Siminov Framework Version 0.9.1-Beta Developer Guide

Siminov Framework

http://www.siminov.github.com/android-orm

Contents

Preface

Working with both Object-Oriented software and Relational Databases can be cumbersome and time consuming. Development costs are signi cantly higher due to a paradigm mismatch between how data is represented in objects versus relational databases. Siminov is an Object/Relational Mapping solution for Android environments. The term Object/Relational Mapping refers to the technique of mapping data from an object model representation to a relational data model representation (and visa versa). See `http://en.wikipedia.org/wiki/Object-relational_mapping` for a good high-level discussion.

Note

While having a strong background in SQL is not required to use Android-Siminov, having a basic understanding of the concepts can greatly help you understand Siminov more fully and quickly. Probably the single best background is an understanding of data modeling principles. You might want to consider these resources as a good starting point: `http://en.wikipedia.org/wiki/Data_modeling`

Siminov not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities. It can signi cantly reduce development time otherwise spent with manual data handling in SQLit Siminov design goal is to relieve the developer from 99

Siminov may not be the best solution for data-centric applications that only use stored-procedures to implement the business logic in the database, it is most useful with object-oriented domain models and business logic in the Java-based. However, Siminov can certainly help you to remove or encapsulate vendor-speci c SQLite code and will help with the common task of result set translation from a tabular representation to a graph of objects.

0.1 Get Involved

Use Siminov and report any bugs or issues you nd. See `https://github.com/Siminov/android-orm/issues` for details. Try your hand at xing some bugs or implementing enhancements. Again, see `https://github.com/Siminov/android-orm/issues`. Engage with the community using mailing lists, forums, IRC, or other ways listed at `https://github.com/Siminov`. Help improve or translate this documentation. Contact us on the developer mailing list if you have interest.Spread the word. Let the rest of your organization know about the bene ts of Siminov.

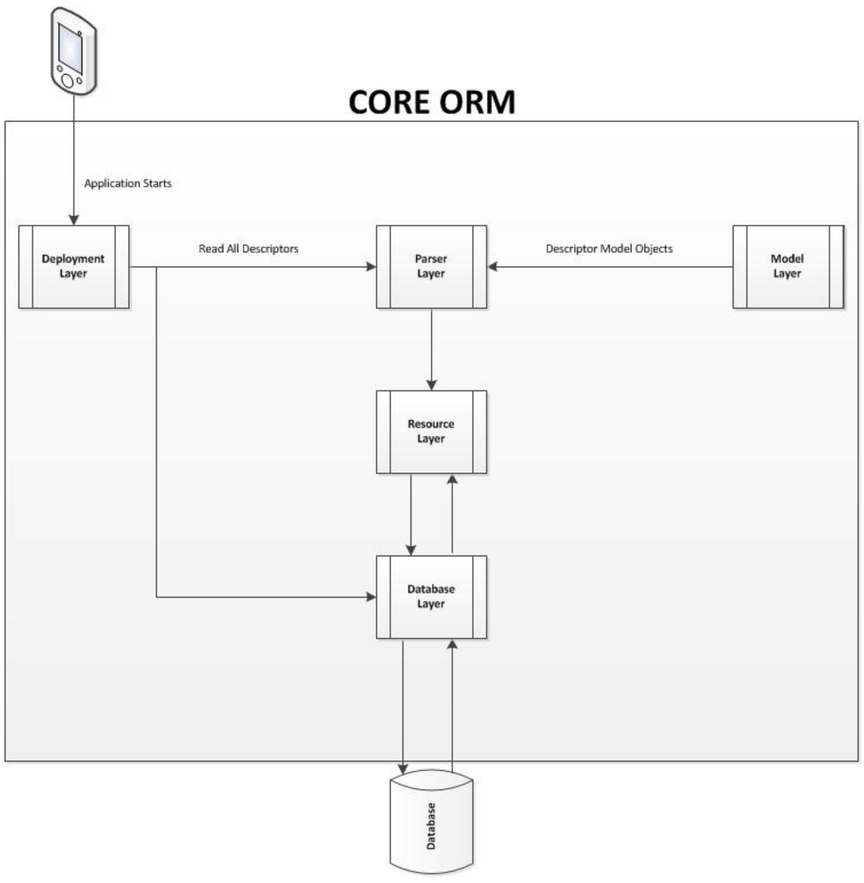
0.2 Getting Started Guide

New users may want to rst look through the Siminov Getting Started Guide for basic information as well as tutorials. Even seasoned veterans may want to considering perusing the sections pertaining to build artifacts for any changes.

Chapter 1

Overview

Siminov ORM component is a native object relationship mapping framework, which handles all activities related to database.



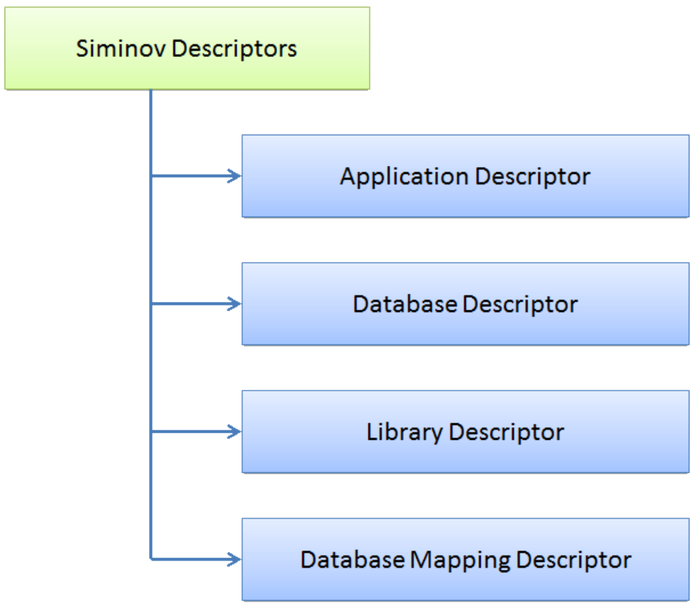
Siminov ORM Layers

1.1 Features

1. Handles application deployment.
2. Provides simple and clean orm for every platform.
3. Easy to con gure and develop using xml or annotations.
4. Provides encryption implementations.

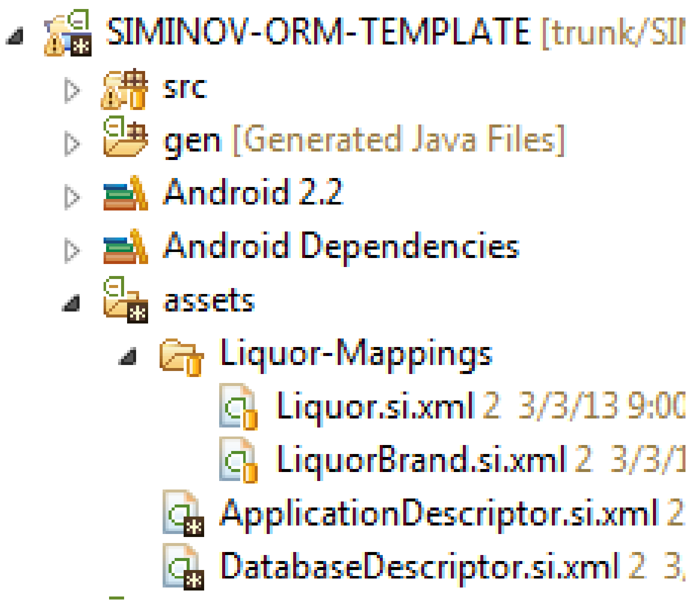
Chapter 2

Con guration



Siminov Framework works based on a set of de ned descriptors which can be broadly classi ed as ApplicationDescriptor.si.xml, DatabaseDescriptor.si.xml, Library-Descriptor.si.xml, DatabaseMappingDescriptor.si.xml.

All these descriptor should will be placed in application assests folder.



SIMINOV-ORM-TEMPLATE [trunk/SII

src

gen [Generated Java Files]

Android 2.2

Android Dependencies

assets

Liquor-Mappings

Liquor.si.xml 2 3/3/13 9:00

LiquorBrand.si.xml 2 3/3/1

ApplicationDescriptor.si.xml 2

DatabaseDescriptor.si.xml 2 3,

Note

1. All descriptor le name should end with .si.xml.

2.1 Application Descriptor (ApplicationDescriptor.si.xml) Con guration

Application Descriptor is the one who connects application to Siminov framework. It provide basic information about application, which de nes the behaviour of application.

```
        <!-- Design Of ApplicationDescriptor.si.xml -->

<siminov>

  <!-- General Application Description Properties -->
    <!-- Mandatory Field -->
  <property name="name">application_name</property>

    <!-- Optional Field -->
  <property name="description">application_description</property>

    <!-- Mandatory Field (Default is 0.0) -->
  <property name="version">application_version</property>



  <!-- Siminov Framework Performance Properties -->
    <!-- Optional Field (Default is true)-->
  <property name="load_initially">true/false</property>



  <!-- Database Descriptors Used By Application (zero-to-many) -->
    <!-- Optional Field's -->
  <databases-descriptors>
    <database-descriptor>full_path_of_database_descriptor_file</
```

Example: ApplicationDescriptor.si.xml File Of Siminov Template Application.

```
                    <!-- Example: ApplicationDescriptor.si.xml Of Siminov
                         ORM Tempalte -->

<siminov>

  <property name="name">SIMINOV TEMPLATE</property>
  <property name="description">Siminov Template Application</property
      >
  <property name="version">0.9</property>

  <property name="load_initially">true</property>

  <!-- DATABASE-DESCRIPTORS -->
  <database-descriptors>
    <database-descriptor>DatabaseDescriptor.si.xml</database-
        descriptor>
  </database-descriptors>


  <!-- SIMINOV EVENTS -->
  <event-handlers>
      <event-handler>siminov.orm.template.events.SiminovEventHandler<
          /event-handler>
      <event-handler>siminov.orm.template.events.DatabaseEventHandler
          </event-handler>
  </event-handlers>

</siminov>
```

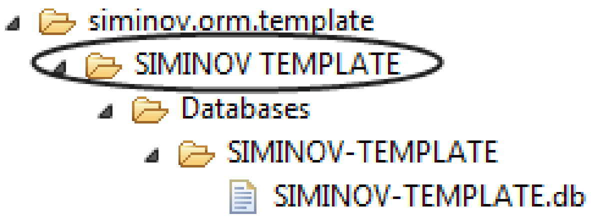
Note

Application Developer can provide their own properties also, and by using following API's they can use properties.

1. Get Properties - getProperties(): It will return all properties associated with Application Descriptor.
2. Get Property - getProperty(Name of Property): It will return property value associated with property name provided.
3. Contains Property - containsProperty(Name of Property): It will return TRUE/FALSE whether property exists or not.
4. Add Property - addProperty(Name of Property, Value of Property): It will add new property to the collection of Application Descriptor properties.
5. Remove Property - removeProperty(Name of Property): It will remove property from Application Descriptor properties based on name provided.

Application Descriptor Elements:

1. General Properties About Application.
   (a) name* : Name of application. It is mandatory eld. If any resources is created by Siminov then it will be under this folder name.



   (b) description: Description of application. It is optional eld.
   (c) version: Version of application. It is mandatory eld. Default is 0.0.
2. Framework Performance Properties.
   (a) load_initially: *TRUE/FALSE*: ORM(Object Relational Mapping) to be done at start of application or at the time when its needed. It is optional eld. By default its false, means mapping is done when its required.

Note

i. If load_initially is false then application will start quickly.

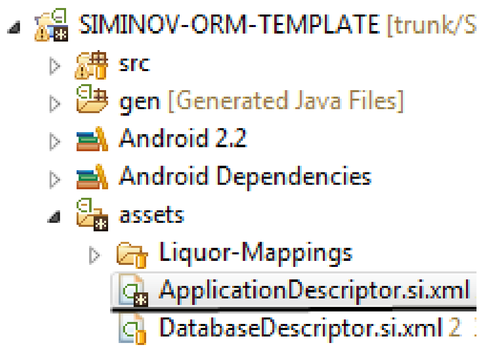
3. Path Of Database Descriptor's Used In Application.
   - Path of all database descriptor's used in application.
   - Every database descriptor will have its own database object.

4. Event Handlers Implemented By Application
   - Siminov Framework provides two type of event handlers
     - ISiminovEvents: It contains events associated with life cycle of Siminov Framework. such as siminovInitialized, rstTimeSiminovInitialized, siminovStopped.
     - IDatabaseEvents: It contains events associated with database operations. such as databaseCreated, databaseDropped, tableCreated, tableDropped, indexCreated, indexDropped.
   - Application can implement these event handlers based on there requirement.

Note

1. Application descriptor le name should always be same as ApplicationDescriptor.si.xml only.
2. It should always be in root folder of application assests.

Example: Siminov Template Application.


SIMINOV-ORM-TEMPLATE [trunk/S
src
gen [Generated Java Files]
Android 2.2
Android Dependencies
assets
Liquor-Mappings
ApplicationDescriptor.si.xml
DatabaseDescriptor.si.xml 2

2.2 Database Descriptor (DatabaseDescriptor.si.xml) Con guration

Database Descriptor is the one who de nes the schema of database.

```
    <!-- Design Of DatabaseDescriptor.si.xml -->

<database-descriptor>

    <!-- General Database Descriptor Properties -->
      <!-- Mandatory Field -->
  <property name="database_name">name_of_database_file</property>

    <!-- Optional Field (Default is sqlite) -->
  <property name="type">database_type</property>

    <!-- Optional Field -->
  <property name="description">database_description</property>

    <!-- Optional Field (Default is false) -->
  <property name="is_locking_required">true/false</property>

    <!-- Optional Field (Default is false) -->
  <property name="external_storage">true/false</property>



  <!-- Database Mapping Descriptor Paths Needed Under This Database
      Descriptor -->
  <!-- Optional Field -->
  <database-mappings>
    <database-mapping path="
        full_path_of_database_mapping_descriptor_file" />
  </database-mappings>



  <!-- Libraries Needed Under This Database Descriptor -->
  <!-- Optional Field -->
  <libraries>
    <library>full_path_of_library_descriptor_file</library>
  </libraries>

</database-descriptor>
```

Example: DatabaseDescriptor.si.xml File Of Siminov Template Application.

```
<database-descriptor>

  <property name="database_name">SIMINOV-TEMPLATE</property>
  <property name="description">Siminov Template Database Config</
     property>
  <property name="is_locking_required">true</property>
  <property name="external_storage">false</property>
  <property name="database_implementer"></property>
  <property name="password"></property>

  <database-mappings>
    <database-mapping path="Liquor-Mappings/Liquor.si.xml" />
    <database-mapping path="Liquor-Mappings/LiquorBrand.si.xml" />
  </database-mappings>

  <libraries>
    <library>siminov.orm.template.resources</library>
  </libraries>

</database-descriptor>
```

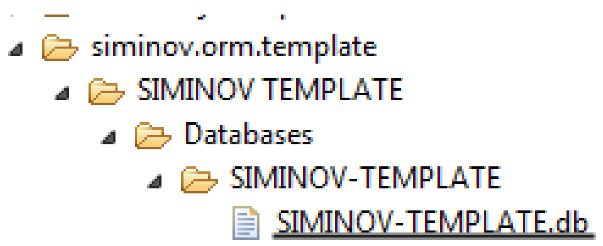
Note

Application Developer can provide their own properties also, and by using following API's they can use properties.

1. Get Properties - getProperties(): It will return all properties associated with Database Descriptor.
2. Get Property - getProperty(Name of Property): It will return property value associated with property name provided.
3. Contains Property - containsProperty(Name of Property): It will return TRUE/FALSE whether property exists or not.
4. Add Property - addProperty(Name of Property, Value of Property): It will add new property to the collection of Database Descriptor properties.
5. Remove Property - removeProperty(Name of Property): It will remove property from Database Descriptor properties based on name provided.

Database Descriptor Elements:

1. General Properties About Database.
   (a) **database_name***: Name of database. It is mandatory eld. All database les (.db)'s will be placed under the this folder name.



   (b) **type**: It de nes the type of database. It is optional eld. Default is sqlite.
   (c) **description**: Description of database. It is optional eld.
   (d) **is_locking_required**: TRUE/FALSE, Control whether or not the database is made thread-safe by using locks around critical sections.

       This is pretty expensive, so if you know that your DB will only be used by a multi threads then you should set this to true.

       The default is false. It is optional eld.

> Note
>
> Siminov does not provide any security for database. If you want your database data needs to encrypted, then you can include SQLCipher implementation provided by Siminov framework in your application. For more detail see SQLCipher Encryption section of this developer guide.

   (e) **external_storage**:It speci es whether database resources needs to be saved on external storage or not (SDCard). It is optinal eld. Default is false.

2. Paths Of Database Mapping Descriptor Needed Under This Database Descriptor.

> Note
>
> (a) Provide full database mapping descriptor le path if you have used xml format to de ne ORM.
> (b) Provide full class path of database mapping descriptor POJO class if you have used annotation to de ne ORM.

3. Paths Of Library Descriptor Needed Under This Database Descriptor.

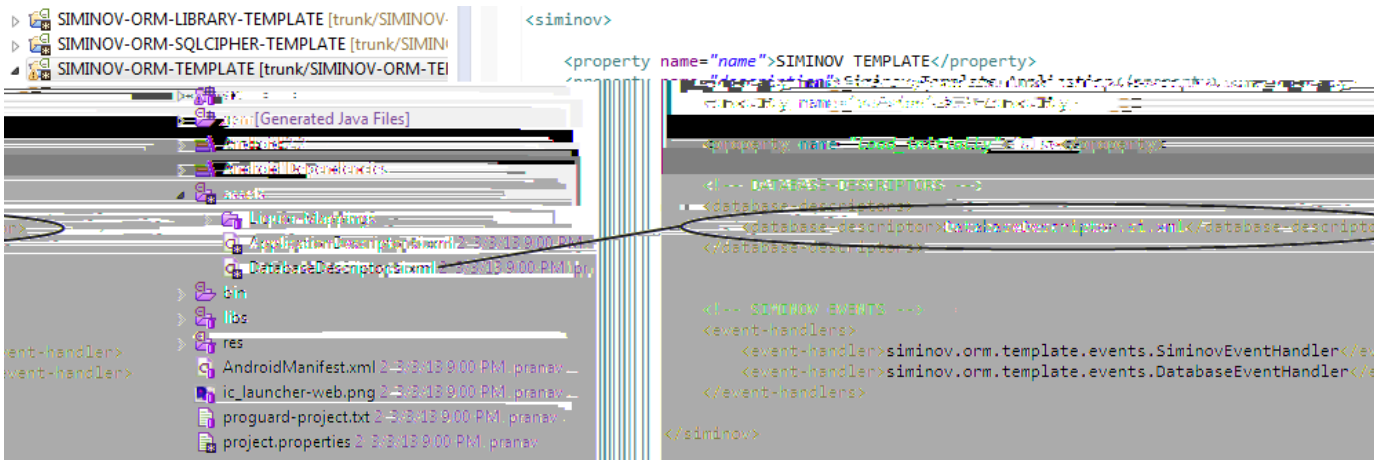
Note

(a) Provide full package name under which LibraryDescriptor.si.xml le is placed.
(b) Siminov framework will automatically read LibraryDescriptor.si.xml le de ned under package name provided.

Note

1. You can specify any name for DatabaseDescriptor.si.xml le.
2. If any database folder is created, it will be on the name of database de ned in DatabaseDescriptor.si.xml le.

Example: DatabaseDescriptor.si.xml le path


SIMINOV-ORM-LIBRARY-TEMPLATE [trunk/SIMINOV-
SIMINOV-ORM-SQLCIPHER-TEMPLATE [trunk/SIMIN
SIMINOV-ORM-TEMPLATE [trunk/SIMINOV-ORM-TE
gen [Generated Java Files]
assets
ApplicationDescriptor.si.xml
DatabaseDescriptor.si.xml
bin
libs
res
AndroidManifest.xml 2/3/13 9:00 PM pranav
ic_launcher-web.png 2/3/13 9:00 PM pranav
proguard-project.txt 2/3/13 9:00 PM pranav
project.properties 2/3/13 9:00 PM pranav

```
<siminov>

    <property name="name">SIMINOV TEMPLATE</property>
    [illegible]

    <property name="load_initially">[illegible]</property>

    <!-- DATABASE-DESCRIPTORS -->
    <database-descriptors>
        <database-descriptor>DatabaseDescriptor.si.xml</database-descripto
    </database-descriptors>


    <!-- SIMINOV EVENTS -->
    <event-handlers>
        <event-handler>siminov.orm.template.events.SiminovEventHandler</e
        <event-handler>siminov.orm.template.events.DatabaseEventHandler</e
    </event-handlers>

</siminov>
```

2.3 Library Descriptor (LibraryDescriptor.si.xml) Con guration

Library Descriptor is the one who de nes the properties of library.

```
    <!−− Design Of LibraryDescriptor.si.xml −−>

<library>

    <!−− General Library Properties −−>
        <!−− Mandatory Field −−>
  <property name="name">name_of_library</property>

    <!−− Optional Field −−>
  <property name="description">description_of_library</property>



  <!−− Database Mapping Descriptor Paths Needed Under This Library
      Descriptor −−>
    <!−− Optional Field −−>
  <database−mappings>
    <database−mapping path="
        full_path_of_database_mapping_descriptor_file" />
  </database−mappings>

</library>
```

Example: LibraryDescriptor.si.xml File Of Siminov Library Template.

```
<library>

  <property name="name">SIMINOV LIBRARY TEMPLATE</property>
  <property name="description">Siminov Library Template</property>

  <!−− Database Mappings −−>
  <database−mappings>
    <database−mapping path="Credential.si.xml" />
  </database−mappings>

</library>
```

Note

Application Developer can provide their own properties also, and by using following API's they can use properties.

1. Get Properties - getProperties(): It will return all properties associated with Library Descriptor.
2. Get Property - getProperty(Name of Property): It will return property value associated with property name provided.
3. Contains Property - containsProperty(Name of Property): It will return TRUE/FALSE whether property exists or not.
4. Add Property - addProperty(Name of Property, Value of Property): It will add new property to the collection of Library Descriptor properties.
5. Remove Property - removeProperty(Name of Property): It will remove property from Library Descriptor properties based on name provided.

Library Descriptor Elements:

1. General Properties Of Library.
   (a) **name***: Name of library. It is mandatory eld.
   (b) **description**: Description of library. It is optional eld.
2. Database Mapping Paths Needed Under This Database Descriptor.

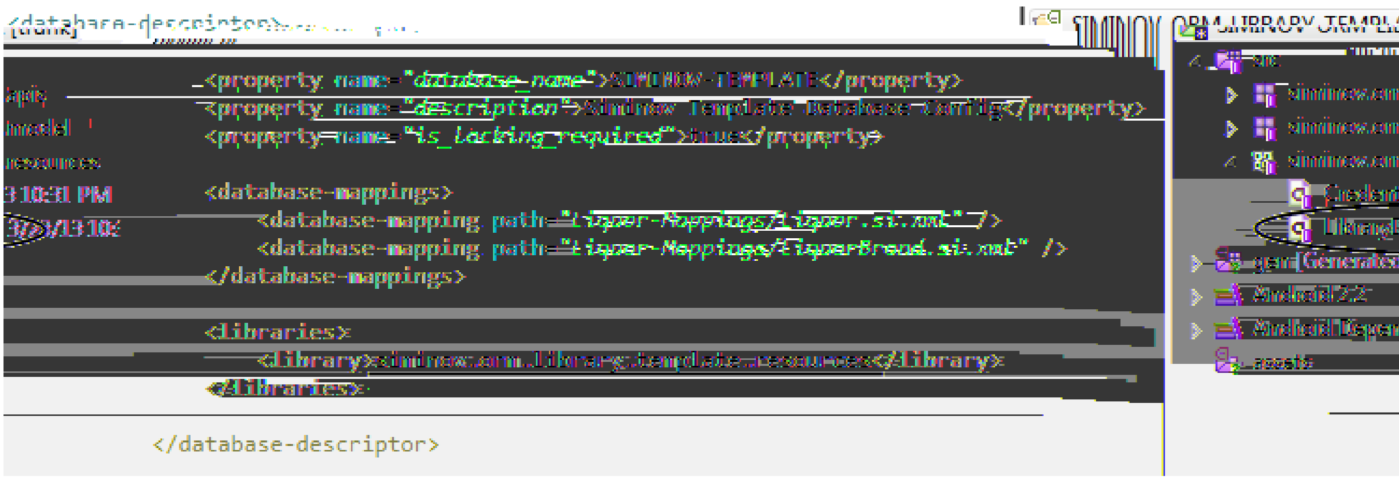
Note

(a) Provide full database mapping descriptor le path if you have used xml format to de ne ORM.
(b) Provide full class path of database mapping descriptor POJO class if you have used annotation to de ne ORM.

Note

1. Library descriptor le name should be same as LibraryDescriptor.si.xml.
2. It should always be in root package speci ed in DatabaseDescriptor.si.xml le.

Example: LibraryDescriptor.si.xml le path



```
<database-descriptor>
        <property name="database_name">SIMINOV-TEMPLATE</property>
        <property name="description">Siminov Template Database Config</property>
        <property name="is_locking_required">true</property>

        <database-mappings>
            <database-mapping path="Liquor-Mappings/Liquor.si.xml" />
            <database-mapping path="Liquor-Mappings/LiquorBrand.si.xml" />
        </database-mappings>

        <libraries>
            <library>siminov.orm.library.template.resources</library>
        </libraries>

</database-descriptor>
```

2.4 Database Mapping Descriptor (DatabaseMappingDescriptor.si.xml) Con guration

Database Mapping Descriptor is one which does ORM, it maps POJO class to database table.

```
    <!-- Design Of DatabaseMappingDescriptor.si.xml -->

<database-mapping>

    <!-- General Properties Of Table And Class -->

      <!-- TABLE_NAME: Mandatory Field -->
      <!-- CLASS_NAME: Mandatory Field -->
  <table table_name="name_of_table" class_name="
     mapped_pojo_class_name">

      <!-- Column Properties Required Under This Table -->

    <!-- Optional Field -->

      <!-- VARIABLE_NAME: Mandatory Field -->
      <!-- COLUMN_NAME: Mandatory Field -->
    <column variable_name="class_variable_name" column_name="
       column_name_of_table">

          <!-- Mandatory Field -->
      <property name="type">int/java.lang.Integer/long/java.lang.Long
         /float/java.lang.Float/boolean/java.lang.Boolean/char/java.
         lang.Character/java.lang.String/void/java.lang.Void/short/
         java.lang.Short</property>

        <!-- Optional Field (Default is false) -->
      <property name="primary_key">true/false</property>

        <!-- Optional Field (Default is false) -->
      <property name="not_null">true/false</property>

        <!-- Optional Field (Default is false) -->
      <property name="unique">true/false</property>

        <!-- Optional Field -->
      <property name="check">condition_to_be_checked (Eg:
         variable_name 'condition' value; variable_name > 0)</
         property>

        <!-- Optional Field -->
      <property name="default">default_value_of_column (Eg: 0.1)</
```

```
        property>

    </column>



    <!-- Index Properties -->

    <!-- Optional Field -->
      <!-- NAME: Mandatory Field -->
      <!-- UNIQUE: Optional Field (Default is false) -->
    <index name="name_of_index" unique="true/false">
      <column>column_name_needs_to_add</column>
    </index>



    <!-- Map Relationship Properties -->

    <!-- Optional Field -->
    <relationships>

          <!-- REFER: Mandatory Field -->
          <!-- REFER_TO: Mandatory Field -->
      <one-to-one refer="class_variable_name" refer_to="
          map_to_pojo_class_name" on_update="cascade/restrict/
          no_action/set_null/set_default" on_delete="cascade/restrict/
          no_action/set_null/set_default">

          <!-- Optional Field (Default is false) -->
        <property name="load">true/false</property>
      </one-to-one>

        <!-- REFER: Mandatory Field -->
          <!-- REFER_TO: Mandatory Field -->
      <one-to-many refer="class_variable_name" refer_to="
          map_to_pojo_class_name" on_update="cascade/restrict/
          no_action/set_null/set_default" on_delete="cascade/restrict/
          no_action/set_null/set_default">

          <!-- Optional Field (Default is false) -->
        <property name="load">true/false</property>
      </one-to-many>

        <!-- REFER: Mandatory Field -->
          <!-- REFER_TO: Mandatory Field -->
      <many-to-one refer="class_variable_name" refer_to="
          map_to_pojo_class_name" on_update="cascade/restrict/
          no_action/set_null/set_default" on_delete="cascade/restrict/
          no_action/set_null/set_default">

          <!-- Optional Field (Default is false) -->
        <property name="load">true/false</property>
      </many-to-one>

        <!-- REFER: Mandatory Field -->
          <!-- REFER_TO: Mandatory Field -->
      <many-to-many refer="class_variable_name" refer_to="
          map_to_pojo_class_name" on_update="cascade/restrict/
          no_action/set_null/set_default" on_delete="cascade/restrict/
          no_action/set_null/set_default">

          <!-- Optional Field (Default is false) -->
        <property name="load">true/false</property>
      </many-to-many>

    </relationships>

  </table>

</database-mapping>
```

Example: DatabaseMappingDescriptor.si.xml File Of Siminov Library Template.

```
<database-mapping>

  <table table_name="LIQUOR" class_name="siminov.orm.template.model.
      Liquor">

    <column variable_name="liquorType" column_name="LIQUOR_TYPE">
      <property name="type">TEXT</property>
      <property name="primary_key">true</property>
      <property name="not_null">true</property>
      <property name="unique">true</property>
    </column>

    <column variable_name="description" column_name="DESCRIPTION">
      <property name="type">TEXT</property>
    </column>

    <column variable_name="history" column_name="HISTORY">
      <property name="type">TEXT</property>
    </column>

    <column variable_name="link" column_name="LINK">
      <property name="type">TEXT</property>
      <property name="default">www.wikipedia.org</property>
    </column>

    <column variable_name="alcholContent" column_name="ALCHOL_CONTENT
       ">
      <property name="type">TEXT</property>
    </column>

    <index name="LIQUOR_INDEX_BASED_ON_LINK" unique="true">
      <column>HISTORY</column>
    </index>

    <relationships>

        <one-to-many refer="liquorBrands" refer_to="siminov.orm.
            template.model.LiquorBrand" on_update="cascade" on_delete=
            "cascade">
        <property name="load">true</property>
        </one-to-many>

    </relationships>

  </table>

</database-mapping>
```

Database Mapping Descriptor Elements:

1. TABLE TAG: It map database table to its corresponding POJO class.
   (a) table_name*: Name of table. It is mandatory eld.
   (b) class_name*: Name of POJO class name which is to be mapped to table name. It is mandatory eld.
2. COLUMN TAG: It map database table column to its corresponding variable of POJO class.
   (a) column_name*: Name of column. It is mandatory eld.
   (b) variable_name*: Name of variable. It is mandatory eld.

   Properties Of Column Tag

   (a) type*: Java variable type (*int*, *java.lang.Integer*, *long*, *java.lang.Long*, *oat*, *java.lang.Float*, *boolean*, *java.lang.Boolean*, *char*, *java.lang.Character*, *java.lang.String*, *byte*, *java.lang.Byte*, *void*, *java.lang.Void*, *short*, *java.lang.Short*). It is mandatory property.

       For more details see Data Type section of this developer guide.
   (b) primary_key: *TRUE/FALSE*. It de nes wheather the column is primary key of table or not. It is optional property. Default value is false.
   (c) not_null: *TRUE/FALSE*. It de nes wheather the column value can be empty or not. It is optional property. Default value is false.
   (d) unique: *TRUE/FALSE*. It de nes wheather the column value should be unique or not. It is optional property. Default value is false.
   (e) default: It de ne the default value of column. It is optional property.
   (f) check: It is used to put condition on column value. It is optional property.

Note

Application Developer can provide their own properties also, and by using following API's they can use properties.

(a) Get Properties - getProperties(): It will return all properties associated with Database Mapping Descriptor Column.
(b) Get Property - getProperty(Name of Property): It will return property value associated with property name provided.
(c) Contains Property - containsProperty(Name of Property): It will return TRUE/FALSE whether property exists or not.
(d) Add Property - addProperty(Name of Property, Value of Property): It will add new property to the collection of Database Mapping Descriptor Column properties.
(e) Remove Property - removeProperty(Name of Property): It will remove property from Database Mapping Descriptor Column properties based on name provided.

3. INDEX TAG: It de nes the stucture of index needed on the table.

   (a) name*: Name of the index. It is mandatory eld.

   (b) unique: *TRUE/FALSE*. It de nes wheather index needs to be unique or not. It is not mandatory property. Default value is false.

      (A unique index guarantees that the index key contains no duplicate values and therefore every row in the table is in some way unique).

   Index Column Tag

   (a) column: Name of columns included in index. Atleast one column should be included.

4. RELATIONSHIPS TAG: It de nes relationship between object.

   Relationship can be of four types:

   (a) ONE-TO-ONE:

$$< one - to - one >$$

   In a one-to-one relationship, each row in one database table is linked to 1 and only 1 other row in another table.

   (b) ONE-TO-MANY:

$$< one - to - many >$$

   In a one-to-many relationship, each row in the related to table can be related to many rows in the relating table. This e ectively save storage as the related record does not need to be stored multiple times in the relating table.

(c) MANY-TO-ONE:

$$< many - to - one >$$

In a many-to-one relationship one entity (typically a column or set of columns) contains values that refer to another entity (a column or set of columns) that has unique values.

(d) MANY-TO-MANY:

$$< many - to - many >$$

In a many-to-many relationship, one or more rows in a table can be related to 0, 1 or many rows in another table. A mapping table is required in order to implement such a relationship.

Relationship Attributes

(a) **refer***: Name of variable which needs to be mapped. It is mandatory eld.

(b) **refer_to***: Class name of mapped variable. It is mandatory eld.

(c) **on_update**: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to be done, when update occur.

(d) **on_delete**: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to be done, when delete occur.

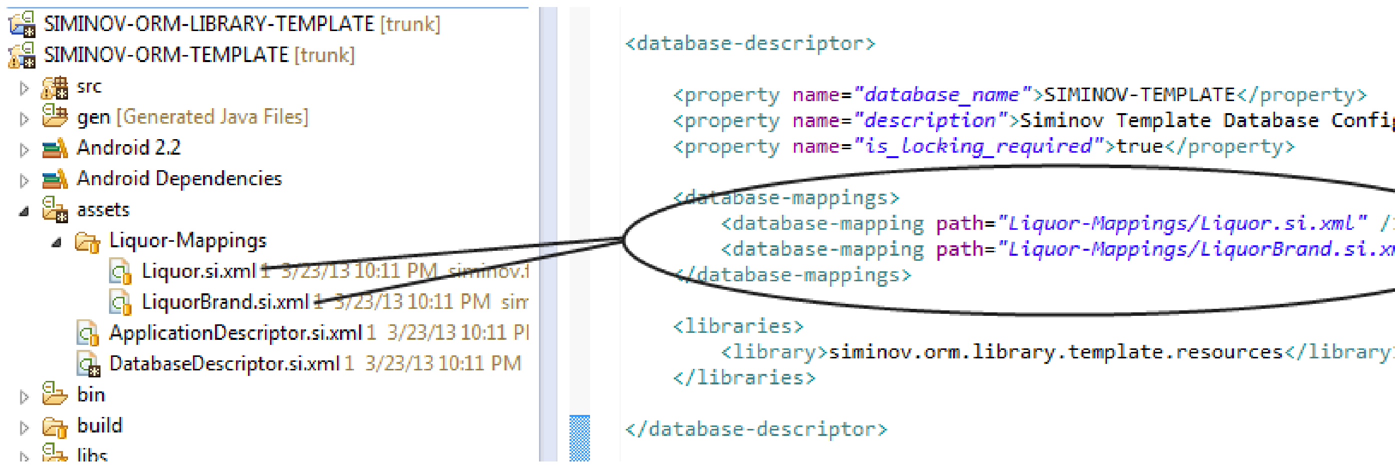
Relationship Properties

(a) **load**: It de nes whether it need to be load or not.

Note

1. Application developer can assign any name to DatabaseMappingDescriptor.si.xml le.
2. It descriptor le should be in same place as per de ned in DatabaseDescriptor.si.xml le.

Example: DatabaseMappingDescriptor.si.xml le path



```
SIMINOV-ORM-LIBRARY-TEMPLATE [trunk]
SIMINOV-ORM-TEMPLATE [trunk]
  src
  gen [Generated Java Files]
  Android 2.2
  Android Dependencies
  assets
    Liquor-Mappings
      Liquor.si.xml 1  3/23/13 10:11 PM  siminov.t
      LiquorBrand.si.xml 1  3/23/13 10:11 PM  sir
    ApplicationDescriptor.si.xml 1  3/23/13 10:11 PI
    DatabaseDescriptor.si.xml 1  3/23/13 10:11 PM
  bin
  build
  libs
```

```
<database-descriptor>

    <property name="database_name">SIMINOV-TEMPLATE</property>
    <property name="description">Siminov Template Database Confi
    <property name="is_locking_required">true</property>

    <database-mappings>
        <database-mapping path="Liquor-Mappings/Liquor.si.xml" /
        <database-mapping path="Liquor-Mappings/LiquorBrand.si.x
    </database-mappings>

    <libraries>
        <library>siminov.orm.library.template.resources</library
    </libraries>

</database-descriptor>
```

2.5 Database Mapping Descriptor Through Annotation

Annotation is another way to do ORM, it maps POJO class to database table.

(Annotations provide data about a program that is not part of the program itself. They have no direct e ect on the operation of the code they annotate)

```
@Table(tableName="name_of_table")
@Indexes({
  @Index(name="name_of_index", unique="true/false", value={
    @IndexColumn(column="column_name_needs_to_add")
  }),
})
public class DatabaseMappingDescriptor {

  @Column(columnName="column_name_of_table",
       properties={
         @Property(name="primary_key", value="true/false"),
         @Property(name="not_null", value="true/false"),
         @Property(name="unique", value="true/false")
         @Property(name="check", value="condition_to_be_checked (Eg:
             variable_name 'condition' value; variable_name > 0)")
         @Property(name="default", value="default_value_of_column (Eg:
              0.1)")
         })
  @OneToOne(onUpdate="cascade/restrict/no_action/set_null/set_default
       ", onDelete="cascade/restrict/no_action/set_null/set_default",
   properties={
     @RelationshipProperty(name=RelationshipProperty.LOAD, value="true
         ")
     })
  @OneToMany(onUpdate="cascade/restrict/no_action/set_null/
       set_default", onDelete="cascade/restrict/no_action/set_null/
       set_default",
   properties={
     @RelationshipProperty(name=RelationshipProperty.LOAD, value="true
         ")
     })
  @ManyToOne(onUpdate="cascade/restrict/no_action/set_null/
       set_default", onDelete="cascade/restrict/no_action/set_null/
       set_default",
   properties={
     @RelationshipProperty(name=RelationshipProperty.LOAD, value="true
         ")
     })
  @ManyToMany(onUpdate="cascade/restrict/no_action/set_null/
       set_default", onDelete="cascade/restrict/no_action/set_null/
```

```
      set_default",
  properties={
    @RelationshipProperty(name=RelationshipProperty.LOAD, value="true
       ")
    })
  private String variableName = null;

}
```

Example: Liquor Class Of Siminov Template Application.

```
@Table(tableName=Liquor.TABLE_NAME)
@Indexes({
  @Index(name="LIQUOR_INDEX_BASED_ON_LINK", unique=true, value={
    @IndexColumn(column=Liquor.LINK)
  }),
})
public class Liquor extends Database implements Serializable {

  @Column(columnName=LIQUOR_TYPE,
      properties={
        @Property(name=Property.PRIMARY_KEY, value="true"),
        @Property(name=Property.NOT_NULL, value="true"),
        @Property(name=Property.UNIQUE, value="true")
        })
  private String liquorType = null;

  @Column(columnName=DESCRIPTION)
  private String description = null;

  @Column(columnName=HISTORY)
  private String history = null;

  @Column(columnName=LINK,
      properties={
        @Property(name=Property.DEFAULT, value="www.wikipedia.org")
        })
  private String link = null;

  @Column(columnName=ALCHOL_CONTENT)
  private String alcholContent = null;

  @OneToMany(onUpdate="cascade", onDelete="cascade",
      properties={
        @RelationshipProperty(name=RelationshipProperty.LOAD, value="
           true")
    })
  private ArrayList<LiquorBrand> liquorBrands = null;

}
```

Di erent Annotation Tags:

1. **@Table**: It map database table to its corresponding POJO class.
   (a) **tableName***: Name of table. It is mandatory eld.
2. **@Indexes**: It contain all index required on table.
3. **@Index**: It de nes the stucture of index needed on the table.
   (a) **name***: Name of the index. It is mandatory eld.
   (b) **unique**: *TRUE/FALSE*. It de nes wheather index needs to be unique or not. It is not mandatory property. Default value is false.

   (A unique index guarantees that the index key contains no duplicate values and therefore every row in the table is in some way unique).

   Index Column Tag

   (a) **column**: Name of columns included in index. Atleast one column should be included.
4. **@Column**: It map database table column to its corresponding variable of POJO class.
   (a) **columnName***: Name of column. It is mandatory eld.

   Properties Of Column: **@Properties**: It contain all properties needed by column.

   Properties Of Column Tag: **@Property**: This tag de nes a perticular property of column.

   (a) **primary_key | ColumnProperty.PRIMARY_KEY**: *TRUE/FALSE*. It de nes wheather the column is primary key of table or not. It is optional property. Default value is false.
   (b) **not_null | ColumnProperty.NOT_NULL**: *TRUE/FALSE*. It de nes wheather the column value can be empty or not. It is optional property. Default value is false.
   (c) **unique | ColumnProperty.UNIQUE**: *TRUE/FALSE*. It de nes wheather the column value should be unique or not. It is optional property. Default value is false.
   (d) **default | ColumnProperty.DEFAULT**: It de ne the default value of column. It is optional property.
   (e) **check | ColumnProperty.CHECK**: It is used to put condition on column value. It is optional property.
5. **@OneToOne**: This tag de nes one-to-one relationship, where each row in one database table is linked to 1 and only 1 other row in another table.

   (a) onUpdate: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to performed, when update occur.

   (b) onDelete: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to be performed, when delete occur.

6. @OneToMany: This tag de nes one-to-many relationship, where each row in the related to table can be related to many rows in the relating table. This e ectively save storage as the related record does not need to be stored multiple times in the relating table.

   (a) onUpdate: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to performed, when update occur.

   (b) onDelete: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to be performed, when delete occur.

7. @ManyToOne: This tag de nes one-to-many relationship, where one entity (typically a column or set of columns) contains values that refer to another entity. (a column or set of columns) that has unique values.

   (a) onUpdate: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to performed, when update occur.

   (b) onDelete: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to be performed, when delete occur.

8. @ManyToMany: This tag de nes one-to-many relationship, where one or more rows in a table can be related to 0, 1 or many rows in another table. A mapping table is required in order to implement such a relationship.

   (a) onUpdate: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to performed, when update occur.

   (b) onDelete: *cascade / restrict / no_action / set_null / set_default*. It de nes action needs to be performed, when delete occur.

   RELATIONSHIP PROPERTY TAG: @RelationshipProperty

   (a) load: *TRUE/FALSE*, It de nes whether it need to be load or not.

Chapter 3

Siminov Initialization

3.1 Initializating Siminov

Every application when it starts they need to rst initialize Siminov. They can do this by invoking initialize method of siminov.orm.Siminov class by passing **ApplicationContext** object as paramter to method.

There are two ways to initialize Siminov

1. **Initializing Siminov From Sub-Class Of Application**

```
public class ApplicationSiminov extends Application {

  public void onCreate() {
    super.onCreate();

    initializeSiminov();
  }

  private void initializeSiminov() {

    IInitializer initializer = siminov.orm.Siminov.initialize();
    initializer.addParameter(this);

    initializer.start();
  }

}
```

Note

Android provides support in every application to create an application wide class. The base class for this is the android.app.Application class.

2. **Initializing Siminov From Sub-Class Of Activity**

```
public class HomeActivity extends Activity {

  public void onCreate(Bundle savedInstanceState) {
    initializeSiminov();
  }

  private void initializeSiminov() {

  IInitializer initializer = siminov.orm.Siminov.initialize();
    initializer.addParameter(getApplicationContext());

    initializer.start();
  }

}
```

Note

If you are initializing Siminov from activity sub class then you should pass application context not the activity context. See above example.

Note

1. Application should call siminov.orm.Siminov initialize only once in the life time of application.
2. Once Siminov initialize it can not be re initialized.

IInitializer Interface: siminov.orm.Siminov.initialize() API returns IInitializer interface implemented class through which we can pass parameters needed by Siminov Framework to work functionally.

```

public interface IInitializer {
  public void addParameter(Object object);

  public void start();

}
```

Steps performed in initializating Siminov Framework

1. **Database Creation**: Siminov provides **Initialization Layer** which handles the creation of databases required by application.

   Siminov follows below steps to create databases required by application.

   (a) **Step 1**: Then application invokes initialize method, it checks wheather database exists or not.

   (b) **Step 2**: If application database does not exists, Siminov will create database required by the application.

   (c) **Step 3**: If application database exists, then it will read all desriptors de ned by application based on load_initially property de ned in ApplicationDescriptor.si.xml le.

2. **Initialize Resources Layer** : Any resource created by Siminov Framework is places in resource layer of Siminov. You can use API's provided by Resources class to get required object.

```
public final class Resources {

  public static Resources getInstance();

  public Context getApplicationContext();

  public ApplicationDescriptor getApplicationDescriptor();

  public Iterator<String> getDatabaseDescriptorsPaths();

  public DatabaseDescriptor getDatabaseDescriptorBasedOnPath(
      final String databaseDescriptorPath);

  public DatabaseDescriptor getDatabaseDescriptorBasedOnName(
      final String databaseDescriptorName);

  public Iterator<DatabaseDescriptor> getDatabaseDescriptors();

  public DatabaseDescriptor
      getDatabaseDescriptorBasedOnClassName(final String
      className);
```

```
public DatabaseDescriptor
    getDatabaseDescriptorBasedOnTableName(final String
    tableName);

public DatabaseMappingDescriptor
    getDatabaseMappingBasedOnClassName(final String className);

public DatabaseMappingDescriptor
    requiredDatabaseMappingDescriptorBasedOnClassName(final
    String className) throws CoreException;

public DatabaseMappingDescriptor
    getDatabaseMappingBasedOnTableName(final String tableName);

public Iterator<
```

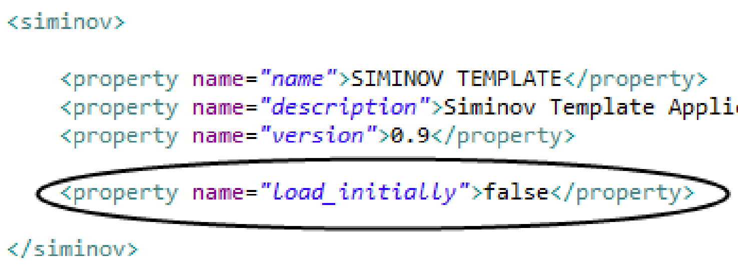
```
}
```

3.2 Lazy Initialization VS Initial Initialization

Siminov Framework provides easy con guration property in ApplicationDescriptor.si.xml le through which application developer can signi cantly improve the performance of Siminov.

1. **load_initially**: *TRUE/FALSE*, It is not mandatory eld, and default value is FALSE.

   Example: Siminov Template Application Example.

   ```
   <siminov>

       <property name="name">SIMINOV TEMPLATE</property>
       <property name="description">Siminov Template Application</property>
       <property name="version">0.9</property>

       <property name="load_initially">false</property>

   </siminov>
   ```

   Basically to map entity and database table, Siminov need DatabaseMappingDescriptor object which de nes relation between entity and database table, this object is created by reading DatabaseMappingDescriptor.si.xml le de ned by application.

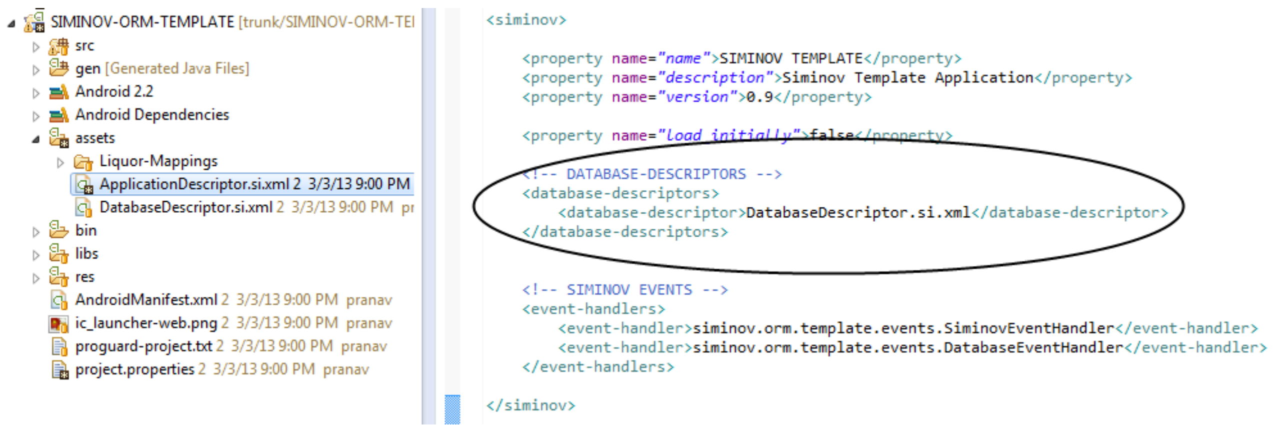
   Lazy Initialization (load_initially=true): If load_initially is set to true then Siminov will load all descriptors at time of Siminov initialization and will create all its corresponding DatabaseMappingDescriptor objects, and place them in resources layer of Siminov.

   Initial Initialization (load_initially=false): If load_initially is set to false then Siminov will not load all descriptors at time of Siminov initialization. It will load descriptor only when it is required by Siminov.

3.3 Handling Multiple Schema's

Siminov framework supports multiple schema's if required by application. Basically each schema is de ned using DatabaseDescriptor.si.xml le. You need to specify all DatabaseDescriptor.si.xml le path in ApplicationDescriptor.si.xml.

Example: Siminov Template Application Example.



```
SIMINOV-ORM-TEMPLATE [trunk/SIMINOV-ORM-TE
  src
  gen [Generated Java Files]
  Android 2.2
  Android Dependencies
  assets
    Liquor-Mappings
    ApplicationDescriptor.si.xml 2  3/3/13 9:00 PM
    DatabaseDescriptor.si.xml 2  3/3/13 9:00 PM  pr
  bin
  libs
  res
  AndroidManifest.xml 2  3/3/13 9:00 PM  pranav
  ic_launcher-web.png 2  3/3/13 9:00 PM  pranav
  proguard-project.txt 2  3/3/13 9:00 PM  pranav
  project.properties 2  3/3/13 9:00 PM  pranav
```

```xml
<siminov>

    <property name="name">SIMINOV TEMPLATE</property>
    <property name="description">Siminov Template Application</property>
    <property name="version">0.9</property>

    <property name="load_initially">false</property>

    <!-- DATABASE-DESCRIPTORS -->
    <database-descriptors>
        <database-descriptor>DatabaseDescriptor.si.xml</database-descriptor>
    </database-descriptors>


    <!-- SIMINOV EVENTS -->
    <event-handlers>
        <event-handler>siminov.orm.template.events.SiminovEventHandler</event-handler>
        <event-handler>siminov.orm.template.events.DatabaseEventHandler</event-handler>
    </event-handlers>

</siminov>
```

Chapter 4

Event Noti ers

Siminov framework provides few event noti ers which gets triggered based on particular action. Application have to provide implementation for these event noti ers and register them with Siminov.

4.1 ISiminov Events

It provide API's related to life cycle of Siminov framework

Example: ISiminov Events Noti er

```
public interface ISiminovEvents {

  public void firstTimeSiminovInitialized();

  public void siminovInitialized();

  public void siminovStopped();

}
```

1. **First Time Siminov Initialized - rstTimSiminovInitialized():** It is triggered when Siminov is initialized for rst time. In this you can perform tasks which are related to initialization of things only rst time of application starts.

   Example:

   Preparing initial data for application, which is required by application in its life time, Since it is to be done only once, therefore we will use rstTimeSiminovInitialized API.

```
  public void firstTimeSiminovInitialized() {
    new DatabaseUtils().prepareData();
  }
```

Note
This API will be triggered only once when Siminov is initialized rst time.

2. **Siminov Initialized - siminovInitialized():** It is triggered whenever Siminov is initialized.

Note
This doesnot gets triggered when Siminov is rst time initialized, instead of this rstTimeSiminovInitialized APIwill be triggered.

3. **Siminov Stopped - siminovStopped():** It is triggered when Siminov is shut-down.

4.2 IDatabaseEvents

It provide API's related to database operations.

Example: IDatabaseEvents Noti er

```
public interface IDatabaseEvents {

  public void databaseCreated(final DatabaseDescriptor
      databaseDescriptor);

  public void databaseDropped(final DatabaseDescriptor
      databaseDescriptor);


  public void tableCreated(final DatabaseMappingDescriptor
      databaseMapping);

  public void tableDropped(final DatabaseMappingDescriptor
      databaseMapping);


  public void indexCreated(final DatabaseMappingDescriptor
      databaseMapping, Index index);
```

```
    public void indexDropped(final DatabaseMappingDescriptor
        databaseMapping, Index index);

}
```

1. **Database Created - databaseCreated(DatabaseDescriptor)**: It is triggered when database is created based on schema de ned in DatabaseDescriptor.si.xml le. This API provides DatabaseDescriptor object for which database is created.
2. **Database Dropped - databaseDropped(DatabaseDescriptor)**: It is triggered when database is dropped. This API provides DatabaseDescriptor object for which database is dropped.
3. **Table Created - tableCreated(DatabaseMappingDescriptor)**: It is triggered when a table is created in database. This API provides DatabaseMappingDescriptor object which describes table structure.
4. **Table Dropped - tableDropped(DatabaseMappingDescriptor)**: It is triggered when a table is deleted from database. This API provides DatabaseMappingDescriptor object for which table is dropped.
5. **Index Created - indexCreated(DatabaseMappingDescriptor, Index)**: It is triggered when a index is created on table. This API provides DatabaseMappingDescriptor and Index object which de nes table and index structure.
6. **Index Dropped - indexDropped(DatabaseMappingDescriptor, Index)**: It is triggered when a index is dropped from table. This API provides DatabaseMappingDescriptor and Index object which de nes table and index for which index is dropped.

Chapter 5

Database API's

5.1 Data Types

Based on java variable type siminov decides the data type of column.

1. int: Java int primitive data type is converted to INTEGER Sqlite data type.

   Example: Java int Primitive Data Type

   ```
   <column variable_name="name-of-variable" column_name="name-
       of-column">
     <property name="type">int</property>
   </column>
   ```

2. Integer: Java Integer class data type is converted to INTEGER Sqlite data type.

   Example: Java Integer Class Data Type

   ```
   <column variable_name="name-of-variable" column_name="name-
       of-column">
     <property name="type">java.lang.Integer</property>
   </column>
   ```

3. long: Java long primitive data type is converted to INTEGER Sqlite data type.

   Example: Java long Primitive Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">long</property>
</column>
```

4. **Long**: Java Long class data type is converted to **INTEGER** Sqlite data type.

   Example: Java Long Class Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">java.lang.Long</property>
</column>
```

5. **oat**: Java oat primitive data type is converted to **REAL** Sqlite data type.

   Example: Java oat Primitive Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">float</property>
</column>
```

6. **Float**: Java Float class data type is converted to **REAL** Sqlite data type.

   Example: Java Float Class Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">java.lang.Float</property>
</column>
```

7. **boolean**: Java boolean primitive data type is converted to **NUMERIC** Sqlite data type.

   Example: Java boolean Primitive Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">boolean</property>
</column>
```

8. **Boolean**: Java Boolean class data type is converted to NUMERIC Sqlite data type.

   Example: Java Boolean Class Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">java.lang.Boolean</property>
</column>
```

9. **char**: Java char array primitive data type is converted to TEXT Sqlite data type.

   Example: Java char Array Primitive Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">char</property>
</column>
```

10. **Character**: Java Character class data type is converted to TEXT Sqlite data type.

    Example: Java Character Class Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">java.lang.Character</property>
</column>
```

11. **String**: Java String class data type is converted to TEXT Sqlite data type.

    Example: Java String Class Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">java.lang.String</property>
</column>
```

12. **byte**: Java byte array data type is converted to NONE Sqlite data type.

    Example: Java byte Array Primitive Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">byte</property>
</column>
```

13. **Byte**: Java Byte class data type is converted to **NONE** Sqlite data type.

    Example: Java Byte Class Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">java.lang.Byte</property>
</column>
```

14. **void**: Java void primitive data type is converted to **NONE** Sqlite data type.

    Example: Java void Primitive Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">void</property>
</column>
```

15. **Void**: Java Void class data type is converted to **NONE** Sqlite data type.

    Example: Java Void Class Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">java.lang.Void</property>
</column>
```

16. **short**: Java short primitive is convered to **INTEGER** Sqlite data type.

    Example: Java short Primitive Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">short</property>
</column>
```

17. **Short**: Java Short class data type is converted to **INTEGER** Sqlite data type.

    Example: Java Short Class Data Type

    ```
        <column variable_name="name-of-variable" column_name="name-
            of-column">
          <property name="type">java.lang.Sort</property>
        </column>
    ```

Sqlite data types:

1. **INTEGER Sqlite Data Type**: This sqlite data type generaly contain *INT*, *INTEGER*, *TINYINT*, *SMALLINT*, *MEDIUMINT*, *BIGINT*, *UNSIGNED BIG INT*, *INT2*, *INT8*.

2. **TEXT Sqlite Data Type**: This sqlite data type generaly contain *CHARACTER(20)*, *VARCHAR(255)*, *VARYING CHARACTER(255)*, *NCHAR(55)*, *NATIVE CHARACTER(70)*, *NVARCHAR(100)*, *TEXT*, *CLOB*.

3. **REAL Sqlite Data Type**: This sqlite data type generaly contain *REAL*, *DOUBLE*, *DOUBLE PRECISION*, *FLOAT*.

4. **NONE Sqlite Data Type**: This sqlite data type generaly contain *BLOB*, *NO DATA TYPE SPECIFIED*.

5. **NUMERIC Sqlite Data Type**: This sqlite data type generaly contain *NUMERIC*, *DECIMAL(10,5)*, *BOOLEAN*, *DATE*, *DATETIME*.

Note

If you de ne ORM using Annotation then you dont have to specify data type, it will automatically be con gured based on variable data type.

5.2 Database API's

5.2.1 Create Database

Siminov provides APIs to create database, based on schema de ned in DatabaseDescriptor.si.xml le.

API: Basically Use IDatabase Interface

```
public void openOrCreate(final String path) throws
    DatabaseException;
```

Note

Generally database creation will be automatically handled by Siminov, but you can do it manually also.

Example: Manually Creating Database.

```
    DatabaseDescriptor databaseDescriptor = Resources.
        getInstance().getDatabaseDescriptorBasedOnName(
        name-of-database);

    IDatabase database = null;
    try {
      database = Database.createDatabase(
          databaseDescriptor);
    } catch(DatabaseException databaseException) {
      //Log It.
    }


    try {
      database.openOrCreate(database_path +
          database_name);
    } catch {
      //Log It.
    }
```

5.2.2 Drop Database

Database class provides API to drop complete database of an application.

API: Drop Database.

```
public static final void dropDatabase(final DatabaseDescriptor
    databaseDescriptor) throws DatabaseException;
```

Example

```
DatabaseDescriptor databaseDescriptor = new Liquor().
    getDatabaseDescriptor();
```

```
try {
  Database.dropDatabase(databaseDescriptor);
} catch(DatabaseException databaseException) {
  //Log It.
}
```

5.2.3 Create Table

Using this API you can create table on database.

> **Note**
> Generally database and its table creation will automatically be handled by Siminov, but u can do it manually also.

There are three ways to create table in database.

1. De ning table structure using DatabaseMappingDescriptor.si.xml le

   Example: Liquor.si.xml le Of Siminov Template Application.

```
<database-mapping>

  <table table_name="LIQUOR" class_name="siminov.orm.template.
      model.Liquor">

    <column variable_name="liquorType" column_name="LIQUOR_TYPE"
        >
      <property name="type">TEXT</property>
      <property name="primary_key">true</property>
      <property name="not_null">true</property>
      <property name="unique">true</property>
    </column>

    <column variable_name="description" column_name="DESCRIPTION
        ">
      <property name="type">TEXT</property>
    </column>

    <column variable_name="history" column_name="HISTORY">
      <property name="type">TEXT</property>
    </column>

    <column variable_name="link" column_name="LINK">
      <property name="type">TEXT</property>
      <property name="default">www.wikipedia.org</property>
    </column>

    <column variable_name="alcholContent" column_name="
        ALCHOL_CONTENT">
      <property name="type">TEXT</property>
    </column>

    <index name="LIQUOR_INDEX_BASED_ON_LINK" unique="true">
      <column>HISTORY</column>
    </index>

    <relationships>

        <one-to-many refer="liquorBrands" refer_to="siminov.orm.
            template.model.LiquorBrand" on_update="cascade"
            on_delete="cascade">
        <property name="load">true</property>
      </one-to-many>

    </relationships>

  </table>

</database-mapping>
```

2. De ning table structure using Annotations

   Example: Liquor Class Of Siminov Template Application.

```
@Table(tableName=Liquor.TABLE_NAME)
@Indexes({
  @Index(name="LIQUOR_INDEX_BASED_ON_LINK", unique=true, value={
    @IndexColumn(column=Liquor.LINK)
  }),
})
public class Liquor extends Database implements Serializable {

  //Table Name
  transient public static final String TABLE_NAME = "LIQUOR";

  //Column Names
  transient public static final String LIQUOR_TYPE = "
      LIQUOR_TYPE";
  transient public static final String DESCRIPTION = "
      DESCRIPTION";
  transient public static final String HISTORY = "HISTORY";
  transient public static final String LINK = "LINK";
transient public static final String ALCHOL_CONTENT = "
    ALCHOL_CONTENT";

//Liquor Types
transient public static final String LIQUOR_TYPE_GIN = "Gin";
transient public static final String LIQUOR_TYPE_RUM = "Rum";
transient public static final String LIQUOR_TYPE_TEQUILA = "
    Tequila";
transient public static final String LIQUOR_TYPE_VODKA = "
    Vodka";
transient public static final String LIQUOR_TYPE_WHISKEY = "
    Whiskey";
transient public static final String LIQUOR_TYPE_BEER = "Beer"
    ;
transient public static final String LIQUOR_TYPE_WINE = "Wine"
    ;


//Variables

@Column(columnName=LIQUOR_TYPE,
     properties={
       @ColumnProperty(name=ColumnProperty.PRIMARY_KEY, value="
           true"),
       @ColumnProperty(name=ColumnProperty.NOT_NULL, value="
           true"),
       @ColumnProperty(name=ColumnProperty.UNIQUE, value="true"
           )
       })
private String liquorType = null;

@Column(columnName=DESCRIPTION)
private String description = null;

@Column(columnName=HISTORY)
private String history = null;

@Column(columnName=LINK,
     properties={
       @ColumnProperty(name=ColumnProperty.DEFAULT, value="www.
           wikipedia.org")
       })
private String link = null;

@Column(columnName=ALCHOL_CONTENT)
private String alcholContent = null;

@OneToMany(onUpdate="cascade", onDelete="cascade",
     properties={
       @RelationshipProperty(name=RelationshipProperty.LOAD,
           value="true")
   })
private ArrayList<LiquorBrand> liquorBrands = null;

//Methods

public String getLiquorType() {
   return this.liquorType;
}

public void setLiquorType(String liquorType) {
   this.liquorType = liquorType;
}

public String getDescription() {
   return this.description;
}

public void setDescription(String description) {
   this.description = description;
}

public String getHistory() {
   return this.history;
}

public void setHistory(String history) {
   this.history = history;
}

public String getLink() {
   return this.link;
}

public void setLink(String link) {
   this.link = link;
}

public String getAlcholContent() {
   return this.alcholContent;
}

public void setAlcholContent(String alcholContent) {
   this.alcholContent = alcholContent;
}

public ArrayList<LiquorBrand> getLiquorBrands() {
   return this.liquorBrands;
   }

   public void setLiquorBrands(ArrayList<LiquorBrand>
       liquorBrands) {
       this.liquorBrands = liquorBrands;
   }
}
```

3. **Creating table programmatically**

   Example: Creating Liquor Table Programmatically.

```
DatabaseMapping databaseMapping = new DatabaseMapping();
databaseMapping.setTableName("LIQUOR");
databaseMapping.setClassName(Liquor.class.getName());

//Add Liquor Type.
DatabaseMapping.Column liquorType = databaseMapping.new Column()
    ;
liquorType.setVariableName("liquorType");
liquorType.setColumnName("LIQUOR_TYPE");

liquorType.setType("TEXT");

liquorType.setPrimaryKey(true);
liquorType.setNotNull(true);
liquorType.setUnique(false);

liquorType.setGetterMethodName("getLiquorType");
liquorType.setSetterMethodName("setLiquorType");

databaseMapping.addColumn(liquorType);

//Add Liquor Description.
DatabaseMapping.Column description = databaseMapping.new Column
    ();
description.setVariableName("description");
description.setColumnName("DESCRIPTION");

description.setType("TEXT");

description.setGetterMethodName("getDescription");
description.setSetterMethodName("setDescription");

databaseMapping.addColumn(description);

//Add History.
DatabaseMapping.Column history = databaseMapping.new Column();
```

```
history.setVariableName("history");
history.setColumnName("HISTORY");

history.setType("TEXT");

history.setGetterMethodName("getHistory");
history.setSetterMethodName("setHistory");

databaseMapping.addColumn(history);

//Add Link.
DatabaseMapping.Column link = databaseMapping.new Column();
link.setVariableName("history");
link.setColumnName("HISTORY");

link.setType("TEXT");
link.setDefault("www.wikipedia.org");

link.setGetterMethodName("getLink");
link.setSetterMethodName("setLink");

databaseMapping.addColumn(link);

//Add Alchol Content.
DatabaseMapping.Column alcholContent = databaseMapping.new
    Column();
alcholContent.setVariableName("alcholContent");
alcholContent.setColumnName("ALCHOL_CONTENT");

alcholContent.setType("TEXT");

alcholContent.setGetterMethodName("getAlcholContent");
alcholContent.setSetterMethodName("setAlcholContent");

databaseMapping.addColumn(alcholContent);

//Create Index On Liquor table.
DatabaseMapping.Index indexOnLiquor = databaseMapping.new Index
    ();
indexOnLiquor.setName("LIQUOR_INDEX_BASED_ON_LINK");
indexOnLiquor.setUnique(true);

//Add Columns on which we need index.
indexOnLiquor.addColumn("LINK");

databaseMapping.addIndex(indexOnLiquor);

Collection<DatabaseMapping> databaseMappings = new ArrayList<
    DatabaseMapping> ();
```

```
databaseMappings.add(databaseMapping);

try {
  Database.createTables(databaseMappings.iterator());
} catch(DatabaseException databaseException) {
  //Log It.
}
```

Siminov **Database** class provide few APIs to create table programmtically.

1.
```
public static final void createTables(final Iterator<
    DatabaseMappingDescriptor> databaseMappings) throws
    DatabaseException;
```

Example:

```
DatabaseDescriptor databaseDescriptor = Resources.getInstance().
    getDatabaseDescriptorBasedOnName(name-of-database);

try {
  Database.createTables(databaseDescriptor.
      orderedDatabaseMappings());
} catch(DatabaseException databaseException) {
  \\Log It.
}
```

2.
```
  public static final void createTable(final
      DatabaseMappingDescriptor databaseMapping) throws
      DatabaseException;
```

Example: Creating Liquor Table Programmatically.

```
    //Defines structure for Liquor table.
    DatabaseMapping databaseMapping = new DatabaseMapping();
    databaseMapping.setTableName("LIQUOR");
    databaseMapping.setClassName(Liquor.class.getName());

    //Add Liquor Type.
    DatabaseMapping.Column liquorType = databaseMapping.new
        Column();
```

```
liquorType.setVariableName("liquorType");
liquorType.setColumnName("LIQUOR_TYPE");

liquorType.setType("TEXT");

liquorType.setPrimaryKey(true);
liquorType.setNotNull(true);
liquorType.setUnique(false);

liquorType.setGetterMethodName("getLiquorType");
liquorType.setSetterMethodName("setLiquorType");

databaseMapping.addColumn(liquorType);

//Add Liquor Description.
DatabaseMapping.Column description = databaseMapping.new
    Column();
description.setVariableName("description");
description.setColumnName("DESCRIPTION");

description.setType("TEXT");

description.setGetterMethodName("getDescription");
description.setSetterMethodName("setDescription");

databaseMapping.addColumn(description);

//Add History.
DatabaseMapping.Column history = databaseMapping.new Column
    ();
history.setVariableName("history");
history.setColumnName("HISTORY");

history.setType("TEXT");

history.setGetterMethodName("getHistory");
history.setSetterMethodName("setHistory");

databaseMapping.addColumn(history);

//Add Link.
DatabaseMapping.Column link = databaseMapping.new Column();
link.setVariableName("history");
link.setColumnName("HISTORY");

link.setType("TEXT");
link.setDefault("www.wikipedia.org");

link.setGetterMethodName("getLink");
link.setSetterMethodName("setLink");

databaseMapping.addColumn(link);

//Add Alchol Content.
DatabaseMapping.Column alcholContent = databaseMapping.new
    Column();
alcholContent.setVariableName("alcholContent");
alcholContent.setColumnName("ALCHOL_CONTENT");

alcholContent.setType("TEXT");

alcholContent.setGetterMethodName("getAlcholContent");
alcholContent.setSetterMethodName("setAlcholContent");

databaseMapping.addColumn(alcholContent);

//Create Index On Liquor table.
DatabaseMapping.Index indexOnLiquor = databaseMapping.new
    Index();
indexOnLiquor.setName("LIQUOR_INDEX_BASED_ON_LINK");
indexOnLiquor.setUnique(true);

//Add Columns on which we need index.
indexOnLiquor.addColumn("LINK");

databaseMapping.addIndex(indexOnLiquor);

try {
  Database.createTables(databaseMapping);
} catch(DatabaseException databaseException) {
  //Log It.
}
```

5.2.4 Drop Table

Database class provides following API's to drop a table.

1.
```
public void dropTable() throws DatabaseException;
```

Example: Drop Liquor Table Through Liquor Class Object.

```
  try {
    new Liquor().dropTable();
  } catch(DatabaseException databaseException) {

  }
```

2.
```
public static final void dropTable(final
    DatabaseMappingDescriptor databaseMapping) throws
    DatabaseException;
```

Example: Drop Liquor Table Using Static API Of Database.

```
  DatabaseMapping databaseMapping = new Liquor().
      getDatabaseMapping();


    Database.dropTable(databaseMapping);
```
■

2. Creating Index Programmatically

Database class provides few API's two create index programmatically

(a)

```
public static final void createIndex(final
    DatabaseMappingDescriptor databaseMapping, final Index
    index) throws DatabaseException;
```

Example:

```
DatabaseMapping.Index indexOnLiquor = databaseMapping.new
    Index();
indexOnLiquor.setName("LIQUOR_INDEX_BASED_ON_LINK");
indexOnLiquor.setUnique(true);

//Add Columns on which we need index.
indexOnLiquor.addColumn("LINK");

DatabaseMapping databaseMapping = new Liquor().
    getDatabaseMapping();

try {
  Database.createIndex(databaseMapping, indexOnLiquor);
} catch(DatabaseException databaseException) {
  //Log It.
}
```

(b)

```
public static final void createIndex(final
    DatabaseMappingDescriptor databaseMapping, final String
    indexName, final Iterator<String> columnNames, final
    boolean isUnique) throws DatabaseException;
```

Example:

```
String indexName = "LIQUOR_INDEX_BASED_ON_LINK";
boolean isUnique = true;

Collection<String> columnNames = new ArrayList<String>();
columnNames.add("LINK");

try {
  new Liquor().createIndex(indexName, columnNames.iterator
      (), isUnique);
} catch(DatabaseException databaseException) {
    //Log It.
  }
```

(c)
```
public void createIndex(final Index index) throws
    DatabaseException;
```

Example:

```
  DatabaseMapping.Index indexOnLiquor = databaseMapping.new
      Index();
  indexOnLiquor.setName("LIQUOR_INDEX_BASED_ON_LINK");
  indexOnLiquor.setUnique(true);

  //Add Columns on which we need index.
  indexOnLiquor.addColumn("LINK");

  try {
    new Liquor().createIndex(indexOnLiquor);
  } catch(DatabaseException databaseException) {
    //Log It.
  }
```

(d)
```
public void createIndex(final String indexName, final
    Iterator<String> columnNames, final boolean isUnique)
    throws DatabaseException;
```

Example:

```
  String indexName = "LIQUOR_INDEX_BASED_ON_LINK";
  boolean isUnique = true;

  Collection<String> columnNames = new ArrayList<String>();
  columnNames.add("LINK");

  try {
    new Liquor().createIndex(indexName, columnNames.iterator
        (), isUnique);
  } catch(DatabaseException databaseException) {
    //Log It.
  }
```

5.2.6 Drop Index

Database class provides following API's to drop index from table.

1.
```
public void dropTable() throws DatabaseException;
```

Example:
```
  try {
    new Liquor().dropTable();
  } catch(DatabaseException databaseException) {
    //Log It.
  }
```

2.
```
public static final void dropTable(final
    DatabaseMappingDescriptor databaseMapping) throws
    DatabaseException;
```

Example:
```
  DatabaseMapping databaseMapping = new Liquor().
      getDatabaseMapping();

  try {
    Database.dropTable(databaseMapping);
  } catch(DatabaseException databaseException) {
    //Log It.
  }
```

5.2.7 Select

Database class provides following API's to fetch tuples from table.

1.
```
  public IFetch select() throws DatabaseException;
```

Fetch tuples from table.

IFetch: Exposes API's to provide information based on which tuples will be fetched from table.

```
public interface IFetch {

  /**
   * Used to specify DISTINCT condition.
   * @return ICount Interface.
   */
  public IFetch distinct();

  /**
   * Column name of which condition will be specified.
   * @param column Name of column.
   * @return IFetchClause Interface.
   */
  public IFetchClause where(String column);

  /**
   * Used to provide manually created Where clause, instead of
      using API's.
   * @param whereClause Manually created where clause.
   * @return IFetch Interface.
   */
  public IFetch whereClause(String whereClause);

  /**
   * Used to specify AND condition between where clause.
   * @param column Name of column on which condition need to be
      specified.
   * @return IFetchClause Interface.
   */
  public IFetchClause and(String column);

  /**
   * Used to specify OR condition between where clause.
   * @param column Name of column on which condition need to be
      specified.
   * @return IFetchClause Interface.
   */
  public IFetchClause or(String column);

  /**
   * Used to specify ORDER BY keyword to sort the result-set.
   * @param columns Name of columns which need to be sorted.
   * @return IFetch Interface.
   */
  public IFetch orderBy(String... columns);
```

```
/**
 * Used to specify ORDER BY ASC keyword to sort the result-set
     in ascending order.
 * @param columns Name of columns which need to be sorted.
 * @return IFetch Interface.
 */
public IFetch ascendingOrderBy(String...columns);

/**
 * Used to specify ORDER BY DESC keyword to sort the result-
    set in descending order.
 * @param columns Name of columns which need to be sorted.
 * @return IFetch Interface.
 */
public IFetch descendingOrderBy(String...columns);

/**
 * Used to specify the range of data need to fetch from table.
 * @param limit LIMIT of data.
 * @return IFetch Interface.
 */
public IFetch limit(int limit);

/**
 * Used to specify GROUP BY statement in conjunction with the
    aggregate functions to group the result-set by one or more
     columns.
 * @param columns Name of columns.
 * @return IFetch Interface.
 */
public IFetch groupBy(String...columns);

/**
 * Used to specify HAVING clause to SQL because the WHERE
    keyword could not be used with aggregate functions.
 * @param column Name of column on which condition need to be
    applied.
 * @return IFetchClause Interface.
 */
public IFetchClause having(String column);

/**
 * Used to provide manually created Where clause, instead of
    using API's.
 * @param havingClause Where clause.
 * @return IFetch Interface.
 */
public IFetch havingClause(String havingClause);
```

```
  /**
   * Used to provide name of columns only for which data will be
       fetched.
   * @param column Name of columns.
   * @return IFetch Interface.
   */
  public IFetch columns(String...columns);

  /**
   * Used to get tuples, this method should be called in last to
       get tuples from table.
   * @return Return array of model objects.
   * @throws DatabaseException Throws exception if any error
      occur while getting tuples from table.
   */
  public Object[] fetch() throws DatabaseException;

}
```

IFetchClause: Exposes API's to provide condition to where clause based on which tuples will be fetched from table.

```
public interface IFetchClause {

  /**
   * Used to specify EQUAL TO (=) condition.
   * @param value Value for which EQUAL TO (=) condition will be
       applied.
   * @return IFetch Interface.
   */
  public IFetch equalTo(String value);

  /**
   * Used to specify NOT EQUAL TO (!=) condition.
   * @param value Value for which NOT EQUAL TO (=) condition
      will be applied.
   * @return IFetch Interface.
   */
  public IFetch notEqualTo(String value);

  /**
   * Used to specify GREATER THAN (>) condition.
   * @param value Value for while GREATER THAN (>) condition
      will be specified.
   * @return IFetch Interface.
   */
  public IFetch greaterThan(String value);
```

```
    /**
     * Used to specify GREATER THAN EQUAL (>=) condition.
     * @param value Value for which GREATER THAN EQUAL (>=)
        condition will be specified.
     * @return IFetch Interface.
     */
    public IFetch greaterThanEqual(String value);

    /**
     * Used to specify LESS THAN (<) condition.
     * @param value Value for which LESS THAN (<) condition will
        be specified.
     * @return IFetch Interface.
     */
    public IFetch lessThan(String value);

    /**
     * Used to specify LESS THAN EQUAL (<=) condition.
     * @param value Value for which LESS THAN EQUAL (<=) condition
         will be specified.
     * @return IFetch Interface.
     */
    public IFetch lessThanEqual(String value);

    /**
     * Used to specify BETWEEN condition.
     * @param start Start Range.
     * @param end End Range.
     * @return IFetch Interface.
     */
    public IFetch between(String start, String end);

    /**
     * Used to specify LIKE condition.
     * @param like LIKE condition.
     * @return IFetch Interface.
     */
    public IFetch like(String like);

    /**
     * Used to specify IN condition.
     * @param values Values for IN condition.
     * @return IFetch Interface.
     */
    public IFetch in(String... values);

}
```

Example: Select Liquor Where Type Equal To RUM

```
LiquorBrand[] liquorBrands = new LiquorBrand().select()
        .where(LiquorBrand.LIQUOR_TYPE).equalTo(liquorType)
        .fetch();
```

2.
```
  public Object[] select(final String query) throws
     DatabaseException;
```

Returns all tuples based on manual query from mapped table for invoked class object.

Example: Select Liquor Object.

```
  String query = "SELECT * FROM LIQUOR";

  Liquor[] liquors = null;
  try {
    liquors = new Liquor().select(query);
  } catch(DatabaseException de) {
    //Log it.
  }
```

5.2.8 Save

```
  public final void save() throws DatabaseException;
```

Example: Saving Liquor Object.

```
  Liquor beer = new Liquor();
  beer.setLiquorType(Liquor.LIQUOR_TYPE_BEER);
  beer.setDescription(applicationContext.getString(R.string.
     beer_description));
  beer.setHistory(applicationContext.getString(R.string.beer_history)
     );
  beer.setLink(applicationContext.getString(R.string.beer_link));
```

```
beer.setAlcholContent(applicationContext.getString(R.string.
    beer_alchol_content));

try {
  beer.save();
} catch(DatabaseException de) {
  //Log it.
}
```

5.2.9 Update

```
public final void update() throws DatabaseException;
```

Example: Updaing Liquor Object.

```
Liquor beer = new Liquor();
beer.setLiquorType(Liquor.LIQUOR_TYPE_BEER);
beer.setDescription(applicationContext.getString(R.string.
    beer_description));
beer.setHistory(applicationContext.getString(R.string.beer_history)
    );
beer.setLink(applicationContext.getString(R.string.beer_link));
beer.setAlcholContent(applicationContext.getString(R.string.
    beer_alchol_content));

try {
  beer.update();
} catch(DatabaseException de) {
  //Log it.
}
```

5.2.10 Save Or Update

```
public final void saveOrUpdate() throws DatabaseException;
```

Example: Save Or Update Liquor Object.

```
  Liquor beer = new Liquor();
  beer.setLiquorType(Liquor.LIQUOR_TYPE_BEER);
  beer.setDescription(applicationContext.getString(R.string.
     beer_description));
  beer.setHistory(applicationContext.getString(R.string.beer_history)
     );
  beer.setLink(applicationContext.getString(R.string.beer_link));
  beer.setAlcholContent(applicationContext.getString(R.string.
     beer_alchol_content));

  try {
    beer.saveOrUpdate();
  } catch(DatabaseException de) {
    //Log it.
  }
```

> **Note**
> If tuple is not present in table then it will insert it into table else it will update the tuple.

5.2.11 Delete

Database class provides following API's to delete tuples from table.

API: Delete API.

```
  public IDelete delete() throws DatabaseException;
```

IDelete: Exposes API's to delete tuples from table.

```
public interface IDelete {

  public String INTERFACE_NAME = IDelete.class.getName();


  public IDeleteClause where(String column);

  public IDelete whereClause(String whereClause);

  public IDeleteClause and(String column);

  public IDeleteClause or(String column);

```

```
    public Object execute() throws DatabaseException;

}
```

IDeleteClause: Exposes API's to provide condition on where clause to delete tuple from table.

```
public interface IDeleteClause {

    public String INTERFACE_NAME = IDeleteClause.class.getName();


    public IDelete equalTo(String value);

    public IDelete notEqualTo(String value);

    public IDelete greaterThan(String value);

    public IDelete greaterThanEqual(String value);

    public IDelete lessThan(String value);

    public IDelete lessThanEqual(String value);

    public IDelete between(String start, String end);

    public IDelete like(String like);

    public IDelete in(String... values);

}
```

Example:

```
    Liquor liquor = new Liquor();
    liquor.delete().execute();
```

5.3 Database Siminov API's

5.3.1 Get Database Descriptor

```
public final DatabaseDescriptor getDatabaseDescriptor() throws
    DatabaseException;
```

Example: Get Database Descriptor Object Which Contain Liquor Table.

```
try {
  DatabaseDescriptor databaseDescriptor = new Liquor().
      getDatabaseDescriptor();
} catch(DatabaseException databaseException) {
  //Log It.
}
```

5.3.2 Get Database Mapping Descriptor

```
public final DatabaseMappingDescriptor getDatabaseMappingDescriptor
    () throws DatabaseException;
```

Example: Get Database Mapping Descriptor Object Related To Liquor Table.

```
DatabaseMapping databaseMapping = null;
try {
  databaseMapping = new Liquor().getDatabaseMapping();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.3 Get Table Name

```
public final String getTableName() throws DatabaseException;
```

Example: Get Liquor Object Table Name.

```
String tableName = null;
try {
  tableName = new Liquor().getTableName();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.4 Get Column Names

```
public final Iterator<String> getColumnNames() throws
    DatabaseException;
```

Example: Get All Column Names Of Liquor Table.

```
String[] columnNames = null;
try {
  columnNames = new Liquor().getColumnNames();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.5 Get Column Values

```
public final Map<String, Object> getColumnValues() throws
    DatabaseException;
```

Example: Get All Column Values Of Liquor Object.

```
Map<String, Object> values = null;
try {
  values = new Liquor().getColumnValues();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.6 Get Column Types

```
public final Map<String, String> getColumnTypes() throws
    DatabaseException;
```

Example: Get All Column Types Of Liquor Table.

```
Map<String, String> columnTypes = null;
try {
  columnTypes = new Liquor().getColumnTypes();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.7 Get Primary Keys

```
public final Iterator<String> getPrimaryKeys() throws
    DatabaseException;
```

Example: Get All Primary Keys Of Liquor Table.

```
Iterator<String> primaryKeys = null;
try {
  primaryKeys = new Liquor().getPrimeryKeys();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.8 Get Mandatory Fields

```
public final Iterator<String> getMandatoryFields() throws
    DatabaseException;
```

Example: Get All Column Names Which Are Marked As Manadatory Fields.

```
Iterator<String> mandatoryFields = null;
try {
  mandatoryFields = new Liquor().getMandatoryFields();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.9 Get Unique Fields

```
public final Iterator<String> getUniqueFields() throws
    DatabaseException;
```

Example: Get All Column Names Which Are Marked As Unique.

```
Iterator<String> uniqueFields = null;
try {
  uniqueFields = new Liquor().getUniqueFields();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.10 Get Foreign Keys

```
public final Iterator<String> getForeignKeys() throws
    DatabaseException;
```

Example: Get All Column Names Which Are Marked As Foreign Keys.

```
Iterator<String> foreignKeys = null;
try {
  foreignKeys = new Liquor().getForeignKeys();
} catch(DatabaseException de) {
  //Log it.
}
```

5.4 Database Aggregation API's

5.4.1 Count

Returns the count of rows based on information provided.

API: Count API.

```
public ICount count() throws DatabaseException;
```

ICount: Exposes API's to get count of the number of times that X is not NULL in a group. The count(*) function (with no arguments) returns the total number of rows in the group.

```
public interface ICount {

  /**
   * Used to specify DISTINCT condition.
   * @return ICount Interface.
   */
  public ICount distinct();

  /**
   * Column name of which condition will be specified.
   * @param column Name of column.
   * @return ICountClause Interface.
   */
  public ICountClause where(String column);

  /**
   * Used to provide manually created Where clause, instead of using
      API's.
   * @param whereClause Manually created where clause.
   * @return ICount Interface.
   */
  public ICount whereClause(String whereClause);

  /**
   * Used to specify AND condition between where clause.
   * @param column Name of column on which condition need to be
      specified.
   * @return ICountClause Interface.
   */
  public ICountClause and(String column);

  /**
```

```
 * Used to specify OR condition between where clause.
 * @param column Name of column on which condition need to be
     specified.
 * @return ICountClause Interface.
 */
public ICountClause or(String column);

/**
 * Used to specify GROUP BY statement in conjunction with the
     aggregate functions to group the result-set by one or more
     columns.
 * @param columns Name of columns.
 * @return ICount Interface.
 */
public ICount groupBy(String...columns);

/**
 * Used to specify HAVING clause to SQL because the WHERE keyword
     could not be used with aggregate functions.
 * @param column Name of column on which condition need to be
     applied.
 * @return ICountClause Interface.
 */
public ICountClause having(String column);

/**
 * Used to provide manually created Where clause, instead of using
     API's.
 * @param havingClause Where clause.
 * @return ICount Interface.
 */
public ICount havingClause(String havingClause);

/**
 * Used to provide name of column for which count will be
     calculated.
 * @param column Name of column.
 * @return ICount Interface.
 */
public ICount column(String column);

/**
 * Used to get count, this method should be called in last to
     calculate count.
 * @return Return count.
 * @throws DatabaseException Throws exception if any error occur
     while calculating count.
 */
public Object execute() throws DatabaseException;
```

```
}
```

ICountClause: Exposes API's to provide condition on where clause to calculate count.

```
public interface ICountClause {

  /**
   * Used to specify EQUAL TO (=) condition.
   * @param value Value for which EQUAL TO (=) condition will be
       applied.
   * @return ICount Interface.
   */
  public ICount equalTo(String value);

  /**
   * Used to specify NOT EQUAL TO (!=) condition.
   * @param value Value for which NOT EQUAL TO (=) condition will be
       applied.
   * @return ICount Interface.
   */
  public ICount notEqualTo(String value);

  /**
   * Used to specify GREATER THAN (>) condition.
   * @param value Value for while GREATER THAN (>) condition will be
       specified.
   * @return ICount Interface.
   */
  public ICount greaterThan(String value);

  /**
   * Used to specify GREATER THAN EQUAL (>=) condition.
   * @param value Value for which GREATER THAN EQUAL (>=) condition
       will be specified.
   * @return ICount Interface.
   */
  public ICount greaterThanEqual(String value);

  /**
   * Used to specify LESS THAN (<) condition.
   * @param value Value for which LESS THAN (<) condition will be
       specified.
   * @return ICount Interface.
   */
  public ICount lessThan(String value);
/**
   * Used to specify LESS THAN EQUAL (<=) condition.
   * @param value Value for which LESS THAN EQUAL (<=) condition will
       be specified.
   * @return ICount Interface.
   */
public ICount lessThanEqual(String value);

/**
   * Used to specify BETWEEN condition.
   * @param start Start Range.
   * @param end End Range.
   * @return ICount Interface.
   */
public ICount between(String start, String end);

/**
   * Used to specify LIKE condition.
   * @param like LIKE condition.
   * @return ICount Interface.
   */
public ICount like(String like);

/**
   * Used to specify IN condition.
   * @param values Values for IN condition.
   * @return ICount Interface.
   */
public ICount in(String...values);

}
```

Example: Get count of liquors type equal to RUM.

```
int count = 0;

try {
  count = new Liquor().count().
        .where(Liquor.LIQUOR_TYPE).equalTo("RUM")
        .execute();

} catch(DatabaseException de) {
  //Log it.
}
```

5.4.2 Average

Returns the average based on column name provided.

API: Average API.

```
  public IAverage avg() throws DatabaseException;
```

IAverage: Exposes API's to get average value of all non-NULL X within a group. String and BLOB values that do not look like numbers are interpreted as 0. The result of avg() is always a oating point value as long as at there is at least one non-NULL input even if all inputs are integers. The result of avg() is NULL if and only if there are no non-NULL inputs.

```
public interface IAverage {

  /**
   * Column name of which condition will be specified.
   * @param column Name of column.
   * @return IAerageClause Interface.
   */
  public IAverageClause where(String column);

  /**
   * Used to provide manually created Where clause, instead of using
      API's.
   * @param whereClause Manually created where clause.
   * @return IAverage Interface.
   */
  public IAverage whereClause(String whereClause);

  /**
   * Used to specify AND condition between where clause.
   * @param column Name of column on which condition need to be
      specified.
   * @return IAerageClause Interface.
   */
  public IAverageClause and(String column);

  /**
   * Used to specify OR condition between where clause.
   * @param column Name of column on which condition need to be
      specified.
   * @return IAverageClause Interface.
   */
  public IAverageClause or(String column);
```

```
    /**
     * Used to specify GROUP BY statement in conjunction with the
        aggregate functions to group the result-set by one or more
        columns.
     * @param columns Name of columns.
     * @return IAverage Interface.
     */
    public IAverage groupBy(String...columns);

    /**
     * Used to specify HAVING clause to SQL because the WHERE keyword
        could not be used with aggregate functions.
     * @param column Name of column on which condition need to be
        applied.
     * @return IAverageClause Interface.
     */
    public IAverageClause having(String column);

    /**
     * Used to provide manually created Where clause, instead of using
        API's.
     * @param havingClause Where clause.
     * @return IAverage Interface.
     */
    public IAverage havingClause(String havingClause);

    /**
     * Used to provide name of column for which average will be
        calculated.
     * @param column Name of column.
     * @return IAverage Interface.
     */
    public IAverage column(String column);

    /**
     * Used to get average, this method should be called in last to
        calculate average.
     * @return Return average.
     * @throws DatabaseException Throws exception if any error occur
        while calculating average.
     */
    public Object execute() throws DatabaseException;

}
```

IAverageClause: Exposes API's to provide condition on where clause to calculate average.

```java
public interface IAverageClause {

  /**
   * Used to specify EQUAL TO (=) condition.
   * @param value Value for which EQUAL TO (=) condition will be
       applied.
   * @return IAverage Interface.
   */
  public IAverage equalTo(String value);

  /**
   * Used to specify NOT EQUAL TO (!=) condition.
   * @param value Value for which NOT EQUAL TO (=) condition will be
       applied.
   * @return IAverage Interface.
   */
  public IAverage notEqualTo(String value);

  /**
   * Used to specify GREATER THAN (>) condition.
   * @param value Value for while GREATER THAN (>) condition will be
       specified.
   * @return IAverage Interface.
   */
  public IAverage greaterThan(String value);

  /**
   * Used to specify GREATER THAN EQUAL (>=) condition.
   * @param value Value for which GREATER THAN EQUAL (>=) condition
       will be specified.
   * @return IAverage Interface.
   */
  public IAverage greaterThanEqual(String value);

  /**
   * Used to specify LESS THAN (<) condition.
   * @param value Value for which LESS THAN (<) condition will be
       specified.
   * @return IAverage Interface.
   */
  public IAverage lessThan(String value);

  /**
   * Used to specify LESS THAN EQUAL (<=) condition.
   * @param value Value for which LESS THAN EQUAL (<=) condition will
       be specified.
   * @return IAverage Interface.
   */
  public IAverage lessThanEqual(String value);

  /**
   * Used to specify BETWEEN condition.
   * @param start Start Range.
   * @param end End Range.
   * @return IAverage Interface.
   */
  public IAverage between(String start, String end);

  /**
   * Used to specify LIKE condition.
   * @param like LIKE condition.
   * @return IAverage Interface.
   */
  public IAverage like(String like);

  /**
   * Used to specify IN condition.
   * @param values Values for IN condition.
   * @return IAverage Interface.
   */
  public IAverage in(String... values);

}
```

Example:

```
  int average = 0;

  try {
    average =
```

API: Sum API.

```
  public ISum sum() throws DatabaseException;
```

ISum: Exposes API's to return sum of all non-NULL values in the group. If there are no non-NULL input rows then sum() returns NULL but total() returns 0.0. NULL is not normally a helpful result for the sum of no rows but the SQL standard requires it and most other SQL database engines implement sum() that way so SQLite does it in the same way in order to be compatible. The result of sum() is an integer value if all non-NULL inputs are integers.

```
public interface ISum {


  /**
   * Column name of which condition will be specified.
   * @param column Name of column.
   * @return ISumClause Interface.
   */
  public ISumClause where(String column);

  /**
   * Used to provide manually created Where clause, instead of using
       API's.
   * @param whereClause Manually created where clause.
   * @return ISum Interface.
   */
  public ISum whereClause(String whereClause);

  /**
   * Used to specify AND condition between where clause.
   * @param column Name of column on which condition need to be
       specified.
   * @return ISumClause Interface.
   */
  public ISumClause and(String column);

  /**
   * Used to specify OR condition between where clause.
   * @param column Name of column on which condition need to be
       specified.
   * @return ISumClause Interface.
   */
  public ISumClause or(String column);

  /**
   * Used to specify GROUP BY statement in conjunction with the
```

```
       aggregate functions to group the result-set by one or more
       columns.
   * @param columns Name of columns.
   * @return ISum Interface.
   */
  public ISum groupBy(String...columns);

  /**
   * Used to specify HAVING clause to SQL because the WHERE keyword
       could not be used with aggregate functions.
   * @param column Name of column on which condition need to be
       applied.
   * @return ISumClause Interface.
   */
  public ISumClause having(String column);

  /**
   * Used to provide manually created Where clause, instead of using
       API's.
   * @param havingClause Where clause.
   * @return ISum Interface.
   */
  public ISum havingClause(String havingClause);

  /**
   * Used to provide name of column for which sum will be calculated.
   * @param column Name of column.
   * @return ISum Interface.
   */
  public ISum column(String column);

  /**
   * Used to get sum, this method should be called in last to
       calculate sum.
   * @return Return sum.
   * @throws DatabaseException Throws exception if any error occur
       while calculating sum.
   */
  public Object execute() throws DatabaseException;

}
```

ISumClause: Exposes API's to provide condition on where clause to calculate sum.

```
public interface ISumClause {

  /**
```

```
 * Used to specify EQUAL TO (=) condition.
 * @param value Value for which EQUAL TO (=) condition will be
     applied.
 * @return ISum Interface.
 */
public ISum equalTo(String value);

/**
 * Used to specify NOT EQUAL TO (!=) condition.
 * @param value Value for which NOT EQUAL TO (=) condition will be
     applied.
 * @return ISum Interface.
 */
public ISum notEqualTo(String value);

/**
 * Used to specify GREATER THAN (>) condition.
 * @param value Value for while GREATER THAN (>) condition will be
     specified.
 * @return ISum Interface.
 */
public ISum greaterThan(String value);

/**
 * Used to specify GREATER THAN EQUAL (>=) condition.
 * @param value Value for which @paa 11.955 l SQ0 g 0 3t0 G0 0.6 0 rg 0 0.6 0 RG0 g 0 G0 0.6 0 rg 0 0.6 0 R
```

```
    * @param start Start Range.
    * @param end End Range.
    * @return ISum Interface.
    */
   public ISum between(String start, String end);

   /**
    * Used to specify LIKE condition.
    * @param like LIKE condition.
    * @return ISum Interface.
    */
   public ISum like(String like);

   /**
    * Used to specify IN condition.
    * @param values Values for IN condition.
    * @return ISum Interface.
    */
   public ISum in(String... values);

}
```

Example:

```
   int sum = 0;

   try {
     sum = new Liquor().sum()
            .column(Liquor.COLUMN_NAME_WHICH_CONTAIN_NUMBRIC_VALUE)
            .where(Liquor.LIQUOR_TYPE).equalTo("RUM")
            .execute();

   } catch(DatabaseException de) {
     //Log it.
   }
```

5.4.4 Total

Returns the total based on column name provided.

API: Total API.

```
   public ITotal total() throws DatabaseException;
```

ITotal: Exposes API's to return total of all non-NULL values in the group. The non-standard total() function is provided as a convenient way to work around this design problem in the SQL language. The result of total() is always a oating point value.

```
public interface ITotal {

  /**
   * Column name of which condition will be specified.
   * @param column Name of column.
   * @return ITotalClause Interface.
   */
  public ITotalClause where(String column);

  /**
   * Used to provide manually created Where clause, instead of using
      API's.
   * @param whereClause Manually created where clause.
   * @return ITotal Interface.
   */
  public ITotal whereClause(String whereClause);

  /**
   * Used to specify AND condition between where clause.
   * @param column Name of column on which condition need to be
       specified.
   * @return ITotalClause Interface.
   */
  public ITotalClause and(String column);

  /**
   * Used to specify OR condition between where clause.
   * @param column Name of column on which condition need to be
       specified.
   * @return ITotalClause Interface.
   */
  public ITotalClause or(String column);

  /**
   * Used to specify GROUP BY statement in conjunction with the
       aggregate functions to group the result-set by one or more
      columns.
   * @param columns Name of columns.
   * @return ITotal Interface.
   */
  public ITotal groupBy(String...columns);
  /**
   * Used to specify HAVING clause to SQL because the WHERE keyword
      could not be used with aggregate functions.
   * @param column Name of column on which condition need to be
      applied.
   * @return ITotalClause Interface.
   */
  public ITotalClause having(String column);

  /**
   * Used to provide manually created Where clause, instead of using
      API's.
   * @param havingClause Where clause.
   * @return ITotal Interface.
   */
  public ITotal havingClause(String havingClause);

  /**
   * Used to provide name of column for which total will be
      calculated.
   * @param column Name of column.
   * @return ITotal Interface.
   */
  public ITotal column(String column);

  /**
   * Used to get total, this method should be called in last to
      calculate total.
   * @return Return total.
   * @throws DatabaseException Throws exception if any error occur
      while calculating total.
   */
  public Object execute() throws DatabaseException;

}
```

ITotalClause: Exposes API's to provide condition on where clause to calculate total.

```
public interface ITotalClause {

  /**
   * Used to specify EQUAL TO (=) condition.
   * @param value Value for which EQUAL TO (=) condition will be
      applied.
   * @return ITotal Interface.
   */
```

```
public ITotal equalTo(String value);

/**
 * Used to specify NOT EQUAL TO (!=) condition.
 * @param value Value for which NOT EQUAL TO (=) condition will be
    applied.
 * @return ITotal Interface.
 */
public ITotal notEqualTo(String value);

/**
 * Used to specify GREATER THAN (>) condition.
 * @param value Value for while GREATER THAN (>) condition will be
    specified.
 * @return ITotal Interface.
 */
public ITotal greaterThan(String value);

/**
 * Used to specify GREATER THAN EQUAL (>=) condition.
 * @param value Value for which GREATER THAN EQUAL (>=) condition
    will be specified.
 * @return ITotal Interface.
 */
public ITotal greaterThanEqual(String value);

/**
 * Used to specify LESS THAN (<) condition.
 * @param value Value for which LESS THAN (<) condition will be
    specified.
 * @return ITotal Interface.
 */
public ITotal lessThan(String value);

/**
 * Used to specify LESS THAN EQUAL (<=) condition.
 * @param value Value for which LESS THAN EQUAL (<=) condition will
     be specified.
 * @return ITotal Interface.
 */
public ITotal lessThanEqual(String value);

/**
 * Used to specify BETWEEN condition.
 * @param start Start Range.
 * @param end End Range.
 * @return ITotal Interface.
 */
public ITotal between(String start, String end);
```

```
  /**
   * Used to specify LIKE condition.
   * @param like LIKE condition.
   * @return ITotal Interface.
   */
  public ITotal like(String like);

  /**
   * Used to specify IN condition.
   * @param values Values for IN condition.
   * @return ITotal Interface.
   */
  public ITotal in(String... values);

}
```

Example:

```
  int total = 0;

  try {
    total = new Liquor().total()
          .column(Liquor.COLUMN_NAME_WHICH_CONTAIN_NUMBRIC_VALUE)
          .where(Liquor.LIQUOR_TYPE).equalTo("RUM")
          .execute();

  } catch(DatabaseException de) {
    //Log it.
  }
```

5.4.5 Minimum

Returns the minimum based on column name provided.

API: Minimum API.

```
  public IMin min() throws DatabaseException;
```

IMin: Exposes API's to returns the minimum non-NULL value of all values in the group. The minimum value is the rst non-NULL value that would appear in an ORDER BY of the column. Aggregate min() returns NULL if and only if there are no non-NULL values in the group.

```
public interface IMin {

  /**
   * Column name of which condition will be specified.
   * @param column Name of column.
   * @return IMinClause Interface.
   */
  public IMinClause where(String column);

  /**
   * Used to provide manually created Where clause, instead of using
       API's.
   * @param whereClause Manually created where clause.
   * @return IMin Interface.
   */
  public IMin whereClause(String whereClause);

  /**
   * Used to specify AND condition between where clause.
   * @param column Name of column on which condition need to be
       specified.
   * @return IMinClause Interface.
   */
  public IMinClause and(String column);

  /**
   * Used to specify OR condition between where clause.
   * @param column Name of column on which condition need to be
       specified.
   * @return IMinClause Interface.
   */
  public IMinClause or(String column);

  /**
   * Used to specify GROUP BY statement in conjunction with the
       aggregate functions to group the result-set by one or more
       columns.
   * @param columns Name of columns.
   * @return IMin Interface.
   */
  public IMin groupBy(String...columns);

  /**
   * Used to specify HAVING clause to SQL because the WHERE keyword
       could not be used with aggregate functions.
   * @param column Name of column on which condition need to be
```

```
        applied.
    * @return IMinClause Interface.
    */
   public IMinClause having(String column);

   /**
    * Used to provide manually created Where clause, instead of using
        API's.
    * @param havingClause Where clause.
    * @return IMin Interface.
    */
   public IMin havingClause(String havingClause);

   /**
    * Used to provide name of column for which max will be calculated.
    * @param column Name of column.
    * @return IMin Interface.
    */
   public IMin column(String column);

   /**
    * Used to get minimum, this method should be called in last to
        calculate minimum.
    * @return Return minimum.
    * @throws DatabaseException Throws exception if any error occur
        while calculating minimum.
    */
   public Object execute() throws DatabaseException;

}
```

IMinClause: Exposes API's to provide condition on where clause to calculate minimum.

```
public interface IMinClause {

   /**
    * Used to specify EQUAL TO (=) condition.
    * @param value Value for which EQUAL TO (=) condition will be
        applied.
    * @return IMax Interface.
    */
   public IMin equalTo(String value);

   /**
    * Used to specify NOT EQUAL TO (!=) condition.
    * @param value Value for which NOT EQUAL TO (=) condition will be
```

```
        applied.
     * @return IMax Interface.
     */
    public IMin notEqualTo(String value);

    /**
     * Used to specify GREATER THAN (>) condition.
     * @param value Value for while GREATER THAN (>) condition will be
        specified.
     * @return IMax Interface.
     */
    public IMin greaterThan(String value);

    /**
     * Used to specify GREATER THAN EQUAL (>=) condition.
     * @param value Value for which GREATER THAN EQUAL (>=) condition
        will be specified.
     * @return IMax Interface.
     */
    public IMin greaterThanEqual(String value);

    /**
     * Used to specify LESS THAN (<) condition.
     * @param value Value for which LESS THAN (<) condition will be
        specified.
     * @return IMax Interface.
     */
    public IMin lessThan(String value);

    /**
     * Used to specify LESS THAN EQUAL (<=) condition.
     * @param value Value for which LESS THAN EQUAL (<=) condition will
         be specified.
     * @return IMax Interface.
     */
    public IMin lessThanEqual(String value);

    /**
     * Used to specify BETWEEN condition.
     * @param start Start Range.
     * @param end End Range.
     * @return IMax Interface.
     */
    public IMin between(String start, String end);

    /**
     * Used to specify LIKE condition.
     * @param like LIKE condition.
     * @return IMax Interface.
```

```
   */
  public IMin like(String like);

  /**
   * Used to specify IN condition.
   * @param values Values for IN condition.
   * @return IMax Interface.
   */
  public IMin in(String... values);

}
```

Example:

```
  int minimum = 0;

  try {
    minimum = new Liquor().min()
          .column(Liquor.COLUMN_NAME_WHICH_CONTAIN_NUMBRIC_VALUE)
          .where(Liquor.LIQUOR_TYPE).equalTo("RUM")
          .execute();

  } catch(DatabaseException de) {
    //Log it.
  }
```

5.4.6 Maximum

Returns the minimum based on column name provided.

API: Maximum API.

```
  public IMax max() throws DatabaseException;
```

IMax: Exposes API's to returns the maximum value of all values in the group. The maximum value is the value that would be returned last in an ORDER BY on the same column. Aggregate max() returns NULL if and only if there are no non-NULL values in the group.

```
public interface IMax {

  /**
```

```
 * Column name of which condition will be specified.
 * @param column Name of column.
 * @return IMaxClause Interface.
 */
public IMaxClause where(String column);

/**
 * Used to provide manually created Where clause, instead of using
    API's.
 * @param whereClause Manually created where clause.
 * @return IMax Interface.
 */
public IMax whereClause(String whereClause);

/**
 * Used to specify AND condition between where clause.
 * @param column Name of column on which condition need to be
    specified.
 * @return IMaxClause Interface.
 */
public IMaxClause and(String column);

/**
 * Used to specify OR condition between where clause.
 * @param column Name of column on which condition need to be
    specified.
 * @return IMaxClause Interface.
 */
public IMaxClause or(String column);

/**
 * Used to specify GROUP BY statement in conjunction with the
    aggregate functions to group the result-set by one or more
    columns.
 * @param columns Name of columns.
 * @return IMax Interface.
 */
public IMax groupBy(String...columns);

/**
 * Used to specify HAVING clause to SQL because the WHERE keyword
    could not be used with aggregate functions.
 * @param column Name of column on which condition need to be
    applied.
 * @return IMaxClause Interface.
 */
public IMaxClause having(String column);

/**
```

```
   * Used to provide manually created Where clause, instead of using
       API's.
   * @param havingClause Where clause.
   * @return IMax Interface.
   */
  public IMax havingClause(String havingClause);

  /**
   * Used to provide name of column for which maximum will be
       calculated.
   * @param column Name of column.
   * @return IMax Interface.
   */
  public IMax column(String column);

  /**
   * Used to get maximum, this method should be called in last to
       calculate maximum.
   * @return Return maximum.
   * @throws DatabaseException Throws exception if any error occur
       while calculating maximum.
   */
  public Object execute() throws DatabaseException;

}
```

IMaxClause: Exposes API's to provide condition on where clause to calculate maximum.

```
public interface IMaxClause {

  /**
   * Used to specify EQUAL TO (=) condition.
   * @param value Value for which EQUAL TO (=) condition will be
       applied.
   * @return IMax Interface.
   */
  public IMax equalTo(String value);

  /**
   * Used to specify NOT EQUAL TO (!=) condition.
   * @param value Value for which NOT EQUAL TO (=) condition will be
       applied.
   * @return IMax Interface.
   */
  public IMax notEqualTo(String value);

```

```
/**
 * Used to specify GREATER THAN (>) condition.
 * @param value Value for while GREATER THAN (>) condition will be
    specified.
 * @return IMax Interface.
 */
public IMax greaterThan(String value);

/**
 * Used to specify GREATER THAN EQUAL (>=) condition.
 * @param value Value for which GREATER THAN EQUAL (>=) condition
    will be specified.
 * @return IMax Interface.
 */
public IMax greaterThanEqual(String value);

/**
 * Used to specify LESS THAN (<) condition.
 * @param value Value for which LESS THAN (<) condition will be
    specified.
 * @return IMax Interface.
 */
public IMax lessThan(String value);

/**
 * Used to specify LESS THAN EQUAL (<=) condition.
 * @param value Value for which LESS THAN EQUAL (<=) condition will
     be specified.
 * @return IMax Interface.
 */
public IMax lessThanEqual(String value);

/**
 * Used to specify BETWEEN condition.
 * @param start Start Range.
 * @param end End Range.
 * @return IMax Interface.
 */
public IMax between(String start, String end);

/**
 * Used to specify LIKE condition.
 * @param like LIKE condition.
 * @return IMax Interface.
 */
public IMax like(String like);

/**
 * Used to specify IN condition.
```

```
   * @param values Values for IN condition.
   * @return IMax Interface.
   */
  public IMax in(String... values);

}
```

Example:

```
  int maximum = 0;

  try {
    maximum = new Liquor().max()
           .column(Liquor.COLUMN_NAME_WHICH_CONTAIN_NUMBRIC_VALUE)
           .where(Liquor.LIQUOR_TYPE).equalTo("RUM")
           .execute();

  } catch(DatabaseException de) {
    //Log it.
  }
```

5.4.7 Group Concat

Returns the group concat based on column name provided.

API: Group Concat API.

```
  public IGroupConcat groupConcat() throws DatabaseException;
```

IGroupConcat: Exposes API's to get group concat that returns a string which is the concatenation of all non-NULL values of X. If parameter Y is present then it is used as the separator between instances of X. A comma (",") is used as the separator if Y is omitted. The order of the concatenated elements is arbitrary.

```
public interface IGroupConcat {

  /**
   * Used to specify separator if Y is omitted.
   * @param delimiter Delimiter.
   * @return IGroupConcat Interface.
   */
  public IGroupConcat delimiter(String delimiter);
```

```
/**
 * Column name of which condition will be specified.
 * @param column Name of column.
 * @return IGroupConcatClause Interface.
 */
public IGroupConcatClause where(String column);

/**
 * Used to provide manually created Where clause, instead of using
    API's.
 * @param whereClause Manually created where clause.
 * @return IGroupConcat Interface.
 */
public IGroupConcat whereClause(String whereClause);

/**
 * Used to specify AND condition between where clause.
 * @param column Name of column on which condition need to be
    specified.
 * @return IGroupConcatClause Interface.
 */
public IGroupConcatClause and(String column);

/**
 * Used to specify OR condition between where clause.
 * @param column Name of column on which condition need to be
    specified.
 * @return IGroupConcatClause Interface.
 */
public IGroupConcatClause or(String column);

/**
 * Used to specify GROUP BY statement in conjunction with the
    aggregate functions to group the result-set by one or more
    columns.
 * @param columns Name of columns.
 * @return IGroupConcat Interface.
 */
public IGroupConcat groupBy(String...columns);

/**
 * Used to specify HAVING clause to SQL because the WHERE keyword
    could not be used with aggregate functions.
 * @param column Name of column on which condition need to be
    applied.
 * @return IGroupConcatClause Interface.
 */
public IGroupConcatClause having(String column);
```

```
  /**
   * Used to provide manually created Where clause, instead of using
      API's.
   * @param havingClause Where clause.
   * @return IGroupConcat Interface.
   */
  public IGroupConcat havingClause(String havingClause);

  /**
   * Used to provide name of column for which average will be
      calculated.
   * @param column Name of column.
   * @return IGroupConcat Interface.
   */
  public IGroupConcat column(String column);

  /**
   * Used to get average, this method should be called in last to
      calculate group concat.
   * @return Return group concat.
   * @throws DatabaseException Throws exception if any error occur
      while calculating group concat.
   */
  public Object execute() throws DatabaseException;

}
```

IGroupConcatClause: Exposes API's to provide condition on where clause to calculate group concat.

```
public interface IGroupConcatClause {

  /**
   * Used to specify EQUAL TO (=) condition.
   * @param value Value for which EQUAL TO (=) condition will be
      applied.
   * @return IGroupConcat Interface.
   */
  public IGroupConcat equalTo(String value);

  /**
   * Used to specify NOT EQUAL TO (!=) condition.
   * @param value Value for which NOT EQUAL TO (=) condition will be
      applied.
   * @return IGroupConcat Interface.
   */
```

```java
public IGroupConcat notEqualTo(String value);

/**
 * Used to specify GREATER THAN (>) condition.
 * @param value Value for while GREATER THAN (>) condition will be
     specified.
 * @return IGroupConcat Interface.
 */
public IGroupConcat greaterThan(String value);

/**
 * Used to specify GREATER THAN EQUAL (>=) condition.
 * @param value Value for which GREATER THAN EQUAL (>=) condition
     will be specified.
 * @return IGroupConcat Interface.
 */
public IGroupConcat greaterThanEqual(String value);

/**
 * Used to specify LESS THAN (<) condition.
 * @param value Value for which LESS THAN (<) condition will be
     specified.
 * @return IGroupConcat Interface.
 */
public IGroupConcat lessThan(String value);

/**
 * Used to specify LESS THAN EQUAL (<=) condition.
 * @param value Value for which LESS THAN EQUAL (<=) condition will
      be specified.
 * @return IGroupConcat Interface.
 */
public IGroupConcat lessThanEqual(String value);

/**
 * Used to specify BETWEEN condition.
 * @param start Start Range.
 * @param end End Range.
 * @return IGroupConcat Interface.
 */
public IGroupConcat between(String start, String end);

/**
 * Used to specify LIKE condition.
 * @param like LIKE condition.
 * @return IGroupConcat Interface.
 */
public IGroupConcat like(String like);
```

```
  /**
   * Used to specify IN condition.
   * @param values Values for IN condition.
   * @return IGroupConcat Interface.
   */
  public IGroupConcat in(String... values);

}
```

Example:

```
  int groupConcat = 0;

  try {
    groupConcat = new Liquor().groupConcat()
            .column(Liquor.COLUMN_NAME_WHICH_CONTAIN_NUMBRIC_VALUE)
            .where(Liquor.LIQUOR_TYPE).equalTo("RUM")
            .execute();

  } catch(DatabaseException de) {
    //Log it.
  }
```

5.5 Database Transaction API's

5.5.1 Begin Transaction

```
  public static final void beginTransaction(final DatabaseDescriptor
      databaseDescriptor) throws DatabaseException;
```

Begins a transaction in EXCLUSIVE mode.

Transactions can be nested. When the outer transaction is ended all of the work done in that transaction and all of the nested transactions will be committed or rolled back. The changes will be rolled back if any transaction is ended without being marked as clean(by calling commitTransaction). Otherwise they will be committed.

Example: Saving Liquor Within Transaction.

```
Liquor beer = new Liquor();
beer.setLiquorType(Liquor.LIQUOR_TYPE_BEER);
beer.setDescription(applicationContext.getString(R.string.
    beer_description));
beer.setHistory(applicationContext.getString(R.string.beer_history)
    );
beer.setLink(applicationContext.getString(R.string.beer_link));
beer.setAlcholContent(applicationContext.getString(R.string.
    beer_alchol_content));

DatabaseDescriptor databaseDescriptor = beer.getDatabaseDescriptor
    ();

try {
  Database.beginTransaction(databaseDescriptor);

  beer.save();

  Database.commitTransaction(databaseDescriptor);
} catch(DatabaseException de) {
  //Log it.
} finally {
  Database.endTransaction(databaseDescriptor);
}
```

5.5.2 Commit Transaction

```
public static final void commitTransaction(final DatabaseDescriptor
     databaseDescriptor) throws DatabaseException;
```

Marks the current transaction as successful. Finally it will End a transaction.

Example: Save And Commt Liquor Transaction.

```
Liquor beer = new Liquor();
beer.setLiquorType(Liquor.LIQUOR_TYPE_BEER);
beer.setDescription(applicationContext.getString(R.string.
    beer_description));
beer.setHistory(applicationContext.getString(R.string.beer_history)
    );
beer.setLink(applicationContext.getString(R.string.beer_link));
beer.setAlcholContent(applicationContext.getString(R.string.
    beer_alchol_content));
```

```
DatabaseDescriptor databaseDescriptor = beer.getDatabaseDescriptor
    ();

try {
  Database.beginTransaction(databaseDescriptor);

  beer.save();

  Database.commitTransaction(databaseDescriptor);
} catch(DatabaseException de) {
  //Log it.
} finally {
  Database.endTransaction(databaseDescriptor);
}
```

5.5.3 End Transaction

```
public static final void endTransaction(final DatabaseDescriptor
    databaseDescriptor);
```

End the current transaction.

Example: End Transaction After Save And Commit Transaction.

```
Liquor beer = new Liquor();
beer.setLiquorType(Liquor.LIQUOR_TYPE_BEER);
beer.setDescription(applicationContext.getString(R.string.
    beer_description));
beer.setHistory(applicationContext.getString(R.string.beer_history)
    );
beer.setLink(applicationContext.getString(R.string.beer_link));
beer.setAlcholContent(applicationContext.getString(R.string.
    beer_alchol_content));

DatabaseDescriptor databaseDescriptor = beer.getDatabaseDescriptor
    ();

try {
  Database.beginTransaction(databaseDescriptor);

  beer.save();

  Database.commitTransaction(databaseDescriptor);
} catch(DatabaseException de) {
  //Log it.
} finally {
  Database.endTransaction(databaseDescriptor);
}
```

5.6 Making Transaction Thread Safe

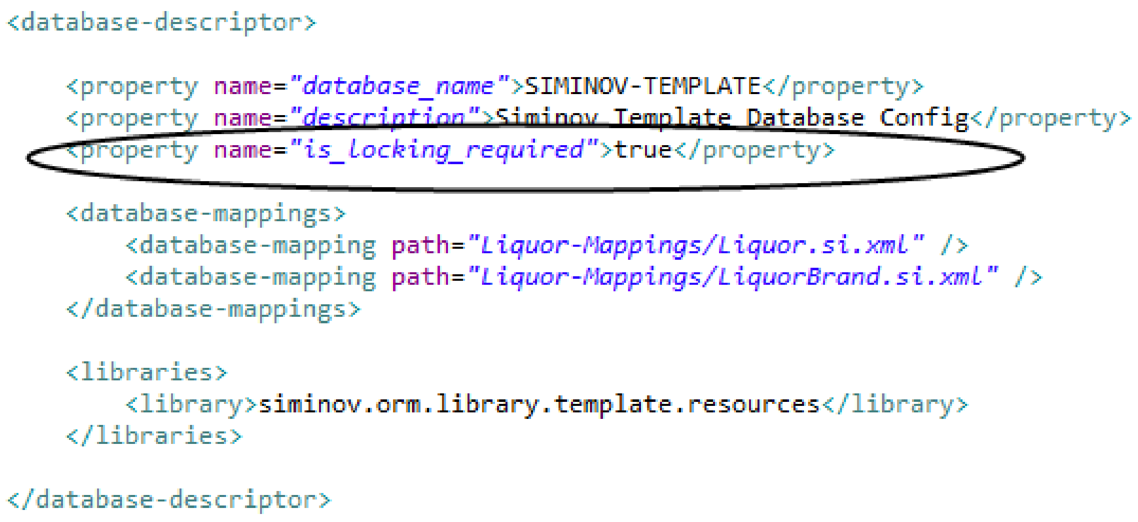
By default any transaction executed on database is not thread safe, android provides API to make all transaction thread-safe by using locks around critical sections. This is pretty expensive, so if you know that your DB will nly be used by a single thread then you should not use this in your application.

Android API: setLockingEnabled Enable/Disable

```
public void setLockingEnabled (boolean lockingEnabled);
```

Con guring transaction thread-safe in Siminov. To enable/disable transaction thread-safe in Siminov you have to use property is_locking_required de ned in DatabaseDescriptor.si.xml le.

Example: Siminov Template Application DatabaseDescriptor.si.xml le.



```
<database-descriptor>

    <property name="database_name">SIMINOV-TEMPLATE</property>
    <property name="description">Siminov Template Database Config</property>
    <property name="is_locking_required">true</property>

    <database-mappings>
        <database-mapping path="Liquor-Mappings/Liquor.si.xml" />
        <database-mapping path="Liquor-Mappings/LiquorBrand.si.xml" />
    </database-mappings>

    <libraries>
        <library>siminov.orm.library.template.resources</library>
    </libraries>

</database-descriptor>
```

5.7 Preparing Where Clause

Siminov does not provide any mechanism to prepare where-clause, Providing where-clause for fetch API is easy. It take same syntax as we form where-clause for SQLite statement.

Example: Fetch Liquor tuple where liquor type is Beer.

```
String whereClause = Liquor.LIQUOR_BRAND + "='" + Liquor.
    LIQUOR_TYPE_BEER + "'";

Liquor[] liquors = null;
try {
  liquors = new Liquor().fetch(whereClause);
} catch(DatabaseException de) {
  //Log it.
}

}
```

5.8 Handling Database Relationships

5.8.1 One to One

One-To-One relationship is in which each row in one database table is linked to 1 and 1 other row in another table.

Example: Relationship between Table A and Table B, each row in Table A is linked to another row in Table B. The number of rows in Table A must equal the number of rows in Table B.

One To One Syntax:

```
<database-mapping>

  <table table_name="name-of-table" class_name="full-class-path-of-
      pojo-class">

    <relationships>

      <one-to-one refer="name-of-variable" refer_to="full-class-path-
          of-refer-variable" on_update="cascade/restrict/no_action/
          set_null/set_default" on_delete="cascade/restrict/no_action/
          set_null/set_default">
```

```
        <property name="load">true/false</property>
      </one-to-one>

    </relationships>

  </table>

</database-mapping>
```

5.8.2 One to Many

One-To-Many relationship is in which each row in the related to table can be related to many rows in the relating table. This e ectively save storage as the related record does not need to be stored multiple times in the relating table.

Example: All the customers belonging to a business is stored in a customer table while all the customer invoices are stored in an invoice table. Each customer can have many invoices but each invoice can only be generated for a single customer.

One To Many Syntax:

```
<database-mapping>

  <table table_name="name-of-table" class_name="full-class-path-of-
      pojo-class">

    <relationships>

      <one-to-many refer="name-of-variable" refer_to="full-class-path
          -of-refer-variable" on_update="cascade/restrict/no_action/
          set_null/set_default" on_delete="cascade/restrict/no_action/
          set_null/set_default">
        <property name="load">true/false</property>
      </one-to-many>

    </relationships>

  </table>

</database-mapping>
```

5.8.3 Many to One

Many-To-One relationship is in which one entity (typically a column or set of columns) contains values that refer to another entity (a column or set of columns) that has unique values.

Example: In a geography schema having tables Region, State, and City, there are many states that are in a given region, but no states are in two regions.

Many To One Syntax:

```
<database-mapping>

  <table table_name="name-of-table" class_name="full-class-path-of-
      pojo-class">

    <relationships>

      <many-to-one refer="name-of-variable" refer_to="full-class-path
          -of-refer-variable" on_update="cascade/restrict/no_action/
          set_null/set_default" on_delete="cascade/restrict/no_action/
          set_null/set_default">
        <property name="load">true/false</property>
      </many-to-one>

    </relationships>

  </table>

</database-mapping>
```

5.8.4 Many to Many

Many-To-Many relationship is in which one or more rows in the table can be related to 0, 1 or many rows in another table. A mapping table is required in order to implement such a relationship.

Example: All the customers belonging to a bank is stored in a customer table while all the bank's products are stored in a product table. Each customer can have many products and each product can be assigned to many customers.

Many To Many Syntax:

```
<database-mapping>
```

```
  <table table_name="name-of-table" class_name="full-class-path-of-
      pojo-class">

    <relationships>

      <many-to-many refer="name-of-variable" refer_to="full-class-
          path-of-refer-variable" on_update="cascade/restrict/
          no_action/set_null/set_default" on_delete="cascade/restrict/
          no_action/set_null/set_default">
        <property name="load">true/false</property>
      </many-to-many>

    </relationships>

  </table>

</database-mapping>
```

Chapter 6

Database Encryption

Data Secuirty plays important role when we talk about database. It protect your database from desctructive forces and the unwanted actions of unauthorized users.

Android sqlite does not provide any protection against your database. There are many third party security implementation's which application developer can use in their application to protect their database.

6.1 SQLCipher

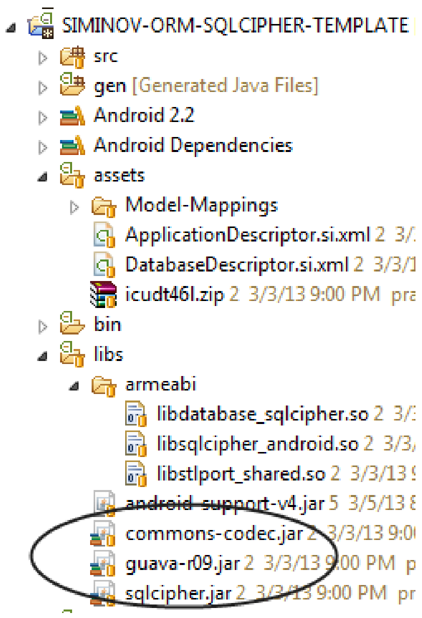
SQLCipher is an open source extension to SQLite that provides transparent 256-bit AES encryption of database le. SQLCipher has a small footprint and great performance so it's ideal for protecting embedded application databases and is well suited for mobile development.

Siminov provide implementation for SQLCipher database encryption security. Its easy and secured to use it.

Below are steps to make your application totally secured.

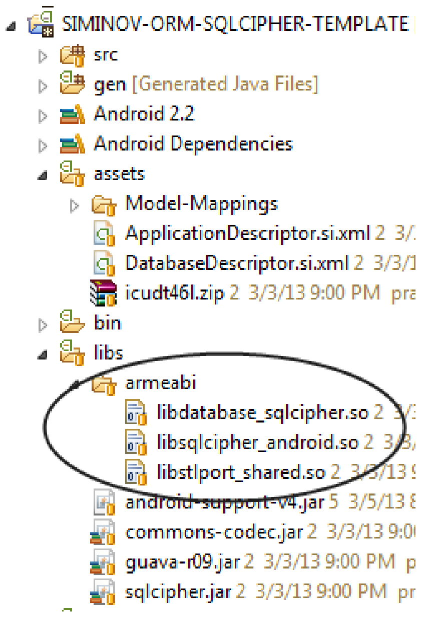
1. Download SQLCipher from their website for android (`http://sqlcipher.net/downloads/`).
2. Con gure SQLCipher in your application. Follow below steps:

(a) Copy sqlcipher.jar, guava-r09.jar, commons-codec.jar in your application libs folder.



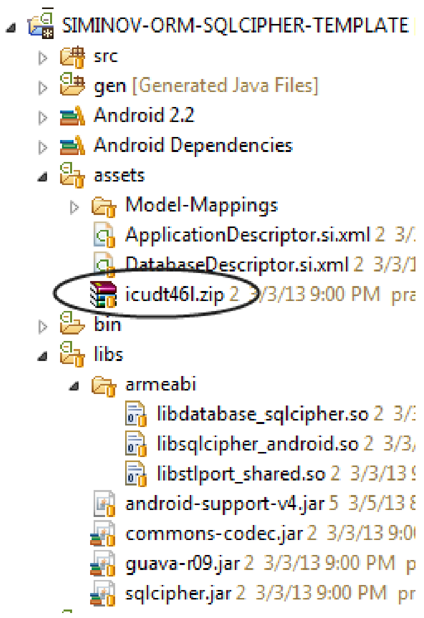
```
SIMINOV-ORM-SQLCIPHER-TEMPLATE
  src
  gen [Generated Java Files]
  Android 2.2
  Android Dependencies
  assets
    Model-Mappings
    ApplicationDescriptor.si.xml 2  3/.
    DatabaseDescriptor.si.xml 2  3/3/1
    icudt46l.zip 2  3/3/13 9:00 PM  pra
  bin
  libs
    armeabi
      libdatabase_sqlcipher.so 2  3/:
      libsqlcipher_android.so 2  3/3,
      libstlport_shared.so 2  3/3/13 !
    ~~android-support-v4.jar~~ 5  3/5/13 8
    commons-codec.jar 2  3/3/13 9:0
    guava-r09.jar 2  3/3/13 9:00 PM  p
    sqlcipher.jar 2  3/3/13 9:00 PM  pr
```

(b) Copy dll le in your libs folder.



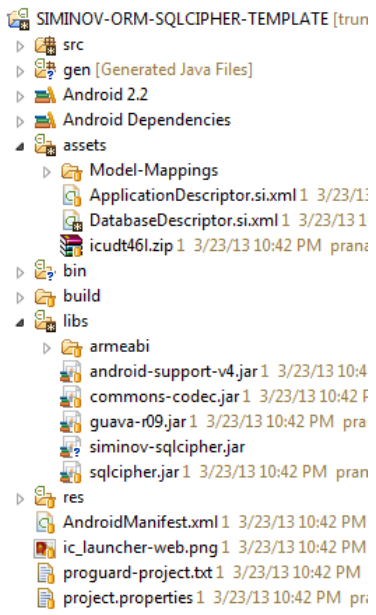
```
SIMINOV-ORM-SQLCIPHER-TEMPLATE
  src
  gen [Generated Java Files]
  Android 2.2
  Android Dependencies
  assets
    Model-Mappings
    ApplicationDescriptor.si.xml 2  3/.
    DatabaseDescriptor.si.xml 2  3/3/1
    icudt46l.zip 2  3/3/13 9:00 PM  pra
  bin
  libs
    armeabi
      libdatabase_sqlcipher.so 2  3/:
      libsqlcipher_android.so 2  3/3,
      libstlport_shared.so 2  3/3/13 !
    ~~android-support-v4.jar~~ 5  3/5/13 8
    commons-codec.jar 2  3/3/13 9:0
    guava-r09.jar 2  3/3/13 9:00 PM  p
    sqlcipher.jar 2  3/3/13 9:00 PM  pr
```

(c) Copy icudt461.zip in your application assets folder.



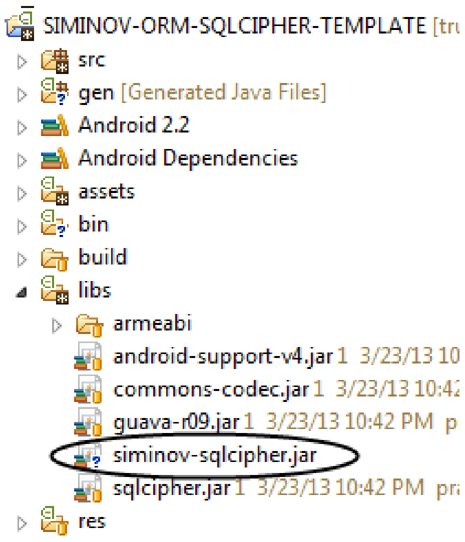
```
SIMINOV-ORM-SQLCIPHER-TEMPLATE
  src
  gen [Generated Java Files]
  Android 2.2
  Android Dependencies
  assets
    Model-Mappings
    ApplicationDescriptor.si.xml 2  3/
    DatabaseDescriptor.si.xml 2  3/3/1
    icudt46l.zip 2  3/3/13 9:00 PM  pra
  bin
  libs
    armeabi
      libdatabase_sqlcipher.so 2  3/
      libsqlcipher_android.so 2  3/3,
      libstlport_shared.so 2  3/3/13
    android-support-v4.jar 5  3/5/13
    commons-codec.jar 2  3/3/13 9:0
    guava-r09.jar 2  3/3/13 9:00 PM  p
    sqlcipher.jar 2  3/3/13 9:00 PM  pr
```

The stucture of your application should look like this.



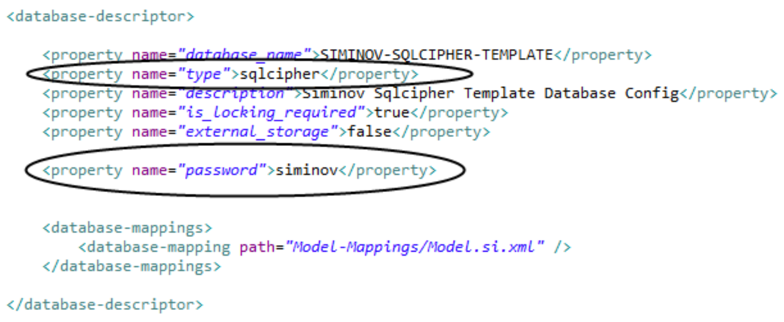
```
SIMINOV-ORM-SQLCIPHER-TEMPLATE [trun
  src
  gen [Generated Java Files]
  Android 2.2
  Android Dependencies
  assets
    Model-Mappings
    ApplicationDescriptor.si.xml 1  3/23/1
    DatabaseDescriptor.si.xml 1  3/23/13 1
    icudt46l.zip 1  3/23/13 10:42 PM  prana
  bin
  build
  libs
    armeabi
    android-support-v4.jar 1  3/23/13 10:4
    commons-codec.jar 1  3/23/13 10:42 F
    guava-r09.jar 1  3/23/13 10:42 PM  pra
    siminov-sqlcipher.jar
    sqlcipher.jar 1  3/23/13 10:42 PM  pran
  res
  AndroidManifest.xml 1  3/23/13 10:42 PM
  ic_launcher-web.png 1  3/23/13 10:42 PM
  proguard-project.txt 1  3/23/13 10:42 PM
  project.properties 1  3/23/13 10:42 PM  pr
```

3. Download and copy Siminov SQLCipher jar in application libs folder.



```
SIMINOV-ORM-SQLCIPHER-TEMPLATE [tru
  ▷ src
  ▷ gen [Generated Java Files]
  ▷ Android 2.2
  ▷ Android Dependencies
  ▷ assets
  ▷ bin
  ▷ build
  ◢ libs
      ▷ armeabi
        android-support-v4.jar 1  3/23/13 10
        commons-codec.jar 1  3/23/13 10:4
        guava-r09.jar 1  3/23/13 10:42 PM  p
        siminov-sqlcipher.jar
        sqlcipher.jar 1  3/23/13 10:42 PM  pr
  ▷ res
```

4. To con gure SQLCipher with Siminov add type property as sqlcipher and password attribute in DatabaseDescriptor.si.xml.



```
<database-descriptor>

    <property name="database_name">SIMINOV-SQLCIPHER-TEMPLATE</property>
    <property name="type">sqlcipher</property>
    <property name="description">Siminov Sqlcipher Template Database Config</property>
    <property name="is_locking_required">true</property>
    <property name="external_storage">false</property>

    <property name="password">siminov</property>


    <database-mappings>
        <database-mapping path="Model-Mappings/Model.si.xml" />
    </database-mappings>

</database-descriptor>
```

Note

For any future reference you can download SIMINOV-SQLCIPHER-TEMPLATE Application and can check how we have con gured application with SQLCipher.

Chapter 7

Database Layer

Database is the most important part of Siminov ORM. It provides an easy way to implement your own database layer, all you have to do is provide implementation for below interface's.

7.1 IDatabase Interface

Exposes methods to deal with actual database object. It has methods to open, create, close, and execute query's.

```
public interface IDatabase {

  public void openOrCreate(final DatabaseDescriptor
      databaseDescriptor) throws DatabaseException;

  public void close(final DatabaseDescriptor databaseDescriptor)
      throws DatabaseException;

  public void executeQuery(final DatabaseDescriptor
      databaseDescriptor, final DatabaseMappingDescriptor
      databaseMappingDescriptor, final String query) throws
      DatabaseException;

  public void executeBindQuery(final DatabaseDescriptor
      databaseDescriptor, final DatabaseMappingDescriptor
      databaseMappingDescriptor, final String query, final Iterator<
      Object> columnValues) throws DatabaseException;

  public Iterator<Map<String, Object>> executeFetchQuery(final
      DatabaseDescriptor databaseDescriptor, final
```

```
    DatabaseMappingDescriptor databaseMappingDescriptor, final
    String query) throws DatabaseException;

  public void executeMethod(final String methodName, final Object
    parameters) throws DatabaseException;

}
```

1. **Open Or Create - openOrCreate(Path)**: Open/Create the database through Database Descriptor. By default add CREATE_IF_NECESSARY ag so that if database does not exist it will create.

   Example:

```
    DatabaseDescriptor databaseDescriptor = Resources.
       getInstance().getDatabaseDescriptorBasedOnName(database-
       descriptor-name);
    IDatabase database = null;

    try {
      database = Database.createDatabase(databaseDescriptor);
    } catch(DatabaseException databaseException) {
      //Log It.
    }


    try {
      database.openOrCreate(databasePath + databaseDescriptor.
         getDatabaseName());
    } catch(DatabaseException databaseException) {
      // Log It.
    }
```

2. **Close - close()**: Close the existing opened database through Database Descriptor.

   Example:

```
    DatabaseDescriptor databaseDescriptor = Resources.
       getInstance().getDatabaseDescriptorBasedOnName(database-
       descriptor-name);
    IDatabase database = null;

    try {
      database = Database.createDatabase(databaseDescriptor);
    } catch(DatabaseException databaseException) {
```

```
    //Log It.
  }


  try {
    database.close();
  } catch(DatabaseException databaseException) {
    // Log It.
  }
```

3. **Execute Query - executeQuery(Query)**: Execute a single SQL statement that is NOT a SELECT or any other SQL statement that returns data. It has no means to return any data (such as the number of a ected rows). Instead, you're encouraged to use insert, update, delete, when possible.

   Example:

```
  DatabaseDescriptor databaseDescriptor = Resources.
      getInstance().getDatabaseDescriptorBasedOnName(database-
      descriptor-name);
  IDatabase database = null;

  try {
    database = Database.createDatabase(databaseDescriptor);
  } catch(DatabaseException databaseException) {
    //Log It.
  }
```

```
    try {
      database = Database.createDatabase(databaseDescriptor);
    } catch(DatabaseException databaseException) {
      //Log It.
    }


    try {
      database.executeBindQuery(Query, Column Values);
    } catch(DatabaseException databaseException) {
      // Log It.
    }
```

5. **Execute Fetch Query - executeFetchQuery(Query)**: Query the given table, returning a Cursor over the result set.

   Example:

```
    DatabaseDescriptor databaseDescriptor = Resources.
        getInstance().getDatabaseDescriptorBasedOnName(database-
        descriptor-name);
    IDatabase database = null;

    try {
      database = Database.createDatabase(databaseDescriptor);
    } catch(DatabaseException databaseException) {
      //Log It.
    }


    try {
      database.executeFetchQuery(Query);
    } catch(DatabaseException databaseException) {
      // Log It.
    }
```

6. **Execute Method - executeMethod(Method Name, Parameters)**: Executes the method on database object.

   Example:

```
    DatabaseDescriptor databaseDescriptor = Resources.
        getInstance().getDatabaseDescriptorBasedOnName(database-
        descriptor-name);
    IDatabase database = null;
    try {
      database = Database.createDatabase(databaseDescriptor);
    } catch(DatabaseException databaseException) {
      //Log It.
    }


    try {
      database.executeMethod(Method Name, Parameters);
    } catch(DatabaseException databaseException) {
      // Log It.
    }
```

7.2 IDataTypeHandler

Exposes convert API which is responsible to provide column data type based on java variable data type.

```
public interface IDataTypeHandler {

  public String convert(String dataType);

}
```

1. **Convert Data Type - convert()**: Converts java variable data type to database column data type.

7.3 IQueryBuilder

Exposes API's to build database queries.

```
public interface IQueryBuilder {

  public String formCreateTableQuery(final String tableName, final
      Iterator<String> columnNames, final Iterator<String> columnTypes
      , final Iterator<String> defaultValues, final Iterator<String>
      checks, final Iterator<String> primaryKeys, final Iterator<
      Boolean> isNotNull, final Iterator<String> uniqueColumns, final
      String foreignKeys);
```

```
public String formCreateIndexQuery(final String indexName, final
    String tableName, final Iterator<String> columnNames, final
    boolean isUnique);

public String formDropTableQuery(final String tableName);

public String formDropIndexQuery(String tableName, String indexName
    );

public String formSelectQuery(final String tableName, final boolean
     distinct, final String whereClause, final Iterator<String>
    columnNames, final Iterator<String> groupBys, final String
    having, final Iterator<String> orderBy, final String
    whichOrderBy, final String limit);

public String formSaveBindQuery(final String tableName, final
    Iterator<String> columnNames);

public String formUpdateBindQuery(final String tableName, final
    Iterator<String> columnNames, final String whereClause);

public String formDeleteQuery(final String tableName, final String
    whereClause);

public String formCountQuery(final String tableName, final String
    column, final boolean distinct, final String whereClause, final
    Iterator<String> groupBys, final String having);

public String formAvgQuery(final String tableName, final String
    column, final String whereClause, final Iterator<String>
    groupBys, final String having);

public String formMaxQuery(final String tableName, final String
    column, final String whereClause, final Iterator<String>
    groupBys, final String having);

public String formMinQuery(final String tableName, final String
    column, final String whereClause, final Iterator<String>
    groupBys, final String having);

public String formGroupConcatQuery(final String tableName, final
    String column, final String delimiter, final String whereClause,
     Iterator<String> groupBys, final String having);

public String formSumQuery(final String tableName, final String
    column, final String whereClause, final Iterator<String>
    groupBys, final String having);
```

```
  public String formTotalQuery(final String tableName, final String
     column, final String whereClause, final Iterator<String>
     groupBys, final String having);

  public Iterator<String> formTriggers(final
     DatabaseMappingDescriptor databaseMappingDescriptor);

  public String formForeignKeys(final DatabaseMappingDescriptor
     databaseMappingDescriptor);

}
```

Chapter 8

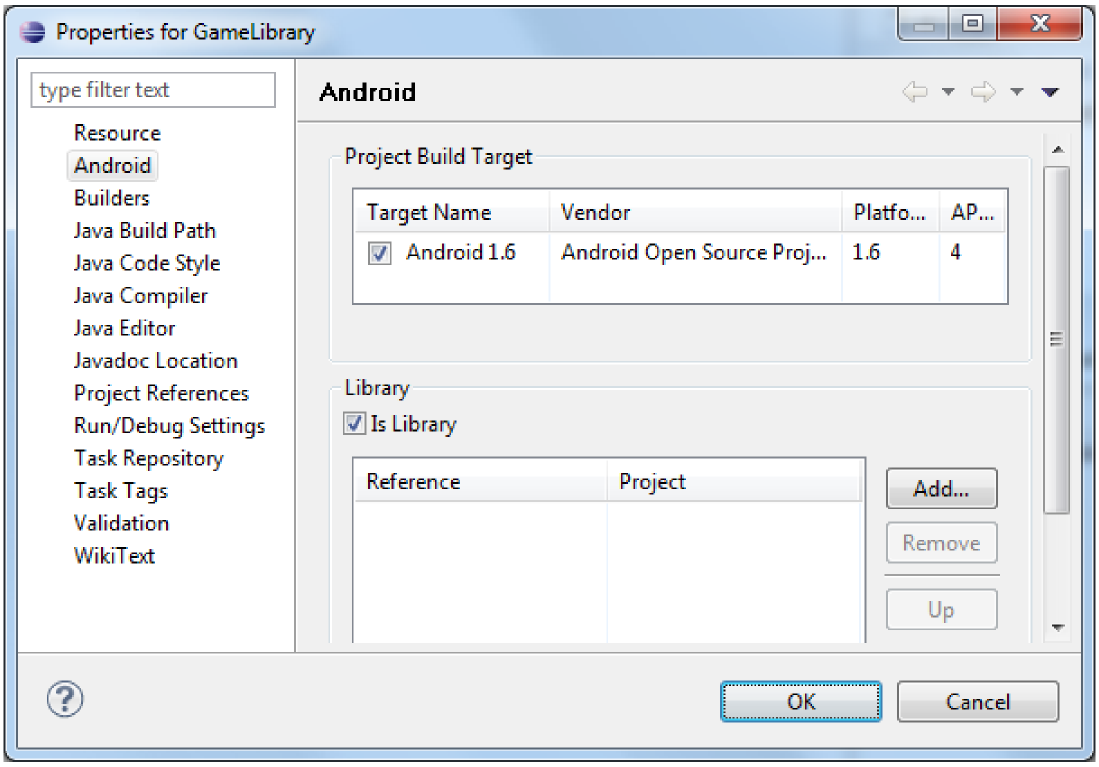
Handling Libraries

An Android library project is a development project that holds shared Android source code and resources. Other Andriod application projects can reference the library project and, at build time, include its compiled sources in their .apk les. Multiple application projects can reference the same library project and any single application project can reference multiple library projects.

Siminov provides mechnism to con gure ORM for your library projects.

8.1 Setting up a Library Project

1. In the Package Explorer, right-click the library project and select Properties.
2. In the Properties window, select the Andorid properties group at left and locate the Library properties at right.
3. Select the is Library checkbox and click Apply.
4. Click OK to close the properties window.



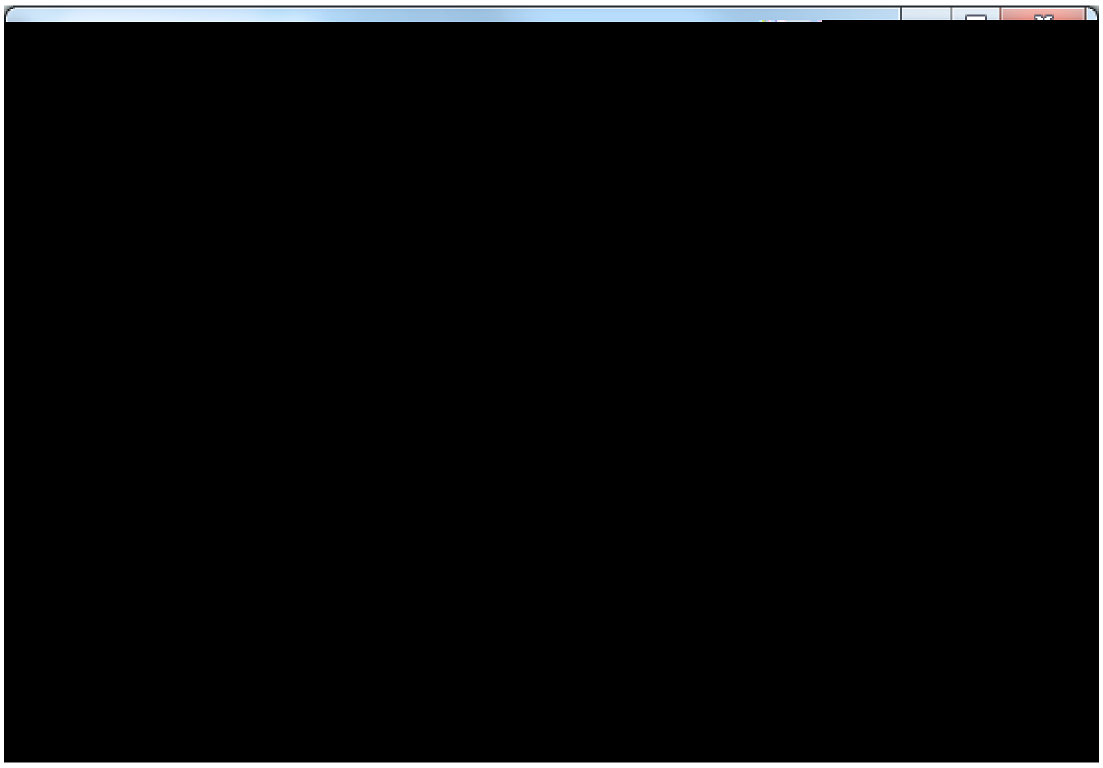
Target Name	Vendor	Platfo...	AP...
✓ Android 1.6	Android Open Source Proj...	1.6	4

✓ Is Library

Reference	Project

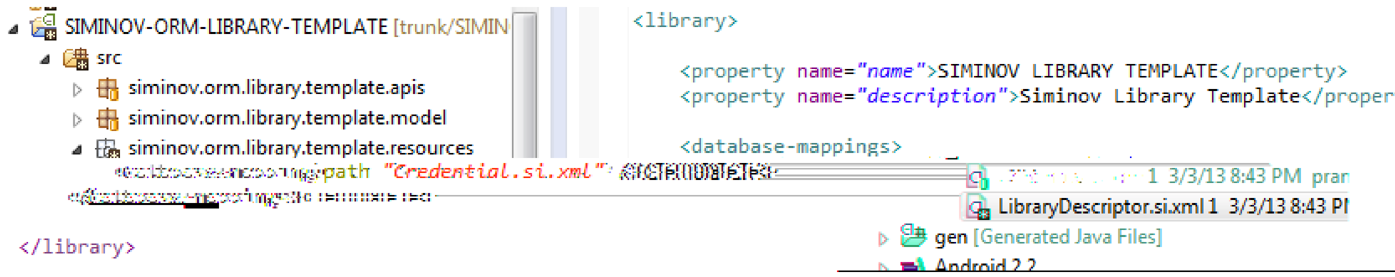
8.2 Referencing a library project

1. In the Package Explorer, right-click the depedent project and select Properties.
2. In the Properties window, select the Android properties group at left and locate the Library properties at right.
3. Click Add to open the Project Selection dialog.
4. From the list of available library project, select a project and click OK.
5. When the dialog closes, click Apply in the Properties window.
6. Click OK to close the Properties window.



8.3 Con gure Application With Library

1. De ne LibraryDescriptor.si.xml le for your library project.

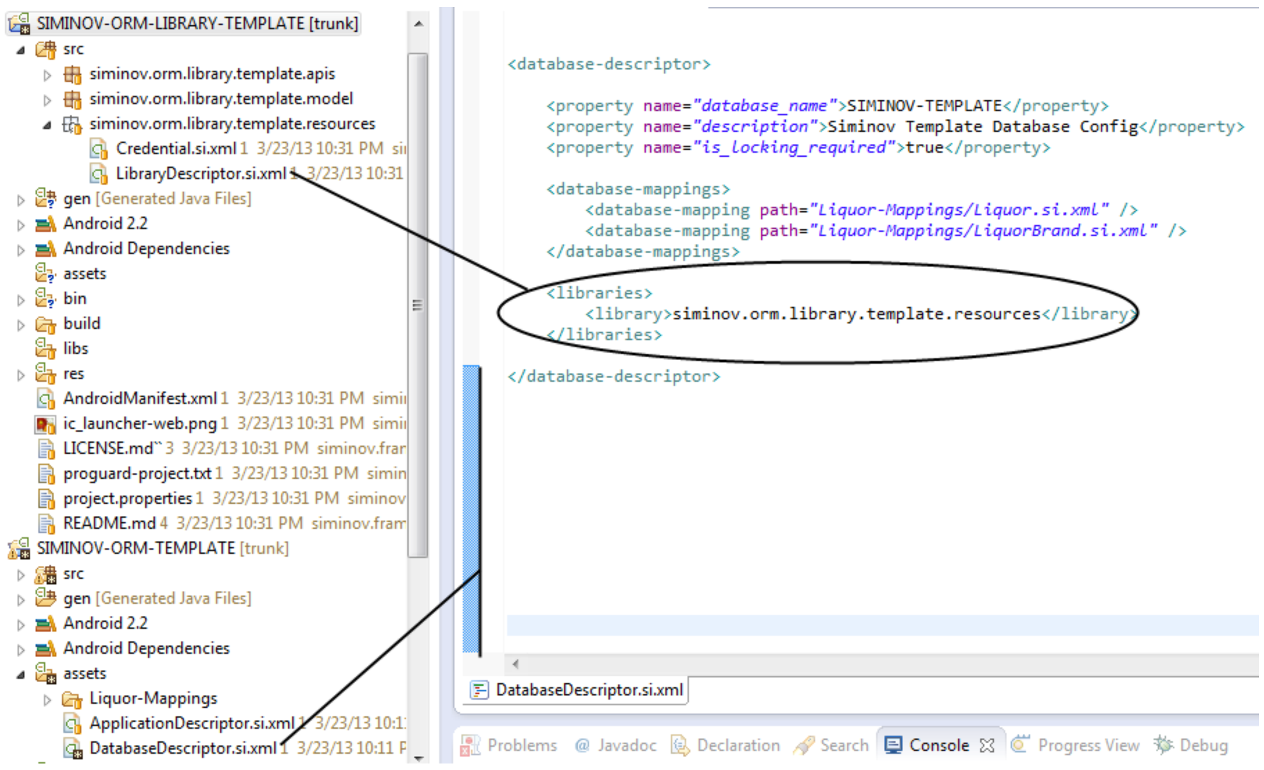


SIMINOV-ORM-LIBRARY-TEMPLATE [trunk/SIMIN
src
siminov.orm.library.template.apis
siminov.orm.library.template.model
siminov.orm.library.template.resources
1 3/3/13 8:43 PM pran
LibraryDescriptor.si.xml 1 3/3/13 8:43 PI
gen [Generated Java Files]
Android 2.2

```
<library>

    <property name="name">SIMINOV LIBRARY TEMPLATE</property>
    <property name="description">Siminov Library Template</proper

    <database-mappings>
        path "Credential.si.xml"

</library>
```

LibraryDescriptor.si.xml le should not be place in assets folder. Create new package and place all descriptors in that.

2. Con gure LibraryDescriptor.si.xml in your application.



SIMINOV-ORM-LIBRARY-TEMPLATE [trunk]
src
siminov.orm.library.template.apis
siminov.orm.library.template.model
siminov.orm.library.template.resources
Credential.si.xml 1 3/23/13 10:31 PM si
LibraryDescriptor.si.xml 1 3/23/13 10:31
gen [Generated Java Files]
Android 2.2
Android Dependencies
assets
bin
build
libs
res
AndroidManifest.xml 1 3/23/13 10:31 PM simi
ic_launcher-web.png 1 3/23/13 10:31 PM simi
LICENSE.md 3 3/23/13 10:31 PM siminov.fran
proguard-project.txt 1 3/23/13 10:31 PM simin
project.properties 1 3/23/13 10:31 PM siminov
README.md 4 3/23/13 10:31 PM siminov.fram
SIMINOV-ORM-TEMPLATE [trunk]
src
gen [Generated Java Files]
Android 2.2
Android Dependencies
assets
Liquor-Mappings
ApplicationDescriptor.si.xml 1 3/23/13 10:1
DatabaseDescriptor.si.xml 1 3/23/13 10:11 P
DatabaseDescriptor.si.xml
Problems
Javadoc
Declaration
Search
Console
Progress View
Debug

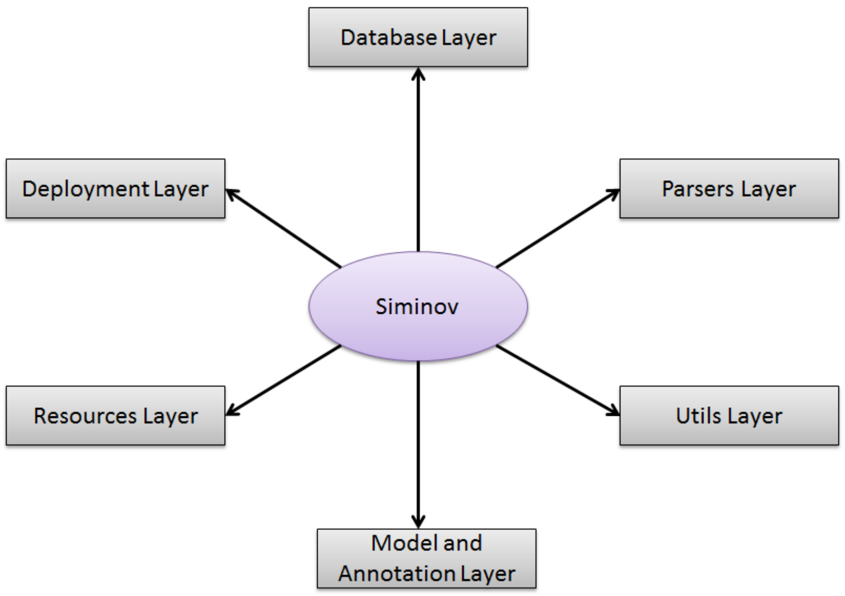
```
<database-descriptor>

    <property name="database_name">SIMINOV-TEMPLATE</property>
    <property name="description">Siminov Template Database Config</property>
    <property name="is_locking_required">true</property>

    <database-mappings>
        <database-mapping path="Liquor-Mappings/Liquor.si.xml" />
        <database-mapping path="Liquor-Mappings/LiquorBrand.si.xml" />
    </database-mappings>

    <libraries>
        <library>siminov.orm.library.template.resources</library>
    </libraries>

</database-descriptor>
```

Note

While con guring DatabaseDescriptor.si.xml le you need to provide full library package name in which LibraryDescriptor.si.xml le is de ned.

Chapter 9

Siminov ORM Architecture



Siminov ORM Layers

9.1 Deployment Layer

When application starts for rst time it does not have its database's. Siminov provides deployment layer, it creates application database's by reading all descriptors from application assests and its libraries.

9.1.1 siminov.orm.Siminov

```
/**
 Exposes methods to deal with SIMINOV FRAMEWORK.
 Such As
  1. Initialize: Entry point to the SIMINOV.
  2. Shutdown: Exit point from the SIMINOV.
 */
public class Siminov {

  /**
   It is used to check weather SIMINOV FRAMEWORK is active or not.
   SIMINOV become active only when deployment of application is
       successful.
   */
public static final void validateCore();


  /**
   It is the entry point to the SIMINOV FRAMEWORK.
When application starts it should call this method to activate
   SIMINOV-FRAMEWORK, by providing ApplicationContext as the
   parameter.

Siminov will read all descriptor defined by application, and do
   necessary processing.

There are two ways to make a call.
      1. Call it from Application class.
2. Call it from LAUNCHER Activity.
   */
  public static final void initialize(final Context context);

  /**
   It is used to stop all service started by SIMINOV.
   When application shutdown they should call this. It do following
       services:
   Close all database's opened by SIMINOV.
   Deallocate all resources held by SIMINOV.
  */
  public static final void shutdown() throws CoreException;

}
```

9.2 Resources Layer

It holds meta data about application needed by Siminov.

9.2.1 siminov.orm.Resources

Only GET APIS shown, for rest visit Javadoc

```
/**
 It handles and provides all resources needed by SIMINOV.
 Such As: Provides Application Descriptor, Database Descriptor,
    Library Descriptor,
    Database Mapping.
 */
public final class Resources {

  /**
   It provides an instance of Resources class.
   */
  public static Resources getInstance();

  /**
   Returns Application Context provided by application.
   */
  public Context getApplicationContext();

  /**
   Get Application Descriptor object of application.
   */
  public ApplicationDescriptor getApplicationDescriptor();

  /**
   Get iterator of all database descriptors provided in Application
       Descriptor file.
   */
  public Iterator<String> getDatabaseDescriptorsPaths();

  /**
   Get DatabaseDescriptor based on path provided as per defined in
       Application Descriptor file.
    */

  public DatabaseDescriptor getDatabaseDescriptorBasedOnPath(final
     String databaseDescriptorPath);
```

```
/**
 Get Database Descriptor based on database descriptor name provided
      as per defined in Database Descriptor file.
*/
public DatabaseDescriptor getDatabaseDescriptorBasedOnName(final
    String databaseDescriptorName);

/**
 Get all Database Descriptors object.
*/
public Iterator<DatabaseDescriptor> getDatabaseDescriptors();

/**
 Get Database Descriptor based on POJO class name provided.
*/
public DatabaseDescriptor getDatabaseDescriptorBasedOnClassName(
    final String className);


/**
 Get Database Descriptor based on table name provided.
*/
public DatabaseDescriptor getDatabaseDescriptorBasedOnTableName(
    final String tableName);

/**
 Get Database Mapping based on POJO class name provided.
*/
public DatabaseMapping getDatabaseMappingBasedOnClassName(final
    String className);

/**
 Get Database Mapping based on table name provided.
*/
public DatabaseMapping getDatabaseMappingBasedOnTableName(final
    String tableName);

/**
 Get all Library Paths as per defined in all Database Descriptor
     file's.
*/
public Iterator<String> getLibraryPaths();

/**
 Get all library paths based on Database Descriptor name.
*/
public Iterator<String>
    getLibraryPathsBasedOnDatabaseDescriptorName(final String
    databaseDescriptorName);
```

```
/**
 Get all Library Descriptor objects as per contain in all Database
     Descriptor's.
*/
public Iterator<LibraryDescriptor> getLibraries();

/**
 Get all Library Descriptor objects based on Database Descriptor
     name.
*/
public Iterator<LibraryDescriptor>
    getLibrariesBasedOnDatabaseDescriptorName(final String
    databaseDescriptorName);

/**
 * Get all Library Database Mapping objects based in library
     descriptor path.
*/
public Iterator<DatabaseMapping>
    getLibraryDatabaseMappingsBasedOnLibraryDescriptorPath(final
    String libraryPath);

/**
 Get IDatabase object based on Database Descriptor name.
*/
public IDatabase getDatabaseBasedOnDatabaseDescriptorName(final
    String databaseName);



/**
 Get IDatabase object based on Database Mapping POJO class name.
*/
public IDatabase getDatabaseBasedOnDatabaseMappingPojoClass(final
    Class<?> classObject);

/**
 Get IDatabase based on Database Mapping POJO class name.
*/
public IDatabase getDatabaseBasedOnDatabaseMappingClassName(final
    String databaseMappingClassName);

/**
 Get IDatabase object based on Database Mapping table name.
*/
public IDatabase getDatabaseBasedOnDatabaseMappingTableName(final
    String databaseMappingTableName);
	/**
	 Get all IDatabase objects contain by application.
	*/
	public Iterator<IDatabase> getDatabases();

	/**
	  Get SIMINOV-EVENT Handler
	*/
	public ISiminovEvents getSiminovEventHandler();

	/**
	 Get DATABASE-EVENT Handler
	*/
	public IDatabaseEvents getDatabaseEventHandler();

}
```

9.3 Database Layer

This layer basically deals with database operations.

9.3.1 siminov.orm.database.Database

```
/**
 Exposes methods to deal with database.
 It has methods to create, delete, and perform other common database
    management tasks.
 */
public abstract class Database implements Constants {

  /**
     Is used to create a new table in an database.
  */
  public static final void createTables(final Iterator<
     DatabaseMapping> databaseMappings) throws DatabaseException;

  /**
     Is used to create a new table in an database.
  */
  public static final void createTable(final DatabaseMapping
     databaseMapping) throws DatabaseException;

  /**
     Is used to create a new index on a table in database.
  */
  public static final void createIndex(final DatabaseMapping
     databaseMapping, final Index index) throws DatabaseException;

  /**
     Is used to create a new index on a table in database.
  */
  public static final void createIndex(final DatabaseMapping
     databaseMapping, final String indexName, final Iterator<String>
     columnNames, final boolean isUnique) throws DatabaseException;

  /**
     Is used to create a new index on a table in database.
  */
  public void createIndex(final Index index) throws DatabaseException
     ;

  /**
```

```
   Is used to create a new index on a table in database.
*/
public void createIndex(final String indexName, final Iterator<
   String> columnNames, final boolean isUnique) throws
   DatabaseException;

/**
   Is used to drop a index on a table in database.
*/
public void dropIndex(final String indexName) throws
   DatabaseException;

/**
   Is used to drop a index on a table in database.
*/
public void dropIndex(final DatabaseMapping databaseMapping, final
   String indexName) throws DatabaseException;

/**
 It drop's the table from database
*/
public void dropTable() throws DatabaseException;

/**
 It drop's the table from database based on database-mapping.
*/
public static final void dropTable(final DatabaseMapping
   databaseMapping) throws DatabaseException;

/**
 It drop's the whole database based on database-descriptor.
*/
public static final void dropDatabase(final DatabaseDescriptor
   databaseDescriptor) throws DatabaseException;

/**
  Begins a transaction in EXCLUSIVE mode.
*/
public static final void beginTransaction(final DatabaseDescriptor
   databaseDescriptor) throws DatabaseException;

/**
 Marks the current transaction as successful.
*/
public static final void commitTransaction(final DatabaseDescriptor
    databaseDescriptor) throws DatabaseException;

/**
 End the current transaction.
```

```
*/
public static final void endTransaction(final DatabaseDescriptor
    databaseDescriptor);

/**
 Returns all tuples from mapped table for invoked class object.
*/
public final Object[] fetch() throws DatabaseException;

/**
 Returns selected column values for all tuples from mapped table
     for invoked class object.
*/
public final Object[] fetch(final Iterator<String> columnNames)
    throws DatabaseException;

/**
 Returns all tuples based on where clause from mapped table for
     invoked class object.
*/
public final Object[] fetch(final String whereClause) throws
    DatabaseException;


/**
 Returns selected column values of selected tuples based on where
     clause from mapped table for invoked class object.
*/
public final Object[] fetch(final String whereClause, final
    Iterator<String> columnNames) throws DatabaseException;

/**
 Returns selected column values of all tuples based on where clause
      across multiple records and group the results by one or more
     columns from mapped table for invoked class object.
*/
public final Object[] fetch(final String whereClause, final
    Iterator<String> columnNames, final Iterator<String> groupBy)
    throws DatabaseException;

/**
 Returns selected column values of all tuples based on where clause
      across multiple records and group the results by one or more
     columns, and can use having clause in combination with group by
     , for mapped table for invoked class object.
*/
public final Object[] fetch(final String whereClause, final
    Iterator<String> columnNames, final Iterator<String> groupBy,
    final String having) throws DatabaseException;
```

```
/**
 Returns A result set with the rows being sorted by the values of
     one or more column values of all tuples based on where clause
     across multiple records and group the results by one or more
     columns, and can use having clause in combination with group by
     , for mapped table for invoked class object.
*/
public final Object[] fetch(final String whereClause, final
    Iterator<String> columnNames, final Iterator<String> groupBy,
    final String having, final Iterator<String> orderBy) throws
    DatabaseException;

/**
 Returns A limited result set with the rows being sorted by the
     values of one or more column values of all tuples based on
     where clause across multiple records and group the results by
     one or more columns, and can use having clause in combination
     with group by, for mapped table for invoked class object.
*/
public final Object[] fetch(final String whereClause, final
    Iterator<String> columnNames, final Iterator<String> groupBy,
    final String having, final Iterator<String> orderBy, final
    String limit) throws DatabaseException;

/**
 Returns all tuples based on manual query from mapped table for
     invoked class object.
*/
public final Object[] fetchManual(final String query) throws
    DatabaseException;

/**
 It adds a record to any single table in a relational database.
*/
public final void save() throws DatabaseException;

/**
 It updates a record to any single table in a relational database.
*/
public final void update() throws DatabaseException;

/**
 It finds out weather tuple exists in table or not.
*/
public final void saveOrUpdate() throws DatabaseException;

/**
 It deletes a record to any single table in a relational database.
```

```
*/
public final void delete() throws DatabaseException;

/**
 It deletes one or more records from any single table in a
     relational database, based on where clause provided.
*/
public final void delete(final String whereClause) throws
   DatabaseException;


/**
 Returns the number of rows based on where clause provided.
*/
public final int count() throws DatabaseException;

/**
 Returns the number of rows based on where clause provided.
*/
public final int count(final String whereClause) throws
   DatabaseException;

/**
 Returns the average based on column name provided.
*/
public final int avg(final String columnName) throws
   DatabaseException;

/**
 Returns the sum based on column name provided.
*/
public final int sum(final String columnName) throws
   DatabaseException;

/**
 Returns the total based on column name provided.
*/
public final int total(final String columnName) throws
   DatabaseException;

/**
 Returns the minimum based on column name provided.
*/
public final int min(final String columnName) throws
   DatabaseException;
```

```
/**
 Returns the minimum based on column name provided.
*/
public final int min(final String columnName, final String groupBy)
     throws DatabaseException;

/**
 Returns the maximum based on column name provided.
*/
public final int max(final String columnName) throws
    DatabaseException;

/**
 Returns the maximum based on column name provided.
*/
public final int max(final String columnName, final String groupBy)
     throws DatabaseException;

/**
 Returns the group concat based on column name provided.
*/
public final int groupConcat(final String columnName) throws
    DatabaseException;

/**
 Returns the group concat based on column name and where clause
     provided.
*/
public final String groupConcat(final String columnName, final
    String whereClause) throws DatabaseException;

/**
 Returns the group concat based on column name and where clause
     provided.
*/
public final String groupConcat(final String columnName, final
    String delimiter, final String whereClause) throws
    DatabaseException;

/**
 Returns database descriptor object based on the POJO class called.
*/
public final DatabaseDescriptor getDatabaseDescriptor() throws
    DatabaseException;

/**
 Returns the actual database mapping object mapped for invoked
     class object.
*/
```

```
public final DatabaseMapping getDatabaseMapping() throws
    DatabaseException;

/**
 Returns the mapped table name for invoked class object.
*/
public final String getTableName() throws DatabaseException;

/**
 Returns all column names of mapped table.
*/
public final Iterator<String> getColumnNames() throws
    DatabaseException;

/**
 Returns all column values in the same order of column names for
     invoked class object.
*/
public final Iterator<Object> getColumnValues() throws
    DatabaseException;

/**
 Returns all columns with there data types for invoked class object
     .
*/
public final Map<String, String> getColumnTypes() throws
    DatabaseException;

/**
 Returns all primary keys of mapped table for invoked class object.
*/
public final Iterator<String> getPrimaryKeys() throws
    DatabaseException;

/**
 Returns all mandatory fields which are associated with mapped
     table for invoked class object.
*/
public final Iterator<String> getMandatoryFields() throws
    DatabaseException;


/**
 Returns all unique fields which are associated with mapped table
     for invoked class object.
*/
public final Iterator<String> getUniqueFields() throws
    DatabaseException;
```

```
  /**
   Returns all foreign keys of mapped table for invoked class object.
  */
  public final Iterator<String> getForeignKeys() throws
     DatabaseException;

}
```

9.4 Model and Annotation Layer

It holds Model POJO classes and Annotation de nation required for database mapping descriptor.

9.4.1 siminov.orm.model.ApplicationDescriptor

```
/**
 Exposes methods to GET and SET Application Descriptor information as
     per define in ApplicationDescriptor.si.xml file by application.
*/
public final class ApplicationDescriptor {

  /**
   Get Application Descriptor Name as per defined in
       ApplicationDescriptor.si.xml file.
  */
  public String getName();

  /**
   Set Description of Application as per defined in
       ApplicationDescriptor.si.xml file.
  */
  public String getDescription();

  /**
   Get Version of Application as per defined in ApplicationDescriptor
       .si.xml file.
  */
  public double getVersion();

  /**
   Check weather database needed by application or not.
  */
  public boolean isDatabaseNeeded();

  /**
   Check weather database descriptor exists in Resources or not.
  */
  public boolean containsDatabaseDescriptor(final DatabaseDescriptor
     databaseDescriptor);

  /**
   Check weather database descriptor exists in Resources or not,
      based on database descriptor path.
```

```
*/
public boolean containsDatabaseDescriptorBasedOnPath(final String
   containDatabaseDescriptorPath);

/**
 Check weather database descriptor exists in Resources or not,
    based on Database Descriptor name.
*/
public boolean containsDatabaseDescriptorBasedOnName(final String
   databaseDescriptorName);

/**
 Get Database Descriptor based on Database Descriptor Name.
*/
public DatabaseDescriptor getDatabaseDescriptorBasedOnName(final
   String databaseDescriptorName);

/**
 Get Database Descriptor based on Database Descriptor Path.
public DatabaseDescriptor getDatabaseDescriptorBasedOnPath(final
   String databaseDescriptorPath);

/**
 Get all database descriptor paths as per contained in
    ApplicationDescriptor.si.xml file.
*/
public Iterator<String> getDatabaseDescriptorPaths();

/**
 Get all database descriptor names as per needed by application.
*/
public Iterator<String> getDatabaseDescriptorNames();

/**
 Get all database descriptor objects contains by Siminov.
*/
public Iterator<DatabaseDescriptor> getDatabaseDescriptors();

/**
 Get all database descriptor objects in sorted order.
*/
public Iterator<DatabaseDescriptor> orderedDatabaseDescriptors();

/**
 It defines the behaviour of SIMINOV. (Should siminov load all
    database mapping at initialization or on demand).
*/
public boolean doLoadInitially();
```

```
  /**
   Get all event handlers as per defined in ApplicationDescriptor.si.
      xml file.
  */
  public  Iterator<String> getEvents();

}
```

9.4.2 siminov.orm.model.DatabaseDescriptor

```
/**
 Exposes methods to GET and SET Database Descriptor information as
    per define in DatabaseDescriptor.si.xml file by application.
*/
public final class DatabaseDescriptor {
  /**
   Get database descriptor name as defined in DatabaseDescriptor.si.
      xml file.
  */
  public String getDatabaseName();

  /**
   Get description as per defined in DatabaseDescriptor.si.xml file.
  */
  public String getDescription();

  /**
   Get database implementer class name as per defined in
      DatabaseDescriptor.si.xml file.
  */
  public String getDatabaseImplementer();

  /**
   Get password.
  */
  public String getPassword();

  /**
   Check weather database implementer class defined in
      DatabaseDescritor.si.xml file or not.
  */
  public boolean isDatabaseImplementer();

  /**
   Check weather database needs to be stored on SDCard or not.
  */
  public boolean isExternalStorageEnable();

  /**
   Check weather database transactions to make multi−threading safe
      or not.
  */
  public boolean isLockingRequired();
```

```
/**
 Check weather database mapping object exists or not, based on
     table name.
*/
public boolean containsDatabaseMappingBasedOnTableName(final String
     tableName);





/**
 Check weather database mapping object exists or not, based on POJO
      class name.
*/
public boolean containsDatabaseMappingBasedOnClassName(final String
     className);

/**
 Get all database mapping paths as per defined in
     DatabaseDescriptor.si.xml file.
*/
public Iterator<String> getDatabaseMappingPaths();

/**
 Get all database mapping objects contained.
*/
public Iterator<DatabaseMappingDescriptor> getDatabaseMappings();

/**
 Get database mapping object based on table name.
*/
public DatabaseMappingDescriptor getDatabseMappingBasedOnTableName(
   final String tableName);

/**
 Get database mapping object based on POJO class name.
*/
public DatabaseMappingDescriptor getDatabseMappingBasedOnClassName(
   final String className);

/**
 Get database mapping object based on path.
*/
public DatabaseMappingDescriptor getDatabseMappingBasedOnPath(final
     String databaseMappingPath);

/**
```

```
 Get all database mapping objects in sorted order. The order will
    be as per defined in DatabaseDescriptor.si.xml file.
*/
public Iterator<DatabaseMappingDescriptor> orderedDatabaseMappings
   ();

/**
 Check weather library exists or not based on library name provided
    .
*/
public boolean containsLibraryBasedOnName(final String libraryName)
   ;

/**
 Check weather library exists or not based on library path provided
    .
*/
public boolean containsLibraryBasedOnPath(final String libraryPath)
   ;



/**
 Get all library paths as per defined in DatabaseDescriptor.si.xml
    file.
*/
public Iterator<String> getLibraryPaths();

/**
 Get all library descriptor paths contained.
*/
public Iterator<LibraryDescriptor> getLibraryDescriptors();

/**
 Get all library descriptor objects in sorted order as per defined
    in DatabaseDescriptor.si.xml file.
*/
public Iterator<LibraryDescriptor> orderedLibraryDescriptors();

/**
 Get library descriptor object based on library descriptor path.
*/
public LibraryDescriptor getLibraryDescriptorBasedOnPath(final
   String libraryPath);

/**
 Check weather library is needed by Database Descriptor or not.
*/
public boolean isLibrariesNeeded();
```

```
}
```

9.4.3 siminov.orm.model.LibraryDescriptor

```
/**
 Exposes methods to GET and SET Library Descriptor information as per
     define in LibraryDescriptor.si.xml file by application.
*/
public final class LibraryDescriptor {

  /**
   Get library name.
  */
  public String getName();

  /**
   Get descriptor as per defined in LibraryDescriptor.si.xml
  */
  public String getDescriptor();

  /**
   Check weather database mapping object exists or not, based on
       table name.
  */
  public boolean containsDatabaseMappingBasedOnTableName(final String
      tableName);

  /**
   Check weather database mapping object exists or not, based on POJO
       class name.
  */
  public boolean containsDatabaseMappingBasedOnClassName(final String
      className);

  /**
   Get all database mapping paths as per defined in
       DatabaseDescriptor.si.xml file.
  */
  public Iterator<String> getDatabaseMappingPaths();

  /**
   Get all database mapping objects contained.
  */
  public Iterator<DatabaseMappingDescriptor> getDatabseMappings();

  /**
   Get database mapping object based on table name.
  */
  public DatabaseMappingDescriptor getDatabseMappingBasedOnTableName(
     final String tableName);

  /**
   Get database mapping object based on POJO class name.
  */
  public DatabaseMappingDescriptor getDatabseMappingBasedOnClassName(
     final String className);


  /**
   Get database mapping object based on path.
  */
  public DatabaseMappingDescriptor getDatabseMappingBasedOnPath(final
      String libraryDatabaseMappingPath);

  /**
   Get all database mapping objects in sorted order. The order will
      be as per defined in DatabaseDescriptor.si.xml file.
  */
  public Iterator<DatabaseMappingDescriptor> orderedDatabaseMappings
     ();


}
```

9.4.4 siminov.orm.model.DatabaseMappingDescriptor

```
/**
Exposes methods to GET and SET Library Descriptor information as per
   define in   DatabaseDescriptor.si.xml or LibraryDescriptor.si.xml
    file by application.
*/
public final class DatabaseMappingDescriptor {

  /**
   Get table name.
  */
  public String getTableName();

  /**
   Get POJO class name.
  */
  public String getClassName();

  /**
   Check weather column exists based on column name.
  */
  public boolean containsColumnBasedOnColumnName(final String
     columnName);

  /**
   Check weather column exists based on variable name.
  */
  public boolean containsColumnBasedOnVariableName(final String
     variableName);

  /**
   Get column based on column name.
  */
  public Column getColumnBasedOnColumnName(final String columnName);

  /**
   Get column based on variable name.
  */
  public Column getColumnBasedOnVariableName(final String
     variableName);

  /**
   Get all column names.
  */
```

```
public Iterator<String> getColumnNames();

/**
 Get all columns.
*/
public Iterator<Column> getColumns();

/**
 Check weather index exists based in index name.
*/
public boolean containsIndex(final String indexName);



/**
 Get index object based on index name.
*/
public Index getIndex(final String indexName);

/**
 Get all index names.
*/
public Iterator<String> getIndexNames();

/**
 Get all indexs.
*/
public Iterator<Index> getIndexes();

/**
 Check weather reference object exists or not based on map to name.
*/
public boolean containsReferenceBasedOnMapTo(final String mapTo);

/**
 Get reference object based on map to.
*/
public Reference getReferenceBasedOnMapTo(final String mapTo);

/**
 Get all reference map to names.
*/
public Iterator<String> getReferenceMapToNames();

/**
 Get all reference objects.
*/
public Iterator<Reference> getReferences();
```

```
/**
 Check weather map object exists based on map name.
*/
public boolean containsMapBasedOnMap(final String map);

/**
 Check weather map object exists based on map to name.
*/
public boolean containsMapBasedOnMapTo(final String mapTo);

/**
 Get map object based on map name.
*/
public Map getMapBasedOnMap(final String map);



/**
 Get map object based on map to name.
*/
public Map getMapBasedOnMapTo(final String mapTo);

/**
 Get all map object map names.
*/
public Iterator<String> getMapMapNames();

/**
 Get all map object map names.
*/
public Iterator<String> getMapMapNames();

/**
 Get all map object map to names.
*/
public Iterator<String> getMapMapToNames();

/**
 Get all maps.
*/
public Iterator<Map> getMaps();



  /**
  Exposes methods to GET and SET Column information as per define
     in DatabaseMappingDescriptor.si.xml file by application.
  */
  public static final class Column {
```

```
/**
Get variable name.
*/
public String getVariableName();

/**
 Get column name.
*/
public String getColumnName();

/**
 Get type of column.
*/
public String getType();

/**
 Get POJO class column getter method name.
*/
public String getGetterMethodName();

/**
 Get POJO class column setter method name.
*/
public String getSetterMethodName();

/**
 Get default value of column.
*/
public String getDefaultValue();

/**
 Get check constraint of column.
*/
public String getCheck();

/**
 Check weather column is primary key.
*/
public boolean isPrimaryKey();

/**
 Check weather column is unique or not.
*/
public boolean isUnique();

/**
 Check weather column value can be not or not.
*/
```

```
    public boolean isNotNull();

  }

  /**
  Exposes methods to GET and SET Reference information as per
     define in DatabaseMappingDescriptor.si.xml file by application
     .
  */
  public static final class Reference {

    /**
     Get action needs to be performed when delete occur.
    */
    public String getOnDeleteAction();

    /**
     Get action needs to be performed when update occur.
    */
    public String getOnUpdateAction();

    /**
     Get map to name.
    */
    public String getMapTo();

    /**
     Check weather map object exists based on refer.
    */
    public boolean containsMapBasedOnRefer(final String refer);

    /**
     Check weather map object exists based on refer to name.
    */
    public boolean containsMapBasedOnReferTo(final String referTo);

    /**
     Get map object based on refer name.
    */
    public Map getMapBasedOnRefer(final String refer);

    /**
     Get map object based on refer to name.
    */
    public Map getMapBasedOnReferTo(final String referTo);

    /**
     Get all map refer names.
    */
```

```
      public Iterator<String> getMapReferNames();

      /**
       Get all map refer to names.
      */
      public Iterator<String> getMapReferToNames();

      /**
       Get all map objects.
      */
      public Iterator<Map> getMaps();

    }

    /**
     Exposes methods to GET and SET Reference Map information as per
        define in DatabaseMappingDescriptor.si.xml file by
        application.
    */
    public static final class Map {

      /**
       Get refer name.
      */
      public String getRefer();

      /**
       Get refer to name.
      */
      public String getReferTo();

    }
  }

    /**
     Exposes methods to GET and SET Reference Map information as per
        define in DatabaseMappingDescriptor.si.xml file by
        application.
    */
    public static final class Index {

      /**
       Get index name.
      */
      public String getName();

      /**
       Check weather index should be unique or not.
      */
    public boolean isUnique();

    /**
     Check weather index contain column or not.
    */
    public boolean containsColumn(final String column);

    /**
     Get all columns.
    */
    public Iterator<String> getColumns();

  }

  /**
   Exposes methods to GET and SET Map information as per define in
       DatabaseMappingDescriptor.si.xml file by application.
  */
  public static final class Map {

    /**
     Get map name.
    */
    public String getMap();

    /**
     Get map to name.
    */
    public String getMapTo();

    /**
     Get getter map method name.
    */
    public String getGetterMapMethodName();

    /**
     Get setter map method name.
    */
    public String getSetterMapMethodName();

    /**
     Get relationship type name.
    */
    public String getRelationshipType();

    /**
     Check weather value is load.
    */
    public boolean isLoad();
```

```
        }

}
```

9.4.5 siminov.orm.annotation.Table

```
/**
 Exposes methods to GET table name as per in POJO Class Annotation by
      application.
 */
public @interface Table {

  /**
   Get table name.
  */
  public String tableName();

}
```

9.4.6 siminov.orm.annotation.Column

```
/**
 Exposes methods to GET column names and its properties as per in
    POJO Class Annotation by application.
 */
public @interface Column {

  /**
   Get column name.
  */
  public String columnName();

  /**
   Get column properties.
  */
  public Property[] properties() default;

}
```

9.4.7 siminov.orm.annotation.Property

```
/**
 Exposes methods to GET properties as per in POJO Class Annotation by
      application.
 */
public @interface Property {

  /**
   Get property name.
  */
  public String name();

  /**
   Get property value.
  */
  public String value();

}
```

9.4.8 siminov.orm.annotation.Index

```
/**
 Exposes methods to GET index properties as per in POJO Class
     Annotation by application.
 */
public @interface Index {

  /**
   Get index name.
  */
  public String name();

  /**
   Get index columns.
  */
  public IndexColumn[] value();

  /**
   Check if index is unique.
  */
  public boolean unique();

}
```

9.4.9 siminov.orm.annotation.IndexColumn

```
/**
 Exposes methods to GET index column as per in POJO Class Annotation
    by application.
 */
public @interface IndexColumn {

  /**
   Get column.
  */
  public String column();

}
```

9.4.10 siminov.orm.annotation.Indexes

```
/**
 Exposes methods to GET indexes properties as per in POJO Class
    Annotation by application.
 */
public @interface Indexes {

  /**
   Get index's.
  */
  public Index[] value();

}
```

9.4.11 siminov.orm.annotation.Reference

```
/**
 Exposes methods to GET Reference properties as per in POJO Class
     Annotation by application.
 */
public @interface Reference {

  /**
   Get map to.
  */
  public String mapTo();

  /**
   Get refer to.
  */
  public String referTo();

  /**
   Get on update.
  */
  public String onUpdate();

  /**
   Get on delete.
  */
  public String onDelete();

}
```

9.4.12 siminov.orm.annotation.Map

```
/**
 Exposes methods to GET Map properties as per in POJO Class
     Annotation by application.
 */
public @interface Map {

  /**
   Get map to name.
  */
  public String mapTo();

  /**
   Get properties.
  */
  public Property[] properties() default;

}
```

9.5 Parsers Layer

It contain parses which parses all descriptor de ned by application.

9.5.1 siminov.orm.parsers.ApplicationDescriptorParser

It is used to parse ApplicationDescriptor.si.xml les de ned by application.

9.5.2 siminov.orm.parsers.DatabaseDescriptorParser

It is used to parse DatabaseDescriptor.si.xml les de ned by application.

9.5.3 siminov.orm.parsers.LibraryDescriptorParser

It is used to parse LibraryDescriptor.si.xml les de ned by application.

9.5.4 siminov.orm.parsers.DatabaseMappingDescriptor

It is used to parse DatabaseMappingDescriptor.si.xml les de ned by application.

9.6 Utils Layer

It provides util classes which provides additional be nites.

9.6.1 siminov.orm.utils.Utils

```
/**
 Exposes utility methods which can be used by both SIMINOV/
    Application.
 */
public class Utils {

  /**
   Get string from input stream.
  */
  public static final String getString(final InputStream inputStream)
      throws SiminovException;


  /**
   Get input stream from string.
  */
  public static final InputStream getInputStream(final String string)
      throws SiminovException;

  /**
   Get input stream from string.
  */
  public static final InputStream getInputStream(final byte[] bytes)
     throws SiminovException;

  /**
   Get generated unique id.
  */
  public static final Long generateUniqueId();

  /**
   Check weather application is running on emulator or not.
  */
  public static boolean isEmulator();

  /**
   Check weather device have network coverage or not.
  */
  public static boolean hasCoverage();
}
```

Chapter 10

Exceptions

10.1 Siminov Exception

```
public class SiminovException extends Exception { }
```

This is general exception, which is thrown through Siminov API, if any exception occur while performing any tasks.

10.2 Deployment Exception

```
public class DeploymentException extends RuntimeException { }
```

This is runtime-time exception, which is thrown if any exception occur at time of initialization of Siminov.

10.3 Database Exception

```
public class DatabaseException extends Exception { }
```

This is general exception, which is thrown through Siminov database API's, if any exception occur which doing any database operations.