## Contents

1. **Contents:**
   1.1 About Framework .................................................. 1
   1.2 Setting up IDE ..................................................... 1
   1.3 Getting Started with Framework ................................. 3
   1.4 Contributing ........................................................ 74

2. **Indices and tables** .......................... 75
1.1 About Framework

Odoo is a powerful open source framework. With help of this framework we can rapidly develop almost any application.

World is contracting with the growth of mobile phone technology. As the number of users is increasing day by day, facilities are also increasing. Now a days mobiles are not used just for making calls but they have innumerable uses and can be used as a Camera , Music player, Tablet PC, T.V. , Web browser etc. And with the new technologies, new software and operating systems are required.

One of the most widely used mobile OS these days is ANDROID. Android is a software bunch comprising not only operating system but also middleware and key applications.

Odoo Mobile framework is an open source mobile application development framework with Odoo integration. With the help of mobile framework we can rapidly develop almost all Odoo supported application as faster as we can develop in Odoo Framework.

This framework contains its own ORM to handle mobile’s local database. So you do not have to worry about data coming from Odoo Server. It has pre-developed services and providers to make your application data synchronized with Odoo.

1.2 Setting up IDE

1.2.1 Downloading & Installing Android Studio


The official Android Development Tool

- Android Studio IDE
- Android SDK tools
- Android 5.0 (Lollipop) Platform
- Android 5.0 emulator system image with Google APIs

1.2.2 System Requirements

Windows
• Microsoft® Windows® 8/7/Vista/2003 (32 or 64-bit)
• 2 GB RAM minimum, 4 GB RAM recommended
• 400 MB hard disk space
• At least 1 GB for Android SDK, emulator system images, and caches
• 1280 x 800 minimum screen resolution
• Java Development Kit (JDK) 7
• Optional for accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

Mac OS X
• Mac® OS X® 10.8.5 or higher, up to 10.9 (Mavericks)
• 2 GB RAM minimum, 4 GB RAM recommended
• 400 MB hard disk space
• At least 1 GB for Android SDK, emulator system images, and caches
• 1280 x 800 minimum screen resolution
• Java Runtime Environment (JRE) 6
• Java Development Kit (JDK) 7
• Optional for accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality
• On Mac OS, run Android Studio with Java Runtime Environment (JRE) 6 for optimized font rendering. You can then configure your project to use Java Development Kit (JDK) 6 or JDK 7.

Linux
• GNOME or KDE desktop
• GNU C Library (glibc) 2.15 or later
• 2 GB RAM minimum, 4 GB RAM recommended
• 400 MB hard disk space
• At least 1 GB for Android SDK, emulator system images, and caches
• 1280 x 800 minimum screen resolution
• Oracle® Java Development Kit (JDK) 7
• Tested on Ubuntu® 14.04, Trusty Tahr (64-bit distribution capable of running 32-bit applications).

1.2.3 Setting up testing device

Testing with virtual device

Testing with physical device
Refer: http://developer.android.com/tools/device.html
1.3 Getting Started with Framework

1.3.1 Download framework from GitHub for Android

Visit Odoo Mobile Repositories

https://github.com/Odoo-mobile

Here you can find all the repositories developed by Odoo S.A.

Click on Framework

https://github.com/Odoo-mobile/framework

You can also choose another repository source code such as crm, notes
Clone or download source code for Odoo Mobile Framework

Import framework code to your Android Studio

Import your project source by selecting Import Project
Test Framework application build for loading customers, suppliers and companies

After successfully load your project to Android Studio you can run it by pressing Run App button from toolbar.
1.3.2 Working with Odoo Mobile Framework

First step to Odoo Mobile

Odoo is the fastest evolving business software in the world. Odoo has a complete suite of business applications covering all business needs, from Website/Ecommerce down to manufacturing, inventory and accounting, all seamlessly integrated. It is the first time ever a software editor managed to reach such a functional coverage.

World is contracting with the growth of mobile phone technology. As the number of users is increasing day by day, facilities are also increasing. Now a days mobiles are not used just for making calls but they have innumerable uses and can be used as a Camera, Music player, Tablet PC, T.V., Web browser etc. And with the new technologies, new software and operating systems are required.

One of the most widely used mobile OS these days is ANDROID. Android is a software bunch comprising not only operating system but also middleware and key applications.

Odoo Mobile framework is an open source mobile application development framework with Odoo integration. With the help of this mobile framework we can rapidly develop almost all Odoo supported application as faster as we can develop in Odoo Framework.
This framework contains its own ORM to handle mobile’s local database. So you do not have to worry about data coming from Odoo Server. It has pre-developed services and providers to make your application data synchronized with Odoo.

Here is some of the application build with Odoo Mobile Framework (Also available on PlayStore):

Understanding architecture

Framework Architecture

Core architecture Odoo Mobile framework uses:

- **Green Layer**: Full user customizable part.

**Application Configuration**

Android Manifest, Gradle configuration are at this layer. Generally used for changing package (application ID), application version, register activities, providers, services, receivers and more.

Activities, Fragments, Drawer menus

An Activity is an application component that provides a screen with which users can interact in order to do something. Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows. More at http://developer.android.com/guide/components/activities.html

A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. More at http://developer.android.com/guide/components/fragments.html

The navigation drawer is a panel that transitions in from the left edge of the screen and displays the app’s main navigation options. More at https://developer.android.com/training/implementing-navigation/nav-drawer.html

Framework provide easy management with the navigation drawer. You just need to provide menu items and its callback. Have a look for Odoo Mobile drawer. Also it is fully customizable.
Controls (Form, Fields, Actionbar menus, FAB Button, and more..)

Odoo Mobile framework provides inbuild controls for faster build your layouts.

- Form
- Fields
  - Boolean
  - Text
  - Radio
  - Checkbox
  - ManyToOne
  - Spinner
  - DateTime
  - and more...
- FAB Button
• Pull to refresh (works with listview)

**Blue Layer**  **Database models, data loaders, custom uris, custom content providers...**

Blue layer contains all your addons model (database table) architecture, business logic for your addon, code behind for your activities and fragments.

Also, there are loaders for your listviews, works with your model content provider. You can also create your custom content provider for custom queries and data loading.

**Orange Layer**  **Orange layer is responsible to get data from odoo server and provide it to framework synchronization base adapter. Framework will take care for offline data. It will compare your latest updated record and synchronize them as well update record in local and server. We have tried to give 100% offline support, but in some cases you need to override synchronization and have to put your own mechanism. Yes, of course it's customizable. :P**

**Red Layer**  **Red layer contains all the core components used by the frameworks. ORM, Synchronization base adapters, Base service providers, base content model providers and some utility classes. Red layer also responsible to manage application accounts, synchronization settings.**

Red layer and Orange layer are dependent part.

**Danger:** Change in this file causes system flow break. Kindly take copy before changes in red layer part.

Framework will create separate database for each of different account. By creating separate database it is possible to create different database version as per odoo versions.

**Directory structure**

Directory structure is same as standard android studio android application structure. Framework has organized some of the packages for faster development and easy understandability of addons (modules).
Addons   All modules (addons) are under `com.odoo.addons` package

All the modules are created under `addons` package along with module package name as shown in figure.

@See Working with addons

Addons Registry  Each addons has its parent view. Such as List of customers, which loaded when user click on Drawer menu item. We have to register each of the parent class for addon in `Addons.java` under `com.odoo.config` package.
Create your feature object with OAddon class named with your feature and provide your feature class (which extends BaseFragment) as below:

```java
public class Addons extends AddonsHelper {

    /**
     * Declare your required module here
     * NOTE: For maintain sequence use object name in asc order.
     * Ex.:
     * OAddon partners = new OAddon(Partners.class).setDefault();
     */
    OAddon customers = new OAddon(Customers.class).setDefault();
}
```

If you want to make your addons default to load when application starts. Just add chaining method `setDefault()`

You can create multiple object as member in this class.

1.3. Getting Started with Framework
Tips: For shorting your features just put alphabetical ordered name to your features.

E.g:

```java
OAddon a_sales = new OAddon(Sale.class).setDefault();
OAddon b_messaging = new OAddon(Messaging.class);
OAddon c_customers = new OAddon(Customer.class);
```

**Note:** System automatically create database as you applied in your `BaseFragment.java` but if some of the model are not creating automatically then you can add those models in `BaseModels.java`

**Base models** Odoo mobile framework comes with some of base models, such as `res.partners`, `res.users`, `mail.message`, `ir.attachments` and more.
1.3. Getting Started with Framework
Contains all the base models (database models) used by framework itself.

You can modify it (add or update columns as you required). Default it contains, ir.attachment, ir.model, mail.message (for chatter), res_* related models such as res_partner, res_users and so on.

You can also add your base model; after adding your model you need to register it in `BaseModels.java` file.

```java
public class BaseModels {
    public static final String TAG = BaseModels.class.getSimpleName();

    public static List<OModel> baseModels(Context context, OUser user) {
        List<OModel> models = new ArrayList<>();
        models.add(new OdooNews(context, user));
        models.add(new IrModel(context, user));
        models.add(new ResPartner(context, user));
        models.add(new ResUsers(context, user));
        models.add(new ResCompany(context, user));
        models.add(new IrAttachment(context, user));
        models.add(new MailMessage(context, user));
        return models;
    }
}
```

**Mail Chatter (Widget)**  Widget under `com.odoo.base.addons.mail.widget`

Mail chatter widget : provide chatter view for each of the record after form view. **Note: works with OForm control only.**
You can integrate mail chatter widget with OForm control by providing `setHasChatter(true)` in models constructor.

```java
public ResPartner(Context context, OUser user) {
    super(context, "res.partner", user);
    setHasMailChatter(true);
}
```
Basic Components

Base Classes

BaseFragment.java  

BaseFragment.java class extends android.support.v4.app.Fragment.Fragment class and implements IBaseFragment interface which contains two methods:

```java
public interface IBaseFragment {
    public List<ODrawerItem> drawerMenus(Context context);
    public <T> Class<T> database();
}
```

We must have to create one file that extends BaseFragment.java which will be registered in Addons.java

These two methods are used by framework for creating database and drawer menu. Here, <T> indicate your model class type. We will see it in OModel.java

BaseFragment.java contains some of useful methods we can used in our fragment. As below:

setTitle() Syntax:

Void setTitle(String title)

Used for setting actionbar title:

setTitle("My Title");

db() Syntax:

OModel db()

Used to get current db object returned in database() method which implemented in our fragment class.

db().select();

OValues values = new OValues();
... ...

db().insert(values);

user() Syntax:

OUser user()

Used to get current active user object. It is easy when you want to do some operation with current user.

user().getName();
user().getPartner_id();
user().getUser_id();
...
parent()  Syntax:

OdooActivity parent()

Returns parent activity (i.e., getActivity()). It will automatically cast Activity object to OdooActivity
So you can easily access public methods of OdooActivity

parent().closeDrawer();
parent().refreshDrawer();
parent().setOnActivityResultListener(callback);

We will see more methods of parent() method in OdooActivity.java

inNetwork()  Syntax:

boolean inNetwork()

Returns true if device in network, otherwise false

if(inNetwork()){  
    // Do some stuff
} else{  
    // Ignore stuff
}

startFragment()  Syntax:

Void startFragment(Fragment fragment, boolean addToBackState)

Void startFragment(Fragment fragment, boolean addToBackState, Bundle data)

Used to start another fragment from current fragment. You can also specify backstate when starting another fragment.

startFragment(new MyNewFragment(), true); // Starting fragment with backstate

Bundle data = new Bundle();
...
...
startFragment(new MyNewFragment(), false, data); // Starting new fragment without backstate but with

setHasFloatingButton()  Syntax:

void setHasFloatingButton(View view, int viewId, ListView listViewObj, View.OnClickListener callback)

By default floating button is hidden. You need to activate floating button to use. It will auto add callback method. Also
you need to add ListView object as parameter so when you scroll your listview FAB will automatically hide/visible on
listview scroll.

setHasFloatingButton(view, R.id.fabButton, listViewObj, this);

// this will implement onClick(View v) method

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.fabButton:
            // Do your stuff
            break;

1.3. Getting Started with Framework
hideFab() and showFab()  Syntax:
void hidFab()
void showFab()

After setting fab button you can call hideFab() and showFab() as per your requirements

```java
if (inNetwork()){
    showFab();
} else {
    hideFab();
}
```

setHasSearchView()  Syntax:
void setHasSearchView(IOnSearchViewChangeListener callback, Menu menu, int menu_id)

If there is a menu with search option. You can directly set setHasSearchView() and framework will work for you. It will give you callback on search text changed and search view close.

It takes callback to its first parameter of IOnSearchViewChangeListener interface which has following methods:

```java
public interface IOnSearchViewChangeListener {
    public static final String TAG = IOnSearchViewChangeListener.class.getSimpleName();

    public boolean onSearchViewTextChange(String newFilter);
    public void onSearchViewClose();
}
```

To apply search view callback just call method when you create your menu:

```java
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    menu.clear();
    inflater.inflate(R.menu.menu_partners, menu);
    setHasSearchView(this, menu, R.id.menu_partner_search);
}
```

It will implement two methods:

```java
@Override
public boolean onSearchViewTextChange(String newFilter) {
    mCurFilter = newFilter;
    // Do any other stuff when change filter text
    return true;
}
```

```java
@Override
public void onSearchViewClose() {
    // Any stuff when user close search view
}
```
setHasSwipeRefreshView() Syntax:
void setHasSwipeRefreshView(View parentView, int resourceId, SwipeRefreshLayout.OnRefreshListener callback)

When using swipe refresh view, you can easily set its call back by calling setHasSwipeRefreshView

```java
setHasSwipeRefreshView(view, R.id.swipe_container, this);
```

It will implement one method for calling swipe event:

```java
@Override
public void onRefresh() {
    if (inNetwork()) {
        // Do your stuff
    } else {
        // Do your stuff
    }
}
```

setSwipeRefreshing() and hideRefreshingProgress() Syntax:
void setSwipeRefreshing(boolean refreshing)
void hideRefreshingProgress()

When using swipe refresh view you can use these method for hiding and showing refreshing operation.

```java
@Override
public void onRefresh() {
    if (inNetwork()) {
        setSwipeRefreshing(true);
    } else {
        hideRefreshingProgress();
    }
}
```

setHasSyncStatusObserver() Syntax:
void setHasSyncStatusObserver(String menuKEY, ISyncStatusObserverListener callback, OModel model)

Used when any of your data are synchronizing in background and you need to notify when sync finished or data set update. By calling this method it is easy to notify on dataset change.

```java
setHasSyncStatusObserver(KEY, this, db());
```

1. It takes drawer KEY or TAG which passed when creating drawer menu
2. Callback for data set change. Implement onStatusChange() method
3. database object on which you need to set observer

```java
@Override
public void onStatusChange(Boolean changed) {
    if (changed) {
        getLoaderManager().restartLoader(0, null, this); // Updating listview
    }
}
```
_s(), _c() and _dim() Syntax:

String _s(int resource_id)
int _c(int resource_id)
int _dim(int resource_id)

Used to get quick string, color and dimension.

String name = _s(R.string.app_name);
int color = _c(R.color.theme_primary);
int height = _dim(R.dimen.statusBarHeight);

Sample class using BaseFragment

public class Messages extends BaseFragment {
    public static final String TAG = Messages.class.getSimpleName();

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.common_listview, container, false);
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        setTitle(_s(R.string.title_messages));
    }

    @Override
    public Class<MailMessage> database() {
        return MailMessage.class;
    }

    @Override
    public List<ODrawerItem> drawerMenus(Context context) {
        return null;
    }
}

OColumn.java Used to create column for model.

Syntax:

OColumn(String label, Class<?> type);
OColumn(String label, Class<?> type, RelationType relationType);

Here,

1. label indicate column label. It auto load in form control as control title.
2. type type of datatype. (Basic type + Relation type)
3. relationType [optional] if type is related to another model class
   • Possible Types
Some chaining methods

*Each method returns, `OColumn` object with updated value.*

**setName()**  Syntax:

```java
OColumn setName(String name)
```

Used to set column name. Generally variable name is considered as column name but if you want to change its name runtime you can change it.

```java
OColumn dummy_column = new OColumn("Name", OVarchar.class).setName("name");
```

**Danger:** Remember that, You can not change column name after model’s constructor call finish. If you trying to change outside of constructor, it will affect your local database.

**setRecordSyncLimit()**  Syntax:

```java
OColumn setRecordSyncLimit(int limit)
```

Limiting syncing record for `OneToMany` and `ManyToMany`.

```java
OColumn tag_ids = new OColumn("Tags", NoteTag.class, RelationType.ManyToMany)
                .setRecordSyncLimit(10);
```

**Note:** Not all records are synced for OneToMany and ManyToMany if you set record sync limit.

**setLabel()**  Syntax:

```java
OColumn setLabel(String label)
```

Sets label for column name. Used as title for column.

```java
OColumn name = new OColumn("Dummy Name", OVarchar.class).setLabel("Name");
```

**setRelatedColumn()**  Syntax:

```java
OColumn setRelatedColumn(String related_column)
```

Used when you have created `OneToMany` relation column. `OneToMany` required related column to maintain relation.

```java
OColumn child_ids = new OColumn("Contacts", ResPartner.class, RelationType.OneToMany)
                   .setRelatedColumn("parent_id");
```

**setSize()**  Syntax:

```java
OColumn setSize(int size)
```

Used to set column size. Takes integer value.
OColumn title = new OColumn("Blog Title", OVarchar.class).setSize(100);

setDefaultValue() Syntax:
OColumn setDefaultValue(Object value)
Sets default value for column. Will store into database if user not pass value for column

OColumn published = new OColumn("Published", OBoolean.class).setDefaultValue(false);

setRequired() Syntax:
OColumn setRequired()
Sets column value required. OForm will automatically show validation error if column is required.

OColumn name = new OColumn("Name", OVarchar.class).setRequired();

setLocalColumn() Syntax:
OColumn setLocalColumn()
Sometime you need some local column that will not available on server. You can make any column local only.

OColumn total_amount = new OColumn("Total Amount", OInteger.class).setLocalColumn();

setType() Syntax:
OColumn setType(Class<?> type)
Sets data type for column. You can change data type for column as runtime but only in constructor. In some cases, such as different type for different odoo version.

    OColumn date = new OColumn("Date", ODate.class);
    public ResPartner(Context context, OUser user) {
        super(context, "crm.lead", user);
        if(getOdooVersion().getVersion_number() > 7){
            date.setType(ODATE.class);
        }
    }

addDomain() Adds default filter domain for column. Basically you need ManyToOne column to be filter on some conditions.

Syntax:
addDomain(String column_name, String operator, Object value);
addDomain(String conditional_operator);

Example:
OColumn parent_id = new OColumn("Related Company", ResPartner.class, RelationType.ManyToOne) .addDomain("is_company","="true);
addSelection() Syntax:

OColumn addSelection(String key, String value)

Used to add key value selection pair. Used with OSelection data type.

OColumn state = new OColumn("State", OSelection.class)
   .addSelection("draft", "Draft")
   .addSelection("confirm", "Confirmed")
   .addSelection("close", "Canceled")
   .addSelection("done", "Done");

Column annotations

@Odoo.api.v7, @Odoo.api.v8 and @Odoo.api.v9alpha api annotations are used when your column name is different in odoo versions. Or may be it is possible that some of column not present in older version and newer version. Framework column annotation provide feature for making your model compitible for different odoo versions.

You need to just add annotation on column with your supported version.

@Odoo.api.v7
OColumn to_read = new OColumn("To Read", OBoolean.class);

@Odoo.api.v8
OColumn is_read = new Column("Is read", OBoolean.class);

Here, api.v7 column will created only if connected odoo server is version 7.0, same as for api.v8

@Odoo.SyncColumnName() Some time you need to create column name that is not supported in SQLite (such as limit) or some variable name are not allowed in java such as class

By using SyncColumnName annotation framework will treat that column in different behaviour. For example, just create _class column and add annotation named with class.

Synchronization will done with class column name but stored in _class also you can treat it with _class name locally.

@Odoo.SyncColumnName("class")
OColumn _class = new OColumn("Class", OVarchar.class);

@Odoo.SyncColumnName("limit")
OColumn _limit = new OColumn("Limit", OInteger.class);

@Odoo.onChange() Compitable with OForm control only

Used when column value changed. It takes method name as first parameter and boolean value as second parameter if you want to execute onchange task in background process.

@Odoo.onChange(method="onParentIdChange")
OColumn parent_id = new OColumn("Company", ResPartner.class, RelationType.ManyToOne);
OColumn city = new OColumn("City", OVarchar.class);

public ODataRow onParentIdChange(ODataRow parent_id) {
    ODataRow newValues = new ODataRow();
    newValues.put("city", parent_id.getString("city"); // get city from parent_id and returning
    return newValues;
}

1.3. Getting Started with Framework
OForm call it automatically and fill the values in form object.

@Odoo.Functional() One can define a field whose value is computed instead of simply being read from the database.

Takes three parameters:

1. method, name of method
2. store, boolean flag for storing value in database (if true, database will create column)
3. depends, array of string, depended column names

```java
OColumn first_name = new OColumn("First name", OVarchar.class);
OColumn last_name = new OColumn("Last name", OVarchar.class);

@Odoo.Functional(method="storeDisplayName", depends = {"first_name","last_name"}, store = true)
OColumn display_name = new OColumn("Display name", OVarchar.class).setLocalColumn();

public String storeDisplayName(OValues values){
    String displayName = "";
    displayName = values.getString("first_name");
    displayName += " " + values.getString("last_name");
    return displayName;
}
```

**Info:**

For ManyToOne, ManyToMany and OneToMany values will be different.

**ManyToOne**

```java
public String storeManyToOne(OValues values){
    String manyToOne = "";
    if(!values.getString("parent_id").equals("false")){
        List<Object> parent_id = (ArrayList<Object>) value.get("parent_id");
        // Here, list index 0 contain record id (server id), and 
        // list index 1 contains record name
        manyToOne = parent_id.get(1).toString();
    }
    return manyToOne;
}
```

**ManyToMany or OneToMany**

```java
public int storeChildCount(OValues values){
    if(!values.getString("child_ids").equals("false")){
        // Contains list of ids (server ids)
        return ((ArrayList<Object>) values.get("child_ids")).size();
    }
    return 0;
}
```

@Odoo.hasDomainFilter() In some cases, you need to filter your record depended on some value change at runtime. For example, by changing country, states are loaded related to country.

By using hasDomainFilter annotation you can deal with it.
Add column domain, and annotation. If system found domains with `hasDomainFilter` annotation it will be treated runtime. **Note: it works with OForm control only**

```java
OColumn country_id = new OColumn("Country", ResCountry.class, RelationType.ManyToOne);
@Odoo.hasDomainFilter()
OColumn state_id = new OColumn("State", ResStates.class, RelationType.ManyToOne)
    .addDomain("country_id","=" , this);
```

**OModel.java** All the model (database class) extends OModel class. It contains all database required methods. Also allow you to add column easily by declaring as member variable type OColumn

- It automatically create relation tables and maintain relations for records.
- Works with Content Provider so faster performance for loading data from SQLite database.
- Properly maintain local relation.

OModel is binded with own ORM. It easy and fast.
OModel support different datatypes which will create dynamic table with its type and return records as per its column type.

**Basic Data Types**

**OVarchar** A string of limited length. Default length : 64

```java
OColumn name = new OColumn("Name", OVarchar.class).setSize(100).setRequired();
```

**OInteger** An integer

```java
OColumn counter = new OColumn("Counter", OInteger.class);
```

**OBoolean** A boolean (true, false). Default false

```java
OColumn is_active = new OColumn("Active", OBoolean.class);
```

**OFloat** A floating point number.

```java
OColumn weight = new OColumn("Weight", OFloat.class);
```

**OText** A text field with no limit in length.

```java
OColumn body = new OColumn("Message body", OText.class);
```

**OHtml** A html (actual text) field with no limit in length.

```java
OColumn body = new OColumn("Message body", OHtml.class);
```

**ODate** A date. Stores yyyy-MM-dd formatted date or false if value not set
OColumn date = new OColumn("Date", ODate.class);

ODatetime Allows to store a date and the time of day in the same field. Stores yyyy-MM-dd HH:mm:ss formatted date or false if value not set
OColumn date = new OColumn("Date", ODateTime.class);

OBlob Allows to store a base64 data in database. Generally used by ir.attachment
OColumn image = new OColumn("Avatar", OBlob.class);

OSelection Allows to store a string value (i.e., key for selection). Used selection for parsing Label for stored key.
OColumn state = new OColumn("State", OSelection.class)
    .addSelection("draft", "Draft")
    .addSelection("confirm", "Confirmed")
    .addSelection("close", "Canceled")
    .addSelection("done", "Done");

OTimestamp Stores current date time to column.
OColumn order_date = new OColumn("Order date", OTimestamp.class);

Relation types

OneToMany One2many field; the value of such a field is the recordset of all the records in comodel_name such that the field inverse_name is equal to the current record.
It required Type as another model's class type and also required realted column (as ManyToOne in related model)
OColumn parent_id = new OColumn("Company", ResPartner.class, RelationType.ManyToOne);
OColumn child_ids = new OColumn("Contacts", ResPartner.class, RelationType.OneToMany).setRelatedColumn("parent_id");

ManyToOne The value of such a field is a recordset of size 0 (no record) or 1 (a single record).
OColumn parent_id = new OColumn("Company", ResPartner.class, RelationType.ManyToOne);

ManyToMany Many2many field; the value of such a field is the recordset.
OColumn tag_ids = new OColumn("Tags", NoteTag.class, RelationType.ManyToMany);

Base structure of class
• extends OModel class
• Contains columns, methods (custom method used for model)
```java
class ResPartner extends OModel{
    public ResPartner(Context context, OUser user){
        super(context,"res.partner",user);
    }
}
```

- Constructor with `Context` and `OUser` parameter only. Pass `model name` in `super`.
- This will create table with some base columns `_id`, `id`, `create_date`, `write_date` and more.

**Note:** Note that database is created when you first time run your application, or when you clean your data from app setting. You need to clean application data everytime when you update your database column.

Adding some columns

```java
class ResPartner extends OModel{
    OColumn name = new OColumn("Name", OVarchar.class);
    OColumn parent_id = new OColumn("Company", ResPartner.class, RelationType.ManyToOne);

    public ResPartner(Context context, OUser user){
        super(context,"res.partner",user);
    }
}
```

Note that, if you pass second parameter `null` while creating model object. It will take current active user object and treat all operation to current user database only.

You can add columns as your requirement. Framework will create each relation column table automatically. But if there is no any relation column for specific model and you need to create that table. You need to register it in `BaseModels.java` @See `Base models`

**Methods**

`setDefaultNameColumn()` Syntax:

```java
void setDefaultNameColumn(String nameColumn)
```

Used when default `name` column is different. Default takes name. Used for storing name column when ManyToOne record arrive.

```java
public class ResPartner extends OModel {
    OColumn display_name = new OColumn("Name", OVarchar.class);

    public ResPartner(Context context, OUser user){
        super(context, "res.partner", user);
        setDefaultNameColumn("display_name");
    }
}
```
getDefaultNameColumn()  Syntax:
String getDefaultNameColumn()
Alternative of setDefaultNameColumn() override getDefaultNameColumn() method for return default
name column.

```java
public class ResPartner extends OModel {
    OColumn display_name = new OColumn("Name", OVarchar.class);

    public ResPartner(Context context, OUser user) {
        super(context, "res.partner", user);
    }

    @Override
    public String getDefaultNameColumn() {
        return "display_name";
    }
}
```

setModelName()  Syntax:
void setModelName(String modelName)
In some cases, you need to change model name (before just creating database table) depends on odoo version.

```java
public class CalendarEvent extends OModel {
    ...
    ...

    public CalendarEvent(Context context, OUser user) {
        super(context, "calendar.event", user);

        // Model name different for calendar.event in odoo version 7.0
        if (getOdooVersion().getVersionNumber() == 7) {
            setModelName("crm.meeting");
        }
    }
}
```

getTableName()  Syntax:
String getTableName()
Returns, table name for model. (Generally, res.partner become res_partner)

```java
ResPartner partner = new ResPartner(mContext, null);
String tableName = partner.getTableName();
```

setHasMailChatter()  Syntax:
void setHasMailChatter(boolean hasChatter)
Used to enable mail chatter below OForm view. takes boolean parameter

```java
public class ResPartner extends OModel {
    ...
    ...
```
```java
public ResPartner(Context context, OUser user) {
    super(context, "res.partner", user);
    setHasMailChatter(true);
}
```

**getUser() Syntax:**

OUser getUser()

Used to get current active User object. returns OUser object

```java
ResPartner partner = new ResPartner(mContext, null);
OUser user = partner.getUser();

String userName = user.getName();
int userId = user.getUserId();
```

**getOdooVersion() Syntax:**

OdooVersion getOdooVersion()

Used to get current user's odoo version information.

```java
ResPartner partner = new ResPartner(mContext, null);
OdooVersion odooVersion = partner.getOdooVersion();

int versionNumber = getOdooVersion().getVersionNumber();
String versionType = getOdooVersion().getVersionType();
```

**getColumns() Syntax:**

List<OColumn> getColumns()

List<OColumn> getColumns(boolean local)

Used to get list of models column. returns, ArrayList<OColumn>

Takes one optional parameter boolean. If you want to get only local column or server columns

- local boolean

```java
ResPartner partner = new ResPartner(mContext, null);

// Getting all columns
for (OColumn column: partner.getColumns()){
    Log.i(column.getName(), column.getLabel());
}

// Getting local columns
for (OColumn column: partner.getColumns(true)){
    Log.i(column.getName(), column.getLabel());
}

// Getting server columns
for (OColumn column: partner.getColumns(false)){
```
Log.i(column.getName(), column.getLabel());
}

**getRelationColumns() Syntax:**

```java
List<OColumn> getRelationColumns()
```

Used when you need to get all relation columns. ManyToMany, ManyToOne and OneToMany

```java
ResPartner partner = new ResPartner(mContext, null);
for (OColumn column : partner.getRelationColumns()){
    Log.i(column.getName(), column.getLabel());
}
```

**getColumn() Syntax:**

```java
OColumn getColumn(String columnName)
```

Used to get OColumn object by using its name

```java
ResPartner partner = new ResPartner(mContext, null);
OColumn display_name = partner.getColumn("display_name");
```

Note, if annotations applied and version not compatible with that column, it will returns null

**getFunctionalColumns() Syntax:**

```java
List<OColumn> getFunctionalColumns()
```

Returns all functional columns (with annotation @Odoo.Functional)

```java
ResPartner partner = new ResPartner(mContext, null);
for (OColumn column : partner.getFunctionalColumns()){
    Log.i(column.getName(), column.getLabel());
}
```

**getManyToManyColumns() Syntax:**

```java
List<OColumn> getManyToManyColumns(OModel relationModel)
```

Returns list of OColumn for many to many table. Takes relation model object as parameter

```java
ResPartner partner = new ResPartner(mContext, null);
for (OColumn col : partner.getManyToManyColumns(new Tags(mContext, null))){
    Log.i(col.getName(), col.getLabel());
}
```

**createInstance() Syntax:**

```java
OModel createInstance(Class<?> type)
```

Used to create OModel object related to model class type.
ResPartner partner = new ResPartner(mContext, null);

Tags tags = (Tags)partner.createInstance(Tags.class);

It will take Context and User from ResPartner object

OModel.get() Syntax:

OModel OModel.get(Context context, String modelName, String userAndroidName)

Used to create OModel object for particular model by its model name.

Takes, three parameters:

1. Context context object
2. String model name (i.e, res.partner or mail.message)
3. String user account name (You can see it under account of android settings). Made of username and database E.g., username: admin, database: production => account name: admin[production] or you can get it by user object. OUser.getAndroidName()

OUser user = OUser.current(mContext);
ResPartner partner = (ResPartner) OModel.get(mContext, "res.partner", user.getAndroidName());

authority() Syntax:

String authority()

returns, default authority. Used by BaseModelProvider. can be change if you have custom ContentProvider

ResPartner partner = new ResPartner(mContext, null);
String authority = partner.authority();

uri() Syntax:

Uri uri()

Returns, model Uri object. Works with BaseModelProvider

ResPartner partner = new ResPartner(mContext, null);

Cursor cr = getContentResolver().query(
   partner.uri(), // URI
   null, // Projection
   null, // Selection
   null, // Selection arguments
   null // sort order
);

buildURI() Syntax:

Uri buildURI(String authority)

Used to create custom uri with different authority and path segments.
public class ResPartner extends OModel {

    public static final String CUSTOMER_FILTER = "com.odoo.base.addons.res.res_partner";
    ...

    public ResPartner(Context context, OUser user) {
        super(context, "res.partner", user);
    }

    @Override
    public Uri uri() {
        return buildURI(CUSTOMER_FILTER);
    }

    public Uri indianCustomers() {
        return uri().buildUpon().appendPath("in_customer_filter").build();
    }
}

projection() Syntax:
String[] projection()
String[] projection(boolean onlyServerColumns)

Returns string array of columns used as projection to ContentResolver query() method.
Optional parameter for getting only local column projection or server column projection.

ResPartner partner = new ResPartner(mContext, null);

Cursor cr = getContentResolver().query(
    partner.uri(), // URI
    partner.projection(), // Projection
    null, // Selection
    null, // Selection arguments
    null // sort order
);

Database Operations

getLabel() Returns label for selection value. Works with OSelection type

CRMLead leads = new CRMLead(mContext, null);

ODataRow row = leads.browse(1);

String stateLabel = leads.getLabel("state", row.getString("state"));

browse() Returns DataRow object for record. null if no record found

ResPartner partner = new ResPartner(mContext, null);
ODataRow row = partner.browse(2); // Here, 2 is belong to SQLite local auto incremented id i.e, _id

browse() with projection:
```java
ResPartner partner = new ResPartner.mContext, null);
ODataRow row = partner.browse(new String[]{
    "name", "parent_id"
}, 2);

browse() with projection, selection and selection arguments

ResPartner partner = new ResPartner.mContext, null);
List<ODataRow> rows = partner.browse(
    new String[]{"name","parent_id"}, // Projection
    "city = ?", // Selection
    new String[]{"Gandhinagar"} // Selection arguments
);

getServerIds()
returns list of server ids.

ResPartner partner = new ResPartner.mContext, null);
List<Integer> serverIds = partner.getServerIds();

isEmptyTable() returns true, if table is empty.

ResPartner partner = new ResPartner.mContext, null);
if(partner.isEmptyTable()){
    // Do synchronization stuff
}

select() Select all records from database
returns, list of ODataRow

ResPartner partner = new ResPartner.mContext, null);
List<ODataRow> rows = partner.select();
for(ODataRow row: rows){
    // code of block
}

select() with projection

ResPartner partner = new ResPartner.mContext, null);
List<ODataRow> rows = partner.select(
    new String[]{"name","parent_id","city"}
);
for(ODataRow row: rows){
    // code of block
}

select() with projection, selection and selection arguments

1.3. Getting Started with Framework 37
ResPartner partner = new ResPartner(mContext, null);

List<ODataRow> rows = partner.select(
    new String[]{"name","parent_id","city"},
    "city = ?",
    new String[]{"Gandhinagar"})
);

for(ODataRow row: rows){
    // code of block
}

insertOrUpdate()  Syntax:
int insertOrUpdate(int serverId, OValues values)
int insertOrUpdate(String selection, String[] selectionArgs, OValues values)
Creates new record if not exists or update if exists

ResPartner partner = new ResPartner(mContext, null);

OValues values = new OValues();
values.put("id",1);
values.put("name", "Dharmang Soni");

int newId = partner.insertOrUpdate(1, values);

insertOrUpdate() with selection and selection arguments

ResPartner partner = new ResPartner(mContext, null);

OValues values = new OValues();
values.put("id",1);
values.put("name", "Dharmang Soni");

int newId = partner.insertOrUpdate("id = ?", new String[]{"1"}, values);
**insert()** create new record with values. returns new created id if successfull, otherwise `OModel.INVALID_ROW_ID` i.e., -1

```java
ResPartner partner = new ResPartner(mContext, null);
OValues values = new OValues();
values.put("id",1);
values.put("name", "Dharmang Soni");
int newId = partner.insert(values);
```

**delete()**  Delete record from local. Server record will be deleted when synchronization done.

**Syntax:**
- delete(int row_id)
- delete(String selection, String[] selectionArgs)

```java
ResPartner partner = new ResPartner(mContext, null);
int count = partner.delete(5);
// or
int count = partner.delete(OColumn.ROW_ID + " = ?", new String[]{"5"});
```

**selectServerId()**  Syntax:

```java
int selectServerId(int row_id)
```

returns server id for local record.

If record not found, returns `OModel.INVALID_ROW_ID` i.e., -1

```java
ResPartner partner = new ResPartner(mContext, null);
int serverId = partner.selectServerId(2);
```

**selectManyToManyRecords()**  Syntax:

```java
List<ODataRow> selectManyToManyRecords(String[] projection, String column_name, int row_id)
```

Returns list of many to many relation records for column and row.

```java
ResPartner partner = new ResPartner(mContext, null);
List<ODataRow> tags = partner.selectManyToManyRecords(new String[]{"name"}, "tag_ids", 2);
```

Here, `projection` is for related table (i.e., tags).

**count()**  Syntax:

```java
int count(String selection, String[] selectionArgs)
```

Returns number or record affecting selection.
ResPartner partner = new ResPartner(mContext, null);
int total = partner.count("is_company = ?", new String[]{"true"});

update() Syntax:
int update(String selection, String[] args, OValues values)
int update(int row_id, OValues values)
Update record value.
ResPartner partner = new ResPartner(mContext, null);
OValues values = new OValues();
values.put("name", "Parth Gajjar");
int updated = partner.update(5, values);

query() Syntax:
List<ODataRow> query(String sql)
List<ODataRow> query(String sql, String[] args)
Returns list of record generated by query.
ResPartner partner = new ResPartner(mContext, null);
String sql = "SELECT _id, name, city FROM res_partner WHERE country_id = ?";
List<ODataRow> records = partner.query(sql, new String[]{"4"});

executeRawQuery() Syntax:
Cursor executeRawQuery(String sql, String[] args)
Used to execute raw queries.
ResPartner partner = new ResPartner(mContext, null);
Cursor cr = partner.executeRawQuery("select * from res_partner where customer = ?", new String[]{"true"});

executeQuery() Syntax:
void executeQuery(String sql)
Execute queries. DROP TABLE, CREATE TABLE, etc...

getName() Syntax:
String getName(int row_id)
Returns name column value for record. (See setDefaultNameColumn() and getDefaultNameColumn())
ResPartner partner = new ResPartner(mContext, null);
String name = partner.getName(3);
countGroupBy() Syntax:

```
ODataRow countGroupBy(String column, String group_by, String having, String[] args)
```

Returns ODataRow object with total column contains total number of records

```java
ResPartner partner = new ResPartner(mContext, null);
int total = partner.countGroupBy("parent_id", "parent_id", "parent_id != ?", new String[]{"false"});
```

Synchronization related methods

getLastSyncDateTime() Syntax:

```
String getLastSyncDateTime()
```

Returns last synchronized date time for model.

```java
ResPartner partner = new ResPartner(mContext, null);
String lastSyncDateTime = partner.getLastSyncDateTime();
```

defaultDomain() Syntax:

```
ODomain defaultDomain()
```

Returns default domain for model called when synchronization occurred. Default it return only blank object new ODomain() with no domain filter. You need to override this method for return you default domain filter for model.

```java
class ResPartner extends OModel {
    ....
    ....
    ....
    @Override
    public ODomain defaultDomain(){
        ODomain domain = new ODomain();
        domain.put("customer","=",true);
        return domain;
    }
}
```

checkForCreateDate() Syntax:

```
boolean checkForCreateDate()
```

Return true if you need to check for create date on synchronization. It will filter create_date with sync limit from setting.

If return false, all the data are re-synchronized every time.

Default is True
class ResPartner extends OModel {
    ...
    ...
    ...

    @Override
    public boolean checkForCreateDate() {
        return true;
    }
}

checkForWriteDate() Syntax:

boolean checkForWriteDate()

Return true if you need to compare each of record with write_date. It will reduce the traffic for requests. Only different write_date records are synchronized.

If return false, nothing will checked with write_date

Default True

class ResPartner extends OModel {
    ...
    ...
    ...

    @Override
    public boolean checkForWriteDate() {
        return true;
    }
}

allowUpdateRecordOnServer() Syntax:

boolean allowUpdateRecordOnServer()

If true, framework will update record on server if local record dirty and write_date is newer that server’s write_date.

If false, framework will never update record on server with locally changed values.

Default True

class ResPartner extends OModel {
    ...
    ...
    ...

    @Override
    public boolean allowUpdateRecordOnServer() {
        return true;
    }
}

allowCreateRecordOnServer() Syntax:

boolean allowCreateRecordOnServer()
If true, framework will create record on server if it found 0 value to id column. (i.e., locally created records)
If false, framework will never create record on server.
Default True

class ResPartner extends OModel {
    ...
    ...
    ...
    
    @Override
    public boolean allowCreateRecordOnServer(){
        return true;
    }
}

 allowDeleteRecordOnServer() Syntax:
boolean allowDeleteRecordOnServer()
If true, framework will delete record from server if it locally inactive (deleted by user locally)
If false, framework will never delete record from server
Default True

class ResPartner extends OModel {
    ...
    ...
    ...
    
    @Override
    public boolean allowDeleteRecordOnServer(){
        return true;
    }
}

 allowDeleteRecordInLocal() Syntax:
boolean allowDeleteRecordInLocal()
If true, framework will remove local record if server record not exist.
If false, framework will not remove any local record if server record not exist.
Default True

class ResPartner extends OModel {
    ...
    ...
    ...
    
    @Override
    public boolean allowDeleteRecordInLocal(){
        return true;
    }
}
onSyncStarted() and onSyncFinished() Syntax:

```java
void onSyncStarted()
void onSyncFinished()
```

Used when you need to perform any operation on sync start and finish.

```java
class ResPartner extends OModel {
    ...
    ...
    ...

    @Override
    public void onSyncStarted() {
        // Code of block
    }

    @Override
    public void onSyncFinished() {
        // code of block
    }
}
```

quickSyncRecords() Syntax:

```java
void quickSyncRecords(ODomain domain)
```

Used when you need to synchronize some of records depends on some domain. You need to run this method in AsynTask or any background service.

```java
new AsyncTask<Void, Void, Void> {
    public Void doInBackground(Void... args) {
        ResPartner partner = new ResPartner(mContext, null);

        ODomain domain = new ODomain();
        domain.add("is_company", "=", true);

        partner.quickSyncRecords(domain);

        return null;
    }
}.execute();
```

quickCreateRecord() Syntax:

```java
void ODataRow quickCreateRecord(ODataRow row)
```

Return fully synced record data row object. You need to pass datarow object which contain id column with its server id value. Method will sync full record and return updated object.

```java
new AsyncTask<Void, Void, Void> {
    public Void doInBackground(Void... args) {
        ResPartner partner = new ResPartner(mContext, null);

        ODataRow row = new ODataRow();
        row.put("id", 49);
```
row = partner.quickCreateRecord(row);  
return null;
}  
].execute();

Other

isInstalledOnServer()  Syntax:
void isInstalledOnServer(String module_name, IModuleInstallListener callback)
Check for module installed on server or not.

ResPartner partner = new ResPartner(mContex, null);  
partner.isInstalledOnServer("notes", new IModuleInstallListener() {
  @Override
  public void installedOnServer(boolean isInstalled) {
    // isInstalled ?
  }
});

getServerDataHelper()  Syntax:
ServerDataHelper getServerDataHelper()
Used to perform some of server operations (works with live network only).
Contains following methods:
• getOdoo() : Returns Odoo object
• nameSearch() : Name search on server
• read() : Read record from server
• searchRecords() : search records with fields, domain and limit
• executeWorkFlow() : execute server workflow with signal
• callMethod() : call model’s custom methods
• createOnServer() : quick create record on server (not create locally)
• updateOnServer() : quick update record on server (not update locally)

BaseModelProvider.java  Provide base database operation with ContentResolver,
extends ContentProvider and works with Uri
We required to use BaseModelProvider when creating custom sync service for model.

public class CustomersSyncProvider extends BaseModelProvider {

  @Override
  public String authority() {
    return ResPartner.AUTHORITY;
  }
}
Adding custom Uri

• Register New Uri in model:

```java
class MailMessage extends OModel {
    public static final String AUTHORITY = "your.custom.authority";
    ...  
    ...  

    @Override
    public Uri uri() {
        return buildURI(AUTHORITY);
    }

    public Uri inboxURI() {
        return uri().buildUpon().appendPath(MailProvider.KEY_INBOX_MESSAGES).build();
    }
}
```

• Adding Uri to MailProvider

```java
public class MailProvider extends BaseModelProvider {
    public static final String TAG = MailProvider.class.getSimpleName();
    public static final int INBOX_MESSAGES = 234;
    public static final String KEY_INBOX_MESSAGES = "inbox_messages";

    @Override
    public boolean onCreate() {
        String path = new MailMessage(getContext(), null).getModelName()
            .toLowerCase(Locale.getDefault());
        matcher.addURI(authority(), path + "" + KEY_INBOX_MESSAGES, INBOX_MESSAGES);
        return super.onCreate();
    }

    @Override
    public void setModel(Uri uri) {
        super.setModel(uri);
        mModel = new MailMessage(getContext(), getUser(uri));
    }

    @Override
    public String authority() {
        return MailMessage.AUTHORITY;
    }

    @Override
    public Cursor query(Uri uri, String[] base_projection, String selection, String[] selectionArgs, String sortOrder) {
        int match = matcher.match(uri);
        if (match != INBOX_MESSAGES)
            return super.query(uri, base_projection, selection, selectionArgs, sortOrder);
    }
}
```
else {
    MailMessage mail = new MailMessage(getContext(), getUser(uri));
    return mail.executeRawQuery("select * from mail_message", null);
}
}

**OSyncService.java**  Provide support for managing your sync requests and perform operation with your model data. When uses sync adapter for custom sync service. Use **OSyncService** We need to extends **OSyncService** class which implement two methods:

1. getSyncAdapter()
2. performDataSync()

**getSyncAdapter()**  Syntax:

```java
OSyncAdapter getSyncAdapter(OSyncService serviceObj, Context context)
```

Used to return initial sync adapter object. (do not use chaining methods in this method)

```java
public class CustomerSyncService extends OSyncService {

    @Override
    public OSyncAdapter getSyncAdapter(OSyncService service, Context context) {
        return new OSyncAdapter(context, ResPartner.class, this, true);
    }

    @Override
    public void performDataSync(OSyncAdapter adapter, Bundle extras, OUser user) {
    }
}
```

**performDataSync()**  Called just before data sync start. You can put some filters (domains), limiting data request in this method. Also you can specify the next sync operation after its sync finish.

```java
public class CustomerSyncService extends OSyncService {

    public static final String TAG = CustomerSyncService.class.getSimpleName();

    @Override
    public OSyncAdapter getSyncAdapter(OSyncService service, Context context) {
        return new OSyncAdapter(context, ResPartner.class, service, true);
    }

    @Override
    public void performDataSync(OSyncAdapter adapter, Bundle extras, OUser user) {

        ODomain domain = new ODomain();
        domain.add("active","=" ,true);
        adapter.syncDataLimit(80).setDomain(domain);
    }
}
```

Specify next sync operation for different model
```java
public class CustomerSyncService extends OSyncService implements ISyncFinishListener {
    public static final String TAG = CustomerSyncService.class.getSimpleName();
    private Context mContext;
    @Override
    public OSyncAdapter getSyncAdapter(OSyncService service, Context context) {
        mContext = context;
        return new OSyncAdapter(context, ResPartner.class, service, true);
    }

    @Override
    public void performDataSync(OSyncAdapter adapter, Bundle extras, OUser user) {
        if (adapter.getModel().getModelName().equals("res.partner")) {
            ODomain domain = new ODomain();
            domain.add("active", ",", true);
            adapter.syncDataLimit(80).setDomain(domain);
            adapter.onSyncFinish(this);
        }
    }

    @Override
    public OSyncAdapter performNextSync(OUser user, SyncResult syncResult) {
        return new OSyncAdapter(mContext, ResCountry.class, this, true);
    }
}
```

Here, You can see we have checked for model name. Each time when you have chaining sync adapters you need to check for model name in `performDataSync`. Otherwise, your service will go in infinite loop.

`onSyncFinish()` will tell service to perform next operation and service will continue to complete all the task in chain.

Note: You have to check everytime for each of model to pass domain and filters by getting its model name.

If you dont wont to continue with next sync operation but you need to do some operation after sync finish. You can do it in two ways:

1. Just add sync finish call back and return `null`. Do your operation before returning `null` or
2. Just override `onSyncFinished()` method in your model. Where you can perform your operations.

**Drawer Menu**

The navigation drawer is a panel that transitions in from the left edge of the screen and displays the app’s main navigation options.

Odoo mobile framework provide feature to add dynamic menu to your application just by providing list of menus.
1.3. Getting Started with Framework
By extending `BaseFragment` class you are allowed to implement `drawerMenu()` method. Which returns list of `DrawerItem` with key, title and your object of fragment or class type for Activity.

Here is simple example for loading drawer items:

```java
class Customers extends BaseFragment {
    ...
    ...

    @Override
    public List<ODrawerItem> drawerMenus(Context context) {
        List<ODrawerItem> items = new ArrayList<>();
        items.add(new ODrawerItem(KEY).setTitle("Customers")
            .setIcon(R.drawable.ic_action_customers)
            .setExtra(extra(Type.Customer))
            .setInstance(new Customers()));
        items.add(new ODrawerItem(KEY).setTitle("Suppliers")
            .setIcon(R.drawable.ic_action_suppliers)
            .setExtra(extra(Type.Supplier))
            .setInstance(new Customers()));
        items.add(new ODrawerItem(KEY).setTitle("Companies")
            .setIcon(R.drawable.ic_action_company)
            .setExtra(extra(Type.Company))
            .setInstance(new Customers()));
        return items;
    }

    public Bundle extra(Type type) {
        Bundle extra = new Bundle();
        extra.putString(EXTRA_KEY_TYPE, type.toString());
        return extra;
    }
    ...
    ...
}
```

**ODrawerItem.java**  
`ODrawerItem` class contains information for your menu item.

**Syntax:**

`ODrawerItem(String key)`

This class contains chaining methods such as:

- `setTitle()`
- `setGroupTitle()`
- `setCounter()`
- `setInstance()`
- `setExtra()`
- `setIcon()`

`setTitle()` Sets title for drawer menu.

**Syntax:**

```
```
```java
ODrawerItem setTitle(String title)

ODrawerItem item = new ODrawerItem("my_menu")
    .setTitle("Dashboard");

setGroupTitle() Works as separator.
Syntax:

ODrawerItem setGroupTitle()

ODrawerItem item = new ODrawerItem("my_menu")
    .setTitle("Dashboard")
    .setGroupTitle();

setCounter() Shows number of record at right of menu.
Syntax:

ODrawerItem setCounter(int counter)

ODrawerItem item = new ODrawerItem("my_menu")
    .setTitle("Dashboard")
    .setCounter(10);

setInstance() Sets loading fragment instance or activity class.
Syntax:

ODrawerItem setInstance(Fragment fragment)
ODrawerItem setInstance(Class<? extends Activity> classType)
ODrawerItem setInstance(Class<? extends ActionbarActivity> classType)
ODrawerItem setInstance(Class<? extends FragmentActivity> classType)

ODrawerItem dashboard = new ODrawerItem("dashboard")
    .setTitle("Dashboard")
    .setCounter(10)
    .setInstance(new Dashboard());

ODrawerItem settings = new ODrawerItem("settings")
    .setTitle("Settings")
    .setInstance(SettingActivity.class);

setExtra() Sets extra bundle passed with fragment or activity.
Syntax:

ODrawerItem setExtra(Bundle data)
Bundle data = new Bundle();
data.putInt("record_id", 10);

ODrawerItem item = new ODrawerItem("my_menu")
    .setTitle("Projects")
    .setInstance(new Projects())
    .setExtra(data);
```

1.3. Getting Started with Framework
**setIcon()**  Shows icon for menu

**Syntax:**

```java
ODrawerItem setIcon(int resource_id)
ODrawerItem item = new ODrawerItem("my_key")
    .setTitle("Projects")
    .setInstance(new Projects())
    .setIcon(R.drawable.ic_action_projects);
```

**Addons**

Here, We take example for Project tasks.

**Creating addon with odoo mobile**  Each of the addon in the odoo mobile app is the feature for your application. It contains, models, providers (for synchronization service and database operation), background services, **BaseFragment** (your addon startup class fragment), another **Activities** or **Fragments** as you required and other utilities used by your addon.

Here you can see one example with some of models, services, providers and other classes.
Your addon main class extends `BaseFragment` and implement two methods:

1. `drawerMenus()`
2. `database()`

`drawerMenus()` For your addons navigation menus, `BaseFragment` provide method to be implement for generating menu under application drawer. Returns list of `DrawerItem`
Tasks

- Customers
- Suppliers
- Companies

SETTINGS

- Profile
- Settings
```java
class Tasks extends BaseFragments {

    ... 
    ...
    ...

    @Override
    public List<ODrawerItem> drawerMenus(Context context) {
        List<ODrawerItem> menu = new ArrayList<>();
        menu.add(new ODrawerItem(TAG).setTitle("Tasks").setInstance(new Tasks()));
        return menu;
    }
}

@See more Drawer Menu

database()  Database method return class type of base database model. It used for creating database tables when application start first time.
return type class for model.

```java
public class Tasks extends BaseFragment {
    public static final String TAG = Tasks.class.getSimpleName();

    @Override
    public List<ODrawerItem> drawerMenus(Context context) {
        List<ODrawerItem> menu = new ArrayList<>();
        menu.add(new ODrawerItem(TAG).setTitle("Tasks").setInstance(new Tasks()));
        return menu;
    }

    @Override
    public Class<ProjectTask> database() {
        return ProjectTask.class;
    }
}

Creating Models for addon Here, we are going to create ProjectProject model class for project.project and ProjectTask model class for project.task under com.odoo.addons.projects.models package

```java
public class ProjectProject extends OModel {
    public static final String TAG = ProjectProject.class.getSimpleName();

    OColumn name = new OColumn("Name", OVarChar.class).setSize(100);

    public ProjectProject(Context context, OUser user) {
        super(context, "project.project", user);
    }
}

public class ProjectTask extends OModel {
    public static final String TAG = ProjectTask.class.getSimpleName();

    OColumn name = new OColumn("Name", OVarChar.class).setSize(100);
    OColumn project_id = new OColumn("Project", ProjectProject.class, OColumn.RelationType.ManyToOne);
}
```

1.3. Getting Started with Framework
ProjectTask contains `project_id` column related to `ProjectProject` class type with ManyToOne relation.

We have passed `ProjectTask.class` to `database()` method. So, when framework creating database, it will take all the relation models in columns and create its master table.

Now, for running app. You need to register your main class to addons registry as below.

### Registering addon to Addons registry

Each of the modules (addons) are registered under `Addons.java` class of `com.odoo.config` package.

```java
public class Addons extends AddonsHelper {

    OAddon customers = new OAddon(Customers.class).setDefault();
    OAddon tasks = new OAddon(Tasks.class);
}
```

To make tasks default just add chaining method, `setDefault()`

To short menu, by addons just rename your addons variable in alphabetical order.

```java
public class Addons extends AddonsHelper {

    OAddon a_tasks = new OAddon(Tasks.class).setDefault();
    OAddon b_customers = new OAddon(Customers.class);
}
```

### Creating Sync service

The sync service component in app encapsulates the code for the tasks that transfer data between the device and a server. Based on the scheduling and triggers provided by application, the sync service framework runs the code in the sync adapter component and perform database operation with received data from server. To create sync service for your addon, you need to add the following pieces:

- Sync Service class (extends `OSyncService` class)
- Custom database provider class with your AUTHORITY (extends `BaseModelProvider` class)
- Sync adapter XML metadata file.
- Declarations in the app manifest.

#### Sync Service class

A component that allows the sync framework to run the code in your sync adapter class for given model.

@See more `O$SyncService.java`

```java
package com.odoo.addons.projects.services;

import android.content.Context;
import android.os.Bundle;
import com.odoo.addons.projects.models.ProjectTask;
```
Custom database provider class with your AUTHORITY (extends BaseModelProvider class)  

The sync adapter framework is designed to work with device data managed by the flexible and highly secure content provider framework. For this reason, the sync adapter framework expects that an app that uses the framework has already defined a content provider for its local data. If the sync adapter framework tries to run your sync adapter, and your app doesn’t have a content provider, your sync adapter crashes.

Here, odoo mobile framework have pre defined methods and mechanism to handle your data with content provider. The one BaseModelProvider is central ContentProvider for all models with base AUTHORITY shared with each of the model.

But, in case of creating custom sync service we required different content provider to be registerd in Android Manifest file. To do so, we just need to extend BaseModelProvider and provide our custom AUTHORITY by overriding authority() method.

Here is snippet:

Adding custom AUTHORITY to Model

```java
public class ProjectTask extends OModel {
    public static final String TAG = ProjectTask.class.getSimpleName();
    public static final String AUTHORITY = "com.odoo.addons.projects.project_tasks";

    OColumn name = new OColumn("Name", OVarchar.class).setSize(100);
    OColumn project_id = new OColumn("Project", ProjectProject.class, OColumn.RelationType.ManyToOne);
    OColumn description = new OColumn("Description", OText.class);

    public ProjectTask(Context context, OUser user) {
        super(context, "project.task", user);
    }
    @Override
    public Uri uri() {
        return buildURI(AUTHORITY);
    }
}
```

Creating provider class:

```java
package com.odoo.addons.projects.providers;
```

1.3. Getting Started with Framework
import com.odoo.addons.projects.models.ProjectTask;
import com.odoo.core.orm.provider.BaseModelProvider;

public class ProjectTaskProvider extends BaseModelProvider {
    public static final String TAG = ProjectTaskProvider.class.getSimpleName();

    @Override
    public String authority() {
        return ProjectTask.AUTHORITY;
    }
}

Sync adapter XML metadata file  A file containing information about your sync adapter. The framework reads this file to find out how to load and schedule your data transfer.

To plug your sync adapter component into the framework, you need to provide the framework with metadata that describes the component and provides additional flags. The metadata specifies the account type you’ve created for your sync adapter, declares a content provider authority associated with your app, controls a part of the system user interface related to sync adapters, and declares other sync-related flags. Declare this metadata in a special XML file stored in the /res/xml/ directory in your app project. You can give any name to the file, although it’s usually called syncadapter.xml.

See more at: https://developer.android.com/training/sync-adapters/creating-sync-adapter.html#CreateSyncAdapterMetadata

<sync-adapter xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="com.odoo.auth"
    android:contentAuthority="com.odoo.addons.projects.project_tasks"
    android:supportsUploading="true"
    android:userVisible="true" />

Declarations in the app manifest  Registering Service in Manifest file

<service android:name="addons.projects.services.ProjectSyncService"
    android:exported="true"
    android:process=":sync_tasks">
    <intent-filter>
        <action android:name="android.content.SyncAdapter" />
    </intent-filter>
    <meta-data
        android:name="android.content.SyncAdapter"
        android:resource="@xml/task_sync_adapter" />
</service>

Registering Provider for sync service :

<provider
    android:name="com.odoo.addons.projects.providers.ProjectTaskProvider"
    android:authorities="com.odoo.addons.projects.project_tasks"
    android:label="Project Tasks"
    android:multiprocess="true" />

All Done!
Launch your application by cleaning app data. (need to clean because we have updated database.)
You can see sync option for project tasks under your account:
1.3. Getting Started with Framework
Working with `R.layout.common_listview` layout  

`common_listview` layout contains `ListView` with `SwipeRefreshLayout` and empty view layout. If you need to load list of rows on your fragment load you can use this common list view which can be easy to integrate with `BaseFragment` as shown below:

Inflating view

```java
public class Tasks extends BaseFragment {
    public static final String TAG = Tasks.class.getSimpleName();

    private View mView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.common_listview, container, false);
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        mView = view;
    }

    // ... ...
}
```

Declare and initialize controls

```java
public class Tasks extends BaseFragment {
    public static final String TAG = Tasks.class.getSimpleName();

    private View mView;
    private ListView listView;
    private OCursorListAdapter listAdapter;

    // ...
    ...

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        mView = view;
        listView = (ListView) mView.findViewById(R.id.listview);
        listAdapter = new OCursorListAdapter(getActivity(), null, android.R.layout.simple_list_item_1);
        listView.setAdapter(listAdapter);
    }

    // ...
    ...
}
```

Registering loader manager

```java
public class Tasks extends BaseFragment implements LoaderManager.LoaderCallbacks<Cursor> {
    public static final String TAG = Tasks.class.getSimpleName();

    // ...
    ...
```
@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    mView = view;
    listView = (ListView) mView.findViewById(R.id.listview);
    listAdapter = new OCursorListAdapter(getActivity(), null, android.R.layout.simple_list_item_1);
    listView.setAdapter(listAdapter);
    getLoaderManager().initLoader(0, null, this);
}

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    return new CursorLoader(getActivity(), db().uri(), null, null, null, null);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    listAdapter.changeCursor(data);
    if (data.getCount() > 0) {
        OControls.setGone(mView, R.id.loadingProgress);
        OControls.setVisible(mView, R.id.swipe_container);
        OControls.setGone(mView, R.id.no_items);
    } else {
        OControls.setGone(mView, R.id.loadingProgress);
        OControls.setGone(mView, R.id.swipe_container);
        OControls.setGone(mView, R.id.no_items);
        OControls.setText(mView, R.id.title, "No Tasks found");
        OControls.setText(mView, R.id.subTitle, "Swipe to check new tasks");
    }
    if (db().isEmptyTable()) {
        // Request for sync
    }
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    listAdapter.changeCursor(null);
}

... ...

Registering sync (SwipeRefresh) control and sync observer  @See more setHasSwipeRefreshView()

@See more setHasSyncStatusObserver()

Swipe refresh view listener:

public class Tasks extends BaseFragment implements LoaderManager.LoaderCallbacks<Cursor>, ISyncStatusObserverListener, SwipeRefreshLayout.OnRefreshListener {
    ...
    ...

    @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        listAdapter.changeCursor(data);
        if (data.getCount() > 0) {
            ...
            ...
            ...

    ... ...

1.3. Getting Started with Framework
OControls.setGone(mView, R.id.loadingProgress);
OControls.setVisible(mView, R.id.swipe_container);
OControls.setGone(mView, R.id.no_items);
setHasSwipeRefreshView(mView, R.id.swipe_container, this);

} else {
OControls.setGone(mView, R.id.loadingProgress);
OControls.setVisible(mView, R.id.swipe_container);
OControls.setGone(mView, R.id.no_items);
setHasSwipeRefreshView(mView, R.id.no_items, this);
OControls.setText(mView, R.id.title, "No Tasks found");
OControls.setText(mView, R.id.subTitle, "Swipe to check new tasks");

} if (db().isEmptyTable()) {
    // Request for sync
    onRefresh();
}

@Override
public void onRefresh() {
    if (inNetwork()) {
        parent().sync().requestSync(ProjectTask.AUTHORITY);
    }
}

... ...

Sync status observer:

public class Tasks extends BaseFragment implements LoaderManager.LoaderCallbacks<Cursor>, ISyncStatusObserverListener {
    ...
    ...

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        mView = view;
        listView = (ListView) mView.findViewById(R.id.listview);
        listAdapter = new OCursorListAdapter(getActivity(), null, android.R.layout.simple_list_item_1);
        listView.setAdapter(listAdapter);
        setHasSyncStatusObserver(TAG, this, db());
        getLoaderManager().initLoader(0, null, this);
    }

    @Override
    public void onStatusChange(Boolean changed) {
        if (changed) {
            getLoaderManager().restartLoader(0, null, this);
        }
    }

    ...
    ...

62 Chapter 1. Contents:
Binding View

```java
public class Tasks extends BaseFragment implements LoaderManager.LoaderCallbacks<Cursor>, ISyncStatusObserverListener, SwipeRefreshLayout.OnRefreshListener, OCursorListAdapter.OnViewBindListener {
    ...
    ...

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        mView = view;
        listView = (ListView) mView.findViewById(R.id.listview);
        listAdapter = new OCursorListAdapter(getActivity(), null, android.R.layout.simple_list_item_1);
        listView.setAdapter(listAdapter);
        listAdapter.setOnViewBindListener(this);
        setHasSyncStatusObserver(TAG, this, db());
        getLoaderManager().initLoader(0, null, this);
    }

    @Override
    public void onViewBind(View view, Cursor cursor, ODataRow row) {
        OControls.setText(view, android.R.id.text1, row.getString("name"));
    }
    ...
    ...
```

1.3. Getting Started with Framework
Chapter 1. Contents:

- Tasks
  - Prepare Requirements Document
  - Make SRS
  - Budget Planning
  - Develop module for Sale Management
  - Develop module for Warehouse
  - Integrate Modules
  - Internal testing + Software Install
  - New portal system
  - Document history management
  - Social network integration
  - User interface improvements
Controls (Form, Fields, actionbar spinner)

To provide faster application development, odoo mobile framework come with some of usefull controls which work with your model record, integrated with chatter view, different data types, on change methods, validations, live search and more.

**OForm widget**  OForm widget used when you need to show model record. It helps record to bind view as per its type. Uses OField as its inner widgets. You need to just add your fields under OForm controls.

It extends LinearLayout so you can easily change its layout property as per LinearLayout does.

Some of other property works with OForm:

- prefix:editableMode: boolean
- prefix:modelName: string
- prefix:autoUIGenerate: boolean
- prefix:controlIconTint: color

Here, prefix is your resource attribute reference.

```xml
<RelativeLayout
    xmlns:android=http://schemas.android.com/apk/res/android-
    xmlns:prefix=http://schemas.android.com/apk/res-auto-
    android:layout_width=match_parent-
    android:layout_height=match_parent-
>
    <odoo.controls.OForm
        android:id=@+id/customerFormEdit-
        prefix:modelName=res.partner-
        android:layout_width=match_parent-
        android:orientation=vertical-
        android:layout_height=wrap_content->
    <!-- OFields controls here //-->

    </odoo.controls.OForm>
</RelativeLayout>
```

You can take any of valid name for prefix, ADT default take app as prefix.

**What if you want to add other controls in OForm widget ?**

Yes, you can also add other controls such as ImageView, TextView or any other controls under OForm.

**editableModel**  Takes boolean value (true|false), default false. Not required property

Make your form work in editable mode or read only mode.

**modelName**  Takes string value (model name). Required property

OForm uses model name to bind your field property. Such as field label, its type and other properties.
**autoUIGenerate**  Takes boolean value (true|false). Default true  Not required property
Responsibility to generate responsive layout for your form fields. (works with OFields only, non of your other controls are affected by autoUIGenerate property)
 Adds icon (if you have provided to OField), label for field, proper spacing and marging and some other UI changes.

**controlIconTint**  Takes color reference value or color code. Not required property
Changes all your OField widget icon tint color.

**Initialize form widget**  OForm form = (OForm) view.findViewById(R.id.myForm);
Methods:

**initForm()**  Syntax:
void initForm(ODataRow record);
Initiate form with given record. If record == null, it will load default values if given and create form for new record.

OForm form = (OForm) view.findViewById(R.id.myForm);
form.initForm(null);

**setEditable()**  Syntax:
void setEditable(Boolean editable)
Changes form behaviour to editable/readonly at runtime.

OForm form = (OForm) view.findViewById(R.id.myForm);
form.initForm(null);
form.setEditable(true);

**loadChatter()**  Syntax:
void loadChatter(boolean loadChatter)
Loads chatter view at bottom of form. (If record is not synced on server chatter view is not loaded.).
This method must be called before initForm()

OForm form = (OForm) view.findViewById(R.id.myForm);
form.loadChatter(true);
form.initForm(record);

**setIconTintColor()**  Syntax:
void setIconTintColor(int color)
Changes fields icon tint color at runtime. This method must be called before initForm()

OForm form = (OForm) view.findViewById(R.id.myForm);
form.setIconTintColor(Color.MAGENTA);
form.initForm(record);
getValues() Syntax:

OValues getValues()

Returns form values, used when you need to create or update record. If returns null, may be validation failed.

OForm form = (OForm) view.findViewById(R.id.myForm);
form.initForm(record);

...
...

OValues updatedValues = form.getValues();
if(updatedValues != null){
    // Store updated values.
}
Interest in your Graphic Design Project

EXPECTED REVENUE
24000.0 € at 0.0%

NEXT ACTION
Mar 23, 2015
Send Catalogue by Email

EXPECTED CLOSING
Feb 23, 2015

CONTACT NAME
Contact Name

COMPANY NAME
Agrolait
**OField widget**  OField widget works with OForm widget. Each of OField is your model’s column or just dummy column.

If OForm autoUIGenerate flag is on, it will create UI with icon (if any), label and column input part. (input box, checkbox, radio button, datetime picker, many to one widget - spinner, and more^^^)

Some of properties you need to know before using OField control.

**fieldName : string**  Model’s column name or your dummy column name.

```xml
<odoo.controls.OField
    app:fieldName=-name-
    android:layout_height=-wrap_content-
    android:layout_width=-match_parent->
</odoo.controls.OField>
```

**iconResource : reference**  Field’s icon resource. Shows left of control.

```xml
<odoo.controls.OField
    android:layout_width=-match_parent-
    app:iconResource=-@drawable/ic_action_message-
    app:fieldName=-email-
    android:layout_height=-wrap_content->
</odoo.controls.OField>
```

**iconTint : reference|color**  Changes icon color. takes color reference or color code.

```xml
<odoo.controls.OField
    android:layout_width=-match_parent-
    app:iconResource=-@drawable/ic_action_message-
    app:fieldName=-email-
    app:iconTint=-@color/android_green-
    android:layout_height=-wrap_content->
</odoo.controls.OField>
```

**showIcon : boolean, showLabel : boolean**  Show/Hide icon and label. Takes true or false. Default is true

```xml
<odoo.controls.OField
    android:layout_width=-match_parent-
    app:iconResource=-@drawable/ic_action_message-
    app:fieldName=-email-
    app:showLabel=-false-
    android:layout_height=-wrap_content->
</odoo.controls.OField>
```

**parsePattern : string**  Used with type, Date, DateTime

```xml
<odoo.controls.OField
    android:layout_width=-match_parent-
    app:fieldName=-create_date-
    app:parsePattern=-MMM dd, yyyy hh:mm a-
    android:layout_height=-wrap_content->
</odoo.controls.OField>
```
**withOutsidePadding**: boolean  
Ignore auto UI generate side padding.

**fieldType**: enum  
Generally it automatically taken from Column type. But you can use when you are creating dummy field for OForm

Possible types:
- Text
- Text
- ManyToOne
- Selection
- Date
- DateTime
- Blob
- Time

**widgetType**: enum  
In some cases you need to change the control behaviour for your column. such as boolean; it can be shown as checkbox or radio or switch. You can specify your control behaviour by using widget type.

**Switch**  
Supported types: boolean
Makes your boolean field behave like switch

**RadioGroup**  
Supported types: boolean
Make your boolean field behave like radio button

**SelectionDialog**  
Supported types: ManyToOne, Selection
Makes dialog to select from available values refer to manytoone model or given selection.

**Searchable**  
Supported types: ManyToOne, Selection
Makes reference values searchable. Open other activity for provide search with available records

**SearchableLive**  
Supported types: ManyToOne
Makes reference value searchable even if there are no any local record regarding your search. If network available it will search on live server and when you click on that record it will available locally

**Image**  
Supported types: Blob
Consider your blob binary data as image and view image in control.

**ImageCircle**  
Same as Image, create circular image only.

**Duration**  
Supported types: Float
Converts float value to duration and display as duration.
**widgetImageSize**: `dimension`  
Changes image size (works with Widget type Image or ImageCircle)

**withBottomPadding**: `boolean`  
Ignore adding bottom padding by auto UI generator if provided false.

**withTopPadding**: `boolean`  
Ignore adding top padding by auto UI generator if provided false.

**controlLabel**: `string|reference`  
Label for control. Takes string or string reference.

**defaultValue**: `reference|string`  
Default value for control. If no data found from record for field it will takes default value.

**defaultImage**: `reference`  
Default image for Image widget type. If no image found.

**valueArray**: `reference`  
Used for dummy column, value array reference (works with selection type)

**fieldTextAppearance**: `reference`  
Field text appearance reference

**fieldTextSize**: `dimension`, **fieldLabelSize**: `dimension`  
Field text and label size in dimension.

**fieldTextColor**: `color`, **fieldLabelColor**: `color`  
Field text and label color

**fieldLabelTextAppearance**: `reference`  
Field label text appearance
Actionbar Spinner  Actionbar spinner provide quick filter and navigations. BaseFragment provide method to implement spinner to your fragment.

By calling following method your can activate spinner control for actionbar from `onViewCreated` method.

```java
parent().setHasActionBarSpinner(true);
...
...

@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    mView = view;
    parent().setHasActionBarSpinner(true);
}
...
...
```
When you call this method, OdooActivity activate spinner for your fragment and returns Spinner object by calling parent().getActionBarSpinner(); method after activating spinner for actionbar.

Now, you can easily manage your spinner with adding items by adapter and custom view.

**Chatter view**

<table>
<thead>
<tr>
<th>Taxes</th>
<th>590.00 €</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total:</td>
<td>3540.00 €</td>
</tr>
</tbody>
</table>

Send a message or Log an internal note

Quotation created

Dharmang Soni  Mar 16 12:31 pm

- Customer: Alpesh Patni
- Untaxed Amount: 2950.0
- Salesperson: Dharmang Soni

Dharmang Soni  Mar 16 12:31 pm

Full history on your record. OForm control integrated with Mail chatter for your record. You just need to activate chatter for your model and OForm will take care for loading chatter view for your record.

Supported features:

- Log internal note with attachments

1.3. Getting Started with Framework  73
• Send message to followers with attachments

Two way to enable chatter for your model:

**Enable from model**  
Call method `setHasMailChatter()` in model’s constructor.

```java
public class ResPartner extends OModel {
    ...
    ...
    public ResPartner(Context context, OUser user){
        super(context, "res.partner", user);
        setHasMailChatter(true);
    }
}
```

It will add chatter view for synced record in OForm automatically.

**Enable at runtime**  
OForm widget contains method to handle chatter view at runtime even if you have specified it in model. By calling `loadChatter()` before `initForm()`

```java
OForm form = (OForm) view.findViewById(R.id.myForm);
form.loadChatter(false);
form.initForm(record);
```

### 1.4 Contributing

If you would like to contribute code you can do so through GitHub by forking the repository and sending a pull request.

When submitting code, please make every effort to follow existing conventions and style in order to keep the code as readable as possible.
Indices and tables

- genindex
- search
Symbols
_s(), _c(), _dim(), 23

A
About, 1
actionbar spinner, 71
Add-on Registry, 14
Addons, 52
allowCreateRecordOnServer(), 42
allowDeleteRecordInLocal(), 43
allowDeleteRecordOnServer(), 43
allowUpdateRecordOnServer(), 42
Aplication Config, 8
authority(), 35
autoUIGenerate, 66

B
Base Models Intro, 16
BaseFragment.java, 20
BaseModelProvider.java, 45
Blue Layer, 12
browse(), 36
buildURI(), 35

C
Chatter View, 73
checkForCreateDate(), 41
checkForWriteDate(), 42
Clone Framework, 5
controlIconTint, 66
controlLabel, 71
Controls, 65
count(), 39
countGroupBy(), 41
Create Model, 55
createInstance(), 34

D
Datatypes, 29
db(), 20
defaultDomain(), 41
defaultImage, 71
defaultValue, 71
delete(), 39
Device testing, 2
Directory Structure, 12
Download Android Studio, 1
drawer menu, 48

E
editableMode, 65
executeQuery(), 40
executeRawQuery(), 40

F
fieldLabelColor, 71
fieldLabelSize, 71
fieldLabelTextAppearance, 71
fieldName, 69
fieldTextAppearance, 71
fieldTextColor, 71
fieldType, 70
First Step, 7
Framework Architecture, 8

G
getColumn(), 34
getColumns(), 33
gGetDefaultTableName(), 32
gGetFunctionalColumns(), 34
gGetLabel() Selection type, 36
gGetLastSyncDateTime(), 41
gGetManyToManyColumns(), 34
gGetName(), 40
gGetOdooVersion(), 33
gGetRelationColumns(), 34
gGetServerDataHelper(), 45
gGetServerIds(), 37
gGetSyncAdapter(), 47
user(), 20

V
v8, 27
valueArray, 71

W
widgetImageSize, 71
widgetType, 70
withBottomPadding, 71
withOutSidePadding, 70
withTopPadding, 71