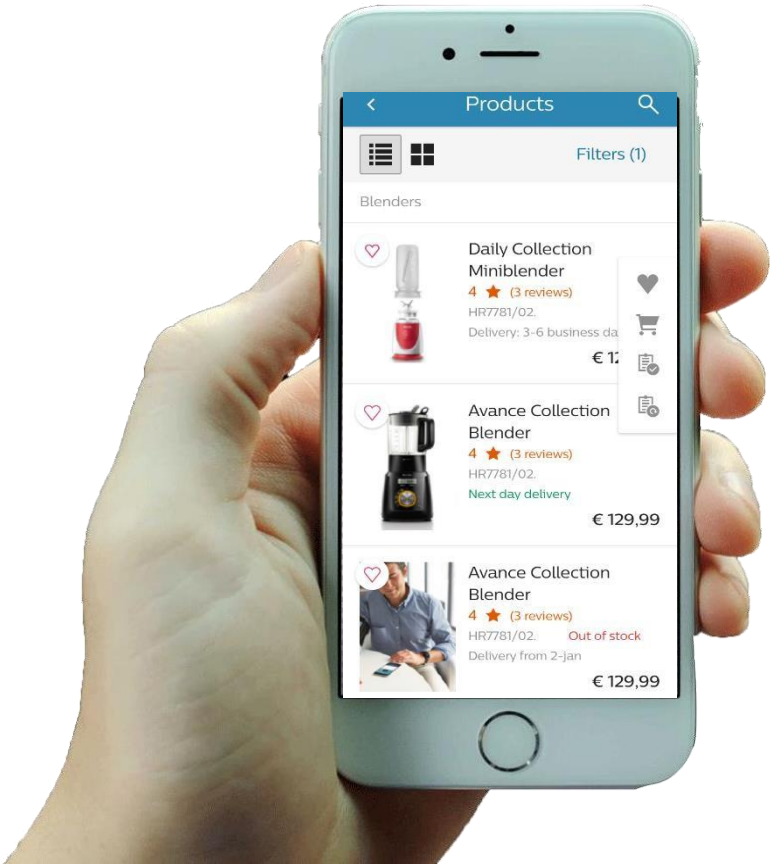
PHILIPS



Mobile E-Commerce

ANDROID INTEGRATION DOCUMENT

ENTERPRISE MOBILE SOLUTION (EMS)

PHILIPS

Document Configuration Management

Document Identification

Name of Project	Mobile E-Commerce
Name of Report	Mobile E-Commerce Android Reference Guide
File Name	MEC_Android_Integration_Document_V1.0

Document Change Control

Version	Date Released	Change Notice
1.0	21-Jan-2020	Gardenia – Release Integration Document

PHILIPS

Contents

Document Configuration Management....1
Document Identification....1
Document Change Control....1
INTEGRATION....3
INITIALIZATION....3
MECSettings....4
MECDependencies....4
LAUNCHING....5
MECInterface....6
Configuration....6
TAGGING....7
SERVICE DISCOVERY....7

PHILIPS

INTEGRATION

The easiest and preferred way to integrate this components is by using gradle (version 3.5.2 or newer). MEC is developed using kotlin and databinding. The below dependencies needs to be added to integrate MEC. Please find the snippet below for reference.

```
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
apply plugin: 'kotlin-kapt'
```

```
ext.kotlin_version = '1.3.50'
```

```
gradle              : 'com.android.tools.build:gradle:3.5.2'
```

```
//Kotlin test
kaptAndroidTest "android.databinding:databinding-
compiler:$kotlin_version"
```

```
androidTestImplementation deps.espresso_web
implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
implementation "org.jetbrains.kotlin:kotlin-test:$kotlin_version"
annotationProcessor deps.livedata_compiler
```

```
dataBinding {
    enabled = true
}
```


INITIALIZATION

The Mobile E-Commerce component can be initialized using the following method:

```
public void init(UappDependencies uappDependencies, UappSettings
uappSettings) {
MECDependencies MECDependencies = (MECDependencies) uappDependencies;
mUserDataInterface = MECDependencies.getUserDataInterface();
mMECSettings = (MECSettings) uappSettings;
mUappDependencies = uappDependencies;
}
```

Note: Please refer MECInterface section for more reference

Basically, proposition app needs to declare as above. This has two main parts like MECSettings and MECDependencies. The object of both type should be passed while initializing MECInterface, which initializes the configuration.

```
public void launch(UiLauncher uiLauncher, UappLaunchInput
uappLaunchInput) throws RuntimeException {
    MECHandler mecHandler = new
MECHandler((MECDependencies)mUappDependencies,mMECSettings,uiLauncher,(M
ECLaunchInput) uappLaunchInput);
    mecHandler.launchMEC();
}
```

Launch method is used to launch MEC component, which takes two input parameters:

uiLauncher : Here uiLauncher is the fragment launcher

uappLaunchInput : This holds all the Configuration values needed to Launch and Customize Mobile E-Commerce behavior

Below, is a snippet of setting the MEC Flow using LaunchInput:

```
MECFlowConfigurator input = new MECFlowConfigurator();
mCategorizedProductList = new ArrayList<>();
mCategorizedProductList.add("HX3631/06")
input.setCTNs(mCategorizedProductList) ;
launchMEC(MECFlowConfigurator.MECLandingView.MEC_PRODUCT_DETAILS_VIEW,
input, null);
```

MECSettings

MEC don’t have any settings to be initialized. So only default initialization of MECSettings is required to be passed while creating MECInterface object.

MECDependencies

This class handles the dependency required for MEC. Currently, MEC has two dependencies i.e AppInfraInterface and UserDataInterface. So propositions need to initialize MECDependencies and set the app infra and user data interface object. This app infra object will be responsible for logging, tagging and some configuration and user data interface object is responsible for all user related data.

```
public MECDependencies(@NonNull AppInfraInterface appInfra, @NonNull
UserDataInterface userDataInterface) {
    super(appInfra);
    this.userDataInterface = userDataInterface;
}
```

LAUNCHING

The MECFlowConfiguration object needs to be initialized with default constructor or with one ctn or ctn list. Based on the landing view proposition would initialize the MECFlowConfiguration.

MEC can be launched with 3 Landing Views, namely:

- **mecProductListView**
  - Launches MEC Product List Screen displaying all the Product List downloaded from Hybris according to the given Configuration like Locale and Proposition ID.

- **mecCategorizedProductListView**
  - Launches MEC Categorized Product List Screen displaying all the Product List downloaded from Hybris according to the given Configuration like Locale and Proposition ID. The difference between this Landing View and mecProductListView is that while launching MEC with this Landing View, we have to pass list of one or more CTNs and only those Products will be displayed in the screen, which is also present in the Hybris.

- **mecProductDetailsView**
  - Launches MEC Product Details Screen with the CTN passed to MEC and displays all the details of the Product with “Buy from Retailers” option.

Example:

```
MECFlowConfigurator input = new MECFlowConfigurator();
mCategorizedProductList = new ArrayList<>();
mCategorizedProductList.add("HX3631/06")
input.setCTNs(mCategorizedProductList) ;

if (getActivity() instanceof LaunchAsActivity) {
launchMEC(MECFlowConfigurator.MECLandingView.MEC_PRODUCT_DETAILS_VIEW,
input, null);
}
else if (getActivity() instanceof LaunchAsFragment) {
launchMECasFragment(MECFlowConfigurator.MECLandingView.MEC_PRODUCT_DETAI
LS_VIEW, input, null);
}
```

MECInterface

MECInterface is the interface class for interacting with the Mobile E-Commerce component. It has two methods for initializing and launching. Following are the two methods:

```
/*For initializing MEC*/
public void init(UappDependencies uappDependencies, UappSettings
uappSettings) {

/*For launching MEC*/
public void launch(UiLauncher uiLauncher, UappLaunchInput
uappLaunchInput) throws RuntimeException {
```

Note: please refer initialization and launching section for more reference.

Configuration

The proposition has to keep the AppConfig.json file.

AppConfig.json

There is a group named “MEC” for InApp Purchase. There is one key named “propositionId” which basically represents the vertical proposition name. For example, Tuscany proposition has id “Tuscany2016”.

```
{
 "UserRegistration": {
        "JanRainConfiguration.RegistrationClientID.Development":
"8kaxdrpvkwyr7pnp987amu4aqb4wmnte",
        "JanRainConfiguration.RegistrationClientID.Testing":
"g52bfma28yjbd24hyjcswudwedcmqy7c",
        "JanRainConfiguration.RegistrationClientID.Evaluation":
"f2stykcygm7enbwfw2u9fbg6h6syb8yd",
        "JanRainConfiguration.RegistrationClientID.Staging":
"f2stykcygm7enbwfw2u9fbg6h6syb8yd",
        "JanRainConfiguration.RegistrationClientID.Production":
"9z23k3q8bhqyfwx78aru6bz8zksga54u",
        "PILConfiguration.CampaignID": "CL20150501_PC_TB_COPPA",
        "Flow.EmailVerificationRequired" : true,
        "Flow.TermsAndConditionsAcceptanceRequired" : true,
        "Flow.MinimumAgeLimit" : { "NL":12 ,"GB":14,"default": 16},
        "SigninProviders.default": ["myphilips","facebook"]
        "SigninProviders.CN": ["myphilips","sinaweibo"]
    },
    "MEC": {
        "propositionId" : "Tuscany2016"  // Proposition Specific ID
needs to be mentioned
    },
    "appinfra": {
        "appidentity.micrositeId": 77000,
        "appidentity.sector": "b2c",
        "appidentity.appState": "Staging",
        "appidentity.serviceDiscoveryEnvironment": "Production",
        "servicediscovery.platformMicrositeId": 77000,
        "servicediscovery.platformEnvironment":"Production"
    }
}
```


TAGGING

Tagging is completely handled by Mobile E-Commerce with AppInfra tagging API's, please refer the AppInfra integration document for any clarification.

SERVICE DISCOVERY

Mobile E-Commerce has two flows i.e Hybris and Non-Hybris (i.e only retailer flow). Mobile E-Commerce is implemented with service discovery feature to decide Hybris or retailer flow. Based on app infra service discovery implementation, Mobile E-commerce flow will be decided.

PHILIPS

Launch MEC as Fragment specific back press handle

If MEC is launched as Fragment, then launching activity must implement BackEventListener interface and handle backpress as shown below:

```
@Override
     public void onBackPressed() {
    ......
    ......
    ......
     FragmentManager fragmentManager = getSupportFragmentManager();
   Fragment currentFrag =
     fragmentManager.findFragmentById(R.id.mainFragmentContainer);
     // to get current fragment
     boolean backState = false;
         if (currentFrag != null && currentFrag instanceof
     BackEventListener) {
             // This handleBackEvent() will be executed in fragments if
     implemented
             // Also this method must be called when user press Android
     back button as well as toolbar back icon.
               backState = ((BackEventListener)
     currentFrag).handleBackEvent();
     if (!backState) {
     super.onBackPressed();
     }
     }
```

---------- End of Document----------