|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document History** | | | | |
| **Version** | **Date** | **Author** | **Section** | **Changes** |
| 1.6.0 | 24-04-2017 | Anurag Gautam,  Adarsha Shetty, G Kavya | All | New Release  Features: Tagging, Logging, Secure Storage, App Identity, Internationalization, Service Discovery, Time Sync, MicroApp Configuration, RestClient , A/B Testing & ContentLoader, Language Pack,AppUpdate |

Mobile App Infrastructure library Integration

|  |  |
| --- | --- |
| Author | Anurag Gautam, Adarsha Shetty, G Kavya & Yogesh HS |
| Approved by |  |
| Email Id | [anurag.gautam@philips.com](mailto:anurag.gautam@philips.com) adarsha.shetty@philips.com  [kavya.g.kurpad@philips.com](mailto:kavya.g.kurpad@philips.com)  [yogesh.hs@philips.com](mailto:yogesh.hs@philips.com) |

1. Synopsys & MODULE 3

Synopsis 3

Modules 3

1. INTRODUCTION 4

2. INTEGRATION 4

2.1 Maven repository (Artifactory based) Integration 4

2.2 Library Integration 4

2.3 Library versioning 5

2.4 .Root gradle changes 5

2.5 Proxy dependencies 6

2.6 Configuration Files: 6

3. INITIALIZATION 8

4. Android Manifest Changes 8

4.1 Other User Permissions 8

5. Supporting apps with Over 65K Methods 9

6. Third Party Library used: 9

 AdobeMobileLibrary 9

 Volley 9

7. Service Discovery 22

Include adobe config json file and give target configuration 36

}, 36

## Synopsis

App Infra provides a range of modules that are the basis for any mobile application. App Infra is integrated as one single library in the application. App Infra in itself is not an operational entity but only exists in the context of an application. App Infra ensures that all modules provided by App Infra are linked to each other where needed. All modules together are exposed to the application through a single set of well-defined interfaces. App Infra is not only used by the application but also by all common components that are integrated in the application. As such App Infra can be seen as a basic layer of functionality in the SW stack that is positioned somewhere between the app and the device’s OS. App Infra is not designed to abstract the operating system; App Infra provides additional functionality on top of the operating system making use of the operating system. Some of the App Infra depend on cloud servers to provide the required functionality. In those cases, App Infra abstracts the cloud server such that the users of App Infra are not directly exposed to the typical problems of remote services.

## Modules

* Secure storage module - Secure Storage is used to store secret value in device storage with encrypted way using AES Encryption. It uses key value pair concept to store data inside the apps.
* Service discovery module - 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.
* App tagging module - 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.
* Logging module - Logging is used to maintain the logs which user access of the propositions or common components with page/action name and several other default values such as UTC timestamps, Log type, Component ID, Event and Message.
* Time module - feature provides an APIs to retrieve the UTC server time accurately. It also perform synchronization for every 24hrs and whenever there is a Data and time change.
* Internationalization module - provided APIs to fetch Locale from Android settings.
* App identity module - 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.
* App config module - The app configuration module maintains configuration settings of the app and it’s included common components, in the form of key value pair.
* REST client module - The REST client module simplifies communication with cloud services that use a REST based interface. Which is built upon AFNetworking Library
* A/B test module - The A/B test client module assists in performing A/B tests in the application. The client communicates with server which distributes users over the test experiences. A test is identified by its test name. when offline, the cached data is used if available instead of the default value to ensure a consistent user experience irrespective of network availability.
* Content loader module - This feature has a dedicated purpose of caching the articles from the CQ5 server for the application. The articles could be in the form of text, images, document etc. This library will download all the articles available at the application specified URL and store it internally
* API signing module - 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.
* Language Pack - 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
* AppUpdate - 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 )

# INTRODUCTION

This document provides an overview of integration procedure for Mobile App Infra library in android mobile applications.

# INTEGRATION

There are two ways to integrate “Mobile App Infrastructure” library with any Android app.

* + 1. **Maven repository based**: At compile time, machine has to be connected with Philips network. Do not follow section 2.2
    2. **Library Integration**: If unable to connect with Philips network then include libraries to your root application. Do not follow section 2.4, 2.5

## Maven repository (Artifactory based) Integration

The easiest and preferred way to use these components is using maven.

All dependent libraries should be downloaded from artifactory.

## Library Integration

Need to copy all aar files in libs folder; below are the libraries needed, Please make gradle changes

dependencies {  
 compile fileTree(**dir**: **'libs'**, **include**: [**'\*.jar'**])  
 androidTestCompile **'org.mockito:mockito-core:1.9.5'** androidTestCompile **'com.google.dexmaker:dexmaker:1.2'** androidTestCompile **'com.google.dexmaker:dexmaker-mockito:1.2'** compile **'com.android.support:appcompat-v7:23.4.0'** compile **'adobeMobileLibrary:adobeMobileLibrary:4.9.0'**  compile **'com.android.volley:volley:1.0.0'**}

## Library versioning

Library version can be obtained by using below API

version = objcdp.getVersion()

## .Root gradle changes

buildscript {  
 repositories {  
maven { url **'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/jcenter'** }  
 }  
  
 dependencies {  
 classpath **'com.android.tools.build:gradle:2.2.0'** *// NOTE: Do not place your application dependencies here; they belong  
 // in the individual module build.gradle files* }  
}  
  
allprojects {  
 repositories {  
 maven { url **'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/jcenter'** }  
 maven {  
 url **'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/ext-release-local'** }  
 maven {  
 url **'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/libs-release-local-android'** }  
 maven {  
 url **'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/libs-stage-local-android'** }  
 maven {  
 url **'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/libs-release-local'** }  
 maven { url **'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/libs-snapshot-local-android'** }  
 }  
}  
task clean(**type**: Delete) {  
 delete rootProject.buildDir  
}

## Proxy dependencies

Gradle dependencies can get some network/proxy related issues. In order to fix this issue, we are using below proxy settings in gradle.properties of root folder.

**systemProp.https.proxyHost**=**42.99.164.34  
systemProp.https.proxyPort**=**10015**

We are using this proxy settings locally. But Eindhoven, does not use above proxy settings.

## Configuration Files:

1. **ADBMobileConfig.json**

Keep this json file in Assets folder. 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",

"analytics": {

"referrerTimeout": 5,

"rsids": "philipsmobileappsdev",

// "server" : "localhost:50000",

"server": "philips.112.2o7.net",

"charset": "UTF-8",

"ssl": true,

"offlineEnabled": true,

"lifecycleTimeout": 30,

"batchLimit": 0,

"privacyDefault": "optunknown",

"poi": [

]

},

"target": {

"clientCode": "philipselectronicsne",

"timeout": 5

},

"audienceManager": {

"server": "",

"analyticsForwardingEnabled": false,

"timeout": 5

},

"acquisition": {

"server": "c00.adobe.com",

"appid": ""

}

}

1. **AppConfig.json**

Developer needs to be crerate AppConfig.json and add microsite, sector & AppState key value. Remaining AppVersion and AppName will be written from gradle & AppLocalName will be written manifest file.

1. {  
    **"UserRegistration"**: {  
    **"JanRainConfiguration.RegistrationClientID.Development"**: **"8kaxdrpvkwyr7pnp987amu4aqb4wmnte"**,  
    **"JanRainConfiguration.RegistrationClientID.Testing"**: **"g52bfma28yjbd24hyjcswudwedcmqy7c"**,  
    **"JanRainConfiguration.RegistrationClientID.Evaluation"**: **"f2stykcygm7enbwfw2u9fbg6h6syb8yd"**,  
    **"JanRainConfiguration.RegistrationClientID.Staging"**: **"f2stykcygm7enbwfw2u9fbg6h6syb8yd"**,  
    **"JanRainConfiguration.RegistrationClientID.Production"**: **"9z23k3q8bhqyfwx78aru6bz8zksga54u"**,  
    **"PILConfiguration.MicrositeID"**: **"77000"**,  
    **"PILConfiguration.CampaignID"**: **"CL20150501\_PC\_TB\_COPPA"**,  
    **"PILConfiguration.RegistrationEnvironment"**: **"PRODUCTION"**,  
    **"Flow.EmailVerificationRequired"** : **true**,  
    **"Flow.TermsAndConditionsAcceptanceRequired"** : **true**,  
    **"Flow.MinimumAgeLimit"** : { **"NL"**:12 ,**"GB"**:0,**"default"**: 16},  
    **"SigninProviders.default"**: [**"facebook"**,**"googleplus"**],  
    **"SigninProviders.NL"**: [**"facebook"**,**"googleplus"**]  
    },  
    **"IAP"**: {  
    **"hostport"**: **"www.occ.shop.philips.com"**,  
    **"propositionid"**: **"Tuscany2016"** },  
    **"appinfra"**:{  
    **"appidentity.micrositeId"**: **"77001"**,  
    **"appidentity.sector"**: **"b2c"**,  
    **"appidentity.appState"**: **"production"**,  
    **"appidentity.serviceDiscoveryEnvironment"**: **"production"**,  
    **"restclient.cacheSizeInKB"**: 51200,  
    **"tagging.sensitiveData"**: [**"bundleId"**, **"language"**] ,  
    **"abtest.precache"**:[**"philipsmobileappabtest1content"**,**"philipsmobileappabtest1success"**],  
    **"contentLoader.limitSize"**:100,  
    **"servicediscovery.platformMicrositeId"**:**"77000"**,  
    **"servicediscovery.platformEnvironment"**:**"production"**,  
    **"appconfig.cloudServiceId"**:**"appinfra.appconfigdownload"**,  
    **"timesync.ntp.hosts"**:[**"0.pool.ntp.org"**,**"1.pool.ntp.org"**,**"2.pool.ntp.org"**,**"3.pool.ntp.org"**,**"0.cn.pool.ntp.org"**],  
    **"LANGUAGEPACK.SERVICEID"**:**"appinfra.languagePack"**,

“appUpdate.serviceId”:”appinfra.testing.version”,

“appUpdate.autoRefresh”:true  
 **"logging.releaseConfig"**:{  
 **"fileName"**:**"AppInfraLog"**,  
 **"numberOfFiles"**:5,  
 **"fileSizeInBytes"**:50000,  
 **"logLevel"**:**"Off"**,  
 **"fileLogEnabled"**:**false**,  
 **"consoleLogEnabled"**:**true**,  
 **"componentLevelLogEnabled"**:**false**,  
 **"componentIds"**: [  
 **"DemoAppInfra"**,  
 **"Registration"** ]  
 },  
 **"logging.debugConfig"**:{  
 **"fileName"**:**"AppInfraLog"**,  
 **"numberOfFiles"**:5,  
 **"fileSizeInBytes"**:50000,  
 **"logLevel"**:**"All"**,  
 **"fileLogEnabled"**:**false**,  
 **"consoleLogEnabled"**:**true**,  
 **"componentLevelLogEnabled"**:**false**,  
 **"componentIds"**: [  
 **"DemoAppInfra"**,  
 **"Registration"**,**"component1"** ]  
 }  
  
 }  
  
}

1. proguard-rules-pro

Proguard rules of AppInfra to be included in application proguard rules.

# INITIALIZATION

AppInfra object should be created in the class which extend Application or anywhre from which is can be consumed as dependency injection.

**public class** FrameworkApplication **extends** Application {  
 **public static** AIAppTaggingInterface *mAIAppTaggingInterface*;

**public static** LoggingInterface AILoggingInterface;

**public static** AppInfraInterface *gAppInfra*;

@Override  
 **public void** onCreate() {  
 **super**.onCreate();  
  
***gAppInfra*=new AppInfra.Builder().build(getApplicationContext());**

*mAIAppTaggingInterface* = *gAppInfra*.getTagging().createInstanceForComponent(**"Component name"**,**"Component ID"**);  
AILoggingInterface=*gAppInfra*.getLogging().createInstanceForComponent(“Component name”,”ID”);

}  
  
}

# Android Manifest Changes

<**uses-permission android:name="android.permission.INTERNET"** />  
<**uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"** />  
<**uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"** />

## Other User Permissions

No special Permission required:

# Supporting apps with Over 65K Methods

This is special case if app which has more than 65K methods do follow below link for reference.

<https://developer.android.com/tools/building/multidex.html>

According to this do following changes in gradle and application class

In Gradle file:

android {  
    compileSdkVersion 21  
    buildToolsVersion "21.1.0"  
  
    defaultConfig {  
        ...  
        minSdkVersion 14  
        targetSdkVersion 21  
        ...  
  
        // Enabling multidex support.  
        multiDexEnabled true  
    }  
    ...  
}  
  
dependencies {  
  compile 'com.android.support:multidex:1.0.1'  
}

In Application Class:

@Override  
public void onCreate() {  
   MultiDex.install(this);

  Super.onCreate();

}

# Third Party Library used:

# AdobeMobileLibrary

https://marketing.adobe.com/resources/help/en\_US/mobile/android/

# Volley

https://developer.android.com/training/volley/index.html?hl=pt-br

**Module Examples:**

**SecureStorage:**

SecureStorageInterface ssInterface = ***gAppInfra***.getSecureStorage();

SecureStorageInterface.SecureStorageError ssError = **new** SecureStorageInterface.SecureStorageError();

**boolean** result = ssInterface.storeValueForKey(“key”, “Value”, ssError);

**if**(**null**==sseStore.getErrorCode() && result==true)  
{  
 // success  
}**else**{  
Toast.makeText(context,sseStore.getErrorCode().toString(),Toast.**LENGTH\_SHORT**).show();  
}

SecureStorageInterface.SecureStorageError ssError = **new** SecureStorageInterface.SecureStorageError();

String decryptedData= ssInterface.fetchValueForKey(“key”,ssError);  
**if**(**null**==sse.getErrorCode() && null!=decryptedData)  
{ context  
 // success   
}**else**{  
 Toast.makeText(context,sse.getErrorCode().toString(),Toast.**LENGTH\_SHORT**).show();  
}

**boolean** result = ssInterface.removeValueForKey(“key”);

**New API added 17.2:**

1. String getDeviceCapability ();

**Checking whether device is jailbroken**

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

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

1. Boolean deviceHasPasscode ();

This method checks whether the device is secured by PIN, pattern or password or a SIM card is currently locked.

**Api Signing:**

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.

To integrate api signing have to follow below steps:

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

*/\*\*  
 \* Create an API signer instance according to HSDP specification  
 \*  
 \** ***@param sharedKey*** *Key shared with server identifying the signing key  
 \** ***@param hexSecretKey*** *secret API signing key formatted as a 128byte hex string  
 \*/*HSDPPHSApiSigning hsdpPhsApiSigning=new HSDPPHSApiSigning (String sharedKey, String hexSecretKey) ;

* And call createSignature () with help of instance and pass appropriate value of RequestMethod,QueryString,Headers, DhpUrl and Requestbody to parameter.

Map<String, String> headers = **new** LinkedHashMap<String, String>();  
headers.put(**"SignedDate"**,**"2016-11-09T13:31:13.492+0000"**);

hsdpPhsApiSigning.createSignature(**"POST"**,**"applicationName=uGrow"**,headers,**"/authentication/login/social"**,**null**)

* createSignature () will return generated api signed key in String type.

String result = **hsdpphsApiSigning**.createSignature(**"POST"**,**"applicationName=uGrow"**,headers,**"/authentication/login/social"**,**null**).trim();

**AppTagging:**

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.

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" : ""

}

}

AIAppTaggingInterface mAIAppTaggingInterface = ***gAppInfra*** .getTagging().createInstanceForComponent("Component name","Component ID");

mAIAppTaggingInterface.setPreviousPage("SomeXpreviousPage");

mAIAppTaggingInterface.trackPageWithInfo("AppTaggingDemoPage", keyValuePair);

AIAppTaggingInterface mAIAppTaggingInterface = ***gAppInfra*** .getTagging().createInstanceForComponent("Component name","Component ID");

mAIAppTaggingInterface.setPreviousPage("SomeXpreviousPage");

mAIAppTaggingInterface.trackPageWithInfo("AppTaggingDemoPage", keyValuePair);

**APIS are below:**

1. **public void** setPrivacyConsent(PrivacyStatus privacyStatus);

setting the privacy status based on user consent

**2)public void** setPreviousPage(String previousPage);  
  
**3) public** PrivacyStatus getPrivacyConsent();

By using getPrivacyConsent method we can get the privacy, which has been set

Privacy Enum states are below :

enum PrivacyStatus {

*OPTIN*, *OPTOUT*, *UNKNOWN*

}

**4) public void** trackPageWithInfo(String pageName, String key, String value);  
  
**5) public void** trackPageWithInfo(String pageName, Map<String, String> paramDict);  
  
**6) public void** trackActionWithInfo(String pageName, String key, String value);  
  
**7) public void** trackActionWithInfo(String pageName, Map<String, String> paramDict);  
  
**8) public void** collectLifecycleInfo(Activity context, Map<String, Object> paramDict);  
  
**9) public void** collectLifecycleInfo(Activity context);  
  
**10) public void** pauseLifecycleInfo();  
  
**11) void** trackVideoStart(String videoName);

This API can be used to track when a video has been started  
  
**12) void** trackVideoEnd(String videoName);

This API can be used to track when a video has been stopped/ended  
  
**13) void** trackSocialSharing(SocialMedium medium, String sharedItem);

This API can be used to track a social share buttons are tapped

SocialMedium is enum value :

enum SocialMedium {

Facebook("facebook"),

Twitter("twitter"),

Mail("mail"),

AirDrop("airdrop");

private final String socialMedium;

SocialMedium(String socialMedium) {

this.socialMedium = socialMedium;

}

public String toString() {

return this.socialMedium;

}

}

**14) void** trackLinkExternal(String url);

This API can be used to track external link opening

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

**15) void** trackFileDownload(String filename);

This API can be used to track file downloading

@param: filename: downloading file name

**16) void** trackTimedActionStart(String actionStart);

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

String value that denotes the action name to track. (**Note**: **Same “action name” must be used in track end API as well**)

@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.

**17) void** trackTimedActionEnd(String actionEnd);

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

String pointer denotes the action name to finish tracking. . (**Note**: **Same “action name” must be used which was used in ‘trackTimedActionStart’ API**

**18) void** setPrivacyConsentForSensitiveData(**boolean** valueContent);

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

"tagging.sensitiveData": ["bundleId", "language"] ,

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

language,

bundleId,

timestamp,

UTCTimestamp,

appsId

**19) boolean** getPrivacyConsentForSensitiveData();

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

20) String getTrackingIdentifier()

Returns the tracking identifier .

**New API added in 17.2:**

21) void registerTaggingData(BroadcastReceiver receiver);

Register for Tagging Data.

21) void unregisterTaggingData(BroadcastReceiver receiver);

Unregister for the Tagging Data.

To provide context information to a crashlog, Apteligent provides the ability to maintain a user activity flow.

In AppInfra, Tagging is used to send a user tracking info via the Adobe SDK to Catalyst.

By hooking Apteligent SDK user flow API into Tagging, the user tracking info can also be send to Apteligent without requiring any proposition code changes.

We have created a LocalBroadCastManger for which the proposition has to register on this receiver to listen to the tagging events.

Sample Demo:

private BroadcastReceiver rec = new BroadcastReceiver() {

@Override

public void onReceive(Context context, Intent intent) {

if (intent! = null) {

if(intent.getAction() == AppTagging.ACTION\_TAGGING\_DATA) {

Log.d("AppInfra APP", "BroadcastReceiver() {...}.onReceive()");

Map textExtra = (Map) intent.getSerializableExtra(AppTagging.EXTRA\_TAGGING\_DATA);

Log.d("APPINFRA-TAGGING”, textExtra.toString());

Crittercism.leaveBreadcrumb(textExtra.toString());

Toast.makeText(getApplicationContext(),

textExtra.toString(), Toast.LENGTH\_LONG).show();

}

}

}

};

// registering for the receiver:

taggingInterface.registerTaggingData (rec);

// Unregister the receiver:

taggingInterface.unregisterTaggingData(rec);

**Logging: (Changed in 17.2)**

loggingInterface.enableConsoleLog(true); is removed

loggingInterface.enableFileLog(true); is removed .

logging.properties file is removed.

Logging update:

Logging properties are migrated from logging.properties file to appconfig.json file.

Proposition needs to add below mentioned key and value under appinfra group. However if this key is not present in appconfig. json, appinfra Logging will pick values from logging.properties 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"**:**true**,  
 **"componentLevelLogEnabled"**:**false**,  
 **"componentIds"**: [  
 **"DemoAppInfra"**,  
 **"Registration"** ]  
 },  
 **"logging.debugConfig"**:{  
 **"fileName"**:**"AppInfraLog"**,  
 **"numberOfFiles"**:5,  
 **"fileSizeInBytes"**:50000,  
 **"logLevel"**:**"All"**,  
 **"fileLogEnabled"**:**false**,  
 **"consoleLogEnabled"**:**true**,  
 **"componentLevelLogEnabled"**:**false**,  
 **"componentIds"**: [  
 **"DemoAppInfra"**,  
 **"Registration"**,**"component1"** ]  
 }

It is recommended to add these new fields in appconfig.json and delete logging.properties file under asset.

Key Details:

1. **logLevel**

Possible Value: “Off”, “Error”, ”Warn”, “Info”, “Debug”, “Verbose” & “All”

If value is set to “Off” then NO log will be printed and rest allkeys will be ignored.

1. **consoleLogEnabled**

Possible Value: true & false

Enables console log if **logLevel** is NOT “Off”

3. **fileLogEnabled**

Possible Value: true & false

Enables file log if **logLevel** is NOT “Off”

1. **filename**

Possible Value: Any String

Generate file log with this name e.g. **AppInfraLog** under proposition root package in device internal memory.

This file can be accessed in terminal by adb shell command:

**Note: Mac users need to give platform-tools path for adb to work:**

**export PATH=${PATH}:/Users/philips/Library/Android/sdk/platform-tools**

>adb shell

$run-as <Proposition package name>

<Proposition package name>$ cd app\_AppInfraLogs

<Proposition package name/ AppInfraLogs >$ cat **AppInfraLog0**

**[2017-04-14 06:41:58.420+0000][ail 1.5.5][INFO][Logger created][NA]**

**[2017-04-14 06:58:01.359+0000][ail 1.5.5][ERROR][tag][message]**

(it is applicable if fileLogEnabled is set true)

1. **fileSizeInBytes**

Possible Value: Integer e.g. **2097152** for 2MB

It demotes maximum size of single file, once a file reached this limit a next new file is generated e.g. **AppInfraLog0 -> AppInfraLog1**

(it is applicable if fileLogEnabled is set true)

1. **numberOfFiles**

Possible Value: Integer e.g. **5 It generates 5 files AppInfraLog0, AppInfraLog1, AppInfraLog2, AppInfraLog3, AppInfraLog4**

Once the last file reaches maximum limit, log start overwriting first file.

(it is applicable if fileLogEnabled is set true)

1. **componentLevelLogEnabled**

Possible Value: true & false

If it is enabled then logs will be filtered out based on list of components mentioned in **componentIds**

1. **componentIds**

Possible Value: ArrayList of String eg [**"DemoAppInfra"**,**"Registration"**]

(it is applicable if componentLevelLogEnabled is set true)

here list of componentID should be mentioned for which log is required. While creating log instance in code:

*gAppInfra*.getLogging().createInstanceForComponent(ComponentId, ComponentVersion);

This same ComponentId must be mentioned here.

API

**public** LoggingInterface createInstanceForComponent(String componentId, String componentVersion);

**public void** log(LogLevel level, String eventId, String message);

Log Level can be **"ERROR"**,**"WARNING"**,**"INFO"**,**"DEBUG"**,**"VERBOSE"  
\\**

e.g.

LoggingInterface loggingInterface= ***AppInfra***.getLogging().createInstanceForComponent(“Component name”,” Component ID”);

**AILoggingInterface**.log(LoggingInterface.LogLevel.VERBOSE, ”TAG”, ”Message”);

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 now (or if they want to live dangerously ensure that no sensitive data ends up in the file log).

Config.*setDebugLogging*(**false**); // to be added in application class to disable adobemobile logs

**Internationalization:**

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

APIS:

public String getUILocaleString();

This API will return the best matched locale supported locale by the Application.

Locale resolution is dependent on the android OS since from Android N it will pick the next preferred language.

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.

Values will be picked from the AppConfiguration file .

For testing purpose , AppIdentity values can be configured dynamically using setPropertyForKey API from AppConfiguration.

If AppState/ServiceDiscoveryEnvironment is set to PRODUCTION , it cannot be modified by AppConFiguration.

*{  
 "micrositeId" : "12345",  
 "sector" : "B2C",   
 "AppState" : "DEVELOPMENT"  
}*

Remaining AppVersion and AppName will be written from gradle & AppLocalName will be picked from gradle file.

**API:**

*public String getAppName();*

returns technical App name.

*public String getAppVersion();*

Validates and returns App version. Throws Exception if not in proper format.

*public String getAppState();*

Validated and returns App state (development, test, acceptance, production).Throws Exception if appstate is other the mentioned states.

*public String getAppLocalizedNAme();*

returns app localized commercial app name.

*public String getMicrositeId();*

Validates and returns micrositeID. Throws Exception if not in proper format

*public String getSector();*

Validates and returns sector. Throws Exception if Sector is other than the mentioned states.

Note :

Microsite ID should be [a-zA-Z0-9] +

Sectors should be from B2C, B2B\_HC, corporate, B2B\_LI (Comparison should be case insensitive)

App State should be from TEST, DEVELOPMENT, STAGING, ACCEPTANCE, PRODUCTION (Comparison should be case insensitive)

Service Discovery Environment should be from TEST, DEVELOPMENT, STAGING, ACCEPTANCE, PRODUCTION (Comparison should be case insensitive)

App version should be  [0-9]+\.[0-9]+\.[0-9]+([\_(-].\*)?

IllegalArgumentException will be thrown in case of not invalid values.

**Service Discovery:**

# 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.state": "<app state>",

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

"servicediscovery.platformMicrositeId":<platform micrositeID>,

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

Keep **appIdentityConfig.json** file in the app bundle to set micrositeId, sector, state and serviceDiscoveryEnvironment. All 6 fields are mandatory (refer App Identity for more details).

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. These entries need to be added in AppConfig json under appinfra group

"servicediscovery.platformMicrositeId":<micrositeID>,

"servicediscovery.platformEnvironment":"<environment>"

Note: 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")](%22https://delete.delete%22)) for that serviceId

In AppConfig.json if set **"servicediscovery.propositionEnabled"**:**true**, means

Proposition is Enabled, by default it’s set to be “true”. And If not set (key is not added) or set non boolean value like numeric or string in that case Url **Downloading from both proposition microsite id and platform microsite id.**

If set false **“servicediscovery.propositionEnabled"**: **false** means proposition is disabled or platform is enabled so **Downloading from platform microsite id and should return the URL's for Service 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 it’s 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.

If the entered serviceId is not present in the proposition it will fall back to platform. Proposition response holds the highest priority.

**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.

**API:**

1. void getHomeCountry(OnGetHomeCountryListener listener)

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. The listener, OnGetHomeCountryListener will get the results back.

2.    String getHomeCountry()

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

Observing home country change

  We can observe to home country change by registering to broadcast receiver, find below code for reference

**mServiceDiscoveryInterface**.registerOnHomeCountrySet(**receiver**)// Pass instance of broadcast receiver which you listen to.

New county code will be sent along with the intent for the action being “ail.servicediscovery.homecountryChanged” and through extras you can get country code with key “ail.servicediscovery.homeCountry”

1. void getServiceUrlWithLanguagePreference (String serviceId, OnGetServiceUrlListener listener)

getServiceUrlWithLanguagePreference will get the URL’s from the response filtering with given ServiceID. The listener, OnGetServiceUrlListener will get the results back.

1. void getServiceUrlWithCountryPreference(String serviceId, OnGetServiceUrlListener listener)

getServiceUrlWithCountryPreference will get the URL’s from the response filtering with given ServiceID. The listener, OnGetServiceUrlListener will get the results back.

1. void getServiceLocaleWithLanguagePreference(String serviceId, OnGetServiceLocaleListener listener)

getServiceLocaleWithLanguagePreference will get the URL’s from the response filtering with given ServiceID. The listener, OnGetServiceLocaleListener will get the results back.

1. void getServiceLocaleWithCountryPreference(String serviceId, OnGetServiceLocaleListener listener)

getServiceLocaleWithCountryPreference will get the URL’s from the response filtering with given ServiceID. The listener, OnGetServiceLocaleListener will get the results back.

1. void getServicesWithLanguagePreference(String serviceIds, OnGetServicesListener listener)

getServicesWithLanguagePreference will get the URL’s from the response filtering with given ServiceID. The listener, OnGetServicesListener will get the results back.

1. void getServicesWithCountryPreference(String serviceIds, OnGetServicesListener listener);

getServicesWithCountryPreference will get the URL’s from the response filtering with given ServiceID. The listener, OnGetServicesListener will get the results back.

1. void refresh(OnRefreshListener listener)

The refresh to Webservice call happens here. And the results will get back to OnRefreshListener.

**Note: if refresh call fails due to server issue then any subsequent service call within 10 second will be ignored. That is if a service call to service discovery fails then next service call should be made only after 10 seconds.**

1. public String getservice(OnRefreshListener listener)

Call the service discovery hard coded single URL without(first run) country code and fetch the country code and save it in shared preference.

There after this service discovery hard coded single URL is called with ‚country‘ also as a parameter to get complete list of service urls

for given service.

1. public void getServiceUrlWithLanguagePreference(String serviceId, OnGetServiceUrlListener listener, Map<String, String> replacement)

Returns 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.

1. public void getServicesWithLanguagePreference(ArrayList<String> serviceId, OnGetServiceUrlMapListener listener, Map<String, String> replacement);

Returns Hashmap with URL mapped 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.

1. public void getServiceUrlWithCountryPreference(String serviceId, OnGetServiceUrlListener listener, Map<String, String> replacement);

Returns 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.

1. public void getServicesWithCountryPreference(ArrayList<String> serviceId, OnGetServiceUrlMapListener listener, Map<String, String> replacement);

Returns Hashmap with URL mapped 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.

1. public URL applyURLParameters(URL url, Map<String, String> replacement);

This API 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

**Sample :**

Map<String, String> parameters = new HashMap<>();

parameters.put("ctn", "HD9740");

parameters.put("sector", "B2C");

parameters.put("catalog", "shavers");

mServiceDiscoveryInterface.getServiceUrlWithCountryPreference(editTextData, mOnGetServiceUrlListener, parameters);

**Internal Document Reference:**

<https://atlas.natlab.research.philips.com/confluence/display/MAIL/Interface+Document>

**Time Sync**

The feature does not maintain and internal real time clock. Rather, the feature determines a time delta between the device local time and the independent accurate time source when the time is synchronized. When the current UTC time is requested, the UTC time is derived from the device local time plus the calculated delta. The feature automatically synchronizes the time at first instantiation, when a large local time change is detected, and every 24 hours.

To fecth the network time required NTP server , this list of Server Pool are fetch from AppConfif.json file , if not able to fetch it will fetch from string arrays from string.xml.

To integrate TimeSync NTP time first have to Copy below line in AppConfig.json.

Server Pool in AppConfig.json:

**"timesync.ntp.hosts"**:[**"0.pool.ntp.org"**,**"1.pool.ntp.org"**,**"2.pool.ntp.org"**,**"3.pool.ntp.org"**,**"0.cn.pool.ntp.org"**]

This is the default NTP server pool list. One can add NTP servers here. It is configurable.

To fetch the NTP network time using SNTP client

**public** String getUTCTime();

To refresh NTP time.

Offset Time= NTP time – Device time

**public void** refreshTime ();

Time sync is used by Logging and Tagging feature

Note : Internet(Private Network) is required atleast for the first time to sync with the NTP server.It doesnt work if you are behind proxy .

**App Configuration:**

This component provides API to get and set the configurations for all micro apps. Configuration file will be in JSON format which will be placed in the assets of the vertical applications for the first time. Later it will be stored in device memory using secure storage.

**Note: App configuration keys (Group and keys both) are case insensitive**

Sample Json file:

{  
 "UR": {  
  
 "Development": "ad7nn99y2mv5berw5jxewzagazafbyhu",  
 "Testing": "xru56jcnu3rpf8q7cgnkr7xtf9sh8pp7",  
 "Evaluation": "4r36zdbeycca933nufcknn2hnpsz6gxu",  
 "Staging": "f2stykcygm7enbwfw2u9fbg6h6syb8yd",  
 "Production": "mz6tg5rqrg4hjj3wfxfd92kjapsrdhy3"  
  
 },  
 "AI": {  
 "MicrositeID": 7750,  
 "RegistrationEnvironment": "Staging",  
 "NL": ["googleplus", "facebook" ],  
 "US": ["facebook","googleplus" ],  
 "EE": [123,234 ]  
 }  
}

API’s:

1. Object getPropertyForKey(String key, String group, ConfigError configError) throws IllegalArgumentException;

This method is used to fetch the value from three configuration. The preference order of fetching value:

1. Dynamic Config (device)

2. Cloud Config (server)

3. Static Config (AppConfig.json)

Requested key is searched in Dynamic Config first, if not found then searched in Cloud Config and if not found then finally searched in Static Configuration.

If requested key or group not found in Static Config then KeyNotExists or GroupNotExists

error is set in ConfigError object.

This implementation takes care of migration of old dynamic Configuration to new.

User has to pass the Coco Name, Key which they are interested in and ConfigError as OUT parameter. The return value will the value for the key mapped.

This method throws IllegalArgumentException if key/groupname are null or improper format.

Example usage of this method:

ConfigInterface.ConfigError configError = new ConfigInterface.ConfigError ();

Object object = mConfigInterface.getPropertyForKey (“UR”, “Development”, configError);

2) boolean setPropertyForKey(String key, String group, Object object, ConfigError configError) throws IllegalArgumentException;

This method is used to set values and update values to the dynamic configuration file. It sets only modified/new value to dynamic config which is persisted to device, dynamic config DOES NOT contain unchanged value from static config (AppConfig.json). User has to pass the Coco Name, Key which they are interested / in case if they want to add new key, Value – it can be any primitive data type, array list of String and Integer and ConfigError as OUT parameter.

User can also add new CoCo and corresponding key-values in the config file.

The return value is true/ false.

This method throws IllegalArgumentException if key/groupname/values are null or improper format.

Example usage of this method:

ConfigInterface.ConfigError configError = new ConfigInterface.ConfigError ();

boolean success = mConfigInterface.setPropertyForKey (“UR”, “newKey”, “newvalue”, configError);

**ConfigError can be: {Invalid Key, GroupNotExists, KeyNotExists, ErrorKeyExists, Fatal Error, DeviceStoreError, NoDataFoundForKey}**

Note:

* For key’s in both app identity and app config:
  + [a-zA-Z0-9\_.-]+
* For values in config:
  + .\* (no check)
* For app identify values:
  + micrositeID, sector: [a-zA-Z0-9]+
  + appstate: enum
  + app name: \* (no check)
  + app version: [a-zA-Z0-9\_. /-]+
* all CocoName/Key/Values is case sensitive
* **To delete any key, pass the null value for the given key:**

boolean setPropertyForKey(“key”, “group”, null, configError);

3)

Object getDefaultPropertyForKey(String key, String group, AppConfigurationError configError) throws IllegalArgumentException;

This method is used to fetch the value from the configuration file always. User has to pass the Key, Coco Name, which they are interested in and ConfigError as OUT parameter. The return value will the value for the key mapped.

This method throws IllegalArgumentException if key/groupname are null or improper format.

Example usage of this method:

ConfigInterface.ConfigError configError = new ConfigInterface.ConfigError ();

Object object = mConfigInterface.getPropertyForKey (“appidentity.appState”, “appinfra”, configError);

Now from PI16 data type for value is also supported for HashMap:

**HashMap<String,String>**

**HashMap<String,Integer>**

**Note:** Key of HashMap must be String and its corresponding value can be either String or Integer

**Ex:**

Map hashMap= new HashMap<String,String>();

hashMap.put("Key1",new Integer(4));

hashMap.put("Key2",new Integer(5));

ConfigInterface.ConfigError configError = new ConfigInterface.ConfigError ();

boolean success = mConfigInterface.setPropertyForKey (“UR”, “newKey”, hashMap , configError);

4)

void refreshCloudConfig (OnRefreshListener onRefreshListener)

**interface** OnRefreshListener {  
 **enum** REFRESH\_RESULT {REFRESHED***\_FROM\_SERVER***, ***NO\_REFRESH\_REQUIRED***, ***REFRESHED\_FAILED***};  
 **void** onError(AppConfigurationError.AppConfigErrorEnum error, String message);  
 **void** onSuccess(REFRESH\_RESULT result);  
}

This method is used to fetch Cloud Configuration. This method is called after initialization of AppInfra however verticals/proposition can call this method anytime thereafter.

5)

*/\*\*  
 \* This method clears dynamic config and downloaded cloud config  
 \* from both local cache and persistence memory.  
 \*/*void resetConfig();

This method clears dynamic config as well as cloud config.

ex usage :

mConfigInterface.resetConfig();

**Demo App link:**

<https://atlas.natlab.research.philips.com/bitbucket/projects/MAIL/repos/app-infra_android/browse/Source/DemoApp/app/src/main/java/com/philips/platform/appinfra/demo/AppConfigurationActivity.java?at=Develop>

**REST Client:**

Volley library is customized for Rest Client and modified class are:

* StringRequest
* JsonObjectRequest
* ImageRequest
* DiskBasedCache

Component and vertical should import above classes from AppInfra pakage:

Eg. **import** com.philips.platform.appinfra.rest.request.StringRequest;

We have 2 different constructor for each of the request class.

For example in StringRequest we have:

1. public StringRequest (int method, String url, Response. Listener<String> listener, Response.ErrorListener errorListener, Map<String, String> header,  
   TokenProviderInterface tokenProviderInterface)

**This constructor should be used when requesting with url. It accepts**

**method**: Request.Method.GET/ Request.Method.POST / Request.Method.PUT/ Request.Method.Delete

**url**: url

**Response.Listener**: success callback

**Response.ErrorListener**: error callback

**header**: custom header to be passed

**tokenProvider**: pass the token provider interface.

Propositions has to implement the TokenProviderInterface to pass the access token and authentication type. Below is the sample.

TokenProviderInterface provider = new TokenProviderInterface () {

@Override

public Token getToken() {

return new Token () {

@Override

public TokenType getTokenType () {

return TokenType.OAUTH2;

}

@Override

public String getTokenValue () {

return accessToken;

}

};

}

};

1. public StringRequest(int method, String serviceID,ServiceIDUrlFormatting.SERVICEPREFERENCE pref,  
   String urlExtension, Response.Listener<String> listener, Response.ErrorListener errorListener)

**This constructor be used when sending request with Service ID.It accepts:**

**method**: Request.Method.GET/ Request.Method.POST / Request.Method.PUT/ Request.Method.Delete

**ServiceId**: pass the serviceId (ex: userreg.janrain.api)

**ServiceIdpreferenceUrl**: ServiceIDUrlFormatting.SERVICEPREFERENCE.BYLANGUAGE/

ServiceIDUrlFormatting.SERVICEPREFERENCE.BYCOUNTRY

**urlExtension**: string/parameters to be appended to url

**Response.Listener**: success callback.

**Response.ErrorListener**: error callback

Apart from StringRequest/JsonObjectRequest/ImageRequest, we have **GsonCustomRequest** which can used to send any type of request.

**Note: http calls are deprecated in this REST Client, use https calls only**

If http url is passed to any of above requests then **HttpForbiddenException** is thrown.

Enable/Disable Caching

By Default Volley enables Cache, to disable it component need to call setShouldCache method on StringRequest, JsonObjectRequest, ImageRequest & Custom Request.

Eg. StringRequest.setShouldCache(false); // set false to disable cache

Encrypt/Decrypt Cache:

Encryption and Decryption is implemented in Appinfra customised **DiskBasedCache** class.

Cache size is taken from App Configuration json file:

**"appinfra"**: {  
 **"appidentity.micrositeId"** : **"77000"**,  
 **"appidentity.sector"** : **"b2c"**,  
 **"appidentity.appState"** : **"Staging"**,  
 **"appidentity.serviceDiscoveryEnvironment"** : **"Production"**,  
 **"restclient.cacheSizeInKB": 1024**}

Public API:

**public RequestQueue getRequestQueue(); // returns volley queue**

**RestInterface mRestInterface** = AppInfraApplication.*gAppInfra*.getRestClient();

**mRestInterface**.getRequestQueue().add(mStringRequest);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public** RequestQueue getRequestQueue() {  
 **if** (**mRequestQueue** == **null**) {  
**mAppConfigurationInterface** = **mAppInfra**.getConfigInterface();  
 AppConfigurationInterface.AppConfigurationError configError = **new** AppConfigurationInterface.AppConfigurationError();  
 Integer cacheSizeinKB = (Integer)**mAppConfigurationInterface**.getPropertyForKey(**"restclient.cacheSizeInKB"**,**"appinfra"**,configError);  
 **if**(cacheSizeinKB==**null** ) {  
 cacheSizeinKB = 1024; *// default fall back* }  
 Cache cache = **new** DiskBasedCache(getCacheDir(), cacheSizeinKB, **mAppInfra**); *//  
// Set up the network to use HttpURLConnection as the HTTP client.* Network network = getNetwork();  
 **mRequestQueue** = **new** RequestQueue(cache,network);  
 **mRequestQueue**.start();  
 }  
 **return mRequestQueue**;  
 }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Demo App link:**

<https://atlas.natlab.research.philips.com/bitbucket/projects/MAIL/repos/app-infra_android/browse/Source/DemoApp/app/src/main/java/com/philips/platform/appinfra/demo/RestClientActivity.java?at=refs%2Fheads%2FDevelop>

**Internet Check:**

Internet Check API is to facilitate the Internet Check in AppInfra module and other apps or vertical application.

The Internet Check have 2 type of API’S. One for check the Internet connection is available or not and another one for type of network Information or type of connection like Wifi or Mobile Data.

* getNetworkReachabilityStatus(): It will return enum NetWorkTypes values like Wifi or Mobile Data or No\_NetWork. If device is connected to Wifi network then this API will return Wifi , similarly if connected to any GSM or CDMA mobile data it will return Mobile Data and If no network available or device is not connected it will return No Network.

To use or call this API follow below guide line.

1. Create Instance of AppInfra or If you have already AppInfra instance use that only.

**private** AppInfra mAppInfra = **new** AppInfra.Builder().build(getApplicationContext());

1. Call getRestClient() followed by getNetworkReachabilityStatus ().

mAppInfra.getRestClient().getNetworkReachabilityStatus();

Example:

NetWorkType networkInfo= mAppInfra.getRestClient().getNetworkReachabilityStatus ();

* isInternetReachable() : It will return boolean value Internet is connected or not. To call this

API follow below guide line.

1. Create Instance of AppInfra or If you have already AppInfra instance use that only.

**private** AppInfra mAppInfra = **new** AppInfra.Builder().build(getApplicationContext());

1. Call getRestClient() followed by isInternetReachable().

mAppInfra.getRestClient().isInternetReachable();

Example:

boolean isNetworkAvailable= mAppInfra.getRestClient().isInternetReachable();

**A/B Testing:**

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 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.

API’s:

1)void updateCache(OnRefreshListener listener);

 This API downloads the experience value from the server. Test names should be mentioned in the Appconfiguration. It accepts only ArrayList of Strings.

"abtest.precache": ["philipsmobileappabtest1content","philipsmobileappabtest1success"]

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

This method gives error/ success callback based on the response.

Error call back

2) CACHESTATUSVALUES getCacheStatus ();

This method returns the status of the cached experiences for the configured list of tests. At initialization of the module, the status is either NO\_TESTS\_DEFINED, NO\_CACHED\_EXPERIENCES, or EXPERIENCES\_NOT\_UPDATED.

An updateCache () may change the state to EXPERIENCES\_PARTIALLY\_UPDATED or EXPERIENCES\_UPDATED.

Cache Status can be any one of these:

NO\_TESTS\_DEFINED, NO\_CACHED\_EXPERIENCES, EXPERIENCES\_NOT\_UPDATED,

EXPERIENCES\_PARTIALLY\_UPDATED, EXPERIENCES\_UPDATED

3) String getTestValue (String testName, String defaultValue, UPDATETYPES updateType, Map<String, Object> parameters);

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

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

Parameters:

testName: TestName for which the testValue is needed.

defaultValue: default value to be returned .

updateTypes: updateType can be App restart and App Update .

**Note :**

**If testname is Apprestart type - it will be only stored in cache.**

**If testname is Appupdate type – it will be stored in both cache and disk.**

**Include adobe config json file and give target configuration**

"target" : {

"clientCode" : "philipselectronicsne",

"timeout" : 15

},

**ContentLoader:**

Content loader downloads data from CQ5 servers. The data is localized; the URL shall be provided via Service Discovery. The data represents information that can be shown to the user. Practical examples are: uGrow articles describing how to feed your baby, or male grooming tips on how to shave. An article consists of meta data, text, and references to images; also called article summary. These articles are stored in a CQ5 database and CQ5 provides one location where a list of articles can be downloaded.

**Usage:**

**Creating a content loader:**

public ContentLoader(Context context, String serviceId, int maxAgeInHours, Class<Content> contentClassType, String contentType, AppInfraInterface appInfra, int *contentLoaderLimitOptional*)

**context:** context

**serviceId:** Id of the service discovery

**maxAgeInHours :** maximum age of the content, a refresh is recommended if cached content is older

**contentClassType :** type of the content class

**contentType :** name of the content as given in the server JSON structure

**appinfra :** appinfra context.

**contentLoaderLimitOptional**: Given value will override the limit given in appconfig.json "*contentLoader.limitSize*" attribute. Give 0 if you don’t want to override appconfig.json "*contentLoader.limitSize*" attribute.

**Content class type should confirms to ContentInterface .**

**For ex :**

ContentLoader mContentLoader = new ContentLoader(getApplicationContext(),”userreg.janrain.api”, 1, ContentArticle.class,”articles”, AppInfraApplication.*gAppInfra, contentLoaderLimitOptional*);

ContentArticle.class is the article model class AppInfra gives that confirms to ContentInterface. For other model types (Beard Style and Assets), propositions have to provide their own model classes that confirms to ContentInterface.

maxAgeInHours is the expiry time for the saved contents.

Download limit of the article can be configured from the Config.json and fetch the value using the AppConfiguration API’s. Below is the sample Config in json: This value will be fetched if contentLoaderLimitOptional is set to 0 else contentLoaderLimitOptional will be the limit.

"appinfra":{

"contentLoader.limitSize":100

}

**Note: Download limit should be greater than the number of contents if its BeardStyle type**

**APIs:**

**Downloading contents:**

void refresh(OnRefreshListener refreshListener);

The downloaded contents will be persistently stored. Propositions needs to check the status of the content loader via getStatus api and needs to call refresh if cached data is outdated.

**Checking Status**

STATE getStatus();

**Getting downloaded contents from the Database :**

**void getAllContent(OnResultListener<String> listener) –** returns the list of available contentIds in success call back

**void getContentById(String id, OnResultListener<Content> listener)** - Returns Content object for the given id in success call back

**void getContentById(String[] ids, OnResultListener<Content> listener)** - Returns list of Content objects for the given array of ids in success call back

**void getContentByTag(String tagID, OnResultListener<Content> listener) -** Returns list of Content objects that have the given tag ID set in success call back

**void getContentByTag(String[] tagIDs, OPERATOR andOr, OnResultListener<Content> listener)** - Returns list of Content objects that have the at least one (OR) or all (AND) given tag IDs set in success call back.

**Removing content loader data :**

void clearCache();

**Language Pack**

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.

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).

*/\*\*  
 \* download language pack overview file.  
 \* If should be called everytime when app is launched  
 \*  
 \** ***@param refreshListener*** *asynchronous callback reporting result of refresh eg {LoadedFromLocalCache, RefreshedFromServer, NoRefreshRequired, RefreshFailed}  
 \*/***void** refresh (OnRefreshListener refreshListener);

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

*/\*\*   
 \* It activates device matching locale from downloaded overview file   
 \* Calling activate will return path of Language pack through call back listener   
 \** ***@param onActivateListener*** *asynchronous callback reporting result of activate  
 \*/***void** activate(OnActivateListener onActivateListener);

Activate API activates stored language pack and returns path through call back listener.

Note:

Refresh and Activate is suggested to be invoked during application launch

Do not include key ail\_locale in strings.xml , as we are depending on app supported locale supported by Philips business.

**Locale Resolution**

App has to specify supported locales in below format in app build.gradle for specific config like below example

defaultConfig {

resConfigs **"en"**, **"nl"**, **"en-rGB"**, **"zh-rCN"**

}

This list indicates that application supports only mentioned locales under this configuration.

Note:

1. App needs to specify exact combination of Locale which is supported by proposition
2. If app does not provide this configuration internationalization API will pick device setting locale
3. Ensure that this list is modified whenever there is addition or modification of locale list supported by proposition.
4. Currently Internationalization API will support standard set of Locales as supported by platform.
5. Whenever there is modification done in platform supported Locale list App-infra team needs to be informed.

**AppUpdate:**

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](https://confluence.atlas.philips.com/download/attachments/21546155/version.json?version=1&modificationDate=1496206621514&api=v2)

* 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.

**Android API's**

/\*\*  
 \* 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  
 \* if "appUpdate.autoRefresh":true is set in appconfig ,  
 \* appupdate json wil be downloaded when appinfra intialization is done.  
 \* @param refreshListener asynchronous callback reporting result of refresh eg {AppUpdate\_REFRESH\_SUCCESS,  
 \* AppUpdate\_REFRESH\_FAILED}.  
 \*/  
void refresh(OnRefreshListener refreshListener);  
  
  
  
interface OnRefreshListener {  
 /\*\*  
 \* AppUpdate\_REFRESH\_SUCCESS : AppUpdate info Downloaded from Server.  
 \* AppUpdate\_REFRESH\_FAILED : Refresh Failed.  
 \*/  
 enum AIAppUpdateRefreshResult {AppUpdate\_REFRESH\_SUCCESS, AppUpdate\_REFRESH\_FAILED}  
  
 void onError(OnRefreshListener.AIAppUpdateRefreshResult error, String message);  
  
 void onSuccess(OnRefreshListener.AIAppUpdateRefreshResult result);  
}  
  
/\*\*  
 \* This will return true if applicationversion < minimumversion  
 \* true when current application version is less than minimum version  
 \* true when deprecatedVersion is greater than current application version and deprecationDate is crossed.  
 \* @return  
 \*/  
boolean isDeprecated();  
  
/\*\*  
 \* minimumVersion <= applicationVersion <= tobeDeprecated.  
 \* @return true if application is not already deprecated and current version is  
 \* lessthan equal to deprecated version.  
 \*/  
boolean isToBeDeprecated();  
  
/\*\*  
 \* applicationVersion < CurrentVersion.  
 \* @return true if current version is less than the  
 \* latest verion available in the appstore.  
 \*/  
boolean isUpdateAvailable();  
  
/\*\*  
 \* Deprecated Version message string.  
 \* @return  
 \*/  
String getDeprecateMessage();  
  
/\*\*  
 \* To be deprecated message string.  
 \* @return  
 \*/  
String getToBeDeprecatedMessage();  
  
/\*\*  
 \* To be deprecated date.  
 \* @return  
 \*/  
String getToBeDeprecatedDate();  
  
/\*\*  
 \* current version message.  
 \* @return  
 \*/  
String getUpdateMessage();  
  
/\*\*  
 \* This method returns the minimum version  
 \* from the appupdate json.  
 \* @return minimum version.  
 \*/  
String getMinimumVersion();  
  
/\*\*  
 \* This method returns the minimumOS version.  
 \* @return minimumOS version.  
 \*/  
String getMinimumOSverion();

Sample Usage :

AppInfra appinfra = new AppInfra.Builder().build(getApplicationContext());

AppUpdateIntreface appupdateInterface = appinfra.getAppUpdate();

 //refresh

appupdateInterface.refresh(new AppUpdateInterface.OnRefreshListener() {  
 @Override  
 public void onError(AIAppUpdateRefreshResult error, String message) {  
 Toast.makeText(AppUpdateActivity.this, error.toString(), Toast.LENGTH\_LONG).show();  
 }  
  
 @Override  
 public void onSuccess(AIAppUpdateRefreshResult result) {  
 Toast.makeText(AppUpdateActivity.this, result.toString(), Toast.LENGTH\_LONG).show();  
 }  
});

other API's can be called in similar way . for ex :

String minimumOsVersion = appupdateInterface.getMinimumOSverion();