Siminov Framework Version 0.9-Beta Developer Guide

Siminov Framework
http://www.siminov.github.com/android-hybrid

Contents

Preface

A hybrid application, by definition is derived from a combination of technologies, approaches or elements of different kinds. With respect to mobile applications, a hybrid application leverages best of both native and mobile web technologies.

In hybrid environment, it is very difficult to map JavaScript/Java objects to relational database, but Siminov makes application developer life easy and simiple by mapping JavaScript/Java objects to relational database.

Note

While having a strong background in SQL is not required to use Android-Siminov Framework, 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 JavsScript/Java classes to database tables (and from JavaScript/Java data types to SQL data types), but also provides data query and retrieval facilities. It can significantly reduce development time otherwise spent with manual data handling in SQLite. Siminov design goal is to relieve the developer from 99% of common data persistence-related programming tasks by eliminating the need for manual, hand-crafted data processing using SQLite. However, unlike many other persistence solutions, Siminov does not hide the power of SQLite from you and guarantees that your investment in relational technology and knowledge is as valid as always.

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 Hybrid object-oriented domain models and business logic in the JavaScript/Java based. However, Siminov can certainly help you to remove or encapsulate vendor-specific 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 find. Try your hand at fixing some bugs or implementing enhancements. 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 benefits of Siminov Framework.

Siminov Native ORM

1. **Web Site**: `https://siminov.github.io.com/android-orm`
2. **Github Website**: `https://github.com/Siminov/android-orm`
3. **Wiki**: `https://github.com/Siminov/android-orm/wiki`
4. **Issue Tracker**: `https://github.com/Siminov/android-orm/issues`

Siminov Hybrid ORM Framework

1. **Web Site**: `https://siminov.github.io.com/android-hybrid`
2. **Github Website**: `https://github.com/Siminov/android-hybrid`
3. **Wiki**: `https://github.com/Siminov/android-hybrid/wiki`
4. **Issue Tracker**: `https://github.com/Siminov/android-hybrid/issues`

0.2 Getting Started Guide

New users may want to first 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

About Siminov

Siminov is a open source framework build and managed by Siminov Software Solution LLP and its community. It is a free software that is distributed under the Apache License, Version 2.0 (the "License"). Aim of this framework is to reduce time and effort spend in building mobile application on various platforms like (Android, iOS, Blackberry, Windows).

Siminov Framework is build by combining various Siminov products like (Siminov ORM , Siminov Hybrid, Siminov Konnect). These components are available or in development stage on all platforms.

1.1 Siminov Android ORM

Siminov Android ORM is an object-relational mapping (ORM) library for the Native Android Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. It primary maps Android Java classes to database tables (and from Java data types to SQLite data types.). It also provides data query and retrieval facilities. It also generates the SQLite calls and attempts to relieve the developer from manual result set handling and object conversion and keep the application protable to all supported SQLite databases with little performance overhead.

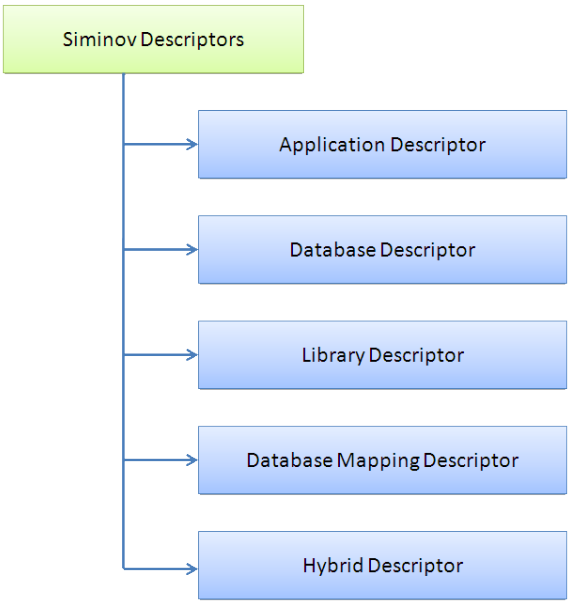
1.2 Siminvo Hybrid

Siminov Hybrid is build using Siminov Android ORM Framework and Siminov JavaScript Framework, together it enables application to map JavaScript/Java object to relational database. Using this application developer can build application which uses the best of both Native and Web technology.

Siminov Hybrid not only take care of mapping from JavaScript/Java classes to database tables (and from JavaScript/Java data types to SQL data types), but also provides data query and retrieval facilities.

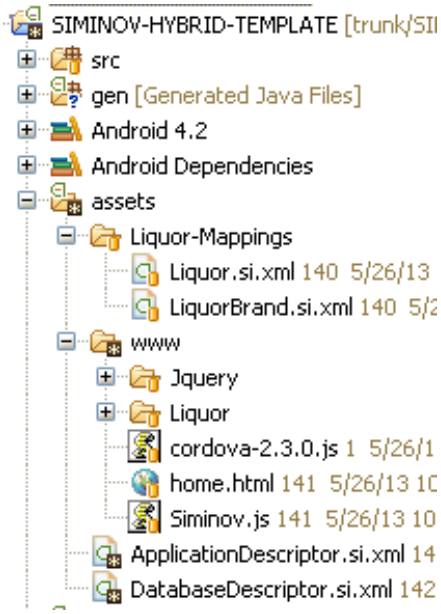
Chapter 2

Configuration



Siminov Framework works based on a set of defined descriptors which can be broadly classified as **ApplicationDescriptor.si.xml**, **DatabaseDescriptor.si.xml**, **LibraryDescriptor.si.xml**, **DatabaseMappingDescriptor.si.xml**, **HybridDescriptor.si.xml**

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



Note

1. All descriptor file name should end with **.si.xml**.

2.1 Application Descriptor (ApplicationDescriptor.si.xml) Configuration

Application Descriptor is the one who connects application to Siminov framework. It provide basic information about application, which defines 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 -->
  <database-descriptors>
    <database-descriptor>full_path_of_database_descriptor_file</
        database-descriptor>
  </database-descriptors>



  <!-- Event Handlers Implemented By Application (zero-to-many) -->
    <!-- Optional Field's -->
  <event-handlers>
    <event-handler>full_java_class_path_of_event_handler/
        javascript_class_path_of_event_handler (ISiminovHandler/
        IDatabaseHandler)</event-handler>
  </event-handlers>

</siminov>
```

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

```
                <!-- Example: ApplicationDescriptor.si.xml Of Siminov
                     Hybrid Template -->

<siminov>

  <property name="name">SIMINOV HYBRID TEMPLATE</property>
  <property name="description">Siminov Hybrid 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.hybrid.template.events.
          SiminovEventHandler</event-handler>
      <event-handler>siminov.hybrid.template.events.
          DatabaseEventHandler</event-handler>
  </event-handlers>

</siminov>
```

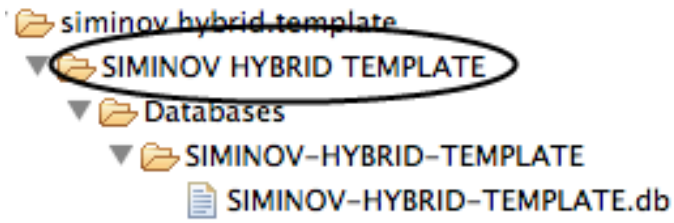
Note

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 field. If any resources is created by Siminov then it will be under this folder name.



   (b) **description**: Description of application. It is optional field.
   (c) **version**: Version of application. It is mandatory field. 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 field. 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**, **firstTimeSiminovInitialized**, **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.

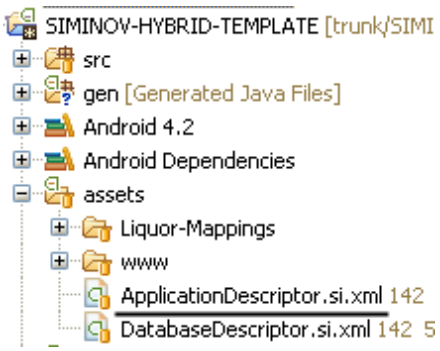
Note

(a) Event Handler can be define in both Native and Web.
(b) Native Event Handler: If you want to handle Event in Native then define Event Handler using Java. Specify full Java class path and name in ApplicationDescriptor.si.xml.
(c) Web Event Handler: If you want to handle Event in Web then define Event Handler using JavaScript. Specify only JavaScript Function name in ApplicationDescriptor.si.xml.
(d) Both (Native/Web) Event Handler: If you want to handle Event in both Native and Web then define Event Handler in both Java and JavaScript. Specify full Java class path and name in ApplicationDescriptor.si.xml, no need to define for JavaScript because Siminov will automatically assume same name for it.

Note

1. Application descriptor file 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.



2.2 Database Descriptor (DatabaseDescriptor.si.xml) Configuration

Database Descriptor is the one who defines 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-HYBRID-TEMPLATE</property>
  <property name="description">Siminov Hybrid Template Database
      Config</property>
  <property name="is_locking_required">true</property>
  <property name="external_storage">false</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 field. All database files (.db)'s will be placed under the this folder name.

   (b) **type**: It defines the type of database. It is optional field. Default is sqlite.
   (c) **description**: Description of database. It is optional field.
   (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 field.
   (e) **external_storage**:It specifies whether database resources needs to be saved on external storage or not (SDCard). It is optinal field. Default is false.

> **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.

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

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

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

Note

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

Note

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

Example: DatabaseDescriptor.si.xml file path

2.3 Library Descriptor (LibraryDescriptor.si.xml) Configuration

Library Descriptor is the one who defines 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>

  <!-- Hybrid Adapters Needed Under This Library Descriptor -->

  <!-- Optional Field -->
    <!-- Hybrid Adapters -->
  <adapters>
      <adapter path="full_path_of_hybrid_adapter_file"></adapter>
  </adapters>

</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 field.
   (b) **description**: Description of library. It is optional field.
2. Database Mapping Paths Needed Under This Database Descriptor.

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

3. Hybrid Adapter Paths Needed Under This Database Descriptor.

> **Note**
>
> (a) Provide full hybrid adapter file path.

> **Note**
>
> 1. Library descriptor file name should be same as LibraryDescriptor.si.xml.
> 2. It should always be in root package specified in DatabaseDescriptor.si.xml file.

Example: LibraryDescriptor.si.xml file path

```
<database-descriptor>

    <property name="database_name">SIMINOV-HYBRID-TEMPLATE</property>
    <property name="description">Siminov Hybrid Template Database Config</property>
    <property name="is_locking_required">true</property>
    <property name="external_storage">false</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) Configuration

Database Mapping Descriptor is one which does ORM, it maps Java/JavaScript 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_java_class_name/mapped_pojo_javascript_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">java_variable_data_type/
          javascript_variable_data_type</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_java_class_name/
      map_to_pojo_javascript_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_java_class_name/
          map_to_pojo_javascript_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_java_class_name/
          map_to_pojo_javascript_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_java_class_name/
          map_to_pojo_javascript_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 Hybrid Template.

```
<database-mapping>

  <table table_name="LIQUOR" class_name="Liquor">
```

```
    <column variable_name="liquorType" column_name="LIQUOR_TYPE">
      <property name="type">String</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">String</property>
    </column>

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

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

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

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

    <relationships>

        <one-to-many refer="liquorBrands" refer_to="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 field.
   (b) **class_name***: Name of Java/JavaScript POJO class name which is to be mapped to table name. It is mandatory field.

Note

i. POJO Class can be define in both Native and Web.
ii. Native Class: If you mapped POJO class is in Native then use Java to define class. Specify full Java class path and name in class_name TAG.
iii. Web Class: If you have mapped POJO class is in Web then use JavaScript to define class. Specify only JavaScript Function name in class_name TAG.
iv. Both (Native/Web) Class: If you mapped POJO class is in both Native and Web then both Java and JavaScript to define classes. Specify full Java class path and name in class_name TAG, no need to define for JavaScript because Siminov will automatically assume same name for it.

2. **COLUMN TAG**: It map database table column to its corresponding variable of POJO class.
   (a) **column_name***: Name of column. It is mandatory field.
   (b) **variable_name***: Name of variable. It is mandatory field.

   Properties Of Column Tag

   (a) **type***: Variable data type. It is mandatory property.

Note

i. POJO Class can be define in both Native and Web.
ii. Native Class: If you have mapped POJO class in Native then specify Java Variable data type.
    Java Data Type's
    A. java.lang.int
    B. java.lang.Integer
    C. java.lang.long
    D. java.lang.Long
    E. java.lang.float
    F. java.lang.Float
    G. java.lang.boolean
    H. java.lang.Boolean
    I. java.lang.char
    J. java.lang.Character
    K. java.lang.String
    L. java.lang.byte
    M. java.lang.Byte
    N. java.lang.void
    O. java.lang.Void
    P. java.lang.short
    Q. java.lang.Short
iii. Web Class: If you have mapped POJO class in Web then specify JavaScript Variable data type.
    JavaScript Data Type's
    A. String
    B. Number
    C. Boolean
    D. Array
    E. Object
    F. Null
    G. Undefined
iv. Native and Web Class: If your mapped POJO class is in both Native and Web then only specify Java Variable data type, for JavaScript Siminov will take care.

For more details see Data Type section of this developer guide.

(b) **primary_key**: *TRUE/FALSE*. It defines wheather the column is primary key of table or not. It is optional property. Default value is false.

(c) **not_null**: *TRUE/FALSE*. It defines wheather the column value can be empty or not. It is optional property. Default value is false.

(d) **unique**: *TRUE/FALSE*. It defines wheather the column value should be unique or not. It is optional property. Default value is false.

(e) **default**: It define 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 defines the stucture of index needed on the table.

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

   (b) **unique**: *TRUE/FALSE*. It defines 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 defines 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 effectively 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 field.

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

Note

i. Class can be define in both Native and Web.
ii. Native Class: If you have mapped POJO class in Native then specify Java class path and name in refer_to TAG.
iii. Web Class: If you have mapped POJO class in Web then specify only JavaScript Function name in refer_to TAG.
iv. Both (Native/Web) Class: If you have mapped POJO class in both Native and Web then specify full Java class path and name in refer_to TAG, no need to define for JavaScript because Siminov will automatically assume same name for it.

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

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

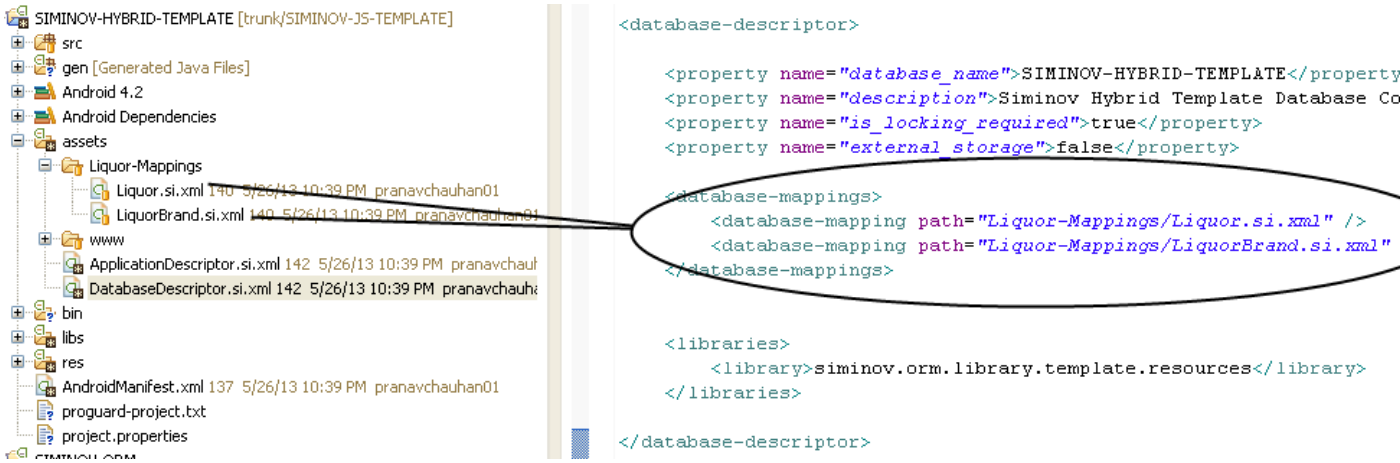
Relationship Properties

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

Note

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

Example: DatabaseMappingDescriptor.si.xml file path



2.5 Database Mapping Descriptor Through Annotation

Annotation is another way to map Java POJO class to database table.

(Annotations provide data about a program that is not part of the program itself. They have no direct effect 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;

}
```

Note

You can only use Annotation with Native Java, it is not supported with Web JavaScript.

Different Annotation Tags:

1. **@Table**: It map database table to its corresponding POJO class.
   (a) **tableName***: Name of table. It is mandatory field.
2. **@Indexes**: It contain all index required on table.
3. **@Index**: It defines the stucture of index needed on the table.
   (a) **name***: Name of the index. It is mandatory field.
   (b) **unique**: *TRUE/FALSE*. It defines 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 field.

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

   Properties Of Column Tag: @Property: This tag defines a perticular property of column.
   (a) **primary_key — ColumnProperty.PRIMARY_KEY**: *TRUE/FALSE*. It defines 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 defines wheather the column value can be empty or not. It is optional property. Default value is false.

(c) **unique** — **ColumnProperty**.**UNIQUE**: *TRUE/FALSE*. It defines wheather the column value should be unique or not. It is optional property. Default value is false.

(d) **default** — **ColumnProperty**.**DEFAULT**: It define 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 defines 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 defines action needs to performed, when update occur.

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

6. **@OneToMany**: This tag defines one-to-many relationship, where each row in the related to table can be related to many rows in the relating table. This effectively 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 defines action needs to performed, when update occur.

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

7. **@ManyToOne**: This tag defines 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 defines action needs to performed, when update occur.

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

8. **@ManyToMany**: This tag defines 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 defines action needs to performed, when update occur.

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

RELATIONSHIP PROPERTY TAG: @RelationshipProperty

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

2.6 Hybrid Descriptor (HybridDescriptor.si.xml) Configuration

Hybrid Descriptor is one which describes properties required to map Web to Native and vice-versa. It is optional descriptor.

```
            <!-- Design of HybridDescriptor.si.xml -->

<hybird-descriptor>

  <!-- Adapters -->
    <adapters>

      <!-- Adapter -->
      <adapter>

          <!-- General Adapter Properties -->
            <!-- Mandatory Field -->
          <property name="name">adapter_name</property>

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

            <!-- Mandatory Field -->
          <property name="type">WEB-TO-NATIVE|NATIVE-TO-WEB</property
              >

            <!-- Optional Field -->
          <property name="map_to">name_of_adapter_class</property>

            <!-- Optional Field (DEFAULT: FALSE)-->
          <property name="cache">true/false</property>

          <!-- Handlers -->
            <!-- Handler -->
          <handlers>

            <handler>

                <!-- General Handler Properties -->
                  <!-- Mandatory Field -->
                <property name="name">handler_name</property>

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

                  <!-- Mandatory Field -->
```

```
            <property name="map_to">name_of_handler_method</
                property>


            <!-- Parameters -->
            <parameters>

                <!-- Parameter -->
                <parameter>

                      <!-- Mandatory Field -->
                    <property name="name">name_of_parameter</
                        property>

                      <!-- Mandatory Field -->
                    <property name="type">parameter_type</
                        property>

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

                </parameter>

            </parameters>

            <return>

                  <!-- Mandatory Field -->
                <property name="type">return_type</property>

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

            </return>

        </handler>

    </handlers>

  </adapter>


    <!-- Adapter Paths -->
  <adapter path="adapter_path" />

</adapters>
```

```
    <!-- Library Needed Under This HybridDescriptor -->
    <libraries>

        <library>full_path_of_library_descriptor_file</library>

    </libraries>

</hybird-descriptor>
```

Example: SiminovAdapter.si.xml File Of Siminov Hybrid Framework.

```
<adapters>

  <adapter>

      <property name="name">SIMINOV</property>
      <property name="description">Siminov Hybrid Handler</property>
      <property name="type">WEB-TO-NATIVE</property>
      <property name="map_to">siminov.hybrid.adapter.handlers.
          SiminovHandler</property>

      <handlers>

          <handler>

              <property name="name">INITIALIZE-SIMINOV</property>
              <property name="map_to">initializeSiminov</property>
              <property name="description">Initialize Siminov</
                  property>
        <property name="type">SYNC</property>

          </handler>

      </handlers>

  </adapter>


  <adapter>

      <property name="name">SIMINOV-NATIVE-TO-WEB</property>
      <property name="description">Siminov Hybrid Handler</property>
      <property name="type">NATIVE-TO-WEB</property>
      <property name="map_to">Adapter</property>

      <handlers>
```

```
        <handler>

            <property name="name">HANDLE_NATIVE_TO_WEB</property>
            <property name="type">SYNC</property>
            <property name="map_to">handle</property>
            <property name="description">Handle Call From Native To
                Hybrid</property>

            <parameters>

                <parameter>

                    <property name="type">java.lang.String</
                        property>
                    <property name="description">Action</property>

                </parameter>

                <parameter>

                    <property name="type">java.lang.String</
                        property>
                    <property name="description">Data</property>

                </parameter>

          </parameters>

        </handler>

    </handlers>

</adapter>


<adapter>

    <property name="name">SIMINOV_EXCEPTION_HANDLER</property>
    <property name="description">Siminov Exception Handler</
        property>
    <property name="type">NATIVE_TO_WEB</property>
    <property name="map_to">SiminovExcpetionHandler</property>

    <handlers>

        <handler>

            <property name="name">DISPLAY</property>
```

```
            <property name="map_to">display</property>
            <property name="description">Display Exception</
                property>

            <parameters>

                <parameter>

                    <property name="type">java.lang.String</
                        property>
                    <property name="description">Data</property>

                </parameter>

            </parameters>

        </handler>


    </handlers>

  </adapter>

</adapters>
```

Hybrid Descriptor Elements:

1. **Adapter TAG**: Adapter allows Web and Native to work together that is normally not possible because of incompatible Technologies. Adapter basically maps JavaScript to Native and vice-versa.
   (a) **name*** : Name of Adapter. It is mandatory field.
   (b) **description**: Description about Adapter. It is optional field.
   (c) **type**: Type Of Adapter. It is mandatory field.
      i. **WEB-TO-NATIVE**: It says this adapter maps JavaScript functions to Native functions.
      ii. **NATIVE-TO-WEB**: It says this adapter maps Native functions to JavaScript functions.
   (d) **map_to**: Name of Class (Web/Native) mapped to this adapter. It is not mandatory field.
   (e) **cache**: true/false: It says that adapter mapped to Class needs to be cached or not. It is optional field. Default is false.

(f) **Handler**: Handler is one which handle request from WEB-TO-NATIVE or NATIVE-TO-WEB.

i. **name***: Name of Handler. It is mandatory field.

ii. **description**: Description about Handler. It is optional field.

iii. **map_to***: Name of Handler function which handles request from WEB-TO-NATIVE or NATIVE-TO-WEB.

i. **Parameter**: Parameters are bacially arguments passed to handler.

A. **name*:** Name of Parameter. It is mandatory field.

B. **description**: Description about Parameter. It is optional field.

C. **type***: Type of Parameter. It is mandatory field.

ii. **Return**: Return defines about data returned from handler.

A. **type***: Type of Returned Data. It is mandatory field.

B. **description**: Description about Return Data. It is optional field.

Note: Adapter can be define in HybridDescriptor.si.xml file or can be defined in seprate xml file.

i. If you define Adapter in HybridDescriptor.si.xml file then define it in adapters TAG.
ii. If you Adapter in seprate xml file then specify Adapter file path in HybridDescriptor.si.xml file.

2. **Libraries TAG**: Library Descriptor Paths Needed Under This Database Descriptor.

(a) Provide full package name under which LibraryDescriptor.si.xml file is placed.

Chapter 3

Siminov Initialization

3.1 Siminov Native ORM Framework

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

> **Note**
>
> Use Siminov Native ORM only if you are buiding Android Native based application.

There are two ways to initialize Siminov Native ORM

1. **Initializing Siminov Native ORM 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 Native ORM 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 Native ORM 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 Native ORM is initialized 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 Native ORM Framework

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

   Siminov Native ORM 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 Native ORM will create database required by the application.

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

2. **Initialize Resources Layer** : Any resource created by Siminov Native ORM Framework is places in resource layer of Siminov Native ORM. 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<String> getLibraryPaths();

public Iterator<String>
    getLibraryPathsBasedOnDatabaseDescriptorName(final String
    databaseDescriptorName);

public Iterator<LibraryDescriptor> getLibraries();

public Iterator<LibraryDescriptor>
    getLibrariesBasedOnDatabaseDescriptorName(final String
    databaseDescriptorName);

public Iterator<DatabaseMappingDescriptor>
    getLibraryDatabaseMappingsBasedOnLibraryDescriptorPath(
    final String libraryPath);

public IDatabase getDatabaseBasedOnDatabaseDescriptorName(
    final String databaseName);

public IDatabase getDatabaseBasedOnDatabaseMappingPojoClass(
    final Class<?> classObject);

public IDatabase getDatabaseBasedOnDatabaseMappingClassName(
    final String databaseMappingClassName);

public IDatabase getDatabaseBasedOnDatabaseMappingTableName(
    final String databaseMappingTableName);

public Iterator<IDatabase> getDatabases();
```

```
    public ISiminovEvents getCoreEventHandler();

    public IDatabaseEvents getDatabaseEventHandler();

}
```

3.2 Siminov Hybrid Framework

Hybrid application is a combination of Native and Web. Siminov Hybrid Framework is build over Siminov Native ORM, therefore it provides both Web ORM as well as Native ORM features.

> **Note**
> If you are using Siminov Hybrid Framework then you don't have to explicitly initialize Siminov Native ORM because Hybrid will take care of that.

Initializing Siminov Native Framework is a two way of initialization through Native and Web.

1. **Intialize Siminov Hybrid From Native When Application Starts**

   Initialize Siminov Hybrid Framework by invoking initialize API of siminov.hybrid.Siminov class and provide ApplicationContext and WebView as a parameter to API. Internally Siminov Hybrid Framework will automatically initialize Siminov Native ORM.

```
public class Siminov extends DroidGap {

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        super.init();
        super.appView.getSettings().setJavaScriptEnabled(true);

        initializeSiminov();

        super.loadUrl("file:///android_asset/www/home.html");

    }


    private void initializeSiminov() {
```

```
    IInitializer initializer = siminov.hybrid.Siminov.initialize
        ();

    initializer.addParameter(getApplicationContext());
    initializer.addParameter(super.appView);
    initializer.addParameter((Activity) this);

    initializer.start();

  }

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

    System.exit(RESULT_OK);
  }

}
```

2. **Initialize Siminov Hybrid From Web When Device Get Ready**

   Intialize Siminov Hybrid Framework by invoking initialize API of Siminov function, once Siminov gets initialized you will get a call back in firstTimeSiminovInitialized/siminovInitialized Event Handler.

```
  //Register Siminov.initialize API For Device Ready Event.
  document.addEventListener("deviceready", Siminov.initialize,
      false);


  //Application Defined Event Handler. Once Siminov gets
      initialize this Event Handler will be invoked.
  function SiminovEventHandler() {

    this.firstTimeSiminovInitialized = function() {
      //Siminov Initialized.
    }

    this.siminovInitialized = function() {
      //Siminov 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();

}
```

Note

(a) If you use Siminov Hybrid Framework it is mandatory to register for Events.
(b) If you use only Siminov Native ORM it is not mandatory to register for Events.

3.3 Lazy Initialization VS Initial Initialization

Siminov Framework provides easy configuration property in ApplicationDescriptor.si.xml file through which application developer can significantly improve the performance of Siminov.

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

   Example: Siminov Template Application Example.

   Basically to map entity and database table, Siminov need DatabaseMappingDescriptor object which defines relation between entity and database table, this object is created by reading DatabaseMappingDescriptor.si.xml file defined 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 itself 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.

```
<siminov>

    <property name="name">SIMINOV HYBRID TEMPLATE</property>
    <property name="description">Siminov Hybrid 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.hybrid.template.events.SiminovEventHandler</event-handler>
        <event-handler>siminov.hybrid.template.events.DatabaseEventHandler</event-handler>
    </event-handlers>

</siminov>
```

3.4 Handling Multiple Schema's

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

Example: Siminov Template Application Example.

Chapter 4

Event Notifiers

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

Note

1. If you use Siminov Hybrid Framework it is mandatory to register for Events.
2. If you use only Siminov Native ORM Framework it is not mandatory to register for events.
3. Event Handler can be define in both Native and Web.
4. Native Event Handler: If you want to handle Event in Native then define Event Handler using Java. Specify full Java class path and name in ApplicationDescriptor.si.xml.
5. Web Event Handler: If you want to handle Event in Web then define Event Handler using JavaScript. Specify only JavaScript Function name in ApplicationDescriptor.si.xml.
6. Both (Native/Web) Event Handler: If you want to handle Event in both Native and Web then define Event Handler in both Java and JavaScript. Specify full Java class path and name in ApplicationDescriptor.si.xml, no need to define for JavaScript because Siminov will automatically accume same name for it.

4.1 ISiminov Events

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

Example: ISiminov Events Notifier

```
public interface ISiminovEvents {

  public void firstTimeSiminovInitialized();

  public void siminovInitialized();

  public void siminovStopped();

}
```

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

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

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

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

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

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

i.ISiminov Native Event Handler

```
public class SiminovEventHandler implements ISiminovEvents {

  public void firstTimeSiminovInitialized() {
    //First Time Siminov Initialized.
  }

  public void siminovInitialized() {
    //Siminov Initialized.
  }

  public void siminovStopped() {
    //Siminov Stopped.
  }

}
```

ii. ISiminov Web Event Handler

```
function SiminovEventHandler() {

    this.firstTimeSiminovInitialized = function() {
    //First Time Siminov Initialized.
    }


    this.siminovInitialized = function() {
    //Siminov Initialized.
    }


    this.siminovStopped = function() {
        //Siminov Stopped.
    }

}
```

4.2 IDatabaseEvents

It provide API's related to database operations.

Example: IDatabaseEvents Notifier

```
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 defined in DatabaseDescriptor.si.xml file. 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 defines 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 defines table and index for which index is dropped.

i.IDatabase Native Event Handler

```
public class DatabaseEventHandler implements IDatabaseEvents {

  public void databaseCreated(DatabaseDescriptor databaseDescriptor)
      {
    //Database Created.
  }

  public void databaseDropped(DatabaseDescriptor databaseDescriptor)
      {
    //Database Dropped.
  }

  public void tableCreated(DatabaseDescriptor databaseDescriptor,
      DatabaseMappingDescriptor databaseMapping) {
    //Table Created.
  }

  public void tableDropped(DatabaseDescriptor databaseDescriptor,
      DatabaseMappingDescriptor databaseMapping) {
    //Table Dropped.
  }

  public void indexCreated(DatabaseDescriptor databaseDescriptor,
      DatabaseMappingDescriptor databaseMapping, Index index) {
    //Index Created.
  }

  public void indexDropped(DatabaseDescriptor databaseDescriptor,
      DatabaseMappingDescriptor databaseMapping, Index index) {
    //Index Dropped.
  }

}
```

ii. IDatabase Web Event Handler

```
function DatabaseEventHandler() {

    this.databaseCreated = function(databaseDescriptor) {
```

```
    //Database Created.
    }


    this.databaseDropped = function(databaseDescriptor) {
    //Database Dropped.
    }


    this.tableCreated = function(databaseDescriptor,
        databaseMappingDescriptor) {
    //Table Created.
    }


    this.tableDropped = function(databaseDescriptor,
        databaseMappingDescriptor) {
    //Table Dropped.
    }


    this.indexCreated = function(databaseDescriptor,
        databaseMappingDescriptor, index) {
    //Index Created.
    }


    this.indexDropped = function(databaseDescriptor,
        databaseMappingDescriptor, index) {
    //Index Dropped.
    }

}
```

Chapter 5

Database API's

5.1 Data Types

Based on Java/JavaScript variable type Siminov decides the data type of column.

Note

1. POJO Class can be define in both Native and Web.
2. Native Class: If you have mapped POJO class in Native then specify Java Variable data type.

   Java Data Type's

   java.lang.int, java.lang.Integer, java.lang.long, java.lang.Long, java.lang.float, java.lang.Float, java.lang.boolean, java.lang.Boolean, java.lang.char, java.lang.Character, java.lang.String, java.lang.byte, java.lang.Byte, java.lang.void, java.lang.Void, java.lang.short, java.lang.Short.
3. Web Class: If you have mapped POJO class in Web then specify JavaScript Variable data type.

   JavaScript Data Type's

   String, Number, Boolean, Array.
4. Native and Web Class: If your mapped POJO class is in both Native and Web then only specify Java Variable data type, for JavaScript Siminov will take care.

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">java.lang.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">java.lang.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. **float**: Java float primitive data type is converted to **REAL** Sqlite data type.

   Example: Java float Primitive Data Type

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

18. **String**: JavaScript String data type is converted to **TEXT** Sqlite data type.

    Example: JavaScript String Data Type

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

19. **Number**: JavaScript String data type is converted to **REAL** Sqlite data type.

    Example: JavaScript Number Data Type

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

20. **Boolean**: JavaScript String data type is converted to **NUMERIC** Sqlite data type.

    Example: JavaScript Boolean Data Type

```
<column variable_name="name-of-variable" column_name="name-
    of-column">
  <property name="type">Boolean</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*, *DOUBLxE 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 define ORM using Annotation then you dont have to specify data type, it will automatically be configured based on variable data type.

5.2 Database API's

5.2.1 Create Database

Siminov provides APIs to create database, based on schema defined in DatabaseDescriptor.si.xml file.

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 these APIs you can create table in database.

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

There are three ways to create table in database.

1. **Defining table structure using DatabaseMappingDescriptor.si.xml file**

   Example: Liquor.si.xml file 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. **Defining 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) {
  //Log It.
}
```

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();

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

5.2.5 Create Index

There are two ways to create index.

1. **Define Index Structure Using DatabaseMappingDescriptor.si.xml file**

```
<database-mapping>

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

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

  </table>

</database-mapping>
```

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

1. **Select API - Native**

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

   (a)

```
public ISelect select() throws DatabaseException;
```

   Select tuples from table.

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

```
public interface ISelect {

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


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

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

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

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

  /**
   * Used to specify OR condition between where clause.
```

```
     * @param column Name of column on which condition need to
         be specified.
     * @return ISelectClause Interface.
     */
    public ISelectClause 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 ISelect Interface.
     */
    public ISelect 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 ISelect Interface.
     */
    public ISelect 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 ISelect Interface.
     */
    public ISelect descendingOrderBy(String...columns);

    /**
     * Used to specify the range of data need to fetch from
        table.
     * @param limit LIMIT of data.
     * @return ISelect Interface.
     */
    public ISelect 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 ISelect Interface.
     */
    public ISelect 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 ISelectClause Interface.
   */
  public ISelectClause having(String column);

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

  /**
   * Used to provide name of columns only for which data
     will be fetched.
   * @param column Name of columns.
   * @return ISelect Interface.
   */
  public ISelect 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 ISelectClause {

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


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

```
 */
public ISelect equalTo(String value);

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

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

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

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

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

/**
 * Used to specify BETWEEN condition.
 * @param start Start Range.
 * @param end End Range.
 * @return ISelect Interface.
 */
```

```
  public ISelect between(String start, String end);

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

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

}
```

Example: Select Liquor Where Type Equal To RUM

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

(b)

```
  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.
  }
```

2. **Select API - Web**

Database function provides following API's to select tuples from table.

(a)

```
this.select = function();
```

Select tuples from table.

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

```
function ISelect(select) {
    return {
        interfaceName : "ISelect",
        distinct : select.distinct,
        where : select.where,
        whereClause : select.whereClause,
        and : select.and,
        or : select.or,
        orderBy : select.orderBy,
        ascendingOrderBy : select.ascendingOrderBy,
        descendingOrderBy : select.descendingOrderBy,
        limit : select.limit,
        groupBy : select.groupBy,
        having : select.having,
        havingClause : select.havingClause,
        columns : select.columns,
        fetch : select.fetch
    }
}
```

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

```
function ISelectClause(clause) {

    return {

        interfaceName : "ISelectClause",

        equalTo : clause.equalTo,

        notEqualTo : clause.notEqualTo,

        greaterThan : clause.greaterThan,

        greaterThanEqual : clause.greaterThanEqual,

        lessThan : clause.lessThan,

        lessThanEqual : clause.lessThanEqual,

        between : clause.between,

        like : clause.like,

        'in' : clause['in']

    }

}
```

Example: Select Liquor Where Type Equal To RUM

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

(b)

```
  this.select = function(query);
```

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

Example: Select Liquor Object.

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

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

5.2.8 Save

1. **Save API - Native**

```
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.
}
```

2. **Save API - Web**

```
this.save = function();
```

Example: Saving Liquor Object.

```
var 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

1. **Update API - Native**

```
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.
}
```

2. **Update API - Web**

```
this.update = function();
```

Example: Updaing Liquor Object.

```
var 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

1. **Save Or Update API - Native**

```
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.
  }
```

2. **Save Or Update API - Web**

```
    this.saveOrUpdate = function();
```

Example: Save Or Update Liquor Object.

```
  var 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

1. **Delete API - Native**

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

```
  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();
```

2. **Delete API - Web**

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

```
this.delete = function();
```

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

```
function IDelete(select) {

  return {

    interfaceName : "IDelete",

    where : select.where,

    whereClause : select.whereClause,

    and : select.and,

        or : select.or,

        execute : select.execute

  }

}
```

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

```
function IDeleteClause(clause) {

    return {

        interfaceName : "IDeleteClause",

        equalTo : clause.equalTo,

        notEqualTo : clause.notEqualTo,
```

```
        greaterThan : clause.greaterThan,
        greaterThanEqual : clause.greaterThanEqual,
        lessThan : clause.lessThan,
        lessThanEqual : clause.lessThanEqual,
        between : clause.between,
        like : clause.like,
        'in' : clause['in']
    }
}
```

Example:

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

5.3 Database Siminov API's

5.3.1 Get Database Descriptor

1. **Get Database Descriptor API - Native**

```
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.
}
```

2. **Get Database Descriptor API - Web**

```
    this.getDatabaseDescriptor = function();
```

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

1. **Get Database Mapping Descriptor API - Native**

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

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

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

2. **Get Database Mapping Descriptor API - Web**

```
    this.getDatabaseMappingDescriptor = function();
```

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

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

5.3.3 Get Table Name

1. **Get Table Name - Native**

```
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.
}
```

2. **Get Table Name - Web**

```
    this.getTableName = function();
```

Example: Get Liquor Object Table Name.

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

5.3.4 Get Column Names

1. **Get Column Names API - Native**

```
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.
}
```

2. **Get Column Names API - Web**

```
    this.getColumnNames = function();
```

Example: Get All Column Names Of Liquor Table.

```
var 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

1. **Get Column Types API - Native**

```
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.
}
```

2. **Get Column Types API - Web**

```
    this.getColumnTypes = function();
```

Example: Get All Column Types Of Liquor Table.

```
var columnTypes = null;
try {
  columnTypes = new Liquor().getColumnTypes();
} catch(DatabaseException de) {
  //Log it.
}
```

5.3.7 Get Primary Keys

1. **Get Primary Keys API - Native**

```
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.
}
```

2. **Get Primary Keys API - Web**

```
    this.getPrimaryKeys = function();
```

Example: Get All Primary Keys Of Liquor Table.

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

5.3.8 Get Mandatory Fields

1. **Get Mandatory Fields API - Native**

```
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.
}
```

2. **Get Mandatory Fields API - Web**

```
    this.getMandatoryFields = function();
```

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

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

5.3.9 Get Unique Fields

1. **Get Unique Fields API - Native**

```
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.
}
```

2. **Get unique Fields API - Web**

```
this.getUniqueFields = function();
```

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

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

5.3.10 Get Foreign Keys

1. **Get Foreign Keys API - Native**

```
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.
}
```

2. **Get Foreign Keys API - Web**

```
this.getForeignKeys = function();
```

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

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

```
}
```

5.4 Database Aggregation API's

5.4.1 Count

1. **Count API - Native**

   Returns the count of rows based on information provided.

```
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 {

  public ICount distinct();

  public ICountClause where(String column);

  public ICount whereClause(String whereClause);

  public ICountClause and(String column);

  public ICountClause or(String column);

  public ICount groupBy(String...columns);

  public ICountClause having(String column);

  public ICount havingClause(String havingClause);

  public ICount column(String column);

  public Object execute() throws DatabaseException;

}
```

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

```
public interface ICountClause {

  public ICount equalTo(String value);

  public ICount notEqualTo(String value);

  public ICount greaterThan(String value);

  public ICount greaterThanEqual(String value);

  public ICount lessThan(String value);

  public ICount lessThanEqual(String value);

  public ICount between(String start, String end);

  public ICount like(String like);

  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.
  }
```

2. **Count API - Web**

   Returns the count of rows based on information provided.

```
    this.count = function();
```

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.

```
function ICount(select) {

    return {

        interfaceName : "ICount",

        distinct : select.distinct,

        where : select.where,

        whereClause : select.whereClause,

        and : select.and,

        or : select.or,

        groupBy : select.groupBy,

        having : select.having,

        havingClause : select.havingClause,

        column : select.column,

        execute : select.execute

    }

}
```

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

```
function ICountClause(clause) {

    return {

        interfaceName : "ICountClause",

        equalTo : clause.equalTo,

        notEqualTo : clause.notEqualTo,

        greaterThan : clause.greaterThan,
```

```
        greaterThanEqual : clause.greaterThanEqual,
        lessThan : clause.lessThan,
        lessThanEqual : clause.lessThanEqual,
        between : clause.between,
        like : clause.like,
        'in' : clause['in']
    }
}
```

Example: Get count of liquors type equal to RUM.

```
var count = 0;

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

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

5.4.2 Average

1. **Average API - Native**

   Returns the average based on column name provided.

```
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 floating 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 {

  public IAverageClause where(String column);

  public IAverage whereClause(String whereClause);

  public IAverageClause and(String column);

  public IAverageClause or(String column);

  public IAverage groupBy(String...columns);

  public IAverageClause having(String column);

  public IAverage havingClause(String havingClause);

  public IAverage column(String column);

  public Object execute() throws DatabaseException;

}
```

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

```
public interface IAverageClause {

  public IAverage equalTo(String value);

  public IAverage notEqualTo(String value);

  public IAverage greaterThan(String value);

  public IAverage greaterThanEqual(String value);

  public IAverage lessThan(String value);

  public IAverage lessThanEqual(String value);

  public IAverage between(String start, String end);

  public IAverage like(String like);

  public IAverage in(String...values);

}
```

Example:

```
int average = 0;

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

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

2. **Average API - Web**

   Returns the average based on column name provided.

```
    this.avg = function();
```

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 floating 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.

```
function IAverage(select) {

    return {

        interfaceName : "IAverage",

        where : select.where,

        whereClause : select.whereClause,

        and : select.and,

        or : select.or,

        groupBy : select.groupBy,
```

```
        having : select.having,
        havingClause : select.havingClause,
        column : select.column,
        execute : select.execute
    }
}
```

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

```
function IAverageClause(clause) {
    return {
        interfaceName : "IAverageClause",
        equalTo : clause.equalTo,
        notEqualTo : clause.notEqualTo,
        greaterThan : clause.greaterThan,
        greaterThanEqual : clause.greaterThanEqual,
        lessThan : clause.lessThan,
        lessThanEqual : clause.lessThanEqual,
        between : clause.between,
        like : clause.like,
        'in' : clause['in']
    }
}
```

Example:

```
var average = 0;

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

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

5.4.3 Sum

1. **Sum API - Native**

   Returns the sum based on column name provided.

```
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 {

  public ISumClause where(String column);

  public ISum whereClause(String whereClause);

  public ISumClause and(String column);

  public ISumClause or(String column);

  public ISum groupBy(String...columns);

  public ISumClause having(String column);

  public ISum havingClause(String havingClause);
```

```
  public ISum column(String column);

  public Object execute() throws DatabaseException;

}
```

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

```
public interface ISumClause {

  public ISum equalTo(String value);

  public ISum notEqualTo(String value);

  public ISum greaterThan(String value);

  public ISum greaterThanEqual(String value);

  public ISum lessThan(String value);

  public ISum lessThanEqual(String value);

  public ISum between(String start, String end);

  public ISum like(String like);

  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.
  }
```

2. **Sum API - Web**

   Returns the sum based on column name provided.

```
    this.sum = function();
```

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.

```
function ISum(select) {

    return {

        interfaceName : "ISum",

        where : select.where,

        whereClause : select.whereClause,

        and : select.and,

        or : select.or,

        groupBy : select.groupBy,

        having : select.having,

        havingClause : select.havingClause,

        column : select.column,

        execute : select.execute

    }

}
```

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

```
function ISumClause(clause) {

    return {

        interfaceName : "ISumClause",

        equalTo : clause.equalTo,

        notEqualTo : clause.notEqualTo,

        greaterThan : clause.greaterThan,

        greaterThanEqual : clause.greaterThanEqual,

        lessThan : clause.lessThan,

        lessThanEqual : clause.lessThanEqual,

        between : clause.between,

        like : clause.like,

        'in' : clause['in']

    }

}
```

Example:

```
  var 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

1. **Total API - Native**

   Returns the total based on column name provided.

```
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 floating point value.

```
public interface ITotal {

  public ITotalClause where(String column);

  public ITotal whereClause(String whereClause);

  public ITotalClause and(String column);

  public ITotalClause or(String column);

  public ITotal groupBy(String...columns);

  public ITotalClause having(String column);

  public ITotal havingClause(String havingClause);

  public ITotal column(String column);

  public Object execute() throws DatabaseException;

}
```

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

```
public interface ITotalClause {

  public ITotal equalTo(String value);

  public ITotal notEqualTo(String value);

  public ITotal greaterThan(String value);
```

```
  public ITotal greaterThanEqual(String value);

  public ITotal lessThan(String value);

  public ITotal lessThanEqual(String value);

  public ITotal between(String start, String end);

  public ITotal like(String like);

  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.
  }
```

2. **Total API - Web**

   Returns the total based on column name provided.

```
    this.total = function();
```

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 floating point value.

```
function ITotal(select) {

    return {
```

```
        interfaceName : "ITotal",
        where : select.where,
        whereClause : select.whereClause,
        and : select.and,
        or : select.or,
        groupBy : select.groupBy,
        having : select.having,
        havingClause : select.havingClause,
        column : select.column,
        execute : select.execute
    }
}
```

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

```
function ITotalClause(clause) {
    return {
        interfaceName : "ITotalClause",
        equalTo : clause.equalTo,
        notEqualTo : clause.notEqualTo,
        greaterThan : clause.greaterThan,
        greaterThanEqual : clause.greaterThanEqual,
        lessThan : clause.lessThan,
        lessThanEqual : clause.lessThanEqual,
        between : clause.between,
        like : clause.like,
```

```
        'in' : clause['in']

    }

}
```

Example:

```
  var 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

1. **Minimum API - Native**

```
  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 first 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 {

  public IMinClause where(String column);

  public IMin whereClause(String whereClause);

  public IMinClause and(String column);

  public IMinClause or(String column);
```

```
  public IMin groupBy(String... columns);

  public IMinClause having(String column);

  public IMin havingClause(String havingClause);

  public IMin column(String column);

  public Object execute() throws DatabaseException;

}
```

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

```
public interface IMinClause {

  public IMin equalTo(String value);

  public IMin notEqualTo(String value);

  public IMin greaterThan(String value);

  public IMin greaterThanEqual(String value);

  public IMin lessThan(String value);

  public IMin lessThanEqual(String value);

  public IMin between(String start, String end);

  public IMin like(String like);

  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.
    }
```

2. **Minimum API - Web**

   Returns the minimum based on column name provided.

```
    this.min = function();
```

IMin: Exposes API's to returns the minimum non-NULL value of all values in the group. The minimum value is the first 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.

```
function IMin(select) {

    return {

        interfaceName : "IMin",

        where : select.where,

        whereClause : select.whereClause,

        and : select.and,

        or : select.or,

        groupBy : select.groupBy,

        having : select.having,

        havingClause : select.havingClause,

        column : select.column,

        execute : select.execute

    }

}
```

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

```
function IMinClause(clause) {

    return {

        interfaceName : "IMinClause",

        equalTo : clause.equalTo,

        notEqualTo : clause.notEqualTo,

        greaterThan : clause.greaterThan,

        greaterThanEqual : clause.greaterThanEqual,

        lessThan : clause.lessThan,

        lessThanEqual : clause.lessThanEqual,

        between : clause.between,

        like : clause.like,

        'in' : clause['in']

    }

}
```

Example:

```
  var 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

1. **Maximum API - Native**

   Returns the maximum based on column name provided.

```
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 {

  public IMaxClause where(String column);

  public IMax whereClause(String whereClause);

  public IMaxClause and(String column);

  public IMaxClause or(String column);

  public IMax groupBy(String...columns);

  public IMaxClause having(String column);

  public IMax havingClause(String havingClause);

  public IMax column(String column);

  public Object execute() throws DatabaseException;

}
```

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

```
public interface IMaxClause {

  public IMax equalTo(String value);
```

```
  public IMax notEqualTo(String value);

  public IMax greaterThan(String value);

  public IMax greaterThanEqual(String value);

  public IMax lessThan(String value);

  public IMax lessThanEqual(String value);

  public IMax between(String start, String end);

  public IMax like(String like);

  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.
  }
```

2. **Maximum API - Hybrid**

   Returns the minimum based on column name provided.

```
    this.max = function();
```

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.

```
function IMax(select) {

    return {

        interfaceName : "IMax",

        where : select.where,

        whereClause : select.whereClause,

        and : select.and,

        or : select.or,

        groupBy : select.groupBy,

        having : select.having,

        havingClause : select.havingClause,

        column : select.column,

        execute : select.execute

    }

}
```

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

```
function IMaxClause(clause) {

    return {

        interfaceName : "IMaxClause",

        equalTo : clause.equalTo,

        notEqualTo : clause.notEqualTo,

        greaterThan : clause.greaterThan,

        greaterThanEqual : clause.greaterThanEqual,

        lessThan : clause.lessThan,
```

```
        lessThanEqual : clause.lessThanEqual,

        between : clause.between,

        like : clause.like,

        'in' : clause['in']

    }

}
```

Example:

```
var 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

1. **Group Concat API - Native**

   Returns the group concat based on column name provided.

```
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 {

  public IGroupConcat delimiter(String delimiter);
```

```
    public IGroupConcatClause where(String column);

    public IGroupConcat whereClause(String whereClause);

    public IGroupConcatClause and(String column);

    public IGroupConcatClause or(String column);

    public IGroupConcat groupBy(String... columns);

    public IGroupConcatClause having(String column);

    public IGroupConcat havingClause(String havingClause);

    public IGroupConcat column(String column);

    public Object execute() throws DatabaseException;

}
```

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

```
public interface IGroupConcatClause {

    public IGroupConcat equalTo(String value);

    public IGroupConcat notEqualTo(String value);

    public IGroupConcat greaterThan(String value);

    public IGroupConcat greaterThanEqual(String value);

    public IGroupConcat lessThan(String value);

    public IGroupConcat lessThanEqual(String value);

    public IGroupConcat between(String start, String end);

    public IGroupConcat like(String like);

    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.
}
```

2. **Group Concat API - Hybrid**

   Returns the group concat based on column name provided.

```
    this.groupConcat = function();
```

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.

```
function IGroupConcat(select) {

    return {

        interfaceName : "IGroupConcat",

        delimiter : select.delimiter,

        where : select.where,

        whereClause : select.whereClause,

        and : select.and,

        or : select.or,

        groupBy : select.groupBy,

        having : select.having,
```

```
        havingClause : select.havingClause,
        column : select.column,
        execute : select.execute
    }
}
```

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

```
function IGroupConcatClause(clause) {
    return {
        interfaceName : "IGroupConcatClause",
        equalTo : clause.equalTo,
        notEqualTo : clause.notEqualTo,
        greaterThan : clause.greaterThan,
        greaterThanEqual : clause.greaterThanEqual,
        lessThan : clause.lessThan,
        lessThanEqual : clause.lessThanEqual,
        between : clause.between,
        like : clause.like,
        'in' : clause['in']
    }
}
```

Example:

```
  var 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

1. **Begin Transaction API - Native**

```
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);
}
```

2. **Begin Transaction API - Web**

```
Database.beginTransaction = function(databaseDescriptor);
```

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.

```
var beer = new Liquor();
beer.setLiquorType(Liquor.LIQUOR_TYPE_BEER);
beer.setDescription(Constants.beer_description));
beer.setHistory(Constants.beer_history));
beer.setLink(Constants.beer_link));
beer.setAlcholContent(Constants.beer_alchol_content));

var 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

1. **Commit Transaction API - Native**

```
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);
}
```

2. **Commit Transaction API - Web**

```
Database.commitTransaction = function(databaseDescriptor);
```

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

Example: Save And Commt Liquor Transaction.

```
var 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

1. **End Transaction API - Native**

```
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);
}
```

2. **End Transaction API - Web**

```
Database.endTransaction = function(databaseDescriptor);
```

End the current transaction.

Example: End Transaction After Save And Commit Transaction.

```
var 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 Handling Database Relationships

5.6.1 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>
```

5.6.2 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 effectively 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>
```

5.6.3 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>
```

5.6.4 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>
```

5.7 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);
```

Configuring transaction thread-safe in Siminov. To enable/disable transaction thread-safe in Siminov you have to use property is_locking_required defined in DatabaseDescriptor.si.xml file.

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

5.8 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;
```

```
<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>
```

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

  }
```

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

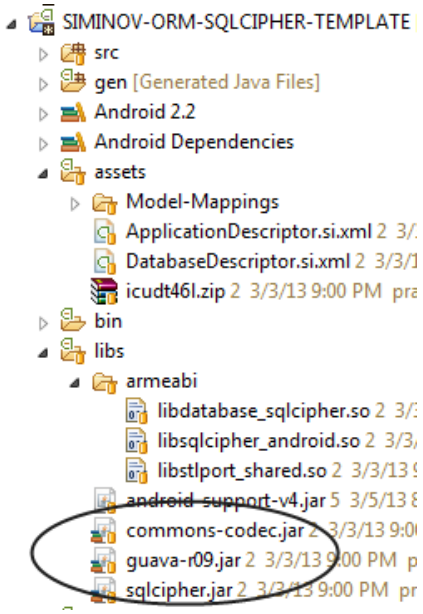
SQLCipher is an open source extension to SQLite that provides transparent 256-bit AES encryption of database file. SQLCipher has a small footprint and great performance so it's ideal for protecting embedded application databases and is well suited for mobile development.

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

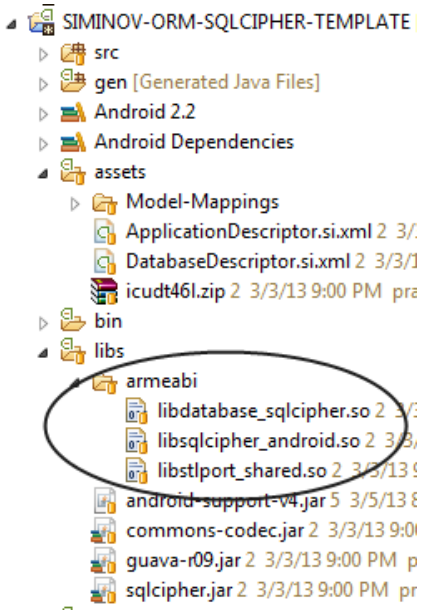
Below are steps to make your application totally secured.

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

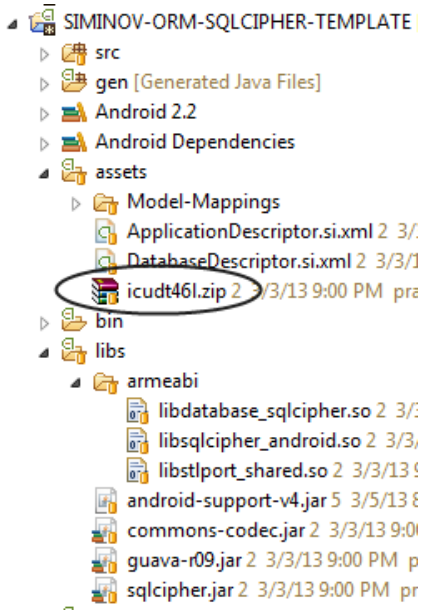
(a) Copy **sqlcipher.jar**, **guava-r09.jar**, **commons-codec.jar** in your application libs folder.



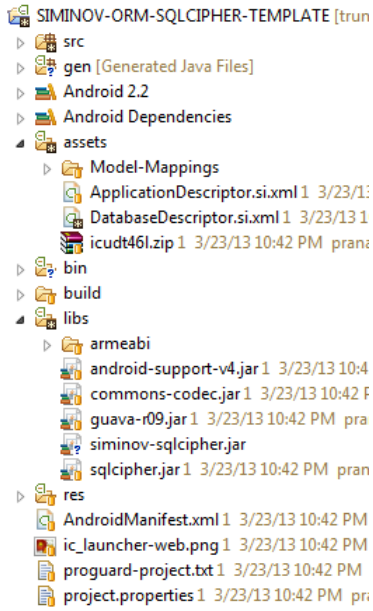
(b) Copy dll file in your libs folder.



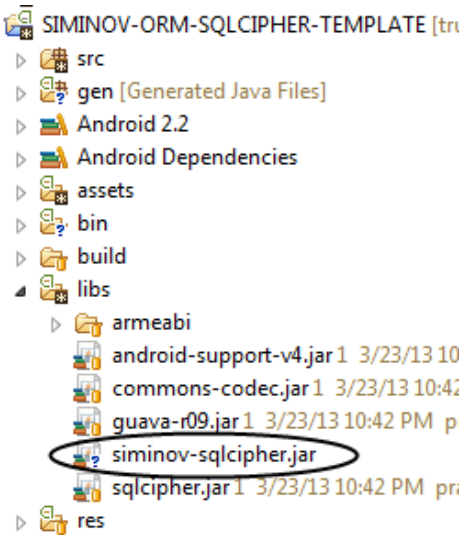
(c) Copy icudt461.zip in your application assets folder.



The stucture of your application should look like this.



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



4. To configure 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 configured 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 flag 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 affected 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.executeQuery(Query);
  } catch(DatabaseException databaseException) {
    // Log It.
  }
```

4. **Execute Bind Query - executeBindQuery(Query, Column Values)**: A pre-compiled statement that can be reused. The statement cannot return multiple rows, but 1x1 result sets are allowed.

   Example:

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

```
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

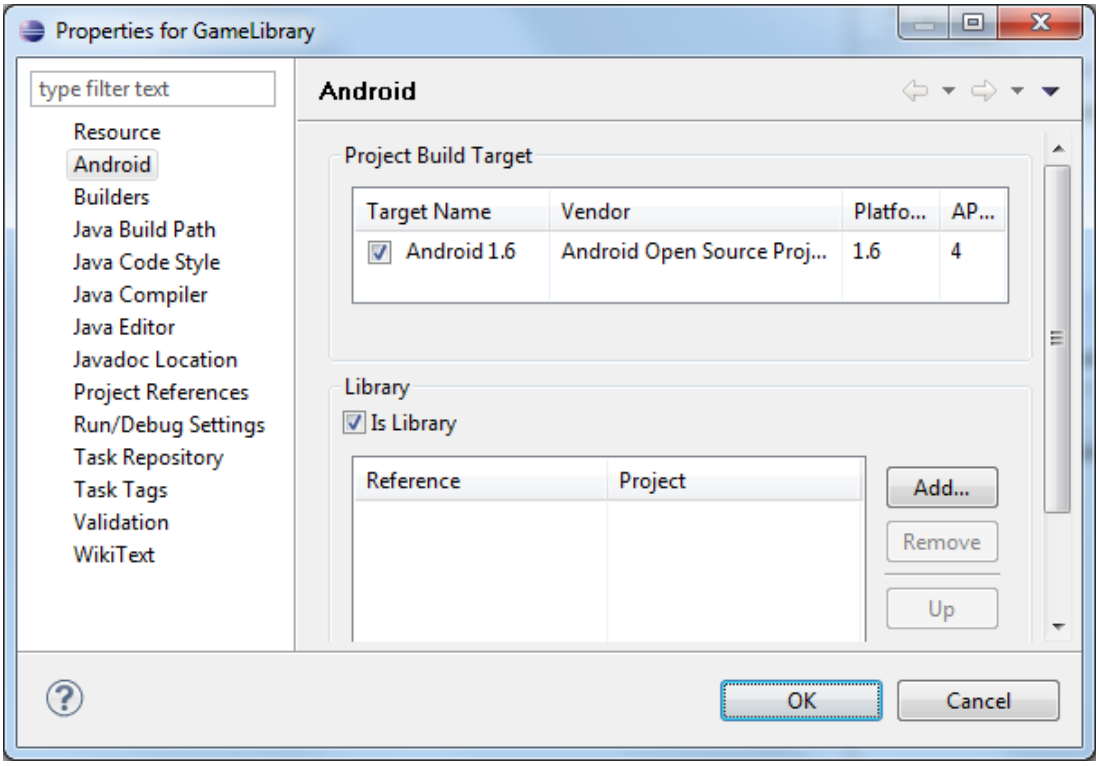
Handling Libraries

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 files. Multiple application projects can reference the same library project and any single application project can reference multiple library projects.

Siminov provides mechnism to configure ORM for your library projects.

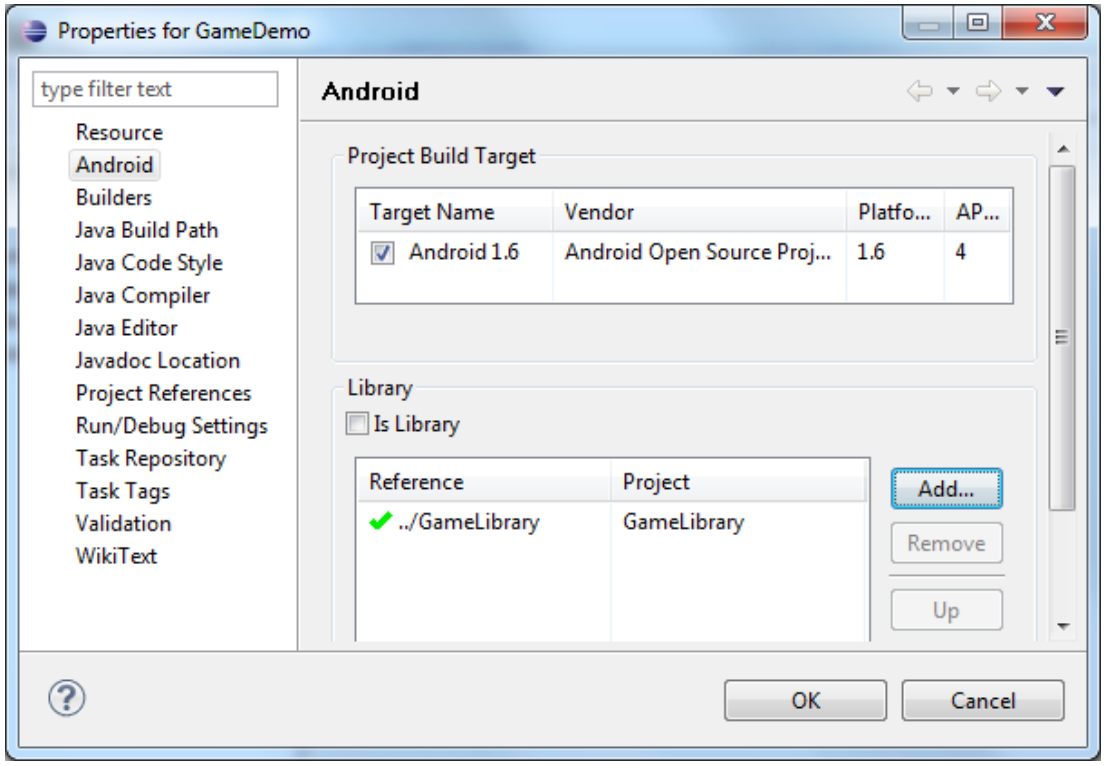
8.1 Setting up a Library Project

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.



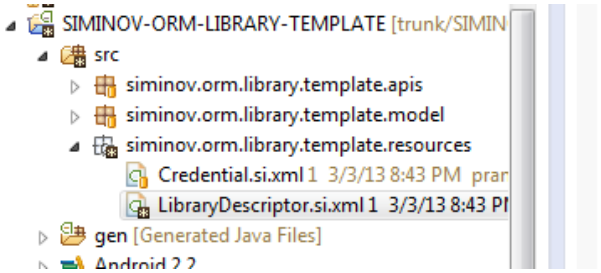
8.2 Referencing a library project

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.


Properties for GameDemo
type filter text
Resource
Android
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Android
Project Build Target
Target Name
Vendor
Platfo...
AP...
Android 1.6
Android Open Source Proj...
1.6
4
Library
Is Library
Reference
Project
../GameLibrary
GameLibrary
Add...
Remove
Up
OK
Cancel

8.3 Configure Application With Library

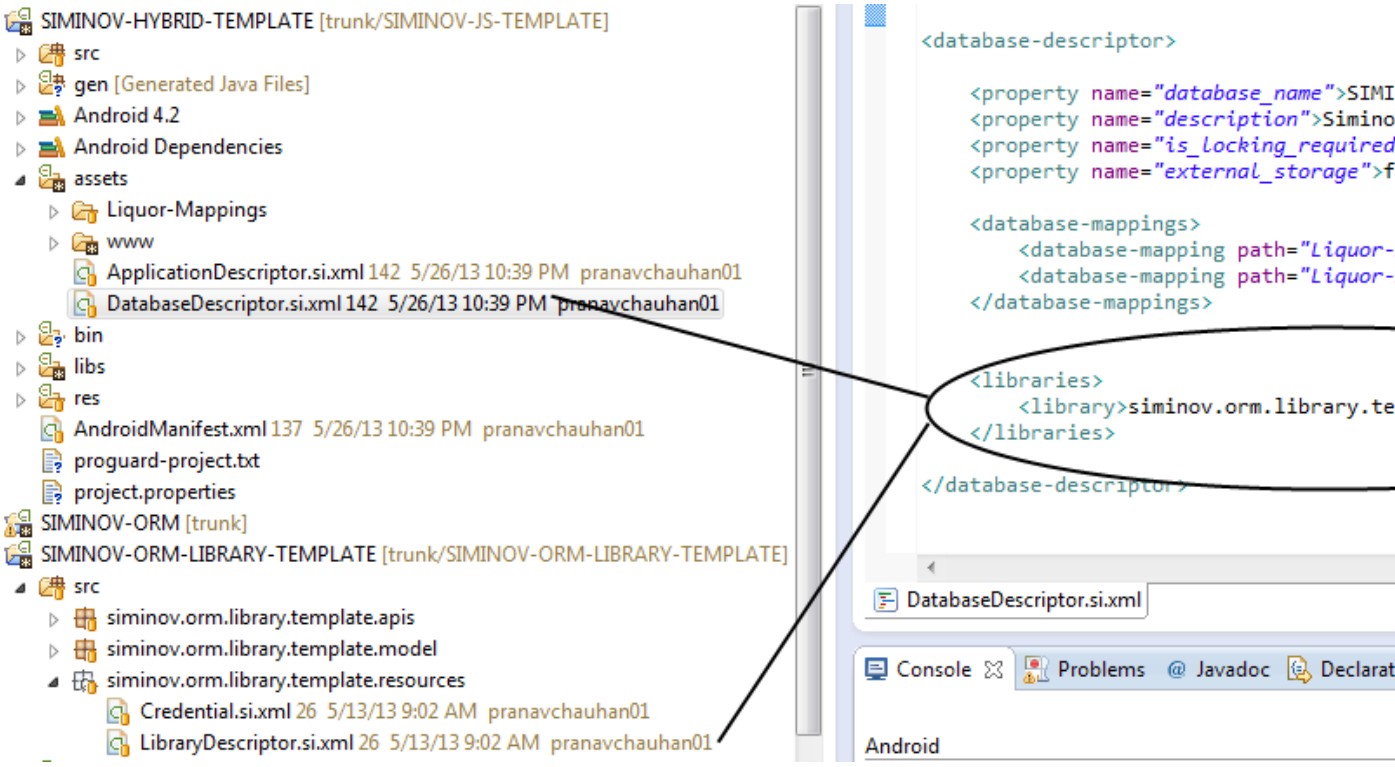
1. Define LibraryDescriptor.si.xml file for your library project.



Note

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

2. Configure LibraryDescriptor.si.xml in your application.



Note

While configuring DatabaseDescriptor.si.xml file you need to provide full library package name in which LibraryDescriptor.si.xml file is defined.

Chapter 9

Exceptions

9.1 Siminov Exception

1. **Siminov Exception - Native**

```
public class SiminovException extends Exception { }
```

2. **Siminov Exception - Web**

```
function SiminovException(className, methodName, message);
```

This is general exception, which is thrown through Siminov API, if any exception occur while performing any tasks.

9.2 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.

9.3 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.