide

1. Upload AppUpdate info to https server.

   check the sample file for the format

   version.json

   - All the values are in string format.
   - All versions comparison will consider X.X.X of the version string
   - deprecated date should be of format YYYY-MM-DD

2. Map appupdate info url in service discovery

   The url containing the appupdate information should be mapped to a service id in service discovery server. AppInfra will download the appupdate information from this file and cache it localy for persistance.

3. provide appconfig key as per your requirement

   The configured app update service is should be added to AppConfig.json file in "appinfra" group for key "appUpdate.serviceId" . AppInfra will be looking for this key in appconfig to get the url for downloading

appupdate information.

4. AutoRefresh Functionality : AutoRefresh of AppUpdate can be enabled by adding the ""appUpdate.autoRefresh":true in the AppConfig.json. If Refresh is successful appupdate json file will be stored in cache .

Next Time When Appinfra initialization is done it will pick from cached file.

iOS APIs

```
/**

 * @brief

 * - AIAppUpdateRefreshSuccess:   App Update info Downloaded from server

 * - AIAppUpdateRefreshFailed:    Refresh Failed

 */

typedef NS_ENUM(NSUInteger,AIAppUpdateRefreshStatus)

{
```

```
    AIAppUpdateRefreshStatusSuccess,
    AIAppUpdateRefreshStatusFailed
};
/**
 * @brief refreshes the appupdate info available in the server
 * refresh will fail if appUpdate.serviceId is missing in appconfig
 * or service discovery is not configured for appupdate
 * or the content of the appupdate file is not in the specified format
 * @param : completionHandler with refresh status and error if refresh failed
 */
-(void)refresh:(nullable void(^)(AIAppUpdateRefreshStatus refreshResult,
                                 NSError * _Nullable error))completionHandler;
/*
 * this will return true if  applicationVersion < minimumVersion
 * true when current application version is less than the minimumVersion
 * true when deprecatedVersion is greater than current application version and deprecationDate is crossed
 */
-(BOOL)isDeprecated;
/*
 * minimumVersion  <= application version <= toBeDeprecated
 * true if application is not already deprecated and current version is lessthan equal to deprecatedVersion
 */
-(BOOL)isToBeDeprecated;
/*
 * applicationversion < currentVersion
 * true if current version is less than the latest version available in the appstore
 */
-(BOOL)isUpdateAvailable;
/*
 * Deprecated Version message string
 */
-(nullable NSString*)getDeprecateMessage;
/*
 * To be deprecated message string
 */
-(nullable NSString*)getToBeDeprecatedMessage;
/*
 * To be deprecated Date
```

```
 */
-(nullable NSDate*)getToBeDeprecatedDate;
```

```
/*
 * currentVersionMessage
 */
-(nullable NSString*)getUpdateMessage;
```

```
/*
 * minimumVersion
 */
-(nullable NSString*)getMinimumVersion;
```

```
/*
 * minimum Os Version
 */
-(nullable NSString*)getMinimumOsVersion;
```