Integration Document- DataServices (Android) – PI-17.1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document History** | | | | |
| **Version** | **Date** | **Author** | **Section** | **Changes** |
| 1.0 | 12/02/2016 | Spoorti Hallur | All | Release for 16.5 |
| 0.2 | 02-09-2017 | Spoorti Hallur | All | Release for 17.1 |
| 0.3 | 04-07-2017 | Spoorti Hallur | All | Platform\_release\_2.1.1 |
| 1.0 | 04-07-2017 | Chithra Boopalan | Device pairing, Insights, Push notifications | New API’s added |
| 1.1 | 12-09-2017 | Arthy Vijayaraghavan Paulo Caetano | Database schema, Synchronization Loop | Moment datamodel updated |

|  |  |  |  |
| --- | --- | --- | --- |
| Administrative Information | | | |
| Project Name | CoCo Data Services | Project Identification | NA |
| Scrum Master | Mahabaleshwara Adiga | Date | 7/20/2017 |
| Quality Leader | Vinay Nair | Form Filled by | Chithra Boopalan |
| Name  Address  Phone  Fax | All Projects in CDP2 | | |

|  |  |
| --- | --- |
| **Authorization** | |
| Quality Leader Signature | Scrum Master Signature |
| Vinay Nair | Mahabaleshwara Adiga |

**For Code:**

**Product : Data-Services**

**Release Version :** **2.2.0 (Android)**

**Release Date: 20-07-2017**

User manual (Optional): NA Customer Version ID (Optional): NA

Contents

[**Authorization** 1](#_Toc487124019)

[**Introduction** 3](#_Toc487124020)

[**Prerequisites** 3](#_Toc487124021)

[**Library Integration** 4](#_Toc487124022)

[**Maven repository Integration** 4](#_Toc487124023)

[**Library Integration** 5](#_Toc487124024)

[**OrmLite Library (Object Relational Mapping Lite)** 6](#_Toc487124025)

[**User Registration** 6](#_Toc487124026)

[**HSDP Login** 6](#_Toc487124027)

[**Service Discovery** 7](#_Toc487124028)

[**Data Base schema** 8](#_Toc487124029)

[**Characteristics:** 10](#_Toc487124030)

[**ConsentDetail:** 10](#_Toc487124031)

[**dcSync:** 10](#_Toc487124032)

[**Measurement:** 11](#_Toc487124033)

[**MeasurementDetail**: 11](#_Toc487124034)

[**MeasurementDetailType:** 11](#_Toc487124035)

[**MeasurementGroup:** 11](#_Toc487124036)

[**MeasurementGroupDetail:** 11](#_Toc487124037)

[**MeasurementGroupDetailType:** 11](#_Toc487124038)

[**Data-Services Interfaces to be implemented by Verticals** 12](#_Toc487124039)

[**Database Interfaces** 12](#_Toc487124040)

[**Characteristics** 12](#_Toc487124041)

[**ConsentDetail** 12](#_Toc487124042)

[**DCSync** 13](#_Toc487124043)

[**Measurement** 13](#_Toc487124044)

[**MeasurementDetail** 13](#_Toc487124045)

[**MeasurementGroup** 13](#_Toc487124046)

[**MeaurementGroupDetial** 13](#_Toc487124047)

[**Moment** 13](#_Toc487124048)

[**MomentDetail** 14](#_Toc487124049)

[**Settings** 14](#_Toc487124050)

[**SynchronisationData** 14](#_Toc487124051)

[**User Insights** 14](#_Toc487124052)

[**Push Notification** 14](#_Toc487124053)

[**Device Pairing** 15](#_Toc487124054)

[**DBDeletingInterface** 16](#_Toc487124055)

[**DBFetchingInterface** 16](#_Toc487124056)

[**DBSavingInterface** 17](#_Toc487124057)

[**DBUpdatingInterface** 17](#_Toc487124058)

[**Setting up Synchronization Loop** 17](#_Toc487124059)

[**Handling Token Expiry** 18](#_Toc487124060)

[**DataServicesManager** 18](#_Toc487124061)

[**DataFetcher** 18](#_Toc487124062)

[**DataSender** 18](#_Toc487124063)

[**Initialization** 18](#_Toc487124064)

[Reference **APP** link 19](#_Toc487124065)

[**Configuring SyncTypes** 19](#_Toc487124066)

[**Way of using Library** 19](#_Toc487124067)

[**Notes** 20](#_Toc487124068)

[**References**: 20](#_Toc487124069)

# **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

1. DB interfaces for storing Data coming from Data-Core.
2. User-Registration Interface for handling token expiry
3. ErrorHandlerInterface for Handling Sync Error
4. Additionally Verticals can register for “SynchronisationCompleteListener” for receiving callback for SyncComplete and SyncFail

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

**Source Path:** <http://tfsemea1.ta.philips.com:8080/tfs/TPC_Region24/CDP2/_git/dsc-android-dataservices>

# **Prerequisites**

1. Vertical project is configured for Android Studio
2. Setting->Developer Options->Don’t Keep Activities should be unchecked. [If Developer Mode is on]
3. Android API version should on 19



# **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:

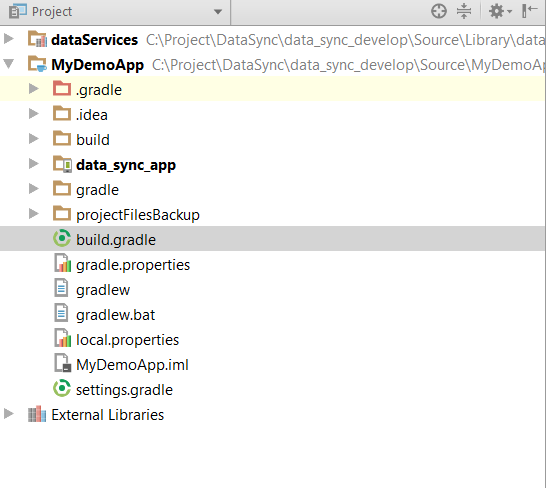
allprojects {  
 repositories {  
 maven { url 'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/jcenter' }  
 maven {  
 url 'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/ext-release-local'  
 }  
 maven {  
 url 'http://maartens-mini.ddns.htc.nl.philips.com:8081/artifactory/libs-release-local-android'  
 }  
 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 :

<http://tfsemea1.ta.philips.com:8080/tfs/TPC_Region24/CDP2/_git/dsc-android-dataservices>

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
4. Please make sure all these are added to application with the latest versions along with Data-Services library aar file under libs folder.
5. Clean project
6. 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:* *8.3.0-SNAPSHOT"  
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"***

}



# **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 – **OneBackend**

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

**public void** initHSDP() {  
 *gAppInfra*.  
 getConfigInterface().setPropertyForKey(  
 **"HSDPConfiguration.ApplicationName"**,  
 URConfigurationConstants.***UR***,  
 loadAppNameFromConfigParams(**APP\_NAME**),  
 **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***,  
 **mDataCoreUrl**,  
 **configError**);  
}

# **Service Discovery**

Service discovery can be used in-order to get the server URLS from cloud.

In the AppConfig.json add the below configuration.

"dataservices": {  
 "appname": "OneBackend"  
}

1. Make sure the App-Infra is initialized
2. Use service discovery API “getServiceUrlWithCountryPreference” for getting the DataServices URL with the key – “ds.dataservice”
3. Please find below the demo code for getting service URL

protected void fetchDataServicesUrl() {  
 serviceDiscoveryInterface.getServiceUrlWithCountryPreference("ds.dataservice", new  
 ServiceDiscoveryInterface.OnGetServiceUrlListener() {  
 @Override  
 public void onError(ERRORVALUES errorvalues, String s) {  
 DSLog.e(DSLog.LOG, "Error");  
 mDataCoreUrl = loadAppNameFromConfigParams(DATACORE\_FALLBACK\_URL);  
 initHSDP();  
 }  
  
 @Override  
 public void onSuccess(URL url) {  
 DSLog.e(DSLog.LOG, "Success = " + url);  
 if (url.toString().isEmpty()) {  
 mDataCoreUrl = loadAppNameFromConfigParams(DATACORE\_FALLBACK\_URL);  
 } else {  
  
 mDataCoreUrl = url.toString();  
 }  
 initHSDP();  
 }  
 });  
}

1. While initializing the HSDP part use service URL from service discover and AppName from AppConfig.json.
2. Find below the code for getting AppName from AppConfig.json

private String loadAppNameFromConfigParams(String propertyKey) {  
 String appname = (String) gAppInfra.getConfigInterface().getPropertyForKey(propertyKey, DATASERVICES\_KEY, configError);  
 if (configError.getErrorCode() != null) {  
 DSLog.e(DSLog.LOG, "VerticalAppConfig ==loadConfigurationFromAsset " + configError.getErrorCode().toString());  
 }  
 return appname;  
}

1. In case the service discovery fails to give the URL, the URL is picked up from AppConfig.json

# **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 9.

**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.

{

                "timestamp": "2018-01-01T07:07:14.000Z",

                "type": "Temperature",

                "details": [{

                                "type": "PHASE",

                                "value": "Morning"

                }],

                "measurementGroups": [{

                                "details": [{

                                                "type": "TEMP\_OF\_DAY",

                                                "value": "temp of day"

                                }],

                                "measurementGroups": [{

                                                "measurements": [{

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

                                                                "type": "Temperature",

                                                                "unit": "celsius",

                                                                "value": 80,

                                                                "details": [{

                                                                                "type": "LOCATION",

                                                                                "value": "Bangalore"

                                                                }]

                                                }]

                                }]

                }]

}

**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 UserCharacteritics data structure:**

UserCharacteristics – For storing user characteristics

{

"characteristics": [

{

"type": "User",

"value": "John",

"characteristics": [

{

"type": "Mouth",

"value": "Upper Teeth",

"characteristics": [

{

"type": "BrokenTeeth",

"value": "1,2,3,4,5",

"characteristics": []

}

]

},

{

"type": "Mouth",

"value": "Lower Teeth",

"characteristics": [

{

"type": "BrokenTeeth",

"value": "6,7,8,9",

"characteristics": []

}

]

}

]

}

]

}

**Find below the Settings data structure:**

UserSetings – For setting locale and unitSystem

{

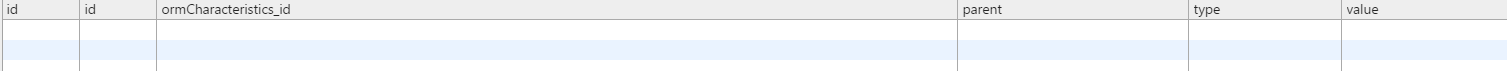
"locale": "en\_US",

"unitSystem": "metric"

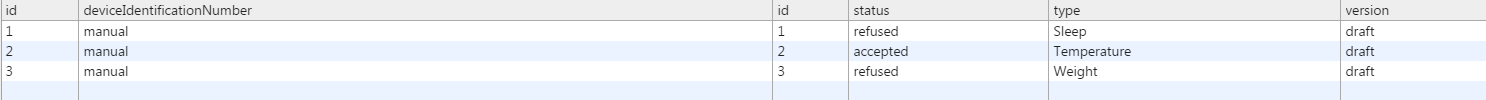
}

**Find Below the table definitions:**

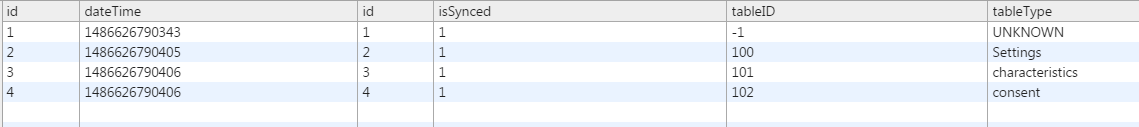
### **Characteristics:**



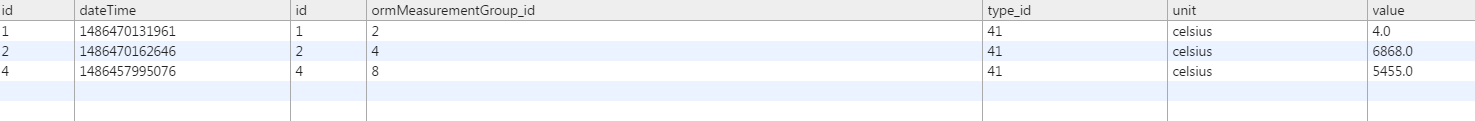
### **ConsentDetail:**



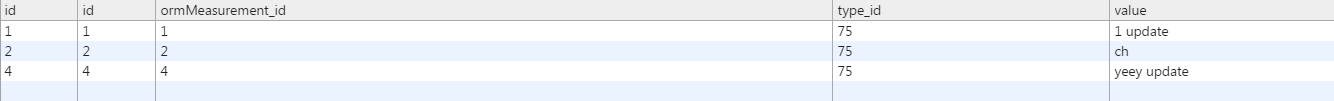
### **dcSync:**



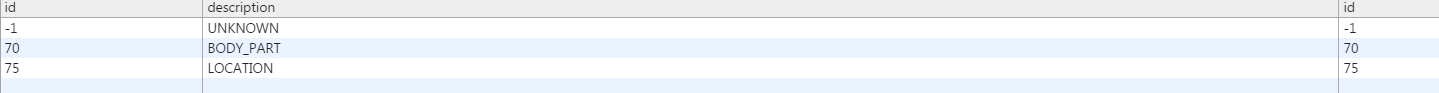
### **Measurement:**



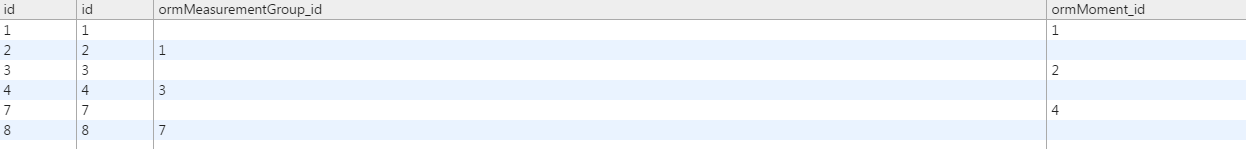
### **MeasurementDetail**:



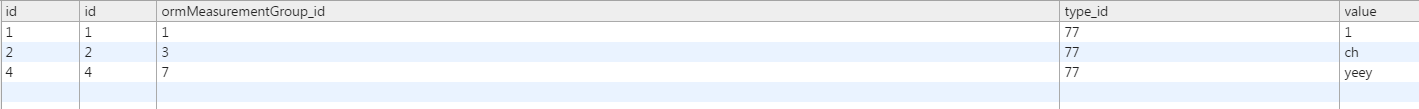
### **MeasurementDetailType:**



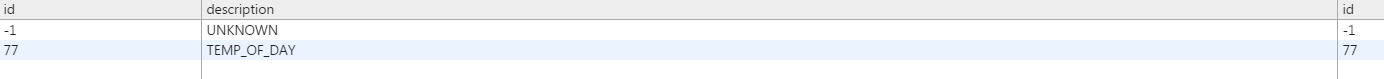
### **MeasurementGroup:**



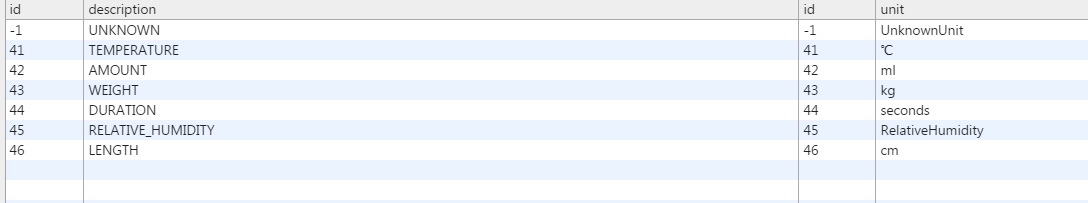
### **MeasurementGroupDetail:**



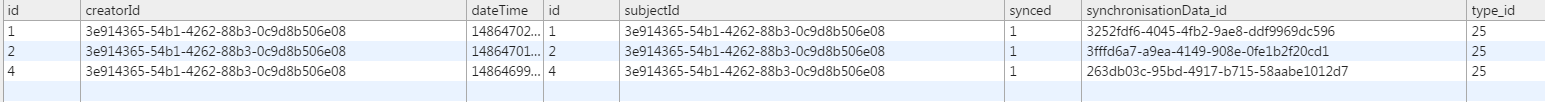
### **MeasurementGroupDetailType:**



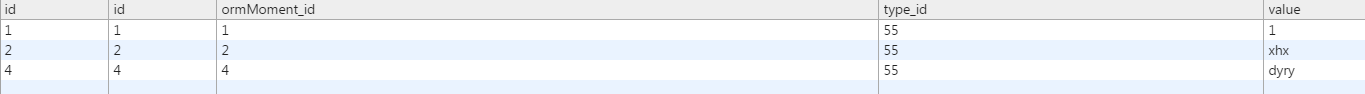
**MeasurementType**



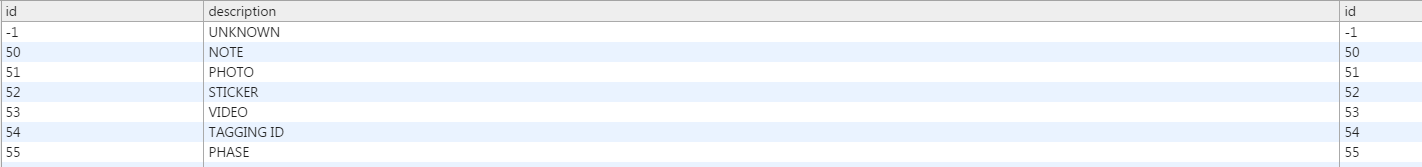
**Moment**



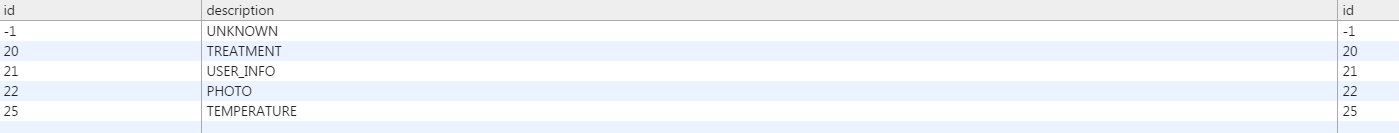
**MomentDetail**



**MomentDetailType**

****

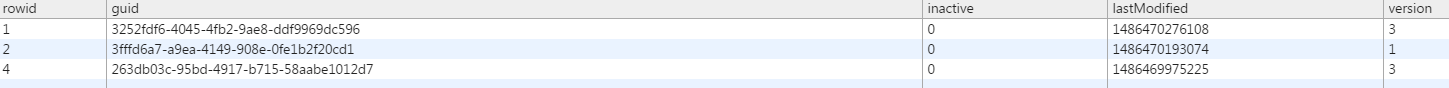
**MomentType**

****

**Settings**

****

**SynchronizationData**

****

### **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.

### **Characteristics**

1. void setType(String type);
2. String getType();
3. void setValue(String value);
4. String getValue();
5. Collection<? extends Characteristics> getCharacteristicsDetail();
6. void setCharacteristicsDetail(Characteristics characteristics);
7. int getParent();

### **ConsentDetail**

1. void setStatus(String status);
2. void setVersion(String version);
3. void setDeviceIdentificationNumber(String deviceIdentificationNumber);
4. String getType();
5. String getStatus();
6. String getVersion();
7. String getDeviceIdentificationNumber();

### **DCSync**

1. String getTableType();
2. boolean isSynced();

### **Measurement**

1. int getId();
2. String getType();
3. String getValue();
4. void setValue(String value);
5. DateTime getDateTime();
6. void setDateTime(DateTime dateTime);
7. Collection<? extends com.philips.platform.core.datatypes.MeasurementDetail> getMeasurementDetails();
8. void addMeasurementDetail(com.philips.platform.core.datatypes.MeasurementDetail measurementDetail);
9. MeasurementGroup getMeasurementGroup();
10. String getUnit();
11. void setUnit(String unit);

### **MeasurementDetail**

1. String getType();
2. String getValue();
3. void setValue(String value);
4. Measurement getMeasurement();

### **MeasurementGroup**

1. Collection<? extends Measurement> getMeasurements();
2. public Collection<? extends MeasurementGroup> getMeasurementGroups();
3. void addMeasurement(Measurement measurement);
4. void addMeasurementGroup(MeasurementGroup measurementGroup);
5. public Collection<? extends MeasurementGroupDetail> getMeasurementGroupDetails();
6. public void addMeasurementGroupDetail(final MeasurementGroupDetail measurementGroupDetail);
7. int getId();
8. void setId(int id);
9. void setMeasurementGroups(Collection<? extends MeasurementGroup> groups);

### **MeaurementGroupDetial**

1. public String getType();
2. public String getValue();
3. public void setValue(final String value);
4. public MeasurementGroup getOrmMeasurementGroup();

### **Moment**

1. String getCreatorId();
2. String getSubjectId();
3. String getType();
4. void setDateTime(@NonNull DateTime dateTime);
5. Collection<? extends MeasurementGroup> getMeasurementGroups();
6. void addMeasurementGroup(MeasurementGroup measurementGroup);
7. Collection<? extends com.philips.platform.core.datatypes.MomentDetail> getMomentDetails();
8. void addMomentDetail(com.philips.platform.core.datatypes.MomentDetail momentDetail);
9. SynchronisationData getSynchronisationData();
10. void setSynchronisationData(com.philips.platform.core.datatypes.SynchronisationData synchronisationData);
11. void setSynced(boolean b);
12. void setId(int id);
13. String getAnalyticsId();
14. DateTime getExpirationDate();

### **MomentDetail**

1. String getType();
2. String getValue();
3. void setValue(String value);
4. Moment getMoment();

### **Settings**

1. String getUnit();
2. String getLocale();
3. void setUnit(String unit);
4. void setLocale(String locale);
5. int getId();
6. void setID(int id);

### **SynchronisationData**

1. String getGuid();
2. boolean isInactive();
3. DateTime getLastModified();
4. int getVersion();
5. void setVersion(int version);
6. void setInactive(boolean inActive);
7. void setGuid(String guid);

### **User Insights**

Insights are generated with respect to the moment temperature. The parameters of the insight response are insightId, momentId, ruleId, lastModified, version, subjectId, insightType, timestamp, progMinVersion, progMaxVersion, title, avg, min, max, metadata

**API exposed for user insights**

1. **fetchInsights**(DBFetchRequestListner dbFetchRequestListner); - Using this api call the synced insights can be fetched from the database. The list of insight will be passed in the onFetchSuccess callback of the DBFetchRequestListner interface.
2. **deleteInsights**(List<? **extends** Insight> insights, DBRequestListener<Insight> dbRequestListener); - Using this api call the requested insights can be deleted.

### **Push Notification**

Push notification feature provides the api to register and unregister the device to get notification from the one backend**.**

**API exposed for push notification**

1. **registerDeviceToken**(String deviceToken, String appVariant, String protocolProvider, RegisterDeviceTokenListener registerDeviceTokenListener); - with this api, the device can register to receive the push notification when the moments are generated
2. **unRegisterDeviceToken**(String appToken, String appVariant, RegisterDeviceTokenListener registerDeviceTokenListener); - With this api, the device can un register from receiving the notification from the server.
3. **handlePushNotificationPayload**(JSONObject jsonObject); - The Json payload received from the push notification tells which component should handle the notification. In case of dsc, the database will be synchronized.

**RegisterDeviceTokenListener** – With this interface listener, the call back can be received on the status of the call made to the register/unregister.

### **Device Pairing**

This feature provides the api to pair and unpair the device to the one backend.

**API exposed for device pairing**

1. **pairDevices**(String deviceID, String deviceType, List<String> subjectIds,  
    List<String> standardObservationNames, String relationshipType, DevicePairingListener devicePairingListener); - With this api the device can be paired with user so that the device can send moments to the server
2. **unPairDevice**(String deviceID, DevicePairingListener devicePairingListener); - With this api, the device can be unpaired
3. **getPairedDevices**(DevicePairingListener devicePairingListener); - with this api, list of devices paired to the user can be fetched.

**DevicePairingListener** - With this interface listener, the call back can be received on the status of the call made.

**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.

1. Moment createMoment(@NonNull final String creatorId, @NonNull final String subjectId, @NonNull final String type, DateTime expirationDate);
2. MomentDetail createMomentDetail(@NonNull final String type, @NonNull final Moment moment);
3. Measurement createMeasurement(@NonNull final String type, @NonNull final MeasurementGroup measurementGroup);
4. MeasurementDetail createMeasurementDetail(@NonNull final String type, @NonNull final Measurement measurement);
5. MeasurementGroup createMeasurementGroup(@NonNull final MeasurementGroup measurementGroup);
6. MeasurementGroup createMeasurementGroup(@NonNull final Moment moment);
7. MeasurementGroupDetail createMeasurementGroupDetail(@NonNull final String type, @NonNull final MeasurementGroup measurementGroup);
8. SynchronisationData createSynchronisationData(@NonNull final String guid, final boolean inactive, @NonNull final DateTime lastModifiedTime, final int version);
9. ConsentDetail createConsentDetail(@NonNull final String type, @NonNull final String status, @NonNull final String version, final String deviceIdentificationNumber);
10. Settings createSettings(String type, String value);
11. Characteristics createCharacteristics(@NonNull final String type, @NonNull final String value, @NonNull final Characteristics characteristics);
12. Characteristics createCharacteristics(@NonNull final String type, @NonNull final String value);

**Database Interfaces**

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:

1. void deleteAll(DBRequestListener dbRequestListener) throws SQLException;
2. void markAsInActive(Moment moment,DBRequestListener dbRequestListener) throws SQLException;
3. void deleteMoment(Moment moment,DBRequestListener dbRequestListener) throws SQLException;
4. void deleteMoments(List<Moment> moments, DBRequestListener dbRequestListener) throws SQLException;
5. void deleteMomentDetail(Moment moment, DBRequestListener dbRequestListener) throws SQLException;
6. void deleteMeasurementGroup(Moment moment,DBRequestListener dbRequestListener) throws SQLException;
7. void deleteFailed(Exception e,DBRequestListener dbRequestListener);
8. void deleteAllMoments(DBRequestListener dbRequestListener) throws SQLException;
9. public void markMomentsAsInActive(final List<Moment> moment, DBRequestListener dbRequestListener) throws SQLException;
10. void deleteAllExpiredMoments(DBRequestListener<Integer> dbRequestListener) throws SQLException;

### **DBFetchingInterface**

The following APIs are expected to be implemented by vertical:

1. void fetchMoments(DBRequestListener dbRequestListener) throws SQLException;
2. void fetchMoments(final @NonNull String type ,DBRequestListener dbRequestListener) throws SQLException;
3. void fetchMoments(DBRequestListener dbRequestListener,final @NonNull Object... types) throws SQLException;
4. void fetchLastMoment(final String type,DBRequestListener dbRequestListener) throws SQLException;
5. Object fetchMomentByGuid(@NonNull final String guid) throws SQLException;
6. List<?> fetchNonSynchronizedMoments() throws SQLException;
7. Object fetchMomentById(final int id, DBRequestListener dbRequestListener) throws SQLException;
8. void fetchConsentDetails(DBRequestListener dbRequestListener) throws SQLException;
9. Map<Class, List<?>> putUserCharacteristicsForSync(Map<Class, List<?>> dataToSync) throws SQLException;
10. List<?> fetchNonSyncConsentDetails() throws SQLException;
11. List<?> fetchConsentDetails() throws SQLException;
12. void postError(Exception e, DBRequestListener dbRequestListener);
13. void fetchUserCharacteristics(DBRequestListener dbRequestListener) throws SQLException;
14. Settings fetchSettings(DBRequestListener dbRequestListener) throws SQLException;
15. Settings fetchSettings() throws SQLException;
16. List<?> fetchNonSyncSettings() throws SQLException;
17. boolean isSynced(int tableID) throws SQLException

### **DBSavingInterface**

The following APIs are expected to be implemented by vertical:

1. boolean saveMoment(final Moment moment, DBRequestListener dbRequestListener) throws SQLException;
2. boolean saveMoments(List<Moment> moments, DBRequestListener dbRequestListener) throws SQLException;
3. boolean saveConsentDetails(final List<ConsentDetail> consentDetails, DBRequestListener dbRequestListener) throws SQLException;
4. void postError(Exception e, DBRequestListener dbRequestListener);
5. boolean saveUserCharacteristics(final List<Characteristics> userCharacteristics, DBRequestListener dbRequestListener) throws SQLException;
6. boolean saveSettings(final Settings settings, DBRequestListener dbRequestListener) throws SQLException;

### **DBUpdatingInterface**

The following APIs are expected to be implemented by vertical:

1. void updateMoment(final Moment ormMoment, DBRequestListener dbRequestListener) throws SQLException;
2. void updateMoments(final List<Moment> ormMoments, DBRequestListener dbRequestListener) throws SQLException;
3. boolean updateConsent(final List<? extends ConsentDetail> consentDetails, DBRequestListener dbRequestListener) throws SQLException;
4. void updateFailed(Exception e, DBRequestListener dbRequestListener);
5. boolean updateCharacteristics(final List<Characteristics> userCharacteristics, DBRequestListener dbRequestListener) throws SQLException;
6. void updateSettings(Settings settings, DBRequestListener dbRequestListener) throws SQLException;
7. boolean updateSyncBit(int tableID,boolean isSynced) throws SQLException;

# **Setting up Synchronization Loop**

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

The proposition can specify the interval the sync should happen.

In the Synchronization loop, vertical app should call “synchronize()”.

Library will take care of syncing the data.

In case Propositions are not interested to sync the data periodically, they can call “synchronize()” API from DataServices anytime they wish to Synchronize.

During the sync, the expired moments are deleted from the local app DB.

**Call-back for Sync-Complete and Sync-Fail**

Propositions can either pass the “SynchronisationCompleteListener” through “initializeSyncMonitors” or register and unregister the “initializeSyncMonitors” through UI.

**Find below the interface that has to be implemented and passed to Library.**

**public interface** SynchronisationCompleteListener {  
 **void** onSyncComplete();  
 **void** onSyncFailed(Exception exception);  
}

**Following are the APIs for registering and unregistering the SynchronisationCompleteListener**

**public void** registerSynchronisationCompleteListener(SynchronisationCompleteListener synchronisationCompleteListener);  
**public void** unRegisterSynchronisationCosmpleteListener();

# **Handling Token Expiry**

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

Library uses “UserRegistrationInterfaceImpl” for getting the hsdp token. The responsibility of giving the right token to library lies within “UserRegistrationInterfaceImpl”. In case the token expires, Library again makes a call to “UserRegistrationInterfaceImpl” for getting the new access token.

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**

Find below the reference code for initializing Dataservices Library.

**private void init() {  
 Set fetchList = new HashSet();  
 fetchList.add(SyncType.MOMENT.getDescription());  
 mDataServicesManager.configureSyncDataType(fetchList);  
  
 OrmCreator creator = new OrmCreator(new UuidGenerator());  
 userRegImple = new UserRegistrationInterfaceImpl(this, new User(this));  
 UserRegistrationInterface userRegistrationInterface = userRegImple;  
 ErrorHandlerInterfaceImpl errorHandlerInterface = new ErrorHandlerInterfaceImpl();  
 mDataServicesManager.initializeDataServices(this, creator, userRegistrationInterface, errorHandlerInterface);  
 injectDBInterfacesToCore();  
 mDataServicesManager.initializeSyncMonitors(this, null, null);  
}**

1. Initialize the Data Core Library
   1. **mDataServicesManager**.initializeDataServices(**this**, creator, userRegistrationInterface, errorHandlerInterface);
2. Initialize DB Monitors
   1. **mDataServicesManager.initializeDatabaseMonitor(this, ORMDeletingInterfaceImpl, dbInterfaceOrmFetchingInterface, ORMSavingInterfaceImpl, dbInterfaceOrmUpdatingInterface);**
3. Initialize Sync Monitors
   1. **mDataServicesManager**.initializeSyncMonitors(**this**, **null**, **null**,synchronisationCompleteListener);

For more details please refer Interface Document

### Reference **APP** link

<http://tfsemea1.ta.philips.com:8080/tfs/TPC_Region24/CDP2/_git/rap-android-reference-app>

# **Configuring SyncTypes**

Propositions can configure the DataType they are interested to sync. With this, only those data types that propositions are interested will be synced.

* 1. Propositions can invoke:

**Set fetchList = new HashSet();  
fetchList.add(SyncType.MOMENT.getDescription());  
mDataServicesManager.configureSyncDataType(fetchList);**

**Please Note:** Make sure to configure the sync types before call to “initializeSyncMonitors”.

# **Way of using Library**

1. All the DB Operations are expected to go through Library. For Ex: fetch(), update(), delete(), save(), create()
2. A listener is passed to all the Calls.
3. Using the Listener, the call-back is passed to Propositions, one’s the job is complete.
4. In case any change happens in DB, the DBChangeListener is used to notify the UI.
5. The Propositions can register and unregister to DBChange.
6. The Propositions can register and unregister “synchronisationCompleteListener” in-order to get call-backs for Sync-Fail and Sync-Success.
7. The Propositions can configure which type of Data they want to sync.

**Find Below the call-back for DB Operations:**

The below callback is used to notify the propositions in case of DB operation complete or failure. Eg: Delete, Update, Create operations – onSuccess or onFailure.

onSuccess – returns a dataset based on the operation performed.

onFailure – returns the SqlException

public interface DBRequestListener<T> {  
 void onSuccess(List<? extends T> data);  
 void onFailure(Exception exception);  
}

**Find below the call-back for DB Change:**

The below should be registered an unregistered by Verticals for receiving notifications for data change on server. Refer Data-Services Demo-App for More details. FileName: “TemperatureTimeLineFragment”

public interface DBChangeListener {  
 void dBChangeSuccess(SyncType type);  
 void dBChangeFailed(Exception e);  
}

**Find below the call-back for Fetch Operations:**

The below listener should be passed by Verticals while fetching data from Data-Base using Data-Services Library. Refer Data-Services Demo-App for More details. FileName: “TemperatureTimeLineFragment”

public interface DBFetchRequestListner<T> {  
 void onFetchSuccess(List<? extends T> data);  
 void onFetchFailure(Exception exception);  
}

**Find below the call-back for SynchronisationCompleteListener:**

The below should be registered an unregistered by Verticals for receiving notifications for sync cycle completion from library. Refer Data-Services Demo-App for More details. FileName: “TemperatureTimeLineFragment”

public interface SynchronisationCompleteListener {  
 void onSyncComplete();  
 void onSyncFailed(Exception exception);  
}

# **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