|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document History** | | | | |
| **Version** | **Date** | **Author** | **Section** | **Changes** |
| 1.0 | 04/08/2016 | Deepthi Shivakumar | All | Release 1 features |
| 2.0 | 03/02/2017 | Rakesh, Yogesh | 7 and above | Release 2 features. |

|  |  |
| --- | --- |
| Author | Deepthi Shivakumar, Richa Bajpai |
| Approved by |  |

Contents

[1. INTRODUCTION 2](#_Toc475455340)

[2. INTEGRATION 2](#_Toc475455341)

[2.1 Maven repository (Artifactory based) Integration 2](#_Toc475455342)

[2.2 Library Integration 2](#_Toc475455343)

[2.3 Library versioning 3](#_Toc475455344)

[2.4 .Root gradle changes 3](#_Toc475455345)

[2.5 Proxy dependencies 4](#_Toc475455346)

[3. INITIALIZATION 4](#_Toc475455347)

[4. Android Manifest Changes 4](#_Toc475455348)

[4.1 Other User Permissions 4](#_Toc475455349)

[5. Base Classes 4](#_Toc475455350)

[5.1 BaseClass for Configuration : 4](#_Toc475455351)

[5.2 Classes for Launching Mechanism 4](#_Toc475455352)

[6. Interfaces 5](#_Toc475455353)

[7. Flow Manager 5](#_Toc475455354)

[7.1.1 Pre-requsites: 5](#_Toc475455355)

[7.1.2 What is Flow Manager? 6](#_Toc475455356)

[7.1.3 API to override in BaseFlowManager are: 6](#_Toc475455357)

[7.2 Listeners 7](#_Toc475455358)

[7.2.1 FlowManagerListener 7](#_Toc475455359)

[7.3 BigBucket Link: 7](#_Toc475455360)

[7.4 Base-App Architecture Overview 8](#_Toc475455361)

[7.4.1 Presenter 8](#_Toc475455362)

[7.4.2 State 8](#_Toc475455363)

[7.4.3 Condition 9](#_Toc475455364)

[7.5 Notes 9](#_Toc475455365)

# INTRODUCTION

This document provides an overview of integration procedure for uAppframework 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:**

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'**])compile **'com.android.support:appcompat-v7:24.2.1'**compile(**group**: **'com.philips.cdp'**, **name**: **'AppInfra'**, **version**: **'1.5.5'**, **ext**: **'aar'**)

## 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/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'}**

**}**

**}**

## 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.

# INITIALIZATION

There is not applicable for uAppFramework. It contains only base classes and interfaces.

# Android Manifest Changes

No special Permission required:

## Other User Permissions

No special Permission required:

# Base Classes

## BaseClass for Configuration:

Class: LaunchType

This class needs to be extended by micro app frameworks for common configuration.

## Classes for Launching Mechanism

An app can launch micro app either as activity or fragment. Each micro app need to facilitate both kind of launching mechanism. Micro app (component) framework enables this by providing set of base classes. App or any component which launches a micro app is expected to create an object of either Fragment launcher or Activity launcher and inject to micro app launch API

1. Abstract Class: UiLauncher

Contains methods for setting animations for the activity

**public void** setAnimation(**int** enterAnimResId, **int** exitAnimation)

* 1. Class: ActivityLauncher

Needs to be instantiated for launching micro app as activity.

**public** ActivityLauncher(ActivityLauncher.ActivityOrientation screenOrientation,Bundle bunde)

ActivityOrientation is a defined set of int defs which can be used to specify ScreenOrientation.

* 1. Class: FragmentLauncher

Needs to be instantiated for launching micro app as Fragment.

**public** FragmentLauncher(FragmentActivity fragmentActivity,  
 **int** parentContainerResId,  
 ActionBarListener actionbarUpdateListener)

# Interfaces

1.ActionBarListener

For Updating title on Action Bar this interface needs to be implemented providing definition of below method:

**void** updateActionBar(String titleText,**boolean** enableBackKey);

2. BackEventListener

Each micro app is expected to implement this interface. Micro app needs to return true if they need to consume and handle back key event else return false if they do not want to any actions there by app calls super.onBackPressed(). The below method needs to be defined.

**boolean** handleBackEvent();

3. uAppInterface

uAppframework provides below interface which has standard APIs to initialise, launch and set configuration.

Each micro app will expose one single class for vertical to launch UI where in it needs to implement this interface.

Below methods needs to be implemented for this interface

**public void** init(Context context, AppInfra appInfra);

**public void** launch(UiLauncher uiLauncher, uAppListener uAppListener);

**public void** setLaunchInput(LaunchInput launchInput);

# Flow Manager

### Pre-requisites:

Some of the pre-requisites for using Flowmanager are as follows:

* BaseFlowManager needs to be extended and populateStateMap and populateConditionMap implemented.
* FlowManager requires a hard coded state that needs to be defined.
* This is required so that there is at least one state that is loaded while the AppFlow.json file is parsed.
* This first state needs to implement the **FlowManagerListener**.
* This listener get the call back once the **AppFlow.json** is parsed.
* After this the first state to be loaded can be fetched by passing the first event and the current state to getNextState method in the flowmanager instance.

### What is Flow Manager?

Flow-Manager is a class which extends BaseFlowManager, responsibility includes to pass context and JSON path to BaseFlowManager and mapping respective State with AppState Constants, mapping respective Conditions with Condition Constants and return respective State based on current State and eventId passed. Find the below Flow Manager File for reference :



For detailed explanation please find below the link:

[https://atlas.natlab.research.philips.com/confluence/display/BA/Flow+Manager](https://atlas.natlab.research.philips.com/confluence/display/BA/Flow+Manager%20)

### API to override in BaseFlowManager are:

**1.public void** initialize(@NonNull **final** Context context, @NonNull **final** String jsonPath, @NonNull **final** FlowManagerListener flowManagerListener) **throws** JsonFileNotFoundException, JsonStructureException, JsonAlreadyParsedException {}

Contains methods to initialize and call the next state from the Proposition classes.

This method needs the context and the json file path which needs to be passed from the propositions. This is the first step in using FlowManager.

2.**public** BaseState getNextState(String eventId) **throws** NoEventFoundException, NoStateException, NoConditionFoundException, StateIdNotSetException, ConditionIdNotSetException {}

This method is used by the propositions to fetch the next state by passing in the eventId that caused the state transition.

**3.public** BaseState getBackState(BaseState currentState) **throws** NoStateException, NoConditionFoundException, StateIdNotSetException, ConditionIdNotSetException {

* This class also contains methods to get the backstate for back handling. Pass the current state of the app to this method. This method then returns the backstate for that particular current state.
* Currently this method only returns the state that the propositions need to move it but the actual logic to navigate to that state needs to be written by the poposition.
* The Backstate stack maintained by the Flowmanager currently goes to the state if it already exists in the backstate and all the states above it are popped.
* If the state returned by this method is a new state , then all the states in the stack are popped and the current state is set to the state returned by this method.
* Current state is then set to the state returned by this method.

**4.public** BaseState getState(String stateId):

This methods returns the state object for the state ID that is passed to it. Example of usage is as follows:

BaseFlowManager targetFlowManager = getApplicationContext().getTargetFlowManager();  
**baseState** = targetFlowManager.getNextState(targetFlowManager.getState(AppStates.***HAMBURGER\_HOME***), eventState);

**5.public** BaseState getCurrentState():

targetFlowManager.getBackState(targetFlowManager.getCurrentState());

This method contains the reference to the current state of the app.

**6.public** BaseCondition getCondition(String conditionId) **throws** ConditionIdNotSetException

This class has method to get the condition object for the condition ID that is passes to it.

## Listeners

### FlowManagerListener

* This call back is used by the flowmanager to get the callback from the AppFlowParser.
* Call back is received by the class implementing this interface in the onParseSuccess() method.
* Parsing is done on a worker thread and the call back is passed to the main thread.
* While this parsing happens and the call back is received by the class implementing this interface, proposition needs to show a progress dialog or wait on the UI.

## BigBucket Link:

<https://bitbucket.atlas.philips.com/projects/MAF/repos/uappframework_android/browse>

## Base-App Architecture Overview

Please find below the overview for Base App architecture in below link:

[https://atlas.natlab.research.philips.com/confluence/display/BA/BaseApp+A rchitecture+Overview](https://atlas.natlab.research.philips.com/confluence/display/BA/BaseApp+A%20rchitecture+Overview%20)

### Presenter

Presenter is responsible for processing events and navigating to corresponding State, sets the presenter to the state and using flow Manager navigates to the corresponding state based on current state and event. Each Fragment must have a presenter which extends the presenter base class UiBasePresenter and it must implement its methods to performing operations on processing events like loading the fragment or on click events inside the fragment

Find the below Presenter file for reference



### State

State class is supposed to extend BaseState Class

And implement the abstract methods for initialization, navigation and handling callbacks

Find the below State file for reference



#### BaseState

This class is the base class for all the state objects. Any new state that is created should extend from this class and implement the abstract classes.

Public methods are:

1. **public abstract void** init(Context context);

This method is used to initliaze any state. For example, initialize any Micro app or common component that needs to be used as a state.

2. **public abstract void** updateDataModel();

This method is used to pass data to a state. This will be used for example to pass the CTN to consumer care Micro app.

3. **public abstract void** navigate(UiLauncher uiLauncher);

This method is used to navigate from the current state to another state.

4. **public** String getStateID()

**public void** setStateID(String stateID)

These methods are used to set and get state ID from the state Objects.

### Condition

As the name says Condition is a class which is responsible to define Conditions under the API isSatisfied(Context context) while navigating to each state, based on the conditions defined FlowManager returns State.

Find the below Condition file for reference



#### BaseCondition

This is the base class for all the conditions that needs to be used by the proposition. This is an abstract class. Any condition that is defined in the **AppFlow.json** will have create a corresponding class and extend from this base class .This ensures the FlowManager can access and check for this condition in getNextState internally.

1. **public** String getConditionID()

This method is used to get the condition ID for this particular Condition Object.

2. **public abstract boolean** isSatisfied(Context context)

This method is used to write the condition by the extending condition class.

## Notes

2. Please refer interface Spec Doc or Java documents for more details on APIs for each individual component.

3. Please refer demo app for implementation details of various CoCo