|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **`Document History** | | | | |
| **Version** | **Date** | **Author** | **Section** | **Changes** |
| 0.1 | 12-01-2016 | Spoorti Hallur | All | Initial draft |

Data Service Integration

**DOCUMENT HISTORY**

|  |  |
| --- | --- |
| **Author** | Spoorti |
| **Approved by** |  |
| **Email Id** | [Spoorti.hallur@philips.com](mailto:Indrajit.kumar@philips.com) |

Table of Contents

[Introduction 3](#_Toc468456799)

[Prerequisites 3](#_Toc468456800)

[Library Integration 4](#_Toc468456801)

[Maven repository Integration 4](#_Toc468456802)

[Library Integration 4](#_Toc468456803)

[OrmLite Library (Object Relational Mapping Lite) 6](#_Toc468456804)

[User Registration 6](#_Toc468456805)

[HSDP Login 6](#_Toc468456806)

[Pre Requisites: 7](#_Toc468456807)

[Data Base schema 8](#_Toc468456808)

[Data-Services Interfaces to be implemented by Verticals 11](#_Toc468456809)

[Database Interfaces sent to Library as callbacks 12](#_Toc468456810)

[Setting up Synchronization Loop 12](#_Toc468456811)

[Handling Token Expiry 13](#_Toc468456812)

[DataServicesManager 13](#_Toc468456813)

[DataFetcher 13](#_Toc468456814)

[DataSender 13](#_Toc468456815)

[INITIALIZATION 13](#_Toc468456816)

[Reference APP link 14](#_Toc468456817)

[https://atlas.natlab.research.philips.com/stash/scm/ip/in-app-purchase\_android.git 14](#_Toc468456818)

[Notes 14](#_Toc468456819)

[References: 14](#_Toc468456820)

# Introduction

Data-Services is a library that provides interfaces for communicating with Data-Core.

The Vertical propositions who ever consumes Data-Services are expected to implement the DB interfaces for storing Data coming from Data-Core.

This document provides an overview of integrating Data-Services Library in Android mobile application.

Source Path: <https://atlas.natlab.research.philips.com/bitbucket/projects/CDS/repos/datasync_android/browse>

# Prerequisites

I. Vertical project is configured for Android Studio

II. Setting->Developer Options->Don’t Keep Activities should be unchecked. [If Developer Mode is on]

III. Android API version should on 10[Gingerbread]



# Library Integration

## Maven repository Integration

The easiest and preferred way to use these components is using maven. Only we need to add maven repositories in app build gradle:

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'  
 }  
 flatDir {  
 dirs 'libs'  
 }  
 }  
}

## Library Integration

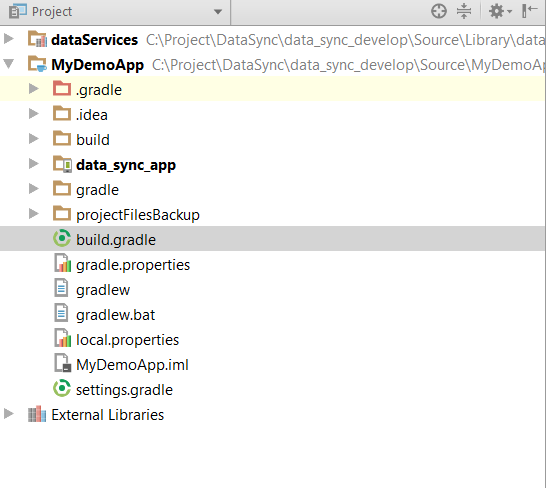
1. Checkout the code from above path wherein developer can find sample app which depends on Data-Services library’s aar file :

<https://atlas.natlab.research.philips.com/bitbucket/scm/cds/datasync_android.git>

1. Data-Services needs other libraries to build which are as below
2. User registration – direct dependency
3. App-Infra – indirectly UR uses App-Infra

Please make sure all these are added to application with the latest versions along with Data-Services library aar file under libs folder.

1. Refer DataServices dependencies along with .aar files in build.gradle as mentioned below :



Note: Since Data-Services Android was built with

**com.android.support:appcompat-v7:24.2.0**

**so vertical app support libs should be greater or equal to mentioned versions**

dependencies {  
**compile ''com.philips.cdp:dataServices:dataServices:+'**

**compile "com.philips.cdp:registrationApi:8.1.0-SNAPSHOT"  
compile "com.philips.cdp:uikitLib:3.2.3"  
compile "com.philips.cdp:AppInfra:1.3.0-SNAPSHOT"  
compile 'com.android.support:support-annotations:23.2.0'**

**compile 'com.android.support:multidex:1.0.1'  
compile 'com.android.support:design:23.1.1'  
compile 'com.android.support:appcompat-v7:+'**

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

}

1. Clean project



# OrmLite Library (Object Relational Mapping Lite)

Object Relational Mapping Lite (ORM Lite) provides some [simple](http://ormlite.com/simple_orm_java.shtml), lightweight functionality for persisting Java objects to SQL databases while avoiding the complexity and overhead of more standard ORM packages.

In our Demo-App, Database implementation is demonstrated using the ORMLite Library. Verticals are free to choose a library of their choice for DB implementation.

# User Registration

User Registration HSDP flow is used to get HSDP token. The Obtained HSDP token is used for restful API calls for communicating to Data-Core

# HSDP Login

Data-Services requires HSDP Login in order to access Data-Core APIs. Data-Services can be tested in DEVELOPMENT or STAGGING Environment.

Following are the sync Space URLs for Staging and development

**DEVELOPMENT:**

URL - "https://platforminfra-ds-platforminfrastaging.cloud.pcftest.com",

App-Name – uGrow

Shared kay and secret keys can be empty but cannot be null.

**STAGING:**

URL - "https://referenceplatform-ds-platforminfradev.cloud.pcftest.com",

App-Name – Datacore

Shared kay and secret keys can be empty but cannot be null.

**Please find below the code snippet for HSDP initialization:**

**public void** initHSDP() {  
 AppConfigurationInterface.AppConfigurationError configError = **new** AppConfigurationInterface.AppConfigurationError();  
 gAppInfra.  
 getConfigInterface().setPropertyForKey(  
 **"HSDPConfiguration.ApplicationName"**,  
 URConfigurationConstants.**UR**,  
 //"Datacore",  
 **"uGrow"**,  
 configError);  
  
 gAppInfra.  
 getConfigInterface().setPropertyForKey(  
 **"HSDPConfiguration.Secret"**,  
 URConfigurationConstants.**UR**,  
 **"ad3d0618-be4d-4958-adc9-f6bcd01fde16"**,  
 configError);  
  
 gAppInfra.  
 getConfigInterface().setPropertyForKey(  
 **"HSDPConfiguration.Shared"**,  
 URConfigurationConstants.**UR**,  
 **"ba404af2-ee41-4e7c-9157-fd20663f2a6c"**,  
 configError);  
  
 gAppInfra.  
 getConfigInterface().setPropertyForKey(  
 **"HSDPConfiguration.BaseURL"**,  
 URConfigurationConstants.**UR**,  
 **"https://platforminfra-ds-platforminfrastaging.cloud.pcftest.com"**,  
 //"https://referenceplatform-ds-platforminfradev.cloud.pcftest.com",  
 configError);  
}

# Pre Requisites:

1. UserRegisteration initialization: Please refer User registration common component document.
2. AppInfra Initialization :

Create instance of AppInfra and use same instance in whole application.

# Data Base schema

Data-Core has a pre-defined Structure for Moments and consents. Based on the data Structure the Database schema is derived.

The Data-services released for the current PI is implemented using Data-Core API version 4.

**Find below the Data-Core Structure for Moments:**

Moments - A key aspect of data storage within DataCore is the concept of Moments. Moments describe a set of measurements that are part of the user input to the system. A Moment is expressed as a JSON format text string.

{

"details" : [

{

"type" : "Note",

"value" : "Note on a Temperature Moment"

}

],

"measurements" : [

{

"timestamp" : "2015-08-13T09:56:17+0200",

"type" : "Temperature",

"unit" : "celsius",

"value" : 38.90000152587891

}],

"timestamp" : "2015-08-13T09:56:04+0200",

"type" : "Temperature"

}

**Find below the consent data structure:**

Consent - A user consent, using which data core sends the consented data to HSDP.

[

{

"name": "Weight",

"status": "accepted",

"documentVersion": "draft",

"deviceIdentificationNumber":"manual"

},

{

"name": "Sleep",

"status": "refused",

"documentVersion": "draft",

"deviceIdentificationNumber":"manual"

}

]

**Find Below the table definations:**

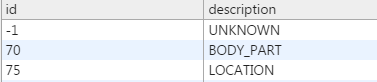
**Measurement:**



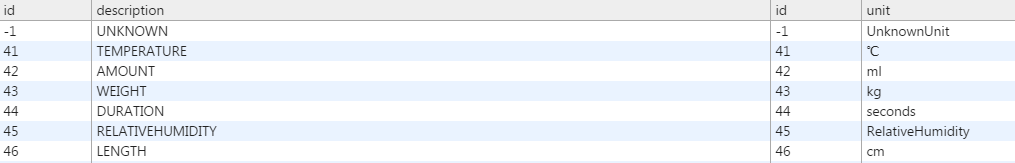
**MeasurementDetail:**



**MeasurementDetailType:**



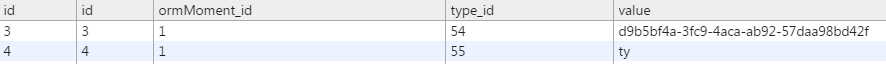
**MeasurementType:**



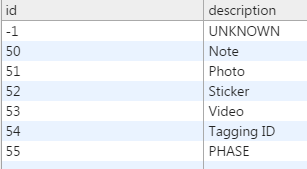
Moment:



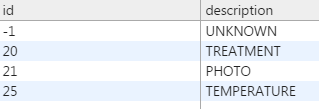
**MomentDetail:**



**MomentDetailType:**



**MomentType:**



**Synchronization Data:**



**Consent:**

Consent table will contain createrID (User uuid) from User-Registartion, Date-Time the consent was created and a primary key (Row-Id).

At any given time consent table will have only one Entry.



**ConsentDetail:**

ConsentDetail table will contain a dirty bit (beSynchronized), deviceIdentificationNumber, status – consented/unconsented, consent type – type of the consent and version – documentVersion



# Data-Services Interfaces to be implemented by Verticals

These are the Data-Services Interfaces that vertical has to implement.

**Database Interfaces**

Following are the interfaces defined by Library for Database Modelling, each corresponding to a data base table. The verticals are supposed to override the same.

* Consent
  + Constructor - consent(String CreaterID)
  + getID()
  + setID(int ID)
  + getCreaterID()
  + getDateTime()
  + Collection<> getConsentDetails()
  + addConsentDetails(ConsentDetail consentDetails)
  + toString()
* ConsentDetail
  + Constructor – consentDetail(final String type, final String status, final String version, final String deviceIdentificationNumber, final consent consent,boolean beSynchronized)
  + Int getID()
  + ConsentDetailType getType()
  + String getStatus()
  + setStatus(final String status)
  + public String getVersion()
  + String getDeviceIdentificationNumber()
  + setBackEndSynchronized(boolean backEndSynchronized)
  + boolean getBackEndSynchronized()
  + setVersion(final String version)
  + setDeviceIdentificationNumber(final String deviceIdentificationNumber)
  + String toString()
* Measurement
  + Constructor – measurement(final OrmMeasurementType type, final OrmMoment ormMoment)
  + int getId()
  + MeasurementType getType()
  + double getValue()
  + void setValue(final double value)
  + DateTime getDateTime()
  + void setDateTime(final @NonNull DateTime date)
  + Collection<? extends OrmMeasurementDetail> getMeasurementDetails()
  + addMeasurementDetail(final MeasurementDetail measurementDetail)
  + Moment getMoment()
  + String toString()
* MeasurementDetail
  + Constructor - MeasurementDetail(final MeasurementDetailType type, final Measurement ormMeasurement)
  + int getId()
  + MeasurementDetailType getType()
  + String getValue()
  + setValue(final String value)
  + Measurement getMeasurement()
  + Measurement getMeasurement()
* Moment
  + Moment(@NonNull final String creatorId, @NonNull final String subjectId, @NonNull final MomentType type)
  + int getId()
  + setId(final int id)
  + String getAnalyticsId()
  + String getCreatorId()
  + String getSubjectId()
  + MomentType getType()
  + DateTime getDateTime()
  + setDateTime(@NonNull final DateTime dateTime)
  + Collection<? extends Measurement> getMeasurements()
  + void addMeasurement(Measurement var1);
  + Collection<? extends MomentDetail> getMomentDetails();
  + void addMomentDetail(MomentDetail var1);
  + SynchronisationData getSynchronisationData();
  + void setSynchronisationData(SynchronisationData var1);
  + void setSynced(boolean var1);
  + void setId(int var1);
  + String getAnalyticsId();
* MomentDetail
  + MomentDetailType getType();
  + String getValue();
  + void setValue(String var1);
  + Moment getMoment();
* SynchronisationData
  + String getGuid();
  + boolean isInactive();
  + DateTime getLastModified();
  + int getVersion();
  + void setVersion(int var1);
  + void setInactive(boolean var1);

**Database Creater**

BaseAppDataCreator interface defines the APIs that have to implemented by verticals for creating the database entries

This implementation is responsible for creating the above interfaces defined.

* + Moment createMoment(@NonNull String var1, @NonNull String var2, @NonNull MomentType var3);
  + Moment createMomentWithoutUUID(@NonNull String var1, @NonNull String var2, @NonNull MomentType var3);
  + MomentDetail createMomentDetail(@NonNull MomentDetailType var1, @NonNull Moment var2);
  + Measurement createMeasurement(@NonNull MeasurementType var1, @NonNull Moment var2);
  + MeasurementDetail createMeasurementDetail(@NonNull MeasurementDetailType var1, @NonNull Measurement var2);
  + SynchronisationData createSynchronisationData(@NonNull String var1, boolean var2, @NonNull DateTime var3, int var4);
  + Consent createConsent(@NonNull String var1);
  + ConsentDetail createConsentDetail(@NonNull ConsentDetailType var1, @NonNull String var2, @NonNull String var3, String var4, boolean var5, @NonNull Consent var6);

# Database Interfaces sent to Library as callbacks

The following interfaces are provided by Library which in turn has to be implemented by vertical and injected to Library for calling DB related queries.

**DBDeletingInterface**

The following APIs are expected to be implemented by vertical:

* **void** deleteAllMoments();
* **void** deleteMoment(Moment var1);
* **void** ormDeletingDeleteMoment(Moment var1);

**DBFetchingInterface**

The following APIs are expected to be implemented by vertical:

* **void** fetchMoments() **throws** SQLException;
* **void** fetchMoments(@NonNull MomentType var1) **throws** SQLException;
* **void** fetchMoments(@NonNull MomentType... var1) **throws** SQLException;
* **void** fetchLastMoment(MomentType var1) **throws** SQLException;
* Object fetchMomentByGuid(@NonNull String var1) **throws** SQLException;
* List<?> fetchNonSynchronizedMoments() **throws** SQLException;
* Object fetchMomentById(**int** var1) **throws** SQLException;
* Map<Class, List<?>> putMomentsForSync(Map<Class, List<?>> var1) **throws** SQLException;

**DBSavingInterface**

The following APIs are expected to be implemented by vertical:

* **boolean** saveMoment(Moment var1) **throws** SQLException;

**DBUpdatingInterface**

The following APIs are expected to be implemented by vertical:

* **int** processMoment(**int** var1, Moment var2);
* **void** updateOrSaveMomentInDatabase(Moment var1);
* Moment getOrmMoment(Moment var1);
* **int** processMomentsReceivedFromBackend(List<? **extends** Moment> var1);
* **void** updateFailed(Exception var1);
* **void** postRetrofitError(Throwable var1);

# Setting up Synchronization Loop

Vertical propositions are expected to set up the Synchronization Loop. In the Demo-App “BroadcastReceiver” is used to start the synchronization loop after every fixed time interval.

The vertical proposition can specify the interval the sync should happen.

In the Synchronization loop, vertical app should call “**mDataServices**.synchchronize()”.

Library will take care of syncing the data.

# Handling Token Expiry

Vertical propositions are expected to implement “UserRegistrationFacade” and inject to library.

Library uses HSDP token from User-Registration. In case the token expires, the call to Data-Core fails. Thus, sending appropriate error message to vertical proposition.

It’s the vertical proposition’s responsibility to give back a valid token to library.

The Data-Services Demo-App contains a reference implementation for the same (UserRegistrationFacadeImpl).

# DataServicesManager

This is an Interface exposed to Vertical propositions for making calls to Data-Services library.

# DataFetcher

This is an abstract class. In case Vertical propositions want to inject the custom fetchers, they can override this class and inject to Library.

Library will use the injected fetcher for pulling data from Data-Core during the sync cycle.

# DataSender

This is an interface exposed to Vertical propositions. In case Vertical propositions want to inject the custom sender, they can implement this class and inject to Library.

Library will use the injected sender for sending data from Application during the sync cycle.

# INITIALIZATION

* Initialize the Data Core Library
  + **mDataServicesManager**.initialize(getContext(), creator, **new** UserRegistrationFacadeImpl(getContext(), **new** User(getContext())));
* **Initialize DB Monitors**
  + **mDataServicesManager**.initializeDBMonitors(ORMDeletingInterfaceImpl,dbInterfaceOrmFetchingInterface,ORMSavingInterfaceImpl,dbInterfaceOrmUpdatingInterface);
* Initialize Sync Monitors
  + **mDataServicesManager**.initializeSyncMonitors(**null**,**null**);

For more details please refer Interface Document

# Reference APP link

# <https://atlas.natlab.research.philips.com/stash/scm/ip/in-app-purchase_android.git>

# Notes

1. Registration is developed as separate library project. App is expected to invoke registration library before calling Data-Core APIs.
2. Library should be initialized as per document or sample app otherwise library would throw runtime exception.
3. Please refer sample application for more details
4. All dependencies can be referred as it is.
5. Configuration can be followed [all sample configurations are provided in sample]

# References:

1. Architecture Document
2. Interface Document
3. PPT on Consents
4. PPT on Moments
5. Postman scripts