CDP

**App Infra Interface spécification**

**Authors: Senthil Kumar, Ravi Kiran**

**Version : 1.1.0**

**Date : 2016-06-24**

**Status : Proposed**

**Platforms: iOS**

**Components covered:** **Secure storage, App Tagging, Logging, App Identity & Service Discovery**

**Table of Contents**

1. Introduction 3

Purpose & Scope

Target Audiance

1. Secure Storage 3

3. Tagging 6

4. Logging 9

5. App Identity 13

6. Service Discovery 14

**DEFINITIONS & ABBREVATIONS**

|  |  |
| --- | --- |
| **Abbreviation** |  |
| CDP | Connected Digital Propositions |
| UI | User Interface |
| API | Application Programming interface |

1. **Introduction:**

**Purpose & Scope**

This document basically contains the information to help the verticals to understand code design flow of each feature of Chassis (AppInfra).

**Target Audiance:**

All component and App developer.

1. **Secure Storage**

Secure Storage is used to store secret value in device storage with encrypted way using AES. It uses key value pair concept to store data inside the apps.

This contains the interface API’s, which defines what the API does along with when and how to call the same.

This doesn't target the Database security, Initially we are targeting to store the custom objects, user data ,text files, Plist files, we could pick up database storage in the coming versions

## Reference: <http://opensource.apple.com/source/CommonCrypto/>

## Guiding Principles

Vertical propositions should get in touch with the secure storage team if there is any need to extend or modify the component or interface to fulfill their requirements.

# **2.1. ARCHITECTURAL ANALYSIS**

## Product Overview

As part of Connected Digital Proposition, Secure storage is a component that can be reused across various applications through App Infra. The subsequent sections provide the detail of the secure storage as a re-usable component through App Infra.

Secure storage is used to securely store user’s sensitive data by encrypting. We have used AES-256 bit key encryption algorithm to encrypt and using keychain to store the key

## Future Extension

The architecture is scalable and can support future extension in the same line. We can include secure DB storage along with cloud support in future

### **Secure storage Interface – Vertical App**

#### **Responsibility/Functions**

Following functions are provided by this class.

1. Ability to encrypt user’s sensitive data using AES encryption
2. Ability to securely store the encryption key in keychain
3. Ability to decrypt data using AES decryption
4. Ability to remove the stored encrypted data

#### **Interfaces & How to integrate/call the API’s**

How to integrate the API:

Create Object for AppInfra

In AppDelegate.h, import AppInfra and create object

#import <AppInfra/AppInfra.h>

@property (nonatomic,strong) AIAppInfra \*objAppInfra;

In AppDelegate.m,

self.objAppInfra =[AIAppInfra buildAppInfraWithBlock:nil];

Use Cases

Developer can perform three operations:

1. Store value in device

Developer can store value in encrypted format assigning a key name to value.

- (void)storeValue:(nonnull id)object

forKey:(nonnull NSString\*)key

error:( NSError \*\_Nullable\*\_Nullable)error;

eg: [[AppDelegate sharedAppDelegate].objAppInfra.storageProvider storeValue:@“abc@123“ forKey:@“myPasswordKey“];

1. Fetch value from device

Developer can fetch stored value from device using assigned key name.

-(nullable id)fetchValueForKey:(nonnull NSString\*)key

error:( NSError \*\_Nullable\*\_Nullable)error;

eg NSString \*strDataFetched = [[AppDelegate sharedAppDelegate].objAppInfra.storageProvider fetchValueForKey:@“myPasswordKey“];error:nil];

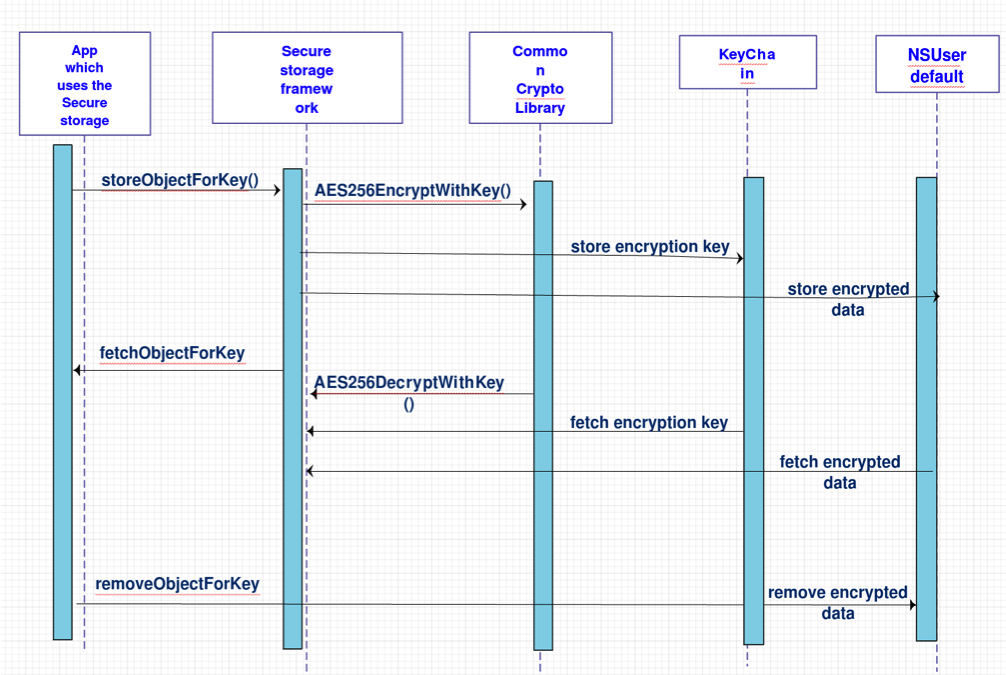
1. Delete value from device

Developer can delete stored value from device using assigned key name.

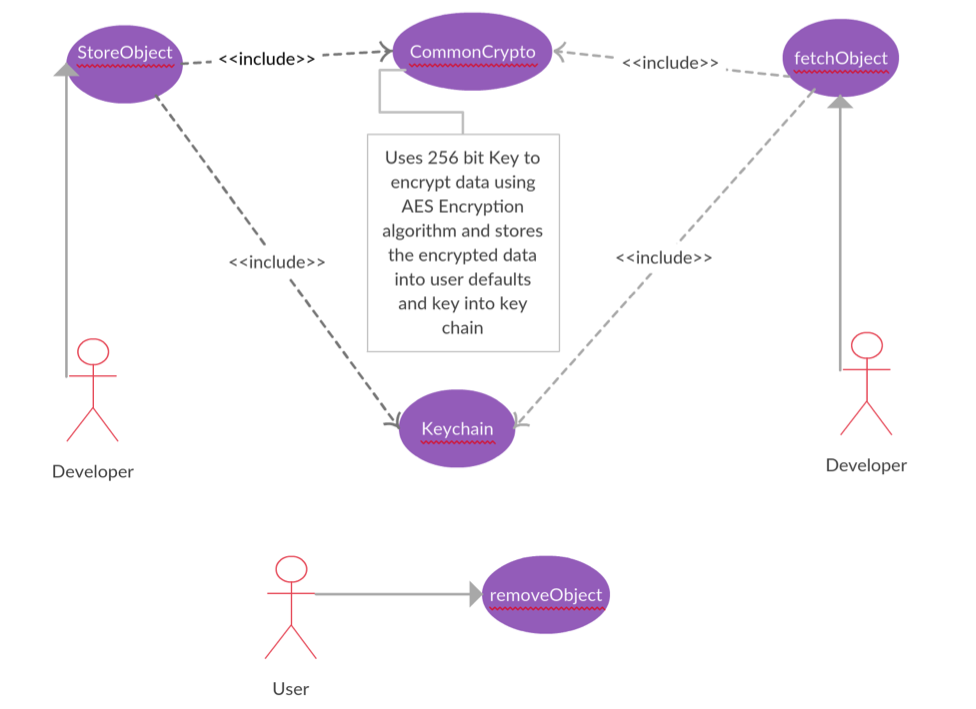
- (void)removeValueForKey:(nonnull NSString\*)key;

eg : [[AppDelegate sharedAppDelegate].objAppInfra.storageProvider removeValueForKey:@“myPasswordKey“];

1. Data Flow

****

**5. Sequence Diagram:**

****

1. **Tagging**

As part of Connected Digital Proposition, App Tagging is a component that can be reused across various applications. The subsequent sections provide the detail of the App Tagging as a re-usable component. it uses the Adobe library (https://marketing.adobe.com/developer/get-started/mobile/c-measuring-mobile-applications) to track and report analytics.

## Reference

* https://marketing.adobe.com/developer/get-started/mobile/c-measuring-mobile-applications

## Guiding Principles

Vertical propositions should get in touch with the App Infra/Analytics team if there is any need to extend or modify the component or interface to fulfill

### **App Tagging Interface – Vertical App**

#### **Responsibility/Functions**

Following functions are provided by this class.

1. Ability to track button action events
2. Ability to track page load events

#### **Interfaces & How to integrate/call the API’s**

AIAppTaggingProtocol **will be have the following methods**

1. **createInstanceForComponent () withComponentId ComponentVersion** - shall be called to create the instance of for each component by providing the Component details
2. ***configureAnalyticsWithFile*** () -it will take a config file as input parameter and configures the analytics accordingly
3. **trackPageWithInfo() withParam key and value** -This method is used to track the page with a single key value pair data
4. **trackPageWithInfo() withParam dictionary -**This method is used to track the page with multiple key value pair data
5. **trackActionWithInfo() withParam key and value** -This method is used to track the button actions with a single key value pair data
6. **trackActionWithInfo() withParam dictionary -**This method is used to track the button actions with multiple key value pair data
7. setP**rivacyConsent()- this method is used to set the privacy consent**
8. **get**P**rivacyConsent()- this method is used to get the privacy consent**

## Use Cases

Create Object for AppInfra

In AppDelegate.h, import AppInfra and create object

#import <AppInfra/AppInfra.h>

@property (nonatomic,strong) AIAppInfra \*objAppInfra;

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

In AppDelegate.m,

self.objAppInfra =[AIAppInfra buildAppInfraWithBlock:nil];

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

[self.objAppInfra.tagging configureAnalyticsWithFilePath:[[NSBundle mainBundle] pathForResource:@"ADBMobileConfigDev" ofType:@"json"]];

[self.objAppInfra.tagging setPrivacyConsent:AIATPrivacyStatusUnknown];

// create app tagging Instances for Demo app

self.objAppTaggingForDemo = [self.objAppInfra.tagging createInstanceForComponent:@"DemoApp" componentVersion:@"1.0.1"];

### **Use case: Track Page**

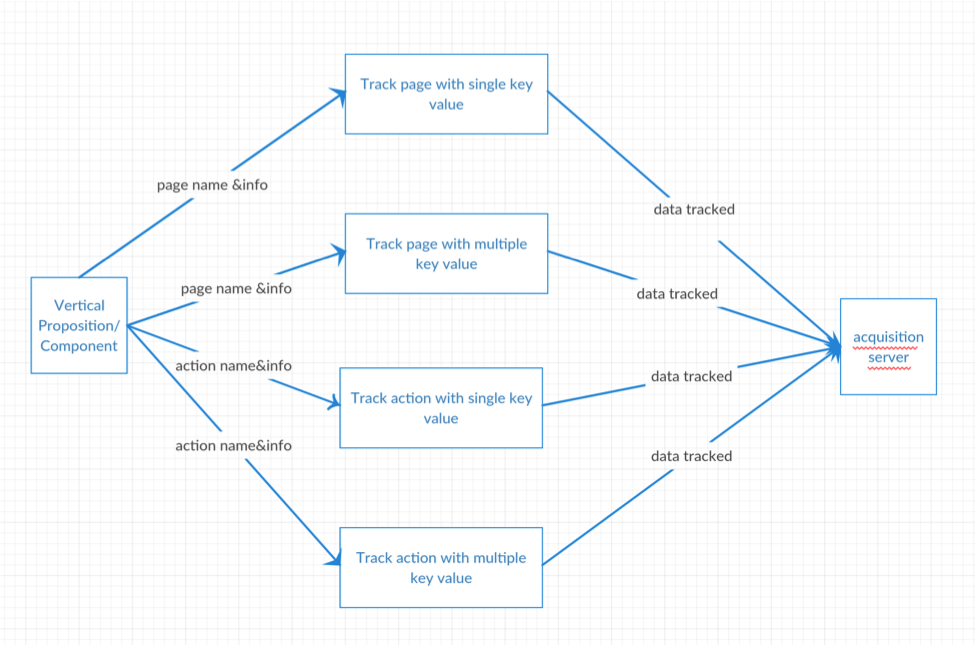
This use case occurs while tracking the page, all the page load events can be tracked with optional additional info parameters

### **Use case: Track Event**

This use case occurs while tracking the button actions, all the button action events can be tracked with optional additional info parameters

**API Usage:**

* **configureAnalyticsWithPrivacyOption** methodshould be called in **AppDelegate’s didFinishLaunchingWithOptions** method by providing all the parameters
* Separate Instances of **AIAppTaggingProtocol** should be created for each component In app delegate class by providing the component ID and name
* **Track action** or T**rack page** methods should be called on the required pages/button actions with the respective instance which are create in app delegate(i.e all the pages which belongs to registration should be tracked by calling trackAction/trackPage on the app tagging instance which is dedicated for registration component+)
  1. Data Flow



1. **Logging**

Logging will be used by various component and App Framework to log events and message in multiple sinks like console and file. Logging provides filtering at log levels and component level.

**CocoaLumberJack** 3rd party library is used for logging logs.

Log can be done at five levels. They are VERBOSE, DEBUG, INFO, WARNING, ERROR.

This contains the interface API’s, which defines what the API does along with when and how to call the same.

This doesn't target Crash Logs, Initially we are targeting to display log in console and file store

## Reference:

* <https://github.com/CocoaLumberjack/CocoaLumberjack>

## Guideline Principle:

Vertical propositions should get in touch with the Logging team if there is any need to extend or modify the component or interface to fulfill their requirements

# ARCHITECTURAL ANALYSIS

## Product Overview

## As part of Connected Digital Proposition, Logging is a horizontal component that can be reused across various applications. The subsequent sections provide the detail of the Logging as a part of AppInfra library.

## Future Extension:

The architecture is scalable and can support future extension in the same line. We can include cloud storage support in future

# Logging Interface – Vertical App

## Responsibility/Functions

Following functions are provided by this class.

1. Ability to maintain Logs such as Info, Warning, Error, Debug and Verbose.
2. Ability to create logs with the help of CocoaLumberjack third party library
3. Connecting CocoaLumberjack with AILogging layer.

## Interfaces & How to integrate/call the API’s

How to integrate the API:

1. [[AppDelegate sharedAppDelegate].objLoggingForAppInfra log: AILogLevelInfo Event:@"Info Event" Message:@"Info Message”]; - shall be called when need to maintain message has to be stored in file/console
2. [[AppDelegate sharedAppDelegate].objLoggingForAppInfra log: AILogLevelError Event:@"Error Event" Message:@"Error Message”]; - shall be called when need to maintain Error message has to be stored in file/console
3. [[AppDelegate sharedAppDelegate].objLoggingForAppInfra log:AILogLevelWarning Event:@"Warning Event" Message:@"Warning Message”]; - shall be called when need to maintain warning message has to be stored in file/console
4. [[AppDelegate sharedAppDelegate].objLoggingForAppInfra log:AILogLevelDebug Event:@"Debug Event" Message:@"Debug Message”]; - shall be called when need to maintain debug message has to be stored in file/console only in debug mode
5. [[AppDelegate sharedAppDelegate].objLoggingForAppInfra log:AILogLevelVerbose Event:@"Verbose Event" Message:@"Verbose Message”]; - shall be called when need to maintain detailed message has to be stored in file/console

Use Cases:

Developer can perform these operations:

1. Initialize log at component level

Prerequisite: Initialize AppInfra object in App and pass its reference to component:

Create Object for AppInfra

In AppDelegate.h, import AppInfra and create object

#import <AppInfra/AppInfra.h>

@property (nonatomic,strong) AIAppInfra \*objAppInfra;

@property (nonatomic,strong) id<AILoggingProtocol> objLoggingForDemo;

In AppDelegate.m,

self.objAppInfra =[AIAppInfra buildAppInfraWithBlock:nil];

// create app Logging Instances for Demo app

self.objLoggingForDemo = [self.objAppInfra.logging createInstanceForComponent:@"DemoApp" componentVersion:@"1.0.1"];

AILoggingConfiguration.h file can be used for configuring framework. App can define different values for each build configuration (i.e. DEBUG /RELEASE).

The following macros can be used for defining the log levels.

1. AI\_LOGLEVEL

This macro is responsible for controlling the current log level.

Different available levels

1. DDLogLevelOff - No logs
2. DDLogLevelError - Error logs only
3. DDLogLevelWarning - Error and warning logs
4. DDLogLevelInfo - Error, warning and info logs
5. DDLogLevelDebug - Error, warning, info and debug logs
6. DDLogLevelVerbose - Error, warning, info, debug and verbose logs
7. DDLogLevelAll - DDLogLevelAll
8. FILE\_LOG\_ENABLED

This macro is responsible for adding logs to file.

* 1 - New logs file will be created for each launch of the app and all logs will be saved.
* 0 - Logs won’t be tracked in files.

1. CONSOLE\_LOG\_ENABLED

This macro is responsible for adding logs to the Apple System Log and XCode console.

Android native log and log4j can be used for adding logs.

* 0 - Console logs disabled
* 1 - Console logs enabled

1. COMPONENT\_LEVEL\_LOG\_ENABLED

This macro is responsible for filtering logs by component name. Filtering component names can be maintained in LIST\_COMPONENTNAME array list.

Android native log and log4j can be used for adding logs.

* 0 - Console logs disabled
* 1 - Console logs enabled

1. MAX\_NUMBER\_OF\_LOG\_FILE

This macro is provided to limit log file creation.

If the app exceeds the max number of files, first created file will be deleted and new one will create.

1. **Get the log files from App**

Testers can find the log files from app data.

1. Download AppData

2. Logs will be available in /AppData/Library/Caches/Logs

3. File name will be in the format <APPIdentifier> <TimeStamp>

**8. Log message format**

Log format is like <UTC timeStamp>|<LogType>|<Component Name : Version> | <Event>|<Message>

Ex:

2016-01-08 10:24:09.365 | INFO | DemoApp: 1.0.1 | "Demo Button Pressed" | "Demo Page Entered"

9. Adopt Best Practices for Logging

* Provide the right amount of information; no more, no less. Avoid creating clutter.
* Avoid logging messages that the user can’t do anything about.
* Release builds should not include the reason for failed authentications/decryption, just the outcome
* No security info
  + Never include information on type of encryption, key exchange states, authentication protocol (i.e. Diffie or Helman tags can help a hacker to exploit weaknesses in key exchange protocol).
  + Never include username/passwords of connections
  + Never include personal information (a UUID should be ok, to be validated with privacy officer)
  + Never include credit card info (only anonymized, i.e. last 4 digits with expiration data, or everything but last 4 digits excl. expiration data; never including CVC)

1. References
   * 1. <https://github.com/CocoaLumberjack/CocoaLumberjack/blob/master/Documentation/GettingStarted.md>
     2. [https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/LoggingErrorsAndWarnings.html#//apple\_ref/doc/uid/10000172i-SW8-SW7](https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/LoggingErrorsAndWarnings.html%2523//apple_ref/doc/uid/10000172i-SW8-SW7)
2. **App Identity**

The App identity feature shall provide an API to get the app release status: development, test, acceptance, production.

The App identity feature shall obtain the technical app name, app version and app release status automatically from the build application build process. developer needs to be crerate appIdentityConfig.plist and add micrositeId, sector & AppState key value:

"micrositeId" : "12345",  
 "sector" : "B2C",  
 "state" : "DEVELOPMENT"

Remaining AppVersion and AppName will be written from info.plist

**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: [[AppDelegate sharedAppDelegate].objAIAppInfra.appIdentity getAppName];

**->** Get Localized App Name

ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.appIdentity getLocalizedAppName];

**-> Get App Version**

ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.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 ([[AppDelegate sharedAppDelegate].objAIAppInfra.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: [[AppDelegate sharedAppDelegate].objAIAppInfra.appIdentity getMicrositeId];

**-> Get Sector**

ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.appIdentity getSector];

1. **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

**Use cases**

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

**-> Get Home Country Code**

ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.serviceDiscovey getHomeCountryCodeWithCompletion:^(NSString \*countryCode, NSString \*sourceType, NSError \*error) {

if (error) {

// Error Display Part

}else {

// Country Code with Source Type Display Part

}

}];

**-> Set Home Country Code**

ex: NSLocale \*countryLocale = [NSLocale currentLocale];

NSString \*countryCode = [countryLocale objectForKey:NSLocaleCountryCode];

[[AppDelegate sharedAppDelegate].objAIAppInfra.serviceDiscovey setHomeCountry:countryCode];

**-> Get ServiceUrl With Language Preference**

ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.serviceDiscovey getServiceUrlWithLanguagePreference:kServiceId :^(NSString \*serviceURL,NSError \*error) {

NSLog(@"getServiceUrlWithLanguagePreference:%@",serviceURL);

NSLog(@"getServiceUrlWithLanguagePreference.error:%@",error.localizedDescription);

}];

#define kServiceId @"ugrow.privacy"

**-> Get ServiceUrl With Country Preference**

ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.serviceDiscovey getServiceUrlWithCountryPreference:kServiceId :^(NSString \*serviceURL,NSError \*error) {

NSLog(@"getServiceUrlWithCountryPreference:%@",serviceURL);

NSLog(@"getServiceUrlWithCountryPreference.error:%@",error.localizedDescription);

}];

**-> Get ServiceLocale With Language Preference**

ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.serviceDiscovey getServiceLocaleWithLanguagePreference:kServiceId :^(NSString \*locale,NSError \*error) {

NSLog(@"getServiceLocaleWithLanguagePreference:%@",locale);

NSLog(@"getServiceLocaleWithLanguagePreference.error:%@",error.localizedDescription);

}];

**-> Get ServiceLocale With Country Preference**

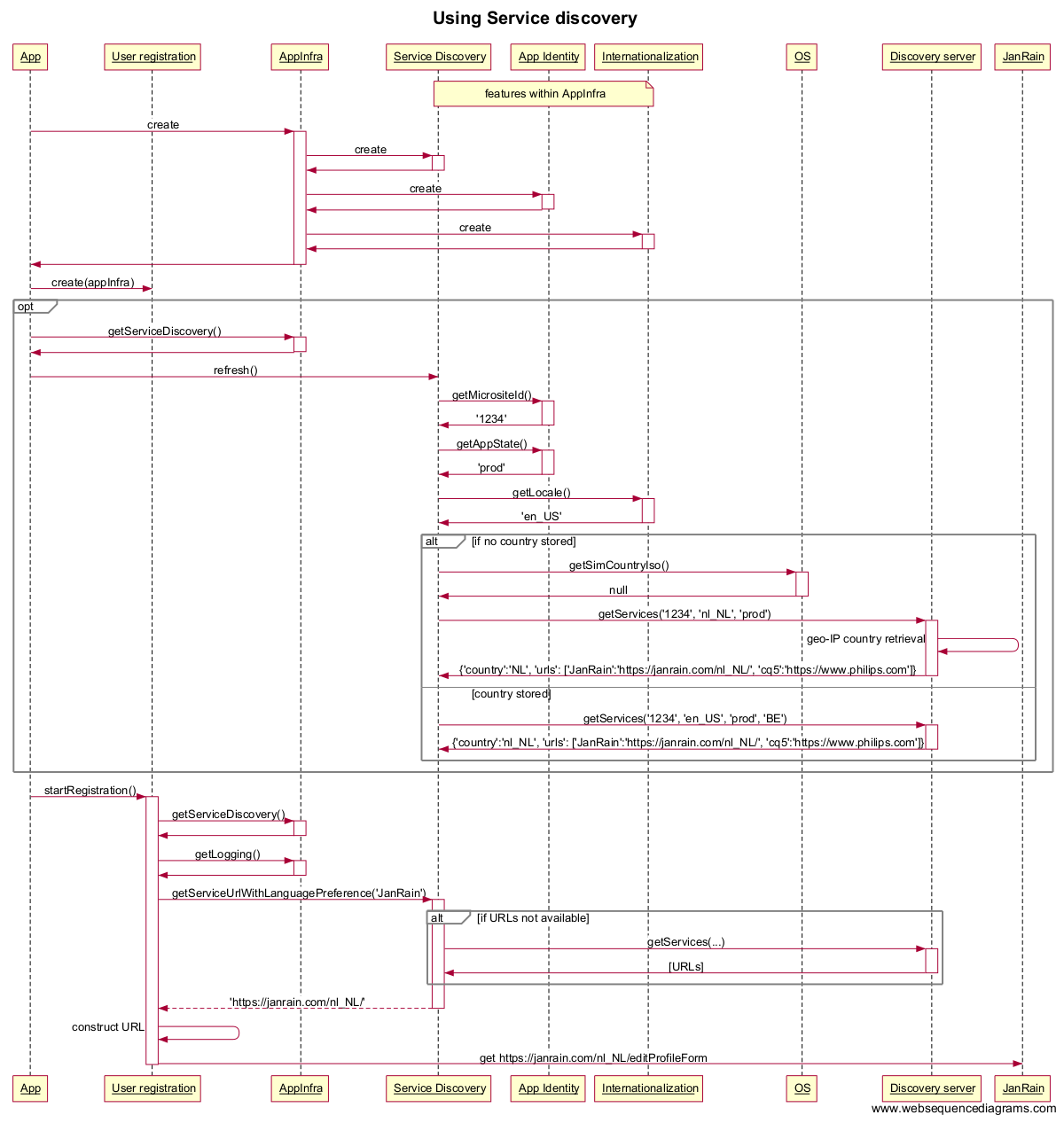
ex: [[AppDelegate sharedAppDelegate].objAIAppInfra.serviceDiscovey getServiceLocaleWithCountryPreference:kServiceId :^(NSString \*locale,NSError \*error) {

NSLog(@"getServiceLocaleWithCountryPreference:%@",locale);

NSLog(@"getServiceLocaleWithCountryPreference.error:%@",error.localizedDescription);

}];

**Flow Diagram:**



**Stat Machine Diagram:**

