**How to use developers manual:**

The introduction explains the existing package structure of the application and explains some of the problems and situations that might have an impact on the application further ahead.

Curious about “The current code structure of the application”, guess where to go.

Want to know how to add things in the application? Go to 3.

The future waits at 4.

**Table of contents:**

# 1. Introduction

## 1.1. Structure of the package design:

The base package of the client side is an abbreviation of the application name with small letters '.ad'. On the client side of the Appsolut Distribute application we are enforcing the MVC –model, the letters 'm', 'v', and 'c' stands for Model, View, and Controller. The ideal usage of this model is to separate the classes and packages into these categories, the purposes of doing so is too keep the code clean and orderly which will increase developers' production rate.
Following description is from www.wikipedia.org:

- A **controller** can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can send commands to the model to update the model's state (e.g., editing a document).
- A **model** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. A *passive* implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.
- A **view** requests from the model the information that it needs to generate an output representation.

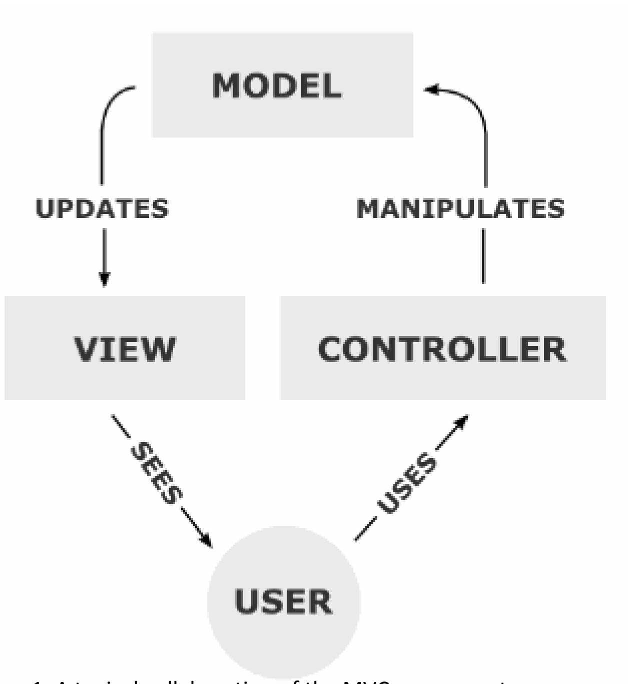


Figure 1: A typical collaboration of the MVC components.

The developed package structure is divided into these three cases. Making the packages in the base package as following:
- '.ad.controller'
- '.ad.model'
- '.ad.view'

## *1.2. Activity and its lifecycle:*

An activity creates the window of the application in which the developer can place an UI and is an important part of the applications overall lifecycle. It can also be a smaller floating window (see API) but this solution will not be applied in this application, for now custom made dialogs will be applied. Appsolut Distribute will consist of several activities, because the plan is to make several features and it is likely that each feature will be an activity of its own. The activities will be placed in their own package in view (since they are the window and contain the UI): '.ad.view.activity'
Figure 2 describes android activities lifecycle, the image is from http://www.androidjavadoc.com/:

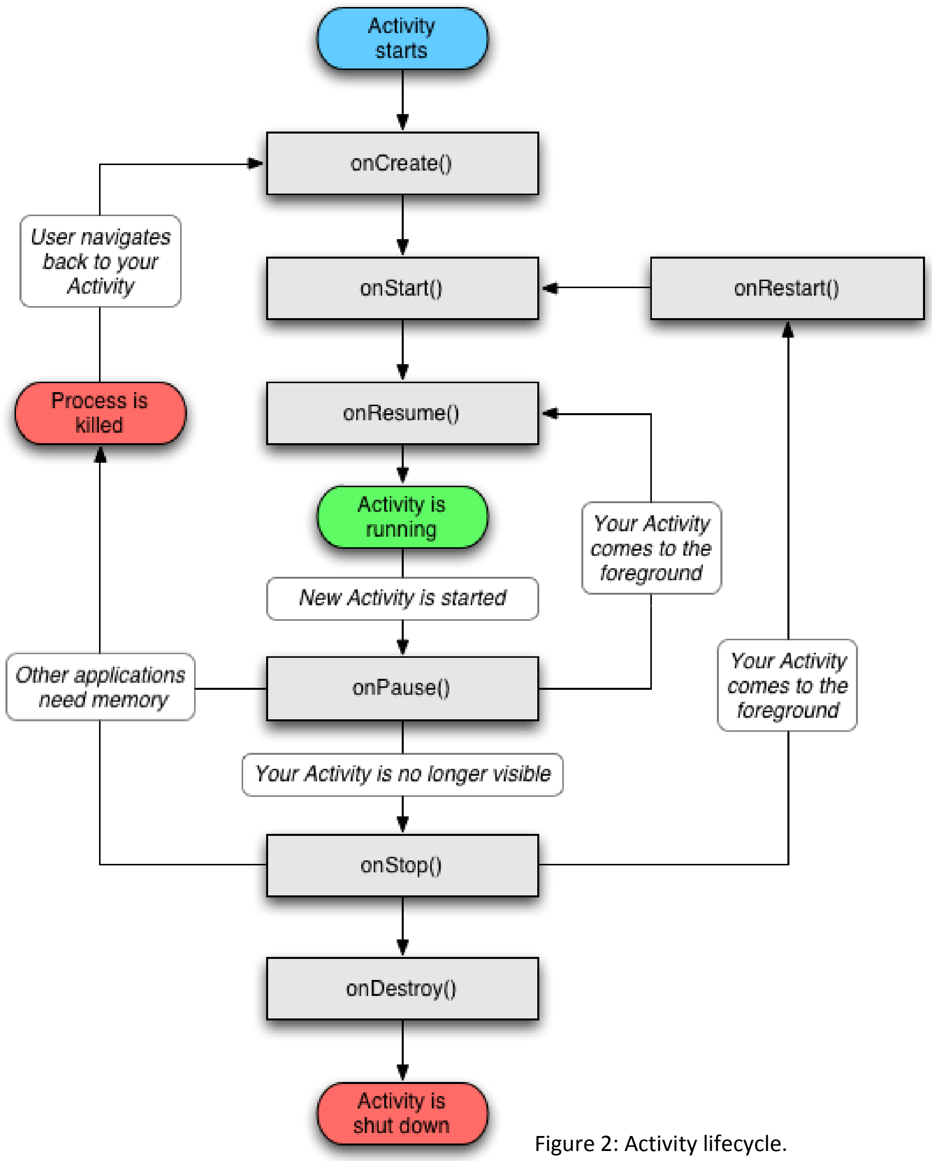

Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 2: Activity lifecycle.

How will this lifecycle effect the application? If the clients activity is doing an operation, being connected to the server or running a feature it may be halted one way or another. It is important to be able to handle such event and following up on what happens in these cases. To save states, use the Bundle class.

### *1.3. Structure of views on android:*

On the android OS there is an underlying structure of how to handle views and their layouts. The views are managed by group views. Each group view and view is managed by several xml files which determine their layouts. Following two images is from API guides on http://developer.android.com.

The figure 3 first is illustrating the relationship between the view group classes and view classes.

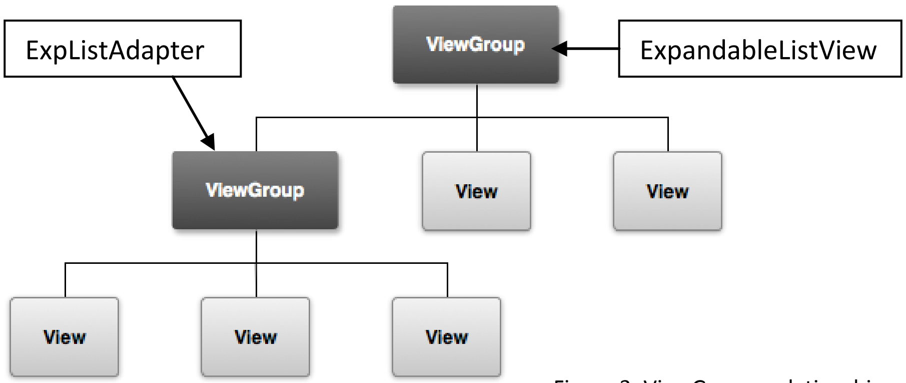
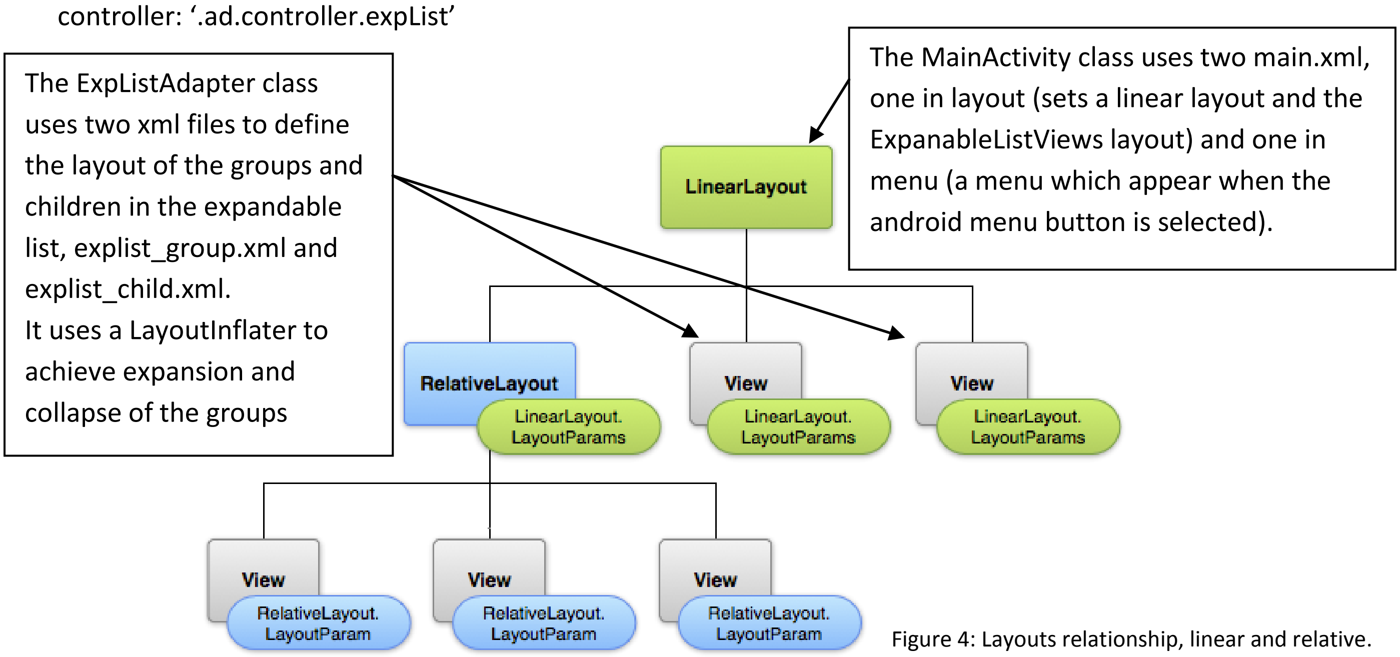


Figure 3: ViewGroups relationship with Views.

This architecture is used by the ExpandableListView class in the main activity class, used by the application. The ExpandableListView needs an Adapter (ExpListAdapter) which acts as a group view for the groups and children (group items) in the expandable list. The adapter updates the model and changes the views presentation, which makes it a controller class located in sub package to controller: '.ad.controller.expList'


The ExpListAdapter class uses two xml files to define the layout of the groups and children in the expandable list, explist_group.xml and explist_child.xml.
It uses a LayoutInflater to achieve expansion and collapse of the groups

The MainActivity class uses two main.xml, one in layout (sets a linear layout and the ExpanableListViews layout) and one in menu (a menu which appear when the android menu button is selected).

Figure 4: Layouts relationship, linear and relative.

Next is the layout of the view groups and views, they are as said managed by xml files. Note that the relationships have a similar structure in figure 3 and 4. They are closely related and it is important to have that in mind while developing this application.

### *1.4. Structure of Appsolut Distribute GUI:*

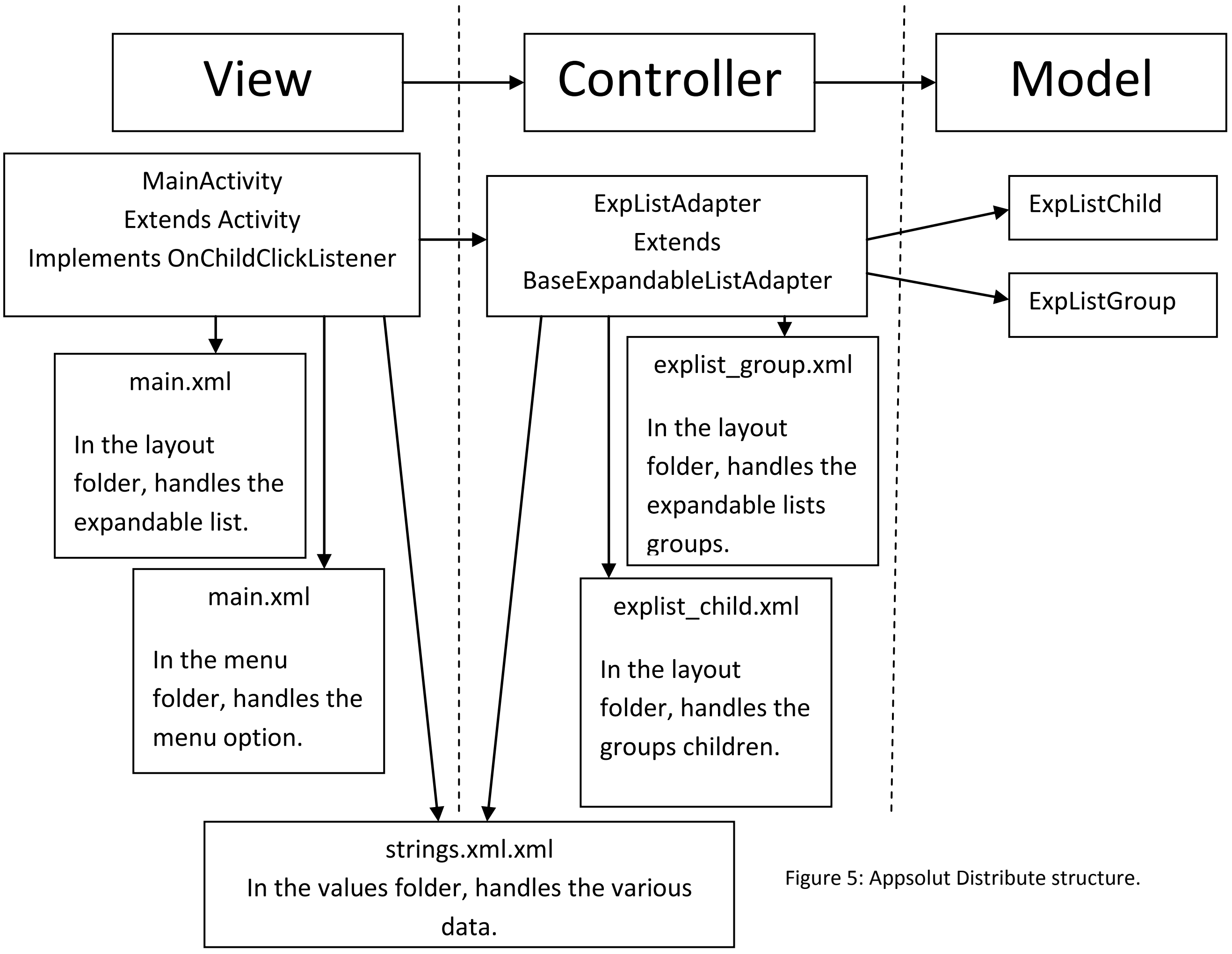

View
Controller
Model
MainActivity Extends Activity Implements OnChildClickListener
ExpListAdapter Extends BaseExpandableListAdapter
ExpListChild
ExpListGroup
main.xml In the layout folder, handles the expandable list.
main.xml In the menu folder, handles the menu option.
explist_group.xml In the layout folder, handles the expandable lists groups.
explist_child.xml In the layout folder, handles the groups children.
strings.xml.xml In the values folder, handles the various data.

Figure 5: Appsolut Distribute structure.

There is also a style.xml used by the android manifest xml file.
To have access to the xml files, located in the res folders subfolders, it is necessary to import the R class from a package specified in the AndroidManifest.xml, in our case the import looks like this: import ad.view.activity.R;
There is also the icon image of the application in the res folder in the drawable subfolders.
An unusual solution was used in the ExpListAdapter due to the lack of support for two dimensional strings for building expandable list menus. The solution was to use a divider in a string array:

```
 <!-- Start menu -->
<string-array name = "start_groups" >
     <item>Connect To Server</item>  <!-- group 1 -->
     <item>Options</item>            <!-- group 2 -->
 </string-array>
 <string-array name = "start_children" >
   <item>Login</item>                <!-- Here are the children belonging to group 1 -->
   <item>Create Account</item>
   <item>Select Server</item>
   <item></item>                     <!-- divider: group 1 ends here and group 2 begins -->
   <item>About</item>                <!-- Here are the children belonging to group 2 -->
   <item>Help</item>
   <item>Exit</item>
 </string-array>
```

Figure 6: Expandable list structure in xml.

The package structure looks like this:

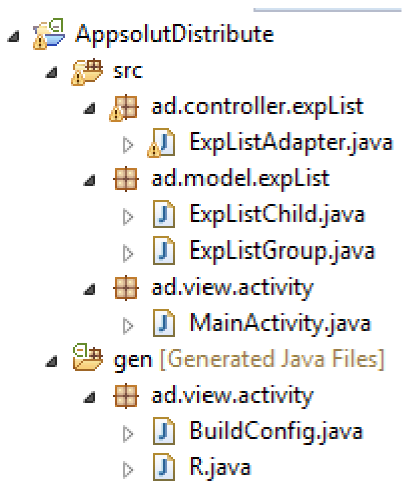


Figure 7: Current package structure in the eclipse package explorer.

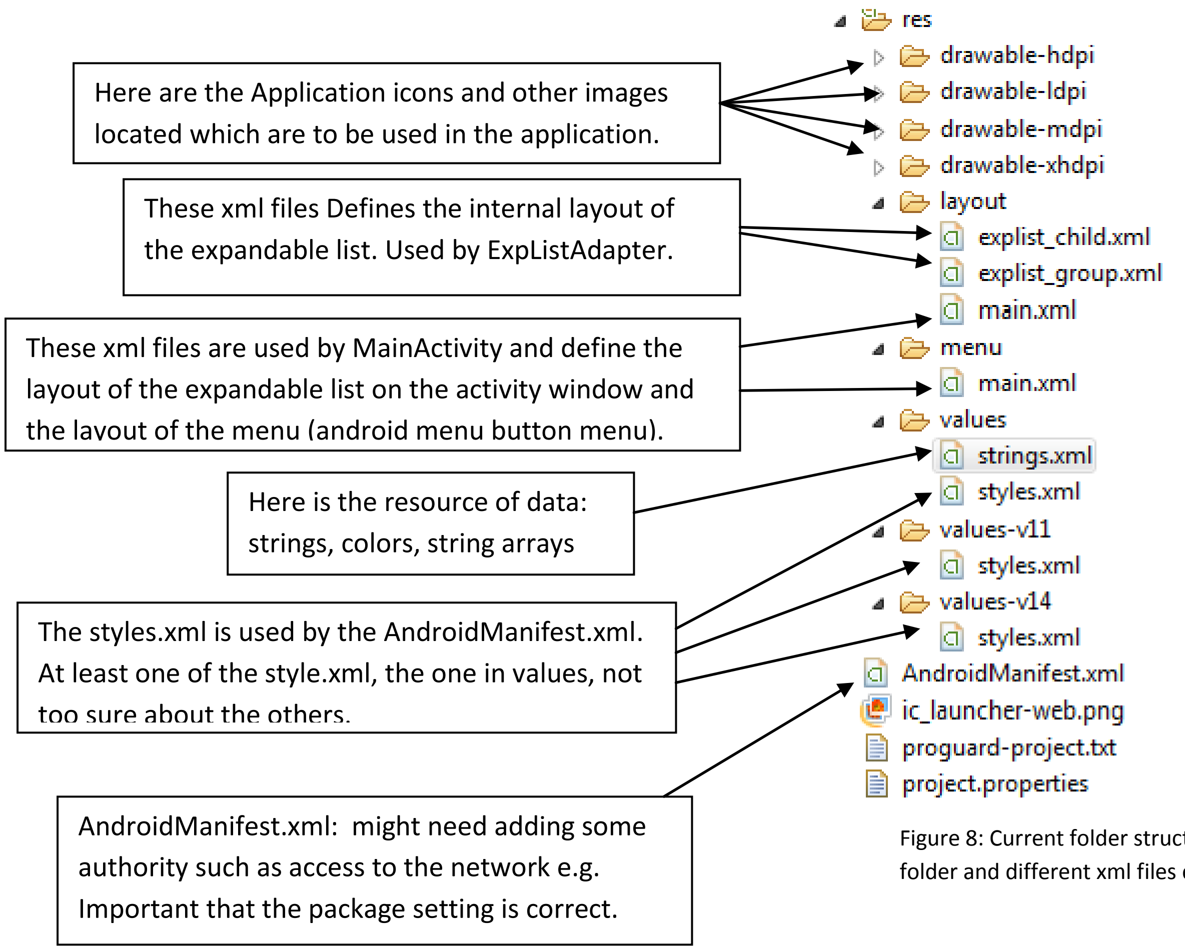
There will be some new packages and classes later, some known classes to be added are:
- Client, in the model package handles communication with server.
- Protocol, in controller package handles the

A class that might be added is:

- OptionDialogs, in View is to handle the communication between user and the Protocol class; it would be used by MainActivity to accomplish this.

- a thread pool class, not yet identified where it should be and how it should be implemented. Most likely it will be in the controller package since it defines when something in the model should be updated.

The structure of the res folder:



Here are the Application icons and other images located which are to be used in the application.

These xml files Defines the internal layout of the expandable list. Used by ExpListAdapter.

These xml files are used by MainActivity and define the layout of the expandable list on the activity window and the layout of the menu (android menu button menu).

Here is the resource of data: strings, colors, string arrays

The styles.xml is used by the AndroidManifest.xml. At least one of the style.xml, the one in values, not too sure about the others.

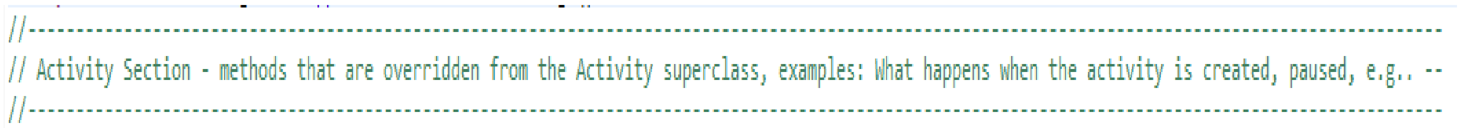
AndroidManifest.xml: might need adding some authority such as access to the network e.g. Important that the package setting is correct.

Figure 8: Current folder structure in the res folder and different xml files current locations.

# 2. Current code structure of the application

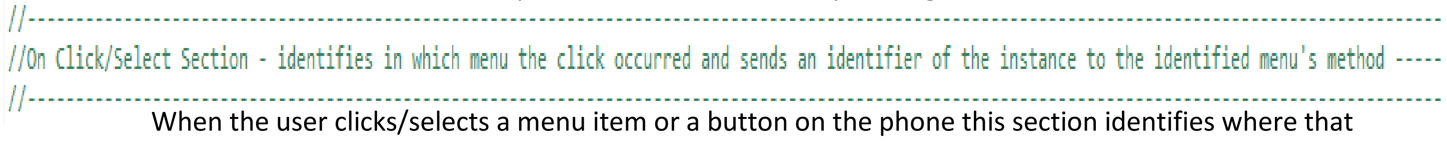
## 2.1. View:

In view there is only the MainActivity class currently and the code in the class is divided into sections to make it easier to develop the class.



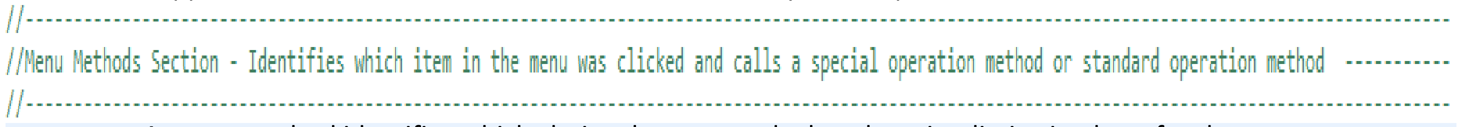
```
//---------------------------------------------------------------------------------------------------------------------------
// Activity Section - methods that are overridden from the Activity superclass, examples: What happens when the activity is created, paused, e.g.. --
//---------------------------------------------------------------------------------------------------------------------------
```

In the first section are the lifecycle methods of the activity. See figure 2 for more information.



```
//---------------------------------------------------------------------------------------------------------------------------
//On Click/Select Section - identifies in which menu the click occurred and sends an identifier of the instance to the identified menu's method -----
//---------------------------------------------------------------------------------------------------------------------------
```

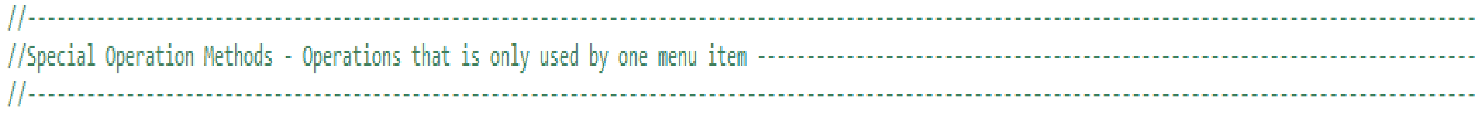
When the user clicks/selects a menu item or a button on the phone this section identifies where that happened and then calls on a menu method to identify which specific choice the user made.



```
//---------------------------------------------------------------------------------------------------------------------------
//Menu Methods Section - Identifies which item in the menu was clicked and calls a special operation method or standard operation method  ----------
//---------------------------------------------------------------------------------------------------------------------------
```

A menu method identifies which choice the user made, but there is a limitation here for the developer. The menu method runs an operation method.

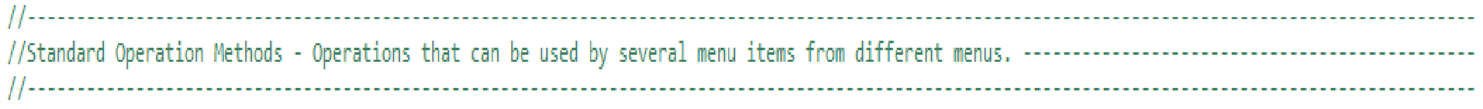
**Limitation:** Do not add the same choice twice or more in the same menu! If the developer has done this and added an operation method or several, multiple operations will be performed.



```
//---------------------------------------------------------------------------------------------------------------------------
//Special Operation Methods - Operations that is only used by one menu item --------------------------------------------------
//---------------------------------------------------------------------------------------------------------------------------
```

Examples on special operations are; login, select server e.g.

Why would these be special operations? Cause they are only called upon by one specific button in one specific menu.



```
//---------------------------------------------------------------------------------------------------------------------------
//Standard Operation Methods - Operations that can be used by several menu items from different menus. -----------------------
//---------------------------------------------------------------------------------------------------------------------------
```

Examples on standard operations is; back, exit, logout e.g.

Why would these be standard operations? Cause they can be used by an expandable list menu or the back button on the phone.

NOTE: Operations can either open a dialog or an expandable list menu. An expandable list menu is constructed from scratch every time, because it was not successful to build all the menus at once on start.

Dialogs are currently implemented on a small scale currently, we have two, exit and settings (settings is not implemented). There will be coming more information here about the dialogs used in the application when they are written.

### 2.2. Controller:

Currently there is only one controller class, ExpListAdapter. Controller classes that will be added are: Protocol class and thread pool class.

```
//-----------------------------------------------------------------------------------------------------------
// Initialize Section - The methods to build the Expandable List Adapter. ----------------------------------
//-----------------------------------------------------------------------------------------------------------
```

Here the expandable list adapter builds expandable list menus. There might be some optimizing, because of the static solution that forces the developer to add an “if statement” for each new expandable list menu.

```
//-----------------------------------------------------------------------------------------------------------
// Expandable List Operations Section - Operations to find out more about the expandable list and do operations on it------------------------------
//-----------------------------------------------------------------------------------------------------------
```

Currently there is a getActiveMenu method (used by MainActivity OnChildClick method) and an addItem method that is currently not used (might be used later on to add feature or privileges for the user).

Note: It might be useful to add a removeItem method.

```
//-----------------------------------------------------------------------------------------------------------
// Extend Methods Section - These methods are used by the class itself since it extends BaseExpandableListAdapter, getters methods. and one isChildSelectable method.
//-----------------------------------------------------------------------------------------------------------
```

These methods were not written by the developers of this application, they are auto generated by the class since it extends the BaseExpandableListAdapter. Most of the content here is retrieved from: http://www.dreamincode.net/forums/topic/270612-how-to-get-started-with-expandablelistview/

Note: There might be needs to alter these methods.

### 2.3. Model:

There are currently two models implemented in the application (client side), ExpListChild and ExpListGroup. They both have two instances, which have each one getter and setter per instance. A possible expansion on the model is if the child becomes a group for other children, but will not be applied in this application.

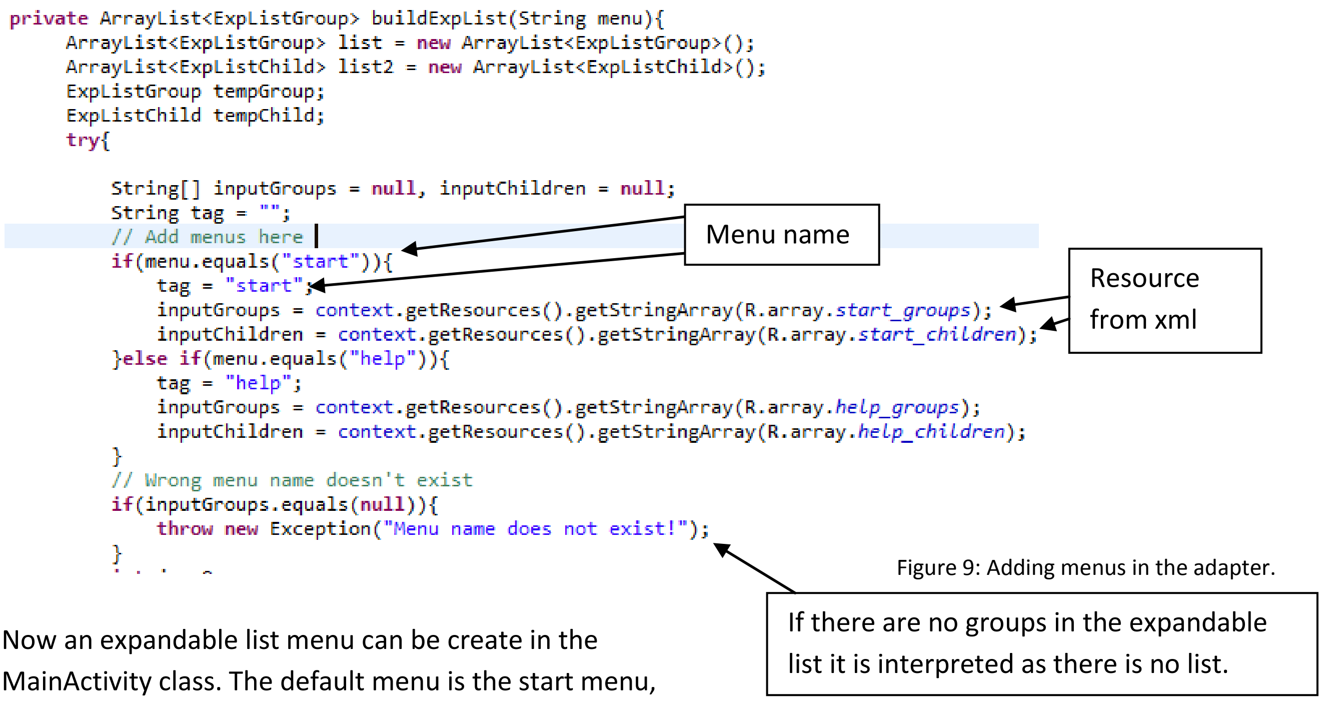
There will be a third class added here, Client, to handle the direct communication with the server.

# 3. How to add menus and menu items in the application

## 3.1. Add expandable list menus and its menu items:

A menu must consist of at least one empty group; it is not allowed to have an empty expandable list menu and it will give an error message when the application tries to create such a menu. The application won't crash but there will be nothing there, not even an exit or back button.

First it is needed to define the content of the expandable list menu with xml files, see figure 6 (Expandable list structure in xml and read why the xml structure is ordered in such a way on page 5. The xml code goes into res/values/strings.xml. NOTE: Every expandable list menu should have a title defined in xml; the only exception is the start menu.
When the menu is defined with groups and children in the xml go to the ExpListAdapter class and add an "else if statement" in the buildExpList in the "Initialize" section.



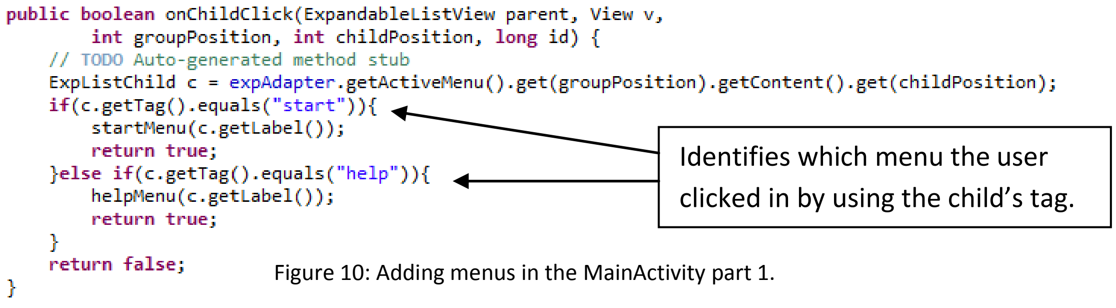
```
private ArrayList<ExpListGroup> buildExpList(String menu){
     ArrayList<ExpListGroup> list = new ArrayList<ExpListGroup>();
     ArrayList<ExpListChild> list2 = new ArrayList<ExpListChild>();
     ExpListGroup tempGroup;
     ExpListChild tempChild;
     try{

         String[] inputGroups = null, inputChildren = null;
         String tag = "";
         // Add menus here
         if(menu.equals("start")){
             tag = "start";
             inputGroups = context.getResources().getStringArray(R.array.start_groups);
             inputChildren = context.getResources().getStringArray(R.array.start_children);
         }else if(menu.equals("help")){
             tag = "help";
             inputGroups = context.getResources().getStringArray(R.array.help_groups);
             inputChildren = context.getResources().getStringArray(R.array.help_children);
         }
         // Wrong menu name doesn't exist
         if(inputGroups.equals(null)){
             throw new Exception("Menu name does not exist!");
         }
```

If there are no groups in the expandable list it is interpreted as there is no list.

Figure 9: Adding menus in the adapter.

Now an expandable list menu can be create in the MainActivity class. The default menu is the start menu, meaning that the user will be sent to the start menu on start. Do **not** change this.
If the developer have made an expandable list menu and wants the user to be able to access it, there are two ways of doing that. Through a dialog or directly, the dialog route is not yet implemented. The direct route means through a click on a child in a group on an expandable list menu. To accomplish this you must create your menu in the MainActivity. First go to the On Click/Select section and add an "else if statement" in the onChildClick method to identify the menu:

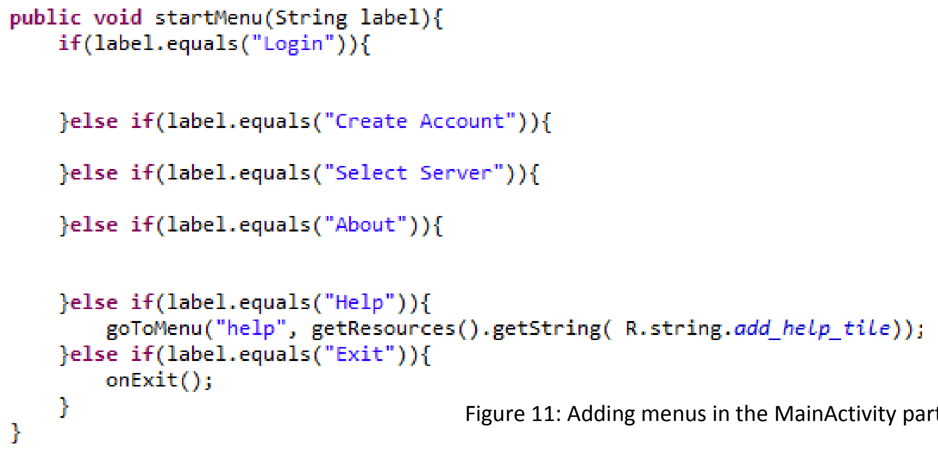


```
public boolean onChildClick(ExpandableListView parent, View v,
        int groupPosition, int childPosition, long id) {
    // TODO Auto-generated method stub
    ExpListChild c = expAdapter.getActiveMenu().get(groupPosition).getContent().get(childPosition);
    if(c.getTag().equals("start")){
        startMenu(c.getLabel());
        return true;
    }else if(c.getTag().equals("help")){
        helpMenu(c.getLabel());
        return true;
    }
    return false;
}
```

Identifies which menu the user clicked in by using the child's tag.

Figure 10: Adding menus in the MainActivity part 1.

Now go to menu methods section and add in one of the other menu's item the goToMenu method, it needs the name of the menu and a title and then make a new menu method for the new menu.



```
public void startMenu(String label){
    if(label.equals("Login")){

    }else if(label.equals("Create Account")){

    }else if(label.equals("Select Server")){

    }else if(label.equals("About")){


    }else if(label.equals("Help")){
        goToMenu("help", getResources().getString( R.string.add_help_tile));
    }else if(label.equals("Exit")){
        onExit();
    }
}
```

Figure 11: Adding menus in the MainActivity part 2.

**Note:** There is an addItem method in the ExpListAdapter, but it is best to leave it alone for now.

**Coming soon: The dialog route to an extendable list menu.**

### *3.2. Other editable menu and navigation features:*

The android menu button will open a default settings menu, currently only one item. Let it be for now.

Might be important to think out what to do with the android buttons, since the application is currently overriding the default method.

### *3.3. Add dialog*

Coming soon

## *4. Future development plans*

### *4.1. High priority plans.*

Add following classes:

Dialogs - view

Protocol - controller

Client - model

### *4.2. Low priority plans.*

Remove menu item method for ExpListAdapter