

Reference App Android Integration Guidelines

| Document History | | | | |
|------------------|--------------|--|---------|---|
| Version | Date | Author | Section | Changes |
| 0.1 | 09-Aug-2016 | Ritesh Jha, Rakesh Krishnamurthy | All | Initial draft |
| 0.2 | 22-Sept-2016 | Richa ,Spoorti | All | Initial draft |
| 0.3 | 12-Dec-2016 | Yogesh H.S, Rakesh Krishnamurthy | All | FlowManager,State,Condition,CoCo Integration |
| 0.4 | 21-Feb-2017 | Rakesh Krishnamurthy | All | All |
| 0.5 | 12-May-2017 | Richa Bajpai | All | All |

| | |
|------------------------|--------------------------------------|
| Author | Ritesh Jha |
| Approved By | Raymond Kloprogge |
| <u>Email Id</u> | <u>Raymond.kloprogge@philips.com</u> |

Table of Contents

| | | |
|----------|--|-----------|
| 1 | INTRODUCTION..... | 3 |
| 1.1 | Reference App Overview | 3 |
| 1.2 | Reference-App dependencies..... | 3 |
| 1.2.1 | Artifactory..... | 3 |
| 1.2.2 | Library dependencies | 3 |
| 1.2.3 | Root gradle changes | 3 |
| 1.2.4 | Reference-App Gradle dependencies..... | 4 |
| 1.2.5 | Package Options | 5 |
| 1.2.6 | Version dependencies..... | 5 |
| 1.2.7 | Proxy dependencies | 5 |
| 1.3 | TFS Link | 5 |
| 1.4 | Reference-App Architecture Overview | 5 |
| 1.5 | Integration of FlowManager in Reference app: | 6 |
| 1.5.1 | Pre-requisites for starting the flowmanager integration | 6 |
| 1.5.2 | Extending BaseFlowManager | 6 |
| 1.5.3 | Extending the condition class and defining custom condition classes..... | 7 |
| 1.5.4 | Define Event ID to trigger new state. | 7 |
| 1.6 | Launching a micro app in Reference app | 8 |
| 1.6.1 | Steps to create a state for a micro app | 8 |
| 1.6.2 | Calling the getNextState method to navigate..... | 9 |
| 2 | Coco Integration..... | 10 |
| 2.1 | App-Infra Integration..... | 10 |
| 2.2 | User Registration..... | 10 |
| 2.2.1 | User Registration State Placeholder | 10 |
| 2.2.2 | Google+ Patch Integration..... | 10 |
| 2.3 | Product Registration | 11 |
| 2.4 | IAP | 11 |
| 2.5 | Data Services | 11 |
| 2.6 | Connectivity | 11 |
| 2.7 | China Flow..... | 11 |
| 2.7.1 | WeChat Social Provider | 11 |
| 2.7.2 | Baidu Map..... | 12 |
| 2.7.3 | Livecom Chat..... | 12 |
| 3 | Handling uApp interfaces in Reference App..... | 12 |
| 3.1 | ActionBar from Coco | 12 |
| 3.2 | Handling Back key from coco..... | 12 |
| 4 | Push Notification Integration:..... | 13 |
| 5 | Notes | 14 |

1 INTRODUCTION

This document provides an overview of integration of Reference App features in Mobile applications.

1.1 Reference App Overview

Reference app is a sample app for propositions to start with from which new apps can be derived quickly by adding or stripping component.

It contains example code to demonstrate usage of various common components. Reference-App has following components integrated:

- Introduction / On boarding screens
- Settings
- User Registration
- Digital Care
- Product Registration
- In App Purchase Retailer
- Connectivity component
- About the application

Note: Please note that Data Services and Connectivity screens present in Reference app is for testing purposes only and it does not conform to any UI standards.

Please also note that about screen is also for representation only. It is the Propostions who have to decide on the design and contents of the above screens.

1.2 Reference-App dependencies

Integration can be done in following ways.

1.2.1 Artifactory

All dependent libraries should be downloaded from artifactory.

Artifactory path:

<http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/simple/libs-release-local-android/com/philips/cdp/appFramework/>

1.2.2 Library dependencies

Please follow the link provided to find the dependent libraries :

<https://confluence.atlas.philips.com/display/CPA/Creation+of+Release+17.1+-+current+status>

1.2.3 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'
        classpath 'org.robolectric:robolectric-gradle-plugin:1.1.0'
    }
}

allprojects {
    ext.androidAssertJVer = '1.1.1'

    repositories {
        maven { url 'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/jcenter' }
        maven { url "https://oss.sonatype.org/content/repositories/snapshots" }
        maven { url "https://mvnrepository.com/artifact/org.robolectric/shadows-core" }
    }
}

```

```

}

tasks.withType(Test) {
    maxHeapSize = '4g'
    scanForTestClasses = false
    include "**/*Test.class"
    test { // set JVM arguments for the test JVM(s)
        jvmArgs '-noverify'
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}

```

1.2.4 Reference-App Gradle dependencies

```

compile ('com.android.support:support-annotations:24.0.0') {
    exclude group: 'com.android.support', module: 'support-annotations'
}

compile 'com.android.support:design:24.0.0'
compile 'com.android.support:appcompat-v7:24.2.1'
debugCompile 'com.squareup.leakcanary:leakcanary-android-no-op:1.5'
leakCanaryCompile 'com.squareup.leakcanary:leakcanary-android:1.5'
releaseCompile 'com.squareup.leakcanary:leakcanary-android-no-op:1.5'
androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
    exclude group: 'com.android.support', module: 'support-annotations'
})

debugCompile 'com.facebook.stetho:stetho:1.2.0'

compile "com.j256.ormlite:ormlite-core:4.48"
compile "com.j256.ormlite:ormlite-android:4.48"

compile(group: 'com.philips.cdp', name: 'product-registration-lib', version: '2.3.5' + objcdp.getVersionSuffix(), ext: 'aar', changing: true) {
    transitive = true
}

compile(group: 'com.philips.cdp', name: 'digitalCare', version: '7.3.5' + objcdp.getVersionSuffix(), ext: 'aar', changing: true) {
    transitive=true
}

compile(group: 'com.philips.cdp', name: 'iap', version: '5.2.5' + objcdp.getVersionSuffix(), ext: 'aar', changing: true) {
    transitive=true
}

compile(group: 'com.philips.cdp', name: 'dataServices', version: '0.3.5' + objcdp.getVersionSuffix(), ext: 'aar') {
    exclude group: 'com.android.support'
    transitive = true
}

compile(group: 'com.philips.cdp', name: 'commlib-all', version: '1.2.2' + objcdp.getVersionSuffix(), classifier: 'release', ext: 'aar'){
    exclude group: 'com.android.support'
    transitive = true
}

testCompile 'junit:junit:4.12'
testCompile 'org.khronos.opengl-api:gl1.1-android-2.1_r1'
testCompile ('org.robolectric:robolectric:3.2.2')

```

```
testCompile "org.mockito:mockito-all:1.10.17"
testCompile "com.squareup.assertj:assertj-android:${androidAssertJVer}"
```

1.2.5 Package Options

```
packagingOptions {
    exclude 'META-INF/DEPENDENCIES.txt'
    exclude 'META-INF/LICENSE.txt'
    exclude 'META-INF/NOTICE.txt'
    exclude 'META-INF/NOTICE'
    exclude 'META-INF/LICENSE'
    exclude 'META-INF/DEPENDENCIES'
    exclude 'META-INF/notice.txt'
    exclude 'META-INF/license.txt'
    exclude 'META-INF/dependencies.txt'
    exclude 'META-INF/LGPL2.1'

    pickFirst 'lib/mips/librsjni.so'
    pickFirst 'lib/mips/libblasV8.so'
    pickFirst 'lib/mips/libRSSupport.so'
    pickFirst 'lib/x86/librsjni.so'
    pickFirst 'lib/x86/libblasV8.so'
    pickFirst 'lib/x86/libRSSupport.so'
    pickFirst 'lib/armeabi-v7a/librsjni.so'
    pickFirst 'lib/armeabi-v7a/libblasV8.so'
    pickFirst 'lib/armeabi-v7a/libRSSupport.so'
    pickFirst 'lib/arm64-v8a/libRSSupport.so'
    pickFirst 'lib/arm64-v8a/librsjni.so'
    exclude 'META-INF/INDEX.LIST'
```

1.2.6 Version dependencies

```
minSdkVersion 19
targetSdkVersion 24
compileSdkVersion 24
buildToolsVersion "24.0.3"
```

1.2.7 Proxy dependencies

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

```
systemProp.https.proxyHost= 165.225.104.34
systemProp.https.proxyPort= 1001
```

Note: Proxy dependencies will not be committed to TFS as this is just a local dependency.

1.3 TFS Link

tfsema1.ta.philips.com:8080/tfs/TPC_Regions/CDP2/_git/rap-android-reference-app

1.4 Reference-App Architecture Overview

Please find below the overview for Reference App architecture:

<https://confluence.atlas.philips.com/display/BA/BaseApp+Architecture+Overview>

1.5 Integration of FlowManager in Reference app:

Before integrating Flowmanager, state and conditions please go through the following link:

<https://confluence.atlas.philips.com/pages/viewpage.action?spaceKey=BA&title=Flow+Manager>

1.5.1 Pre-requisites for starting the flowmanager integration

- Create the **AppFlow.json** file as per the uApp library provided standard.
- Place the json in assets folder of the Application source folder and provide the path when flowmanager is instantiated.
- Associate each state in the json to a class by creating state classes.
- Associate each condition in the json to a condition class by creating condition classes.
- Sample code to create the condition class is in section **1.5.3** and to create a sample state class please refer section **1.6**.
- After this extend the BaseFlowManager class. The details of why and how to extend the BaseFlowManager class is in section **1.5.2**.
- Since it is necessary in Android system to have once activity as launch activity in Manifest file, please add this as the launch activity. Since there are no states available when app is launched as the json is not yet parsed, launch activity can act as the base on to of which the json can be parsed.
- Once the json is parsed, first state is returned and then flowmanager can take over in maintaining the app flow.
- If the proposition chooses to launch a fragment as a state, they can do so as shown below and add these fragments to the launchactivity stack.
- Reference App has two Activities. This is done to differentiate between the screens having action bar and the ones not having them. This is easier to maintain as there is not too many conditions written to hide or unhide the actionbar depending on the screen.
- Define event id, this is the trigger to fetch next state. Each event ID should be defined in AppFlow.json file. This event ID should be same as the one that is passed to the flowManager to fetch next state. Refer to section **1.5.4** for more info on using event ID.

1.5.2 Extending BaseFlowManager

We have to extend the BaseFlowManager because it provides the necessary methods to enable forward and backward navigation in the app. To enable back navigation in Android, please represent a standard event called “back” in AppFlow.json and create an event in the app in the same name. Then call the **getBackState()** method to update the flowmanager with the current state. Please go through the uApp Library link to understand more about back navigation through FlowManager. The link is provided below :

<https://confluence.atlas.philips.com/pages/viewpage.action?spaceKey=BA&title=Flow+Manager>

- Create a class by extending the BaseFlowManager.
- Implement the populateStateMap and populateConditionMap methods.
- Create two maps. One map to map the state ID and the state classes. The other one to map the condition ID with the condition classes.
- We have to ensure that the state ID and the condition ID are the same as the one in the AppConfig.json file.
- To the maps that are passed to the above methods, add the state objects and the condition objects.
- Find below the sample code to implement the above methods:

```
public void populateStateMap(final Map<String, BaseState> uiStateMap) {
    uiStateMap.put(AppStates.WELCOME, new WelcomeState());
    uiStateMap.put(AppStates.ON_BOARDING_REGISTRATION, new UserRegistrationOnBoardingState());
    uiStateMap.put(AppStates.SETTINGS_REGISTRATION, new UserRegistrationSettingsState());
    uiStateMap.put(AppStates.HOME_FRAGMENT, new HomeFragmentState());
    uiStateMap.put(AppStates.ABOUT, new AboutScreenState());
    uiStateMap.put(AppStates.DEBUG, new DebugTestFragmentState());
    uiStateMap.put(AppStates.SETTINGS, new SettingsFragmentState());
    uiStateMap.put(AppStates.IAP, new IAPRetailerFlowState());
    uiStateMap.put(AppStates.PR, new ProductRegistrationState());
}
```

```

uiStateMap.put(AppStates.SUPPORT, new SupportFragmentState());
uiStateMap.put(AppStates.SPLASH, new SplashState());
uiStateMap.put(AppStates.DATA_SYNC, new DataServicesState());
uiStateMap.put(AppStates.CONNECTIVITY, new ConnectivityFragmentState());
uiStateMap.put(AppStates.HAMBURGER_HOME, new HamburgerActivityState());
}

public void populateConditionMap(final Map<String, BaseCondition> baseConditionMap) {
    baseConditionMap.put(AppConditions.IS_LOGGED_IN, new ConditionIsLoggedIn());
    baseConditionMap.put(AppConditions.IS_DONE_PRESSED, new ConditionIsDonePressed());
    baseConditionMap.put(AppConditions.CONDITION_APP_LAUNCH, new ConditionAppLaunch());
}

```

1.5.3 Extending the condition class and defining custom condition classes

1. Extend the BaseCondition abstract class defined in the uAppFramework library.
2. Implement the abstract method isSatisfied().
3. Define the logic that needs to go inside the isSatisfied() method and return a Boolean value.
4. Sample code for creating a condition class is as follows:

```

public class ConditionAppLaunch extends BaseCondition {

    public ConditionAppLaunch() {
        super(AppConditions.CONDITION_APP_LAUNCH);
    }

    @Override
    public boolean isSatisfied(final Context context) {
        AppFrameworkApplication appFrameworkApplication = (AppFrameworkApplication) context;
        final BaseFlowManager targetFlowManager = appFrameworkApplication.getTargetFlowManager();
        final boolean isUserLoggedIn =
            targetFlowManager.getCondition(AppConditions.IS_LOGGED_IN).isSatisfied(context);
        final boolean isDonePressed =
            targetFlowManager.getCondition(AppConditions.IS_DONE_PRESSED).isSatisfied(context);
        return isDonePressed && !isUserLoggedIn;
    }
}

```

1.5.4 Define Event ID to trigger new state.

Since moving to a new state requires a trigger, flowmanager associates these triggers as events.

Here in the below sample code, we have associated the launch of the app as an event(APP_LAUNCH) which triggers the state SplashState which is loaded onto the LaunchActivity.

```

BaseFlowManager targetFlowManager = getApplicationContext().getTargetFlowManager();
BaseState baseState = null;
try {
    if (event.equals(APP_LAUNCH))
        baseState = getSplashState();
}

@NonNull
protected SplashState getSplashState() {
    return new SplashState();
}

```

1.6 Launching a micro app in Reference app

1.6.1 Steps to create a state for a micro app

Before a micro app is integrated into the reference app, each micro app should be represented as a state in Reference app.

1. Extend the BaseState abstract class.
2. Implement the abstract methods init, navigate and updateDataModel.
3. Any initialization of the Micro app that needs to be done should be done in the init() method of the state.
4. Any data that needs to be passed to the state needs to be added in the updateDataModel() method of the state.
5. To launch the micro app, call the navigate method. Inside the navigate method, call the updateDataModel to set the data that is needed for that particular micro app and then call the launch method of the particular micro app conforming to the uAppFramework library standards.

Sample Code is as follows for ConsumerCare micro app. A state called SupportFragmentState is created as follows:

1. Override the updateDataModel to set the data for CosumerCare. Consumer care requires CTN for product information :

```
@Override
public void updateDataModel() {
    String[] ctNList = new String[new
ArrayList<>(Arrays.asList(activityContext.getResources().getStringArray(R.array.productselection_ctnlist))).size()];
    ctNList = (new
ArrayList<>(Arrays.asList(activityContext.getResources().getStringArray(R.array.productselection_ctnlist))).toArray(c
tnList));
    setCtnList(ctNList);
}
```

2. Call the navigate method of the state as shown below. Inside the navigate method, updateDataModel() method is called and then launchCC() is called to launch ConsumerCare micro app as shown below:

```
@Override
public void navigate(UiLauncher uiLauncher) {
    fragmentLauncher = (FragmentLauncher) uiLauncher;
    this.activityContext = getFragmentManager();
    DigitalCareConfigManager.getInstance().registerCcListener(this);

    ((AppFrameworkBaseActivity)activityContext).handleFragmentBackStack(null,null,getUiStateData().getFragmentManagerLaun
chState());
    updateDataModel();
    launchCC();
}

private void launchCC()
{
    ProductModelSelectionType productsSelection = new
com.philips.cdp.productselection.productselectiontype.HardcodedProductList(getCtnList());
    productsSelection.setCatalog(Catalog.CARE);
    productsSelection.setSector(Sector.B2C);
    CcInterface ccInterface = new CcInterface();

    if (ccSettings == null) ccSettings = new CcSettings(activityContext);
    if (ccLaunchInput == null) ccLaunchInput = new CcLaunchInput();
    ccLaunchInput.setProductModelSelectionType(productsSelection);
    ccLaunchInput.setConsumerCareListener(this);
}
```



```

CcDependencies ccDependencies = new CcDependencies(getApplicationContext().getAppInfra());

ccInterface.init(ccDependencies, ccSettings);
ccInterface.launch(fragmentLauncher, ccLaunchInput);
}

```

1.6.2 Calling the getNextState method to navigate

Reference App uses the MVP pattern to interact with different components of the App. The Presenter takes care of acting on the event and the calling the flowmanager to navigate to the next state. So if the propositions require, they can refer to the below steps to create and launch a state through a presenter.

1. Extend the UIBasePresenter abstract class.
2. Override the abstract method onEvent.
3. Inside onEvent method, get the flowManager instance and call the **flowmanager.getNextState** method. To this method send the **eventID** which triggered the flowmanager. This method returns the next state to transition to.
4. Before calling the state.navigate method, call the setUiStateData method defined in the BaseState class. This method is used to handle the backstack of the fragments. This can be defined by the propositions to help in controlling the backstack.
5. Sample code for setUiStateData is as follows:

```

UIStateData homeStateData = new UIStateData();
homeStateData.setFragmentLaunchType(Constants.ADD_HOME_FRAGMENT);
baseState.setUiStateData(homeStateData);

```

6. Then call the state.navigate() method to navigate/transition to the next state in reference app.
7. This navigate() method in the state class has the following method which takes the UIStateData object to handle backstack. Sample code is as follows

```

((AppFrameworkBaseActivity)activityContext).handleFragmentBackStack(null,null,getUiStateData().getFragmentLaunchState());

```

8. handleFragmentBackStack is defined in the AppFrameworkBaseActivity which will help in maintaining the backstack for fragments. Code is as follows:

```

public void handleFragmentBackStack(Fragment fragment, String fragmentTag, int fragmentAddState) {
    containerId = getContainerId();
    try {
        fragmentTransaction = getSupportFragmentManager().beginTransaction();
        switch (fragmentAddState) {
            case Constants.ADD_HOME_FRAGMENT:

                if (null == getSupportFragmentManager().findFragmentByTag(HomeFragment.TAG)) {
                    addToBackStack(containerId, fragment, fragmentTag);
                } else {
                    getSupportFragmentManager().popBackStackImmediate(HomeFragment.TAG, 0);
                }

                break;
            case Constants.ADD_FROM_HAMBURGER:

                getSupportFragmentManager().popBackStack(null, FragmentManager.POP_BACK_STACK_INCLUSIVE);
                addToBackStack(containerId, new HomeFragment(), HomeFragment.TAG);
                fragmentTransaction = getSupportFragmentManager().beginTransaction();

```

```

        addToBackStack(containerId, fragment, fragmentTag);

        break;
    case Constants.CLEAR_TILL_HOME:

        getSupportFragmentManager().popBackStack(null, FragmentManager.POP_BACK_STACK_INCLUSIVE);
        addToBackStack(containerId,new HomeFragment(),HomeFragment.TAG);

        break;
    }
} catch (Exception e){
}
}

```

2 Coco Integration

2.1 App-Infra Integration

App-Infra is component which supports features like Logging, Tagging, Secure Storage, Service Discovery etc., Using App-Infra requires following Configuration

- Place App-Config.json under Android assets directory , find below file for reference



AppConfig.json

- Place ADBMobileConfig.json under Android assets directory, find below file for reference



ADBMobileConfig.json

Place the below code in Application class to initialize and use App-Infra

```

public AppInfraInterface appInfra;
appInfra = new AppInfra.Builder().build(getApplicationContext());

```

2.2 User Registration

2.2.1 User Registration State Placeholder

- Place the below code in Application class to initialize UserRegistration

```

private UserRegistrationState userRegistrationState;
userRegistrationState = new UserRegistrationOnBoardingState();
userRegistrationState.init(this);

```

- Find the User Registration State file for your reference



UserRegistrationSplashState.java

2.2.2 Google+ Patch Integration

Google+ social provider is supported in UR module now with Web browser sign in to enable Google+ for the application, there are certain prerequisite(support libraries, OAuth2.0 etc) are recommended to follow.

Please refer [User Registration's integration guide](#) for in depth explanations.

Note: Key should be application specific.

2.3 Product Registration

- Place the below code in Application class to initialize Product Registration

```
private ProductRegistrationState productRegistrationState;  
productRegistrationState = new ProductRegistrationState();  
productRegistrationState.init(this);
```

- Find the Product Registration State file for your reference



ProductRegistrationState.java

2.4 IAP

- Place the below code in Application class to initialize IAP

```
IAPState iapState = new IAPRetailerFlowState();  
iapState.init(this);
```

- Find the IAP retailer State file for your reference



IAPRetailerFlowState.java

2.5 Data Services

Data Services component is integrated only for testing purpose. The UI for Data Services is created to test the capabilities of the Data Services component. For more information on Data services, please refer to the DataServices folder in the Reference App Source Folder. Data Services screen is launched by creating a state so that it conforms to the flowmanager standards.. Please go through the following link for more information:

https://bitbucket.atlas.philips.com/projects/CDS/repos/datasync_android/browse/Documents/External/DataServices_Integration_Document_Android_PI_17.1.docx

2.6 Connectivity

Connectivity is also integrated only for testing purpose. The screens do not conform to any UI standard defined. Connectivity screen is also launched by creating a state so that it conforms to the flowmanager standards. For more information about connectivity integration, please go to the following link :

<https://bitbucket.atlas.philips.com/projects/COM/repos/dicomm-android/browse/Documents>

2.7 China Flow

There are some specific features completed for China Market. Listing is as below:

2.7.1 WeChat Social Provider

We chat social provider is supported in UR module now. To enable WeChat for application, there are certain prerequisite are recommended to follow. AppId and Secret key has to be generated on the basis on app signature.

Please refer User Registration's integration guide for in depth explanations.

Additional reference:

<https://confluence.atlas.philips.com/display/UR/How+to+integrate+the+WeChat+in+the+Mobile+apps>

Note: Key should be application specific.

2.7.2 Baidu Map

Consumer Care common component has a feature to show Philips's Service Centres on the Baidu Map. There is a common URL we received from ConsumerCare as below, which is added in digitalcare_config.xml.

URL: <http://vip.crmclick.com/wxz/index.aspx>

2.7.3 Livecom Chat

Consumer Care common component has a feature to connect to China service helpdes. There is a common URL we received from ConsumerCare as below, which is added in digitalcare_config.xml.

URL: http://ph-china.livecom.cn/webapp/index.html?app_openid=ph_6idvd4fj&token=PhilipsTest

3 Handling uApp interfaces in Reference App

3.1 ActionBar from Coco

State must implement ActionBarListener to handle the title and back key events from coco

We have following methods to be overridden for getting title

The string in the below method parameter is the title of actionbar that needs to be set by the app

Boolean b : is the value whether back is handled by coco or we need to handle it.

True : Required to show back button and the respective coco will handle back key

False: App needs to handle the back key

```
public void updateActionBar(@StringRes int i, boolean b) {
    setTitle(getResources().getString(i));
    updateActionBarIcon(b);
}

/**
 * For Updating the actionbar title as coming from other components
 * @param s String to be updated on actionbar title
 * @param b Whether back is handled by them or not
 */
@Override
public void updateActionBar(String s, boolean b) {
    setTitle(s);
    updateActionBarIcon(b);
}
```

3.2 Handling Back key from coco

We need to check the value of handleBackEvent() method from BackEventListener from coco .

If true is returned - the respective coco will handle the event,

If false is returned – the application needs to handle the back event

```
if (currentFrag != null && currentFrag instanceof BackEventListener &&
currentFrag instanceof RegistrationFragment) {
    backState = ((BackEventListener) currentFrag).handleBackEvent();
    if (!backState) {
        fragmentManager.popBackStack(); // Do your stuff here
    }
}
```

4 Push Notification Integration:

This component will abstract GCM client implementation as well as provide API callbacks to register token with your app server. It also handles push notification and redirects payload to components you registered for receiving push messages. For using this component you need to copy source from reference app repository. Please refer below URL.

http://tfsemea1.ta.philips.com:8080/tfs/TPC_Region24/CDP2/_git/rap-android-reference-app?path=%2FSource%2FAppFramework&version=GBdevelop&a=contents

To receive push notification in your app you need to configure your project in FCM console. After your project has been configured you will get google-services.json. You need to replace this file in source you copied from above repo as currently this component is configured for Reference App project.

Push Notification Configurations:

You can configure push notification from AppConfig.json.

AppConfig.json changes

```
"ReferenceApp":{
  "PushNotification.autoLogout":"true",
  "PushNotification.polling":"false"
}
```

We can set the above two fields as true or false as per need .

If autologout is enabled : Token will be deregistered from datacore automatically once user logout.

If polling is enabled : Push notification will be disabled and data synchronization will happen every 30 seconds.

Push Notification Initialization:

Push Notification needs to be initialized in application class.

```
public static PushNotificationManager getInstance() {
    if (pushNotificationManager == null) {
        pushNotificationManager = new PushNotificationManager();
    }
    return pushNotificationManager;
}

public void init(AppInfraInterface appInfra, PushNotificationUserRegistrationWrapperInterface
pushNotificationUserRegistrationWrapperInterface){
    PNLog.init(appInfra);
    PNLog.enableLogging();
    this.pushNotificationUserRegistrationWrapperInterface=pushNotificationUserRegistrationWrapperInterface;
}
```

While initialising PN CoCo, it expects AppInfra object as well as URWrapper Interface implementation object. Refer below wrapper interface

```
public interface PushNotificationUserRegistrationWrapperInterface {
    boolean isUserSignedIn(Context appContext);
}
```

You need to implement this interface in your app/component. From this implementation class Push Notification CoCo will fetch User information.

See below sample code.

```
private PushNotificationManager pushNotificationManager;
```

```
pushNotificationManager = PushNotificationManager.getInstance();
pushNotificationManager.init(appInfra, getDataServiceState());
pushNotificationManager.startPushNotificationRegistration(getApplicationContext());
```

Push Notification Api's:

In the class where you need to receive token once app is registered with GCM, you need to implement following interface PushNotificationTokenRegistrationInterface

This interface has two methods to be implemented for registering and deregistering the token

```
public interface PushNotificationTokenRegistrationInterface {
    void registerToken(String deviceToken, String appVariant, String protocolProvider,
        RegistrationCallbacks.RegisterCallbackListener registerCallbackListener);

    void deregisterToken(String appToken, String appVariant, RegistrationCallbacks.DergisterCallbackListener
        dergisterCallbackListener);
}
```

While registering token with your app server, you need to send callback to PushNotification component using following interfaces.

```
public interface RegisterCallbackListener{
    void onResponse(boolean isRegistered);

    void onError(int errorCode,String errorMessage);
}

public interface DergisterCallbackListener{
    void onResponse(boolean isDeRegistered);

    void onError(int errorCode,String errorMessage);
}
```

Also to receive callback,you need to register this class with PushNotificationManager. Use following registration API from PushNotificationManager.

```
PushNotificationManager.getInstance().registerForTokenRegistration(this);
```

We have provided an API to deregister your callback in case your component or app don't want GCM token

If your component/App want to receive payload from Push Notification then you can simply implement following interface.

```
public interface HandleNotificationPayloadInterface {
    void handlePayload(JSONObject payloadObject) throws JSONException;
}
```

And not forget to register implementing class with PushNotificationManager. Refer below code to register for receiving payload

```
PushNotificationManager.getInstance().registerForPayload(this);
```

5 Notes

1. Please refer interface Spec Doc or Java documents for more details on APIs for each individual component.
2. Please refer demo app for implementation details of various CoCo