|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document History** | | | | |
| **Version** | **Date** | **Author** | **Section** | **Changes** |
| 1.2.0 | 19-Jul-2015 | Anurag Gautam,  Ashok Kasthuri, G Kavya | All | New Release  Features: Tagging, Logging, Secure Storage, App Identity, Internationalization, Service Discovery & Time Sync, MicroApp Configuration. |

Mobile App Infrastructure library Integration

|  |  |
| --- | --- |
| Author | Anurag Gautam, Ashok Kasthuri, G Kavya |
| Approved by |  |
| Email Id | [anurag.gautam@philips.com](mailto:anurag.gautam@philips.com) [ashok.kasturi@philips.com](mailto:ashok.kasturi@philips.com)  [kavya.g.kurpad@philips.com](mailto:kavya.g.kurpad@philips.com) |

1. INTRODUCTION 3

2. INTEGRATION 3

2.1 Maven repository Integration 3

2.2 Library Integration 3

2.3 Library versioning 4

2.4 Root gradle changes 4

2.5 Gradle dependencies 4

2.6 Proxy dependencies 5

2.7 Configuration File 5

3. INITIALIZATION 6

6. Android Manifest Changes 7

6.1 Other User Permissions 7

7. **Design Document Reference 7**

# 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.

**Artifactory Path:**

<http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/libs-release-local-android/com/philips/cdp/AppInfra/1.1.0-rc.3/>

If you are inside Philips network then you can directly refer “**2.5 Gradle dependencies**” section. It will automatically download all nested dependencies 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:4.8.3'** compile(**'com.philips.cdp:prx:2.0.0@aar'**) {  
 exclude **group**: **'com.android.support'** }  
 compile(**'com.philips.cdp:localeMatch:2.0.0@aar'**) {  
 exclude **group**: **'com.android.support'** transitive = **true** }  
 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:2.1.0'**

**// 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/libs-release-local-android' }**

**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'}**

**}**

**}**

## Gradle dependencies

Just by adding below gradle dependencies, digitalcare and nested possible libraries will be downloaded from artifcatory. But it has to be inside Philips network.

compile(group: **'com.philips.cdp'**, name: **'prx'**, version: **'2.0.0'**, ext: **'aar'**){  
 exclude group: **'com.android.support'** transitive=**true** }  
 compile(group: **'com.philips.cdp'**, name: **'localeMatch'**, version: **'2.0.0'**, ext: **'aar'**){  
 exclude group: **'com.android.support'** transitive=**true** }

## 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 File

1. **logging.properties**

Copy **logging.properties** file from Documents\Internal folder to integrating Demo/vertical/Library assets folder. Developer can configure/filter/modify console/file logging properties by editing this file. “FileNotFoundException” will be thrown if this file is missing under application assets folder.

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"**,  
 **"acquisition"**: {  
 **"server"**: **"c00.adobe.com"** },  
  
 **"analytics"** : {  
 **"referrerTimeout"**: 5,   
 **"rsids"** : **"philipsmobileappsdev"**,  
  
*// "server" : "localhost:50000",* **"server"** : **"philips.112.2o7.net"**,  
 **"charset"** : **"UTF-8"**,  
 **"ssl"** : **true**,  
 **"offlineEnabled"** : **false**,  
 **"lifecycleTimeout"** : 30,  
 **"batchLimit"** : 0,  
 **"privacyDefault"** : **"optunknown"**,  
 **"poi"** : [  
 ]  
 },  
 **"target"** : {  
 **"clientCode"** : **"amsdk"**,  
 **"timeout"** : 5  
 },  
 **"audienceManager"** : {  
 **"server"** : **""** }  
}

1. AppIdentity.json

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

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

"ServiceDiscoveryEnvironment" : "PRODUCTION"

}

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 using AppInfraSingleton class. (Single App Infra object will exist in entire App Framework.)

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

**public static** LoggingInterface AILoggingInterface;

**public static** AppInfraInterface *gAppInfra*;

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

*gAppInfra*=AppInfraSingleton.*getInstance*();

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

}  
  
}

*AppInfraSingleton.setInstance()* must be called only once from application.

*AppInfraSingleton.getInstance()* can be called multiple times from App and Library as well.

# Android Manifest Changes

No special Permission required:

## 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();

}

**SecureStorage:**

SecureStorageInterface ssInterface = AppInfraSingleton.getInstance().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”);

**AppTagging:**

AIAppTaggingInterface mAIAppTaggingInterface = AppInfraSingleton.getInstance().getTagging().createInstanceForComponent("Component name","Component ID");

mAIAppTaggingInterface.setPreviousPage("SomeXpreviousPage");

mAIAppTaggingInterface.trackPageWithInfo("AppTaggingDemoPage", keyValuePair);

**Logging:**

LoggingInterface loggingInterface= AppInfraSingleton.getInstance().getLogging().createInstanceForComponent(“Component name”,” Component ID”);

loggingInterface.enableConsoleLog(true);

loggingInterface.enableFileLog(true);

loggingInterface.log (LoggingInterface.LogLevel.INFO,”Event”,message”);

1. Filter Logs:

Developer can use logging.properties file to filter Logs based on:

a)Log level

{VERBOSE, DEBUG, INFO, WARNING, ERROR}

*java.util.logging.ConsoleHandler.level=FINE // all five log Levels*

*java.util.logging.FileHandler.level = INFO // INFO, WARNING & ERROR*

*java.util.logging.ConsoleHandler.level=OFF // no output*

b) Component Level

philips.di.cl.appframework.UiKit.level=WARNING // only WARNING and ERROR log will output

philips.di.cl.appframework.UiKit.level=OFF // No log output for this component

Logging can be completely disabled from app by disabling file and console logging in logging.properties file irrespective of release and debug mode as follow:

java.util.logging.ConsoleHandler.level=OFF

java.util.logging.FileHandler.level = OFF

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).

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 appidentity.json and add microsite, sector & AppState key value:

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

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

*public String getAppName();*

Fetch technical App name.

*public String getAppVersion();*

Fetch App version

*public String getAppState();*

Fetch App state (development, test, acceptance, production)

*public String getLocalizedAppName();*

Fetch app localized commercial app name.

*public String getMicrositeId();*

Fetch micrositeID

*public String getSector();*

Fetch sector

*public String getServiceDiscoveryEnvironment();*

Fetch the environment of service discovery

**Service Discovery:**

1. void getHomeCountry(OnGetHomeCountryListener listener)

GetHomeCountry will get the country either from SIM or GEOIP. The country is saved in preferences. The listener, OnGetHomeCountryListener will get the results back.

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.

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.

**Design Document Reference:**

<https://atlas.natlab.research.philips.com/confluence/download/attachments/7439362/Android%20_Code_Design_AppInfra_global_v1.1.0-rc6.docx?api=v2>

**MicroApp Configuration:**

This component provides API to get and set the configurations for all microapps. 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.

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);

This method is used to fetch the value from the configuration file for the first time. Once values are modified /new keys are added it is fetched from the secure storage .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 or null when the key doesn’t exist.

Example usage of this method:

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

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

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

This method is used to set values and update values to the configuration file. User has to pass the Key, Coco Name 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 values for it in the config file.

The return value is true/ false.

Example usage of this method:

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

boolean success = mConfigInterface.setPropertyForKey (“newKey”, “UR”, “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 empty values.**

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.

Example usage of this method:

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

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