Health

Programming Guide

Version 1.0

Revision History

Version	Date	Description
1.0		

Table of Contents

1. Overview

Health allows you to develop applications that use the health data created by the S-Health application. Health supports a number of health, medical, and fitness devices to measure and store the end user health data. You can use the medical devices to receive measurement data.

You can use Health to:

- Provide interfaces to read and update the end user health data.
- Connect to health sensor devices to get the end user health and fitness information.

1.1. Basic Knowledge

Health content data is exposed through the Android content provider, which can be accessed by multiple applications in parallel. The content data can be synced to a PHR server periodically to make the content available in other eco systems. You can also access user profile information.

Sensor service supports multiple sensor devices, such as Blood pressure, Blood glucose, Heart Rate Monitor, Pedometer, GPS, and Fitness Equipment. Some devices provide continuous stream data, such as steps and heart rate, while others give discrete data, such as blood pressure and glucose. Multiple applications can receive data from one sensor at a time. An application can take control of a sensor to start, stop, and issue specific commands to the sensor.

1.2. Architecture

The following figure shows the Health architecture.

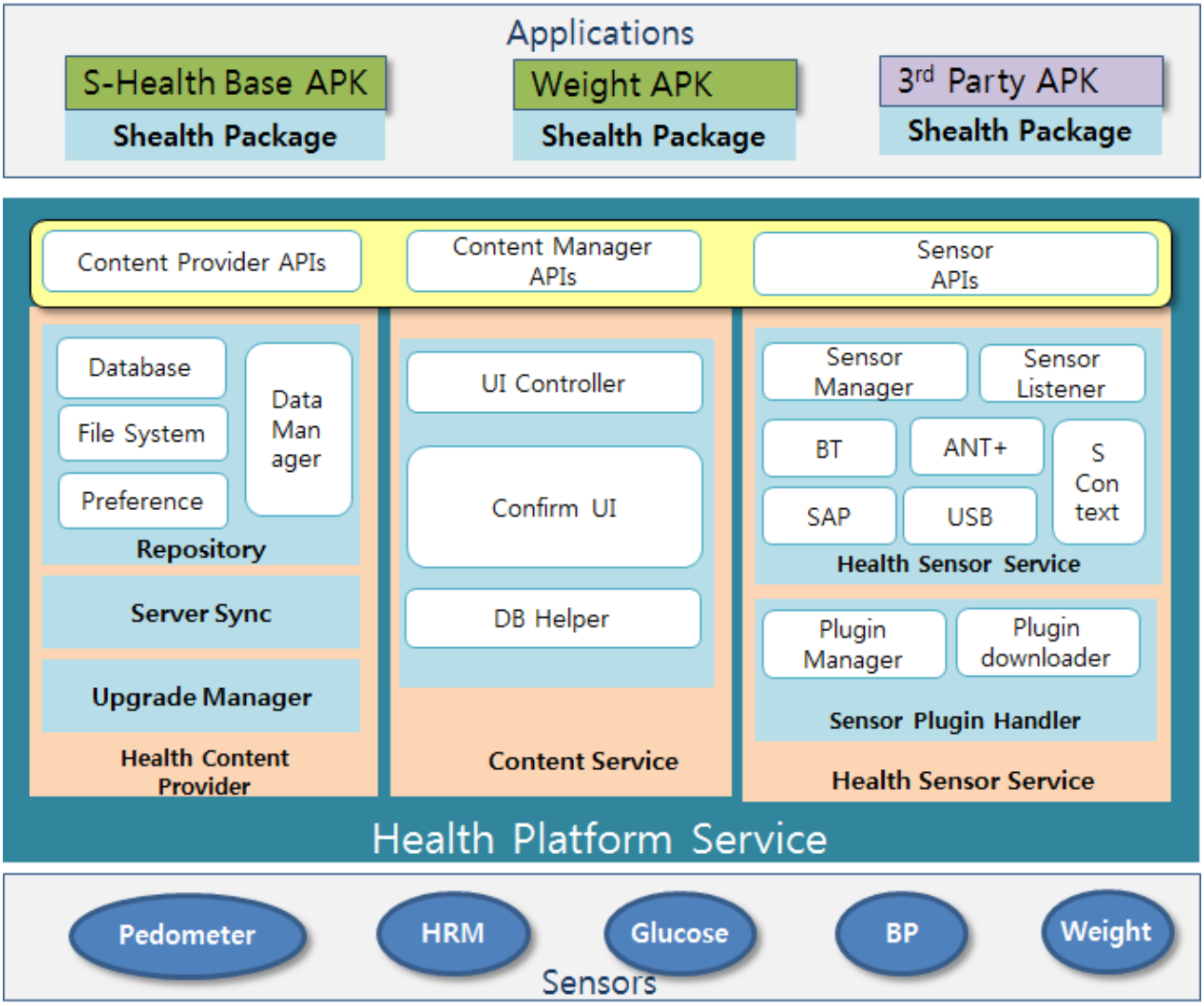


Figure 1: Health architecture

The architecture consists of:

- **Health Content Provider:** Component providing health content; contains modules to upgrade the old database and sync the data to the server.
- **Health Sensor Service:** Service component for communication with various sensor services.
- **Content Service:** Service component for showing a confirmation UI before inserting, updating, and deleting data.

1.3. Class Diagram

The following figure shows the Health classes and interfaces that you can use in your application.

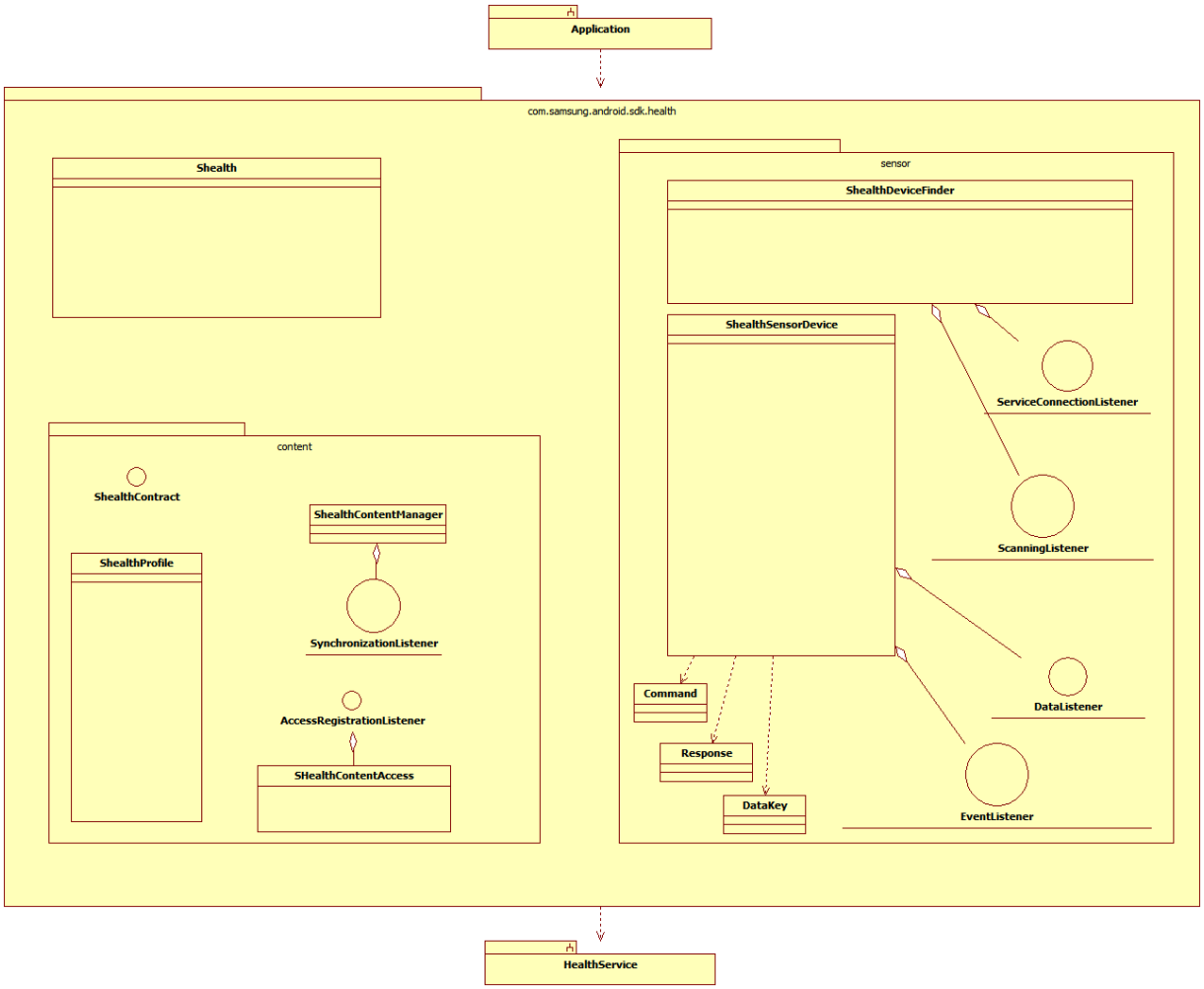


Figure 2: Health classes and interfaces

The Health classes and interfaces include:

- **Shealth:** Provides methods for version and feature check.
- **ShealthContentManager:** Provides the health data through the Android content resolver. Also provides sync methods to sync data to the server.
- **ShealthProfile:** Provides end user profile information.
- **ShealthContract:** Provides contract classes to get the data from the content manager.
- **ShealthDeviceFinder:** Searches for sensors.
- **ShealthSensorDevice:** Provides communication with the sensor.

1.4. Supported Platforms

Android 4.3 (Jellybean API 18) or higher + Gallexy S5 or higher device that support Health.

1.5. Supported Features

Health supports the following features:

- Methods for reading, updating, inserting, and deleting Health content data
- Syncing the health data with the server
- Connecting to different types of sensors and receiving measurement data:
    - Supports Bluetooth, USB, ANT+, SAP, and internal sensors
    - Supports sensors for blood pressure, glucose, weigh scale, HRM, pedometer, GPS, and fitness equipment

1.6. Components

- Components
    - healthsdk.jar
- Imported packages:
    - com.samsung.android.sdk.health

1.7. Installing the Package for Eclipse

To install Health for Eclipse:

1. Add the healthsdk.jar file to the libs folder in Eclipse.

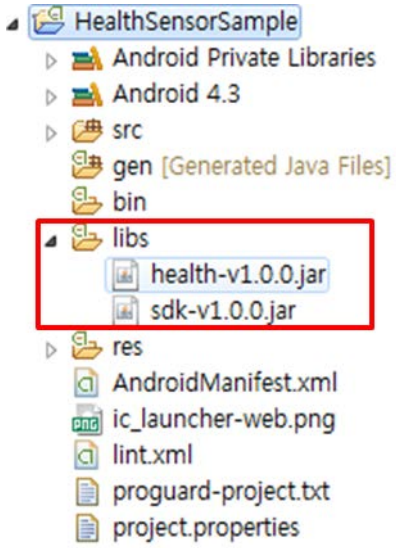


Figure 3: libs folder in Eclipse

2. Hello Health

Hello Health is a simple program that:

2. Scans for a particular sensor.
3. Joins with that sensor.
4. Starts the sensor to receive data.

```
public class SensorActivity extends Activity
{
	private static final String TAG = DummySensorActivity.class.getSimpleName();
	private ShealthDeviceFinder mHealthSensor;
	private static Context context;

	private ShealthSensorDevice mDevice;

	public void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		setContentView(R.layout.dummysensor_layout);
		context = getApplicationContext();

		Shealth health = new Shealth();
		try {
		     // Initialize an instance of Shealth.
		     health.initialize(this);
		} catch (SsdkUnsupportedException e) {
		     if(e.getType()==SsdkUnsupportedException.VENDOR_NOT_SUPPORTED) {
		         // Vendor is not Samsung
		     }
		}


		  mHealthSensor = new ShealthDeviceFinder(context,
mSensorConnectionListener);

	}



	 /*** Service listener Callback called on successful binding to the Sensor
service.
	 */
	ServiceConnectionListener mSensorConnectionListener = new
ServiceConnectionListener()
	{
		@Override
		public void onServiceConnected(int type)
		{
			Log.d(TAG, "Binding is done - Service connected");
		    mHealthSensor.startScan(ShealthSensorDevice.CONNECTIVITY_TYPE_DUMMY,
ShealthSensorDevice.Type.DUMMY, ShealthSensorDevice.DATA_TYPE_ALL, 10,
mSensorScanListener);

		}
```

```
            @Override
            public void onServiceDisconnected(int type)
            {
                  Log.d(TAG, "Service disconnected");
            }
      };


   /**
       * Scan listener callback called by Health Sensor on
       * the device found or if an error occurs
       */
      ScanListener mSensorScanListener = new ScanListener()
      {
            @Override
            public void onScanStopped(int error)
            {
            }

            @Override
            public void onScanStarted(int error)
            {
            }

            @Override
            public void onDeviceFound(ShealthSensorDevice device)
            {
                  mDevice = device;
                  mDevice.join(mEventListener);
            }
      };


      /**
       * Callback called on successful pairing with a sensor device
       */
      EventListener mEventListener = new EventListener()
      {

            @Override
            public void onResponseReceived(Command command, Response response)
            {

            }

            @Override
            public void onLeft(int error)
            {

            }

            @Override
            public void onJoined(int error)
            {

                  mDevice.startReceivingData(mDataListener);

            }
```

```
		@Override
		public void onStateChanged(int state)
		{
		}
	};

	/**
	 * Data callback called when data is received from sensor
	 */
	DataListener mDataListener = new DataListener()
	{
		@Override
		public void onReceived(int dataType, Health data, Bundle extra)
		{
			Pedometer pedo = (Pedometer)data;
			Log.d(TAG, "onReceived - steps : " + pedo.totalStep);

		}

		@Override
		public void onStarted(int dataType, int error)
		{
		}

		@Override
		public void onStoped(int dataType, int error)
		{

		}
	};
}
```

3. Using the Shealth Class

The Shealth class provides the following methods:

- `initialize()` initializes Health. You need to initialize the Health package before you can use it. If the device does not support Health, SsdkUnsupportedException is thrown.
- `getVersionCode()` gets the Health version number as an integer.
- `getVersionName()` gets the Health version name as a string.

```
Shealth health = new Shealth();
try {
     // Initialize an instance of Shealth.
     health.initialize(this);
} catch (SsdkUnsupportedException e) {
     if(e.getType()==SsdkUnsupportedException.VENDOR_NOT_SUPPORTED) {
         // Vendor is not Samsung
     }
}

int versionCode = health.getVersionCode();

String versionName = health.getVersionName();
```

3.1. Using the initialize() Method

The `Shealth.initialize()` method:

- Initializes the Health package.
- Checks if the device is a Samsung device.
- Checks if the Samsung device supports the Health package.
- Checks if the Health package libraries are installed on the device.

```
void initialize(Context context) throws SsdkUnsupportedException
```

If the Health package fails to initialize, the `initialize()` method throws an SsdkUnsupportedException exception. To find out the reason for the exception, check the exception message.

3.2. Handling SsdkUnsupportedException

If an SsdkUnsupportedException exception is thrown, check the exception message type using `SsdkUnsupportedException.getType()`.

The following types of exception messages are defined in the Shealth class:

- **VENDOR_NOT_SUPPORTED:** The device is not a Samsung device.

4. Content Module Overview

The Content module manages the central repository of data in the Health package. You can use health ContentProvider to access the central repository which exposes the database, files and profile information.

- ShealthContract class to get the information for accessing the database and file resources.
- ShealthProfile class to access the user profile information.

The Content module is also responsible for synchronization of data with a backup server and in backing up the user data incrementally.

To use the Content module, you must register its permissions in the platform. Your application can call the `requestPermission()` method to register its permission list. For more information, see Registering for Access Control.

4.1. Content Organization

The following sections define the data that the content provider exposes.

4.1.1. Database

The important tables exposed by the content provider are:

- **Food tables:**
  - FoodInfo (ShealthContract.FoodInfoColumns)
  - FoodNutrient (ShealthContract.FoodNutrientColumns)
  - Meal (ShealthContract.MealColumns)
  - MealPlan (ShealthContract.MealPlanColumns)
  - MealItem (ShealthContract.MealItemColumns)
  - MealImage (ShealthContract.MealImageColumns)
- **Exercise tables:**
  - Exercise (ShealthContract.ExerciseColumns)
  - ExerciseDeviceInfo (ShealthContract.ExerciseDeviceInfoColumns)
  - ExerciseInfo (ShealthContract.ExerciseInfoColumns)
  - ExercisePhoto (ShealthContract.ExercisePhotoColumns)
  - ExerciseActivity (ShealthContract.ExerciseActivityColumns)
  - WalkInfo (ShealthContract.WalkInfoColumns)
  - LocationData (ShealthContract.LocationDataColumns)
  - RealTimeData (ShealthContract.RealTimeDataColumns)
  - StrengthFitness (ShealthContract.StrengthFitnessColumns)

    - FirstBeatCoachingResult (ShealthContract.FirstBeatCoachingResultColumns)
    - FirstBeatCoachingVariable (ShealthContract.FirstBeatCoachingVariableColumns)
- **Measures and goal:**
    - Weight (ShealthContract.WeightColumns)
    - BloodPressure (ShealthContract.BloodPressureColumns)
    - BloodGlucose (ShealthContract.BloodGlucoseColumns)
    - BodyTemperature (ShealthContract.BodyTemperatureColumns)
    - Sleep (ShealthContract.SleepColumns)
    - SleepData (ShealthContract.SleepDataColumns)
    - Stress (ShealthContract.StressColumns)
    - PulseOximeter (ShealthContract.PulseOximeterColumns)
    - ElectroCardiography (ShealthContract.ElectroCardiographyColumns)
    - HeartRate (ShealthContract.HeartRateColumns)
    - HeartRateData (ShealthContract.HeartRateDataColumns)
    - WaterIngestion (ShealthContract.WaterIngestionColumns)
    - Goal (ShealthContract.GoalColumns)
- **Device tables:**
    - UserDevice (ShealthContract.UserDeviceColumns)

The tables are commonly referred to by the names of their contract classes. The classes define constants for content URIs, column names, and column values that the tables use.

The ShealthContract class contains definitions for the supported table URIs and the field information. It is similar to the Contract classes defined by Android. For more information on all the tables and fields, see the ShealthContract API reference.

The access to the URIs is restricted using the HealthService Platfrom Access Control mechanism.

4.1.2. User Profile

The ShealthProfile class is the interface for accessing user profile information, such as user name, height, and weight. This class provides simple methods to get and set the user profile information.

4.1.3. Image Resources

You can access image resources through the content provider. The Content module exposes the images that are added by the user.

5. Using the Content Module

This section describes how to use the Content module in your application.

5.1. Using CRUD Operations

You can use the content provider to insert, update, delete, and read data in the database.

5.1.1. Inserting and Updating Data

To insert or update data, add the values to an instance of the ContentValues class. The `insert()` method accepts a URI and a content value.

The following code shows how to insert a weight entry.

```
ContentValues values = new ContentValues();
             values.put(WeightColumns.VALUE, value);
             values.put(WeightColumns.COMMENT, comment);
             values.put(WeightColumns.SAMPLE_TIME, sampleTime);

Uri rawContactUri =
context.getContentResolver().insert(ShealthContract.Weight.CONTENT_URI, values);
return ContentUris.parseId(rawContactUri);
```

The following code shows how to update the weight value when the existing value is 81.

```
String select;
values = new ContentValues();
values.put(ShealthContract.Weight.VALUE, "80");
select = ShealthContract.Weight.VALUE + " = " + "81";
getContentResolver().update(ShealthContract.Weight.CONTENT_URI, values, select,
null);
```

5.1.2. Deleting Data

To delete data, use a select string to select the required rows to be deleted.

The following code shows how to delete in the Weight table the rows that have the weight value of 81.

```
Float value = 81;
where = ShealthContract.Weight.VALUE + " = " + value;
getContentResolver().delete(ShealthContract.Weight.CONTENT_URI, where, null);
```

5.1.3. Reading Data

You can use the query() method to read the content of the database. The method accepts a URI for the table, a projection array of strings that precisely indicate the required columns, and a select string to select the required rows. It accepts two more parameters for which you can pass null values:

- selectionArgs: In the select string, you can use the special character '?', which is replaced by the values from selectionArgs.
- sortOrder : You can use sortOrder to order the rows.

The following code shows how to retrieve the data of the kcal, comment, and create_time columns from the Exercise table.

```
Cursor cursor = null;
String[] projection1 = { ShealthContract.ExerciseColumns.KCAL,
ShealthContract.ExerciseColumns.COMMENT,
ShealthContract.ExerciseColumns.CREATE_TIME };

cursor = getContentResolver().query(ShealthContract.Exercise.CONTENT_URI,
projection1, null, null, null);
```

5.2. Listening to Database Content Changes

You can listen to any change in the database content using the content observers.

To use a content observer:

- Implement a subclass of ContentObserver.
- Register your content observer to listen for changes for the specific URI.

ContentObserver is an abstract class with no abstract methods. It provides two onChange() methods that are implemented without any logic. You have to override them. For more details, see http://developer.android.com/reference/android/database/ContentObserver.html.

5.3. Accessing Image Resources

You can access image resources by opening an input stream with the applicable URI.

The following code shows how to open an input stream for the image resource associated with the CONTENT_MEAL_IMAGES_READ_URI. To open a writable steam, use CONTENT_MEAL_IMAGES_WRITE_URI.

```
InputStream is =
getApplicationContext().getContentResolver().openInputStream(Uri.withAppendedPath(
ShealthContract.CONTENT_MEAL_IMAGE_URI, "imagefilename.png"));
```

If there is no data associated with the URI, the FileNotFoundException exception is thrown.

5.4. Managing User Profiles

The ShealthProfile class provides simple methods to get and set the user profile information. You can create an ShealthProfile instance by passing the application context, which then loads the current user profile information.

You can use the setter methods provided by the ShealthProfile class to change the profile information. To update the changes to the profile, call the `save()` method. To refresh the profile information after any changes, use the `load()` method .

The following code shows how to retrieve the user name from the profile information and update the gender information.

```
ShealthProfile sProfile = new ShealthProfile(context);
String userName = sProfile.getName();

//update profile
sProfile.setGender(GenderType.TYPE_WOMAN);
sProfile.save();
```

5.5. Synchronizing Database Content to the Server

ContentManager provides the following methods to sync the database content to the server:

- `startSynchronization(int dataSyncType, SynchronizationListener listener)`
  The method starts the synchronization and accepts two parameters. The first parameter is the data type to be synchronized and the second parameter is the listener to receive the synchronization result.
- `stopSynchronization()`
  The method stops the data synchronization with the PHR server.

The SynchronizationListener class provides the following callbacks:

- `onStarted(int dataSyncType, int error)`: Called when data synchronization is started by passing an error code.
- `onFinished(int dataSyncType, int error)`: Called when data synchronization is finished by passing an error code.

stopSynchronization api is used to stop data synchronization with PHR server

Once platform sync is done, there is a message will be sent by braodcast to inform the sync completion.

Application can listen the intent which action is "com.sec.android.service.health.cp.SYNC_STATE". This intent contains state, PHR type and error code as extras. (You can refer to ShealthContentManager.java for detail information is in)

- *Action : com.sec.android.service.health.cp.SYNC_STATE*

- *Extras :*

key	value
type	defined in ShealthContentManager.java
sync_state	SYNC_TPYE_START(0) - started SYNC_TYPE_END(1) - finished
error	defined in ShealthContentManager.SynchronizationListener

This is a sample code to get extras from intent.

```
public class SyncStateBroadcastReceiver extends BroadcastReceiver
{

        private static final String TAG = "SyncBroadcastReceiver";

        private static final String SYNC_STATE_ACTION_NAME =
"com.sec.android.service.health.cp.SYNC_STATE";

        private int mPHRType;
        private int mState;
        private int mErrorCode;

        @Override
        public void onReceive(Context context, Intent intent)
        {
                if(intent.getAction() != null && SYNC_STATE_ACTION_NAME.equals(intent.getAction()))
                {
                   mState = intent.getIntExtra("sync_status", -1);
                   mPHRType = intent.getIntExtra("type", -1);
                   mErrorCode = intent.getIntExtra("error", -1);
                }
        }
}
```

5.6. Registering for Access Control

You can use the access control mechanism of the Content module to validate whether an application has access to specific PHR data.

To register your application for access control:

5. Register your application with the Samsung Server and get app_id for your package.
   During the registration process, you can select permissions on specific PHR data types.
6. At the first launch of the application, call the `ShealthAccessControl.requestPermission()` method.

During this call, the HealthService platform retrieves the permission information from the server and applies it to all the data access later on.

NOTE: Make sure that the registration method gets a success callback. If there is no network connectivity, your application must show a relevant message to the end user and try the registration process later again.

The following code shows how to use the `requestPermission()` method.

```
ShealthAccessControl accessControl = new ShealthAccessControl(this);
              accessControl.requestPermission("Test_App01", "ABCDEFGHIJ",
                     new ShealthAccessControl.AccessRegistrationListener()
              {

                     @Override
                     public void onRegistrationCompleted(int error)
                     {
                            Log.i("ShealthAccessControl",
                            "onRegistrationCompleted " + error);
                     }
              });
```

6. Sensor Module Overview

The Health Sensor Service provides methods to easily scan for, connect to, and obtain data from a variety of sensor-based devices through various protocols. It sends the appropriate response to the calling application through the proper callbacks.

The Health Sensor Service work is internally realized as an Android service. Your application can use the Sensor Service client library to communicate with the Health Sensor Service and get the sensor data.

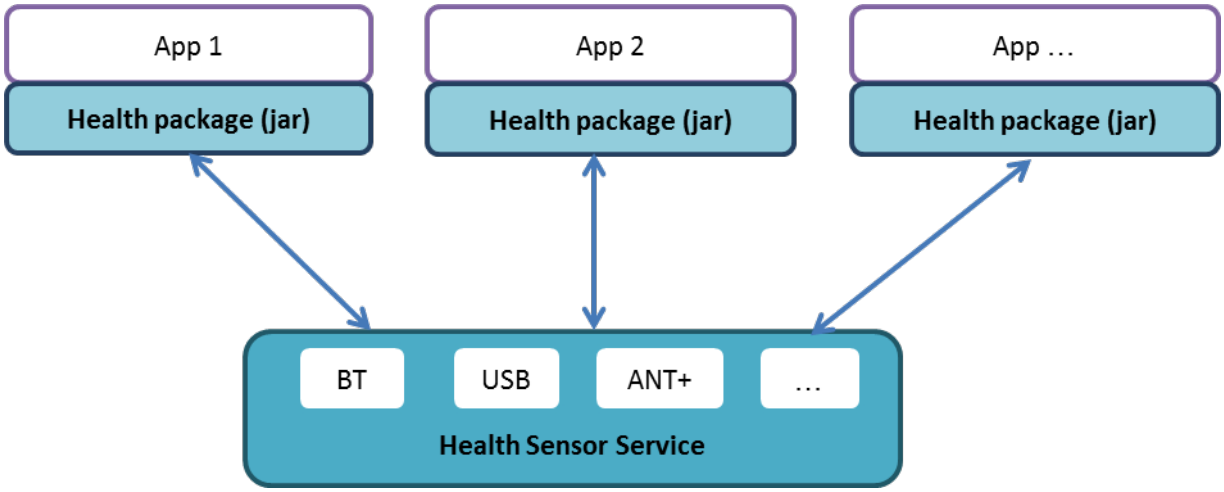


Figure 4: Health Sensor Service

The Health Sensor Service provides methods to facilitate communication between your application and various sensor devices. It also allows communication through command and response to interact with certain sensor devices. In this case, each request to the sensor is sent as a command along with any required parameters.

6.1. Supported Sensors

The Health Sensor Service supports various sensor devices. The design is made extendable to support new sensor devices easily in the future.

The following sections define the supported sensors.

6.1.1. Bluetooth Devices

The Health Sensor Service supports Bluetooth medical sensor devices. The sensors support the following protocols through Bluetooth:

Continua

This follows the protocol IEEE 11073 Personal Health Device standards defined by the Continua Health Alliance. The devices which use this standard include blood pressure monitors, weighing scales and blood glucose monitors. The manufacturers include AND and Omron.

HME

The Health Sensor Service supports devices from the manufacturer i-SENS that use the HME protocol. The supported devices include blood glucose monitors.

6.1.2. ANT+ Devices

The Health Sensor Service supports ANT+ devices, such as weight scale, blood pressure, heart rate monitor, SDM, and watch devices.

6.1.3. USB Devices

The Health Sensor Service supports medical USB devices, such as blood glucose monitors. The main protocols supported through USB are Infopia and JNJ.

6.1.4. SAP Devices

The Health Sensor Service supports heart rate monitors and weight scale of Samsung Accessory Protocol.

6.1.5. Internal Sensors

The internal sensors in the phone device provide data without additional external devices.

SContext Pedometer

Provides data from the pedometer sensor in the device.

Pulse Oximeter

Provides data from the pulse oximeter sensor in the device.

6.1.6. NFC Devices

The Health Sensor Service supports the devices from iSens and Sony through NFC. The supported devices include blood glucose monitors.

7. Using the Sensor Module

This section describes how to use the Sensor module in your application.

7.1. Initializing the Sensor Service

To communicate with the Sensor Service, construct an ShealthDeviceFinder instance. The constructor call internally binds your application to the Sensor Service. Once the Sensor Service is connected, the `onServiceConnected()` callback of the passed listener instance is called. Your application can call other methods of the ShealthDeviceFinder instance to communicate with the Sensor Service.

The following code shows how to initialize the Sensor Service.

```
void startServiceIfNeeded()
{
	mHealthSensor = new ShealthDeviceFinder (context, mSensorConnectionListener);
}

ServiceConnectionListener mServiceConnectionListener = new
ServiceConnectionListener()
{
	@Override
	public void onServiceConnected(int error)
	{
		// Binding is done - Service connected
		// now other APIs can be called
	}

	@Override
	public void onServiceDisconnected(int error)
	{
		// Service disconnected
	}
};
```

7.2. Scanning for Devices

You can obtain the list of available devices by scanning for devices.

The following code shows how to start scanning for Bluetooth devices. The scanning duration is set to 10 seconds. The mHealthSensor in the code is the same object that was constructed when initializing the Sensor Service.

To successfully scan for Bluetooth devices, Bluetooth must be switched on in the device. Otherwise a response error code is sent back. To stop scanning, call the `stopScanning()` method.

```
// To save the detected device
ShealthSensorDevice mDevice;
```

```
// Listener for scanning
ScanListener mSensorScanListener = new ScanListener()
{
	@Override
	public void onScanStopped(int error)
	{
		// Scan is finished or cancelled (depending on error code)
	}

	@Override
	public void onScanStarted(int error)
	{
		// Scan started
	}

	@Override
	public void onDeviceFound(ShealthSensorDevice device)
	{
		// Will be called multiple times
		// For illustration, we are considering one device only
		mDevice = device;
	};
}

// call startScan method to start the scanning procedure
mHealthSensor.startScan(
	ShealthSensorDevice.CONNECTIVITY_TYPE_BLUETOOTH, // Cconnectivity type
	ShealthSensorDevice.Type.BLOOD_PRESSURE, // Type of the device (BP)
	ShealthSensorDevice.DATA_TYPE_BLOODPRESSURE, // Type of data (BP values)
	10,                              // Duration in seconds
	mSensorScanListener              // Listener instance for receiving result
);


// For stopping scan, the following can be used
mHealthSensor.stopScan();
```

7.3. Joining with Devices

After successful scanning, you can select the device to join by calling the `join()` method. The method accepts a listener as the parameter to receive the events related to the device. For more information, see the API documentation.

The following code shows how to join with a given device. When your application has stopped communicating with the device, call the `leave()` method.

```
EventListener mEventListener = new EventListener()
{
	@Override
	public void onResponseReceived(Command command, Response response)
	{
		// Request API's result will be received here
	}
```

```
        @Override
        public void onLeft(int error)
        {
                // Device connection is disconnected
        }

        @Override
        public void onJoined(int error)
        {
                if(error == EventListener.ERROR_NONE)
                {
                        // Device is joined successfully

                        // After this, call startReceivingData
                        // and call leave when done
                }
        }

        @Override
        public void onStateChanged(int status)
        {
                // Will be called when state is changed
        }

};

mDevice.join(mEventListener);

// When communication is not required anymore
mDevice.leave();
```

7.4. Getting Data

To start receiving data from a particular device, call the `startReceivingData()` method. This method accepts as a parameter a listener that is notified when the data is sent from the Health Sensor Service.

When your application does not want to receive any more data, call the `stopReceivingData()` method for streaming data. For non-streaming data, call this method to cancel receiving the data. This removes the DataListener and the application no longer receives the `onDataReceived()` callback.

The following code illustrates how to get and stop getting data for a blood pressure device.

```
// Listener for data receive:
ShealthSensorDevice.DataListener mDataListener = new
ShealthSensorDevice.DataListener()
{
  public void onDataReceived(int dataType, Health data, Bundle extra)
  {
       if(dataType  == ShealthSensorDevice.DATA_TYPE_BLOODPRESSURE)
       {
              BloodPressure receivedData = (BloodPressure)data;
              float bp_sys = receivedData.systolic;
              float bp_dia = receivedData.diastolic;
              float bp_pulse = receivedData.pulse;
```

```
                log(TAG, "BP Sys: " + Float.toString(bp_sys) + " BP Dia: " +
                Float.toString(bp_dia) + " Pulse:" + Float.toString(bp_pulse));
        }
    }
};

// To start receiving data:
mDevice.startReceivingData(mDataListener);

// Exercise id in DB associated with exercise related data

//mDevice.startReceivingData(exerciseId, mDataListener);


// To stop receiving data:
mDevice.stopReceivingData();
```

7.5. Sending Commands

You can use the `ShealthSensorDevice.request()` method to send special commands to the sensor device. The method takes a single parameter of type Command. The command object consists of the following:

- Command ID, which indicates the command operation to be performed.
- Bundle parameter, which can be used to pass extra information to the sensor. The extra information is passed as multiple key/value pairs.

The response to the command comes in the `onResponseReceived()` callback of ShealthSensorDevice.EventListener. This is the same listener that is used in the `join()` method. The response callback contains a response object.

7.5.1. Response Object

The Response object received in `onResponseReceived()` has the following methods to get the response data:

- `getErrorCode()` – Error code
- `getErrorDescription()` – Description of the error
- `getCommandId()` – ID of the command in the sent request
- `getResponse()` – Response Bundle containing all the data.

The following code shows how to send a command to a sensor device and receive the response.

```
// Listener for event receive:
EventListener mEventListener = new EventListener()
{
      @Override
      public void onResponseReceived(Command command, Response response)
      {
            // Request API's result is received here
```

```
                Log.d("response.getCommandId() : " + response.getCommandId());
                Log.d("response.getErrorCode() : " + response.getErrorCode());
                Log.d("response.getErrorDescription() : " +
                response.getErrorDescription());
                Log.d("response.getResponse() : " + response.getResponse());
        }

        @Override
        public void onLeft(int error)
        {
                // Device connection is disconnected
        }

        @Override
        public void onJoined(int error)
        {
                if(error == EventListener.ERROR_NONE)
                {
                        // Creating the command for sending
                        Bundle bundle = new Bundle();

                        bundle.putInt("test", 0);

                        ShealthSensorDevice.Command cmd = new
                        ShealthSensorDevice.Command("TEST", bundle);
                        cmd.setCommandId("TEST1");
                        try
                        {
                                mDevice.request(cmd);
                        }
                        catch(RemoteException e)
                        {
                        }
                        catch(IllegalArgumentException e)
                        {
                        }
                }
        }

        @Override
        public void onStateChanged(int status)
        {
                // Will be called when state is changed
        }

};

}
```

7.6. Taking Control of the Sensor Device

Your application can take control of a sensor device for exclusive communication. This is not applicable for internal sensors which are available in the phone. The exclusive access helps to avoid the interference of other applications when your application is using the device.

The ShealthSensorDevice provides the following methods which can be called by only one application at a time:

- `startReceivingData()`
- `request()`
- `stopReceivingData()`

You can call the `startReceivingData()` or `request()` method after getting a successful `onJoined()` callback. These methods fail if other applications try to call them before the application with the current access calls `stopReceivingData()` first.

If your application calls `startReceivingData()`, other applications can also receive data from the same sensor by calling `startListeningData()`. After you call `stopReceivingData()`, the sensor is stopped and no other application receives data. The device becomes available for other applications to take control.

Note: Stream data (such as HRM) from the sensor is saved to the database only if your application takes control using `startReceivingData()`.

When an application calls the `request()` method, other applications cannot call the `request()` method until the device finishes the request and sends the response to the application.

7.7. Listening for Data

The ShealthSensorDevice class provides methods to listen for data from the sensor devices:

- `startListeningData()`: Listen for data from a sensor which is controlled by another application.
- `stopListeningData()`: Remove the listener set by `startListeningData()`.

If your application has called `startListeningData()` to listen for data, and then calls `startReceivingData()`, an exception is thrown. In this case, your application must call `stopListeningData()` to unregister the current listener before calling `startReceivingData()`.

When an application calls `startReceivingData()`, all applications using the same device are notified with the `onStarted()` event. When an application calls `stopReceivingData()`, all applications using the same device are notified with the `onStopped()` event.

7.8. Checking the Capability of a Sensor

The ShealthDeviceFinder class provides the `isAvailable()` method to check whether a sensor is available based on the Connectivity type, device type, data type, and protocol parameters. The method returns true if the device or sensor with the given parameters is supported by the Sensor framework.

The following code shows how to use the `isAvailable()` method.

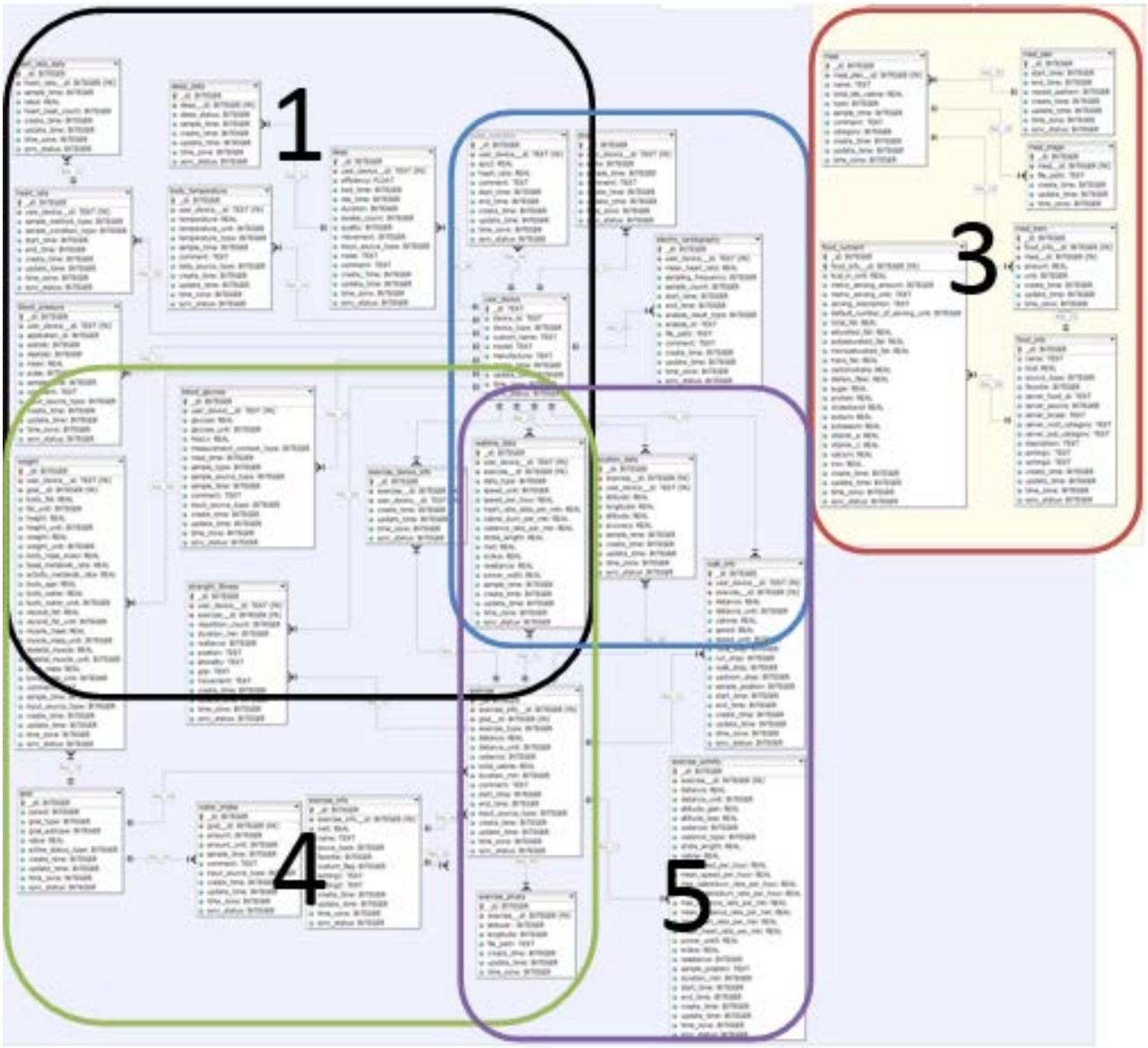
```
try
{
	if(mShealthDeviceFinder!= null)
	{
		boolean b = mShealthDeviceFinder.isAvailable(ShealthSensorDevice.
```

```
                        CONNECTIVITY_TYPE_INTERNAL, //Connectivity type
                        ShealthContract.Constants.DeviceType.PEDOMETER, //Device type
                       ShealthSensorDevice.DATA_TYPE_ALL, //Data type
                        null);//Protocol
        }
}
catch(RemoteException e)
{
        addLogToListView("TEST CASE sensor.isAvailable() FAILED (RemoteException)");
        e.printStackTrace();
}
catch(IllegalArgumentException e1)
{
        addLogToListView("TEST CASE sensor.isAvailable() IllegalArgumentException");
}
```

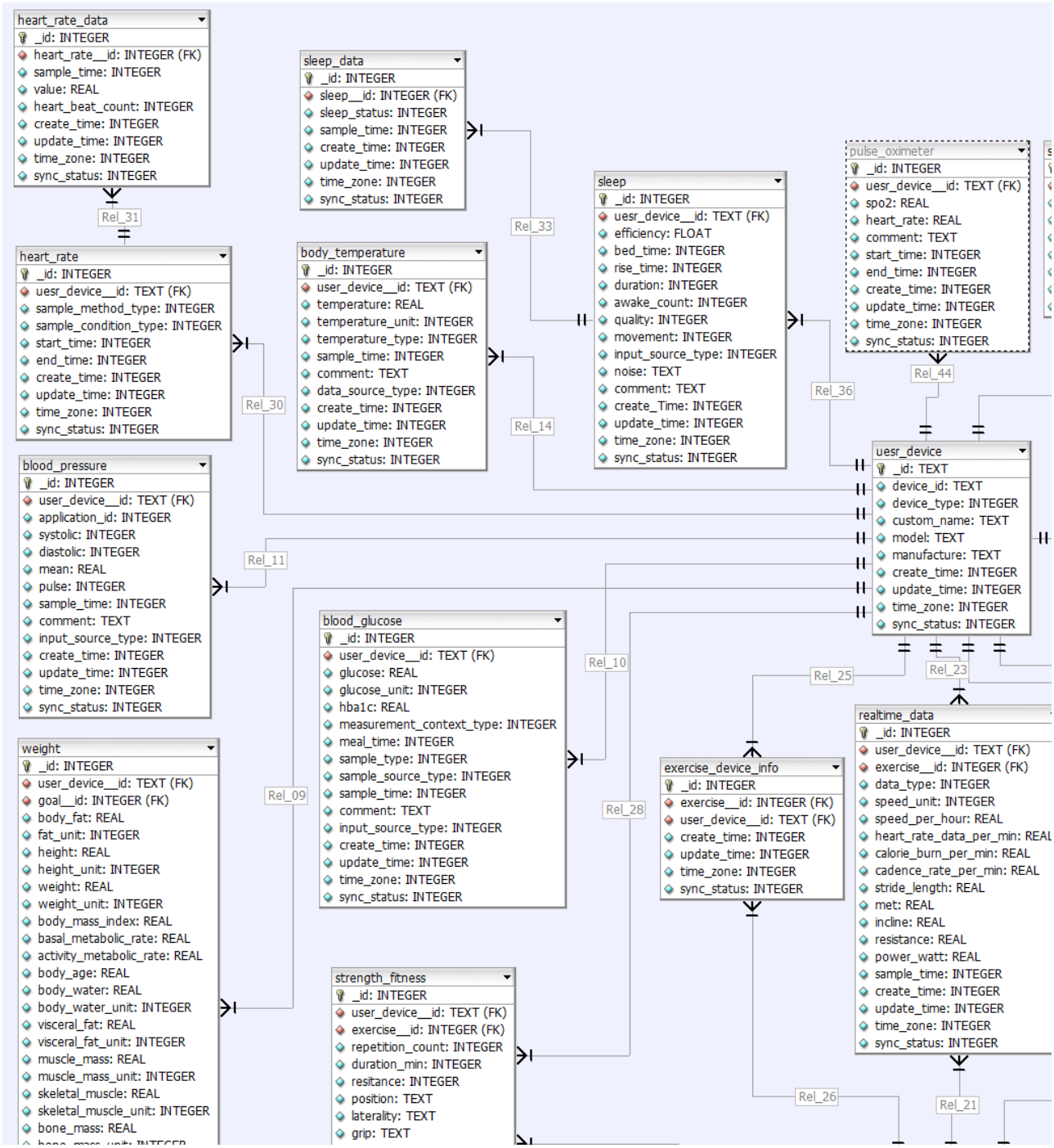
Note: To check the availability of the ANT device type, set the protocol parameter to “null”.

Appendix A. DB Schema

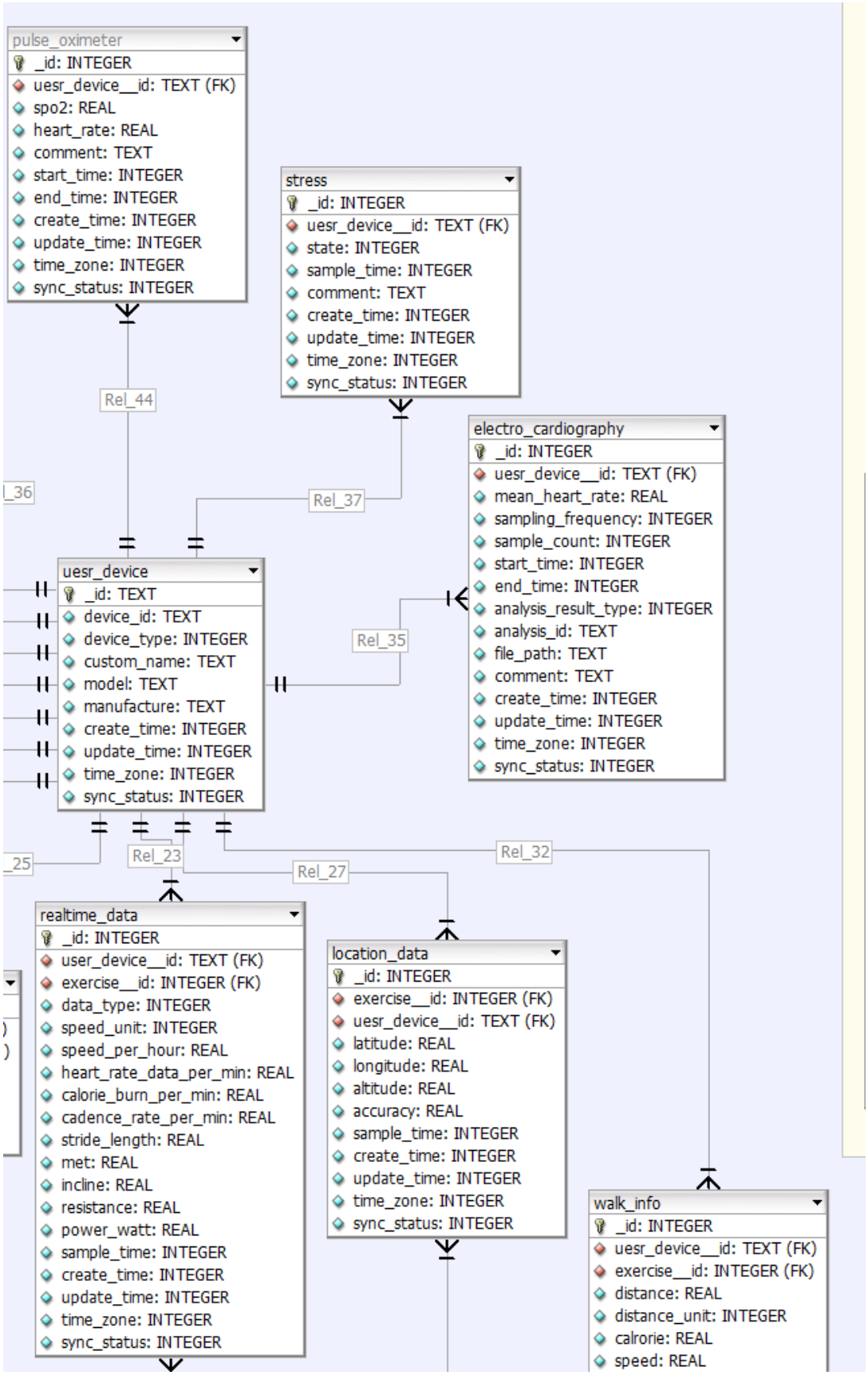
A.1. DB Schema – All



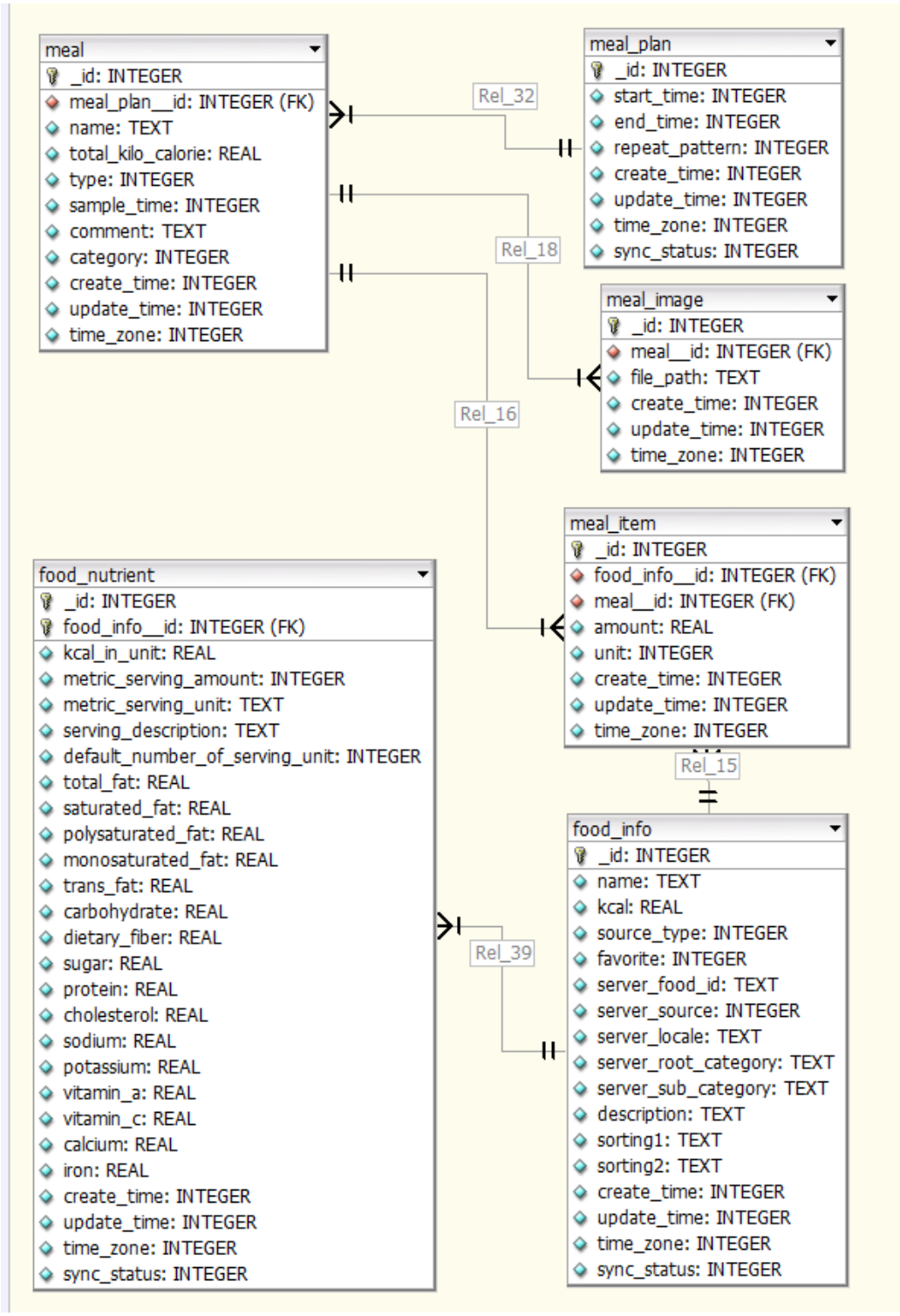
A.2. DB Schema – Part 1/5



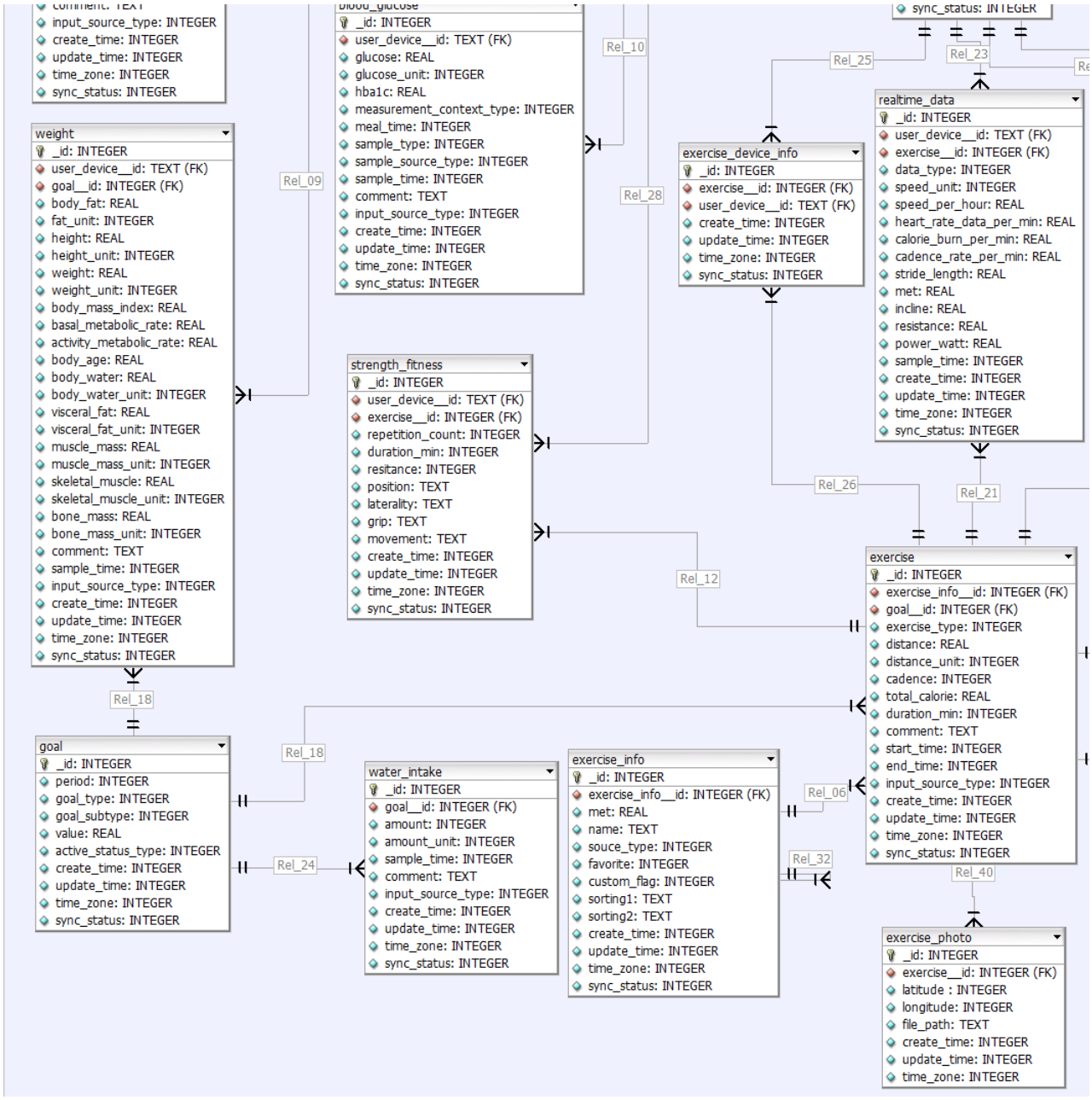
A.3. DB Schema – Part 2/5



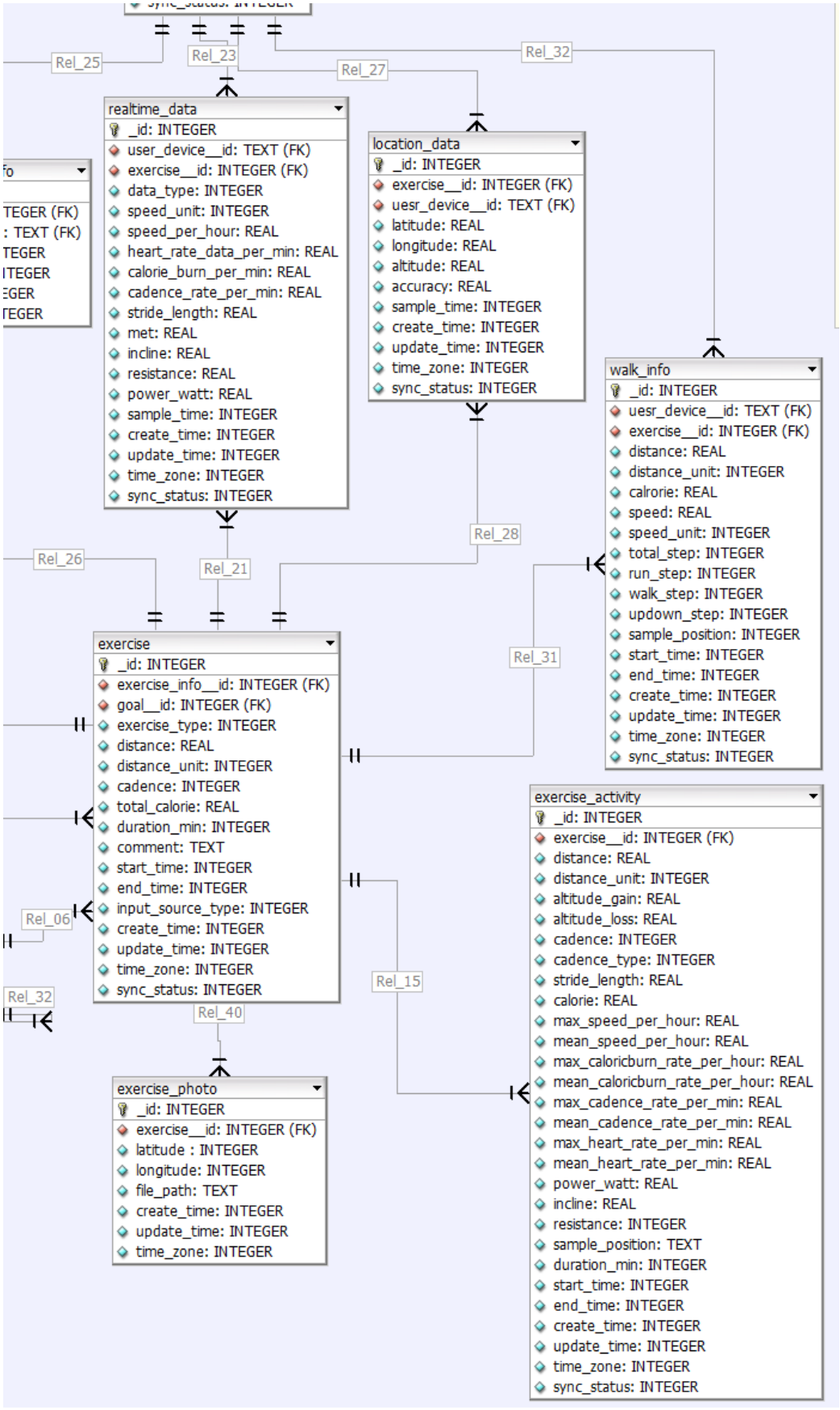
A.4. DB Schema – Part 3/5



A.5. DB Schema – Part 4/5



A.6. DB Schema – Part 5/5



Copyright

Though every care has been taken to ensure the accuracy of this document, Samsung Electronics Co., Ltd. cannot accept responsibility for any errors or omissions or for any loss occurred to any person, whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

Samsung Electronics Co. Ltd. may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

No part of this document may be communicated, distributed, reproduced or transmitted in any form or by any means, electronic or mechanical or otherwise, for any purpose, without the prior written permission of Samsung Electronics Co. Ltd.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

For more information, please visit http://developer.samsung.com/