

# **Final Design Review (FDR)**

## **Cause and Effect**



---

**By:**

**Keith Adler**

**Matthew Brannick**

**Tomin Kozhimala**

**William Vennes**

**Sponsor: Samsung**



**Faculty Advisor: Dr. Don Evans**

# Table of Contents

Vision Statement.....	8
Executive Summary.....	9
User Needs.....	10
Initial User Stories .....	10
Client User Stories .....	10
Developer User Stories.....	11
Iteration User Stories .....	12
Iteration 1 .....	12
Iteration 2 .....	13
Iteration 3 .....	14
Iteration 4 .....	14
Iteration 5 .....	15
Iteration 6 .....	16
Iteration 7 .....	17
Iteration 8 .....	18
Iteration 9 .....	19
Iteration 10 .....	19
Iteration 11 .....	19
Phone/Physical Rules .....	20
3rd Party API Based Rules .....	20
Change Request Forms .....	21
Initial Design Ideas .....	23
Rule View Activity.....	23
Drag and Drop View .....	24
Rule Creation Activity.....	25
My Rules Activity (Version 2) .....	26
Account Sync Activity .....	27
My Rules Activity .....	28

Help Activity .....	29
Android Screen Flow Diagram.....	30
Rule Handling Workflow (Backend) .....	31
Final Design .....	32
User Interface.....	32
NFC Sharing .....	35
The Application Service.....	36
Broadcast Receiver.....	37
Rules Engine .....	38
Structural View .....	39
Thread View.....	40
Expression Tree.....	41
Individual Cause Evaluation.....	42
Database.....	43
Action Executer .....	46
Thread Diagram .....	47
Cause and Effect API .....	48
Risk Assessment and Management Plan .....	52
Agile Development Schedule.....	54
Iteration Burn Down Charts.....	57
Gantt Charts.....	63
Fall .....	63
Iteration 1 .....	64
Iteration 2 .....	64
Iteration 3 .....	65
Iteration 4 .....	65
Iteration 5 .....	66
Spring.....	67
Iteration 6 .....	68

Iteration 7 .....	68
Iteration 8 .....	69
Iteration 9 .....	69
Iteration 10 .....	70
Iteration 11 .....	70
Final Budget .....	71
Assessment of Context .....	74
Lessons Learned .....	76
Future Work .....	78
Source Code Repository .....	79
Javadocs .....	79
Wordpress .....	79
References .....	80
Backlog .....	81
Phone/Physical Rule User Stories .....	81
API Based User Stories .....	84
Glossary of Terms and Acronyms .....	90
Glossary References .....	94
Acceptance Test Plan .....	95
Java Class Diagrams .....	163
Code Appendix .....	172
Java .....	172
rEffect.java .....	172
Effect.java .....	176
rCause.java .....	179
Cause.java .....	188
DatabaseHandler.java .....	191
Help.java .....	219
FileUtils.java .....	223

About.java.....	224
ShareNFC.java.....	225
ShareList.java.....	229
DeleteDialogFragmentRule.java.....	236
Preferences.java .....	238
CategoryList.java.....	242
Security.java.....	243
EditRuleNode.java .....	244
CauseView.java.....	246
TabListener.java.....	257
CEapp.java .....	259
ActionExecuter.java.....	261
RulesEngine.java.....	263
ExpressionTree.java.....	266
General.java.....	272
CEbr.java .....	273
EffectView.java .....	278
MainActivity.java .....	289
DeleteDialogFragment.java .....	295
MyRules.java.....	297
CEservice.java .....	303
Rule.java .....	305
EditRule.java .....	311
MapPicker.java .....	323
ActionShowNotification.java .....	330
ActionRingerMode.java .....	332
ActionVibrate.java .....	334
ActionToast.java .....	335
ActionPlaySound.java .....	336

XML.....	337
AndroidManifest.xml.....	337
activity_about.xml.....	340
activity_cause.xml .....	341
activity_edit_rule.xml.....	342
activity_effect.xml .....	344
activity_general.xml .....	345
activity_help.xml .....	346
activity_main.xml .....	347
activity_my_rules.xml.....	349
activity_preferences.xml .....	350
activity_security.xml.....	351
activity_sharelist.xml.....	352
activity_sharenfc.xml.....	353
ce_activity_map.xml.....	354
ce_dialog_notification.xml .....	355
ce_dialog_ssid.xml.....	356
ce_dialog_switch.xml .....	357
ce_dialog_toast.xml.....	358
ce_dialog_vibrate.xml .....	359
item_list.xml .....	360
menu_edit_task.xml .....	361
menu_help.xml.....	362
menu_my_rules.xml .....	363
menu_preferences.xml.....	364
menu_sync.xml.....	365
values/strings.xml.....	366
values/styles.xml .....	368
values-v11/styles.xml .....	369

values-v14/styles.xml ..... 370

# Vision Statement

**Cause and Effect:** For people wanting to make their smart devices truly intelligent. Today, smart phones are overly complex and jammed pack with functionality. Despite having the power to do so many spectacular things, smart devices lack the ability for users to take control, and put the phone to work for them. Cause and Effect is a software solution that let users easily program their smart devices to respond to a set of rules with a chosen effect. With Cause and Effect, smart devices will become more intelligent, and make users' lives easier by intuitively automating tasks, gathering information, displaying information, and more. Cause and Effect provides more functionality, and a better user interface to allow any type of user to take control of their phone.

# Executive Summary

Today, smart phones are a part of our everyday lives. Users of smart phones are able to do a variety of functions, whether it be text messaging, phone calls, talking to friends on Facebook, listening to music, or sharing and watching videos on YouTube. However, smart phones currently do not have the ability to automatically perform user tasks based on changes that occur in the phone's environment. Users often pay large amounts of money for their smart phones, but never utilize their phone's potential. The typical smart phone user uses only functions such as the phone, the text messaging, email, and some fun games. There are so many features and APIs available for smart phone development, but rarely are these integrated to enrich the life of the user by using the entire phone's brilliance.

The solution to this is Cause and Effect for Android. Cause and Effect allows users to take more control over their phones. Users of Cause and Effect will be able to set "rules" for their phone, which enables the phone to automatically respond to changes in the environment, such as a change in Wi-Fi connection or current phone location via GPS. Users will also be able to share their phone rules with other users of Cause and Effect, whether it is via email, text, or NFC between other Android devices. Cause and Effect will enable users to automate desired functionality, and will increase the usability of their smart phone.

Partnered with our client, Samsung Mobile, we have been given the tools and direction to develop Cause and Effect for Android. We plan on revolutionizing the way how users see their phones and the smart phone development industry. Through much planning and hard work, we have a working prototype here today.

The application is simple and intuitive. It is so easy that your grandma could use it. The user can create rules that can be toggled on and off. These rules consist of Causes and Effects, which is where the name comes from. The Causes are actions from the phone's operating system or from an external source such as Facebook, Gmail, or Twitter. The operating system broadcasts can come from simple calls or messages, or many of the phones sensors and services. Users can choose causes with their own parameters to trigger effects. These effects are actions that the phone can execute. These effect actions allow users to really gain a lot of benefit from this application. With various types of Causes and Effects, users can create complex rule structures to fit their needs. Further variety can be attained using the logical AND and OR structure integrated into the causes chosen by the user. These causes are arranged in a tree structure so that users can apply user friendly Boolean logic to their own rules. Since you only have to setup a rule only once, this functionality allows users to take advantage of their phone from then on. Users can also share their rules and lists of rules with other Cause and Effect users so a community could form around rule creation and rule types that solve everyday problems.

# User Needs

We met with Samsung and outlined our requirements for the project. These requirements are organized into user stories, so that we can develop a project in a structured manner that abides by our agile development methodology. Some user stories can be considered "epic." Epic user stories are high level requirements and are intended to be broken into sub user stories and tasks when developed. For organizational purposes, the stories are organized into the initial user stories that we derived from our first meeting with Samsung, and the user stories that we assigned and completed for each iteration.

## Initial User Stories

These User Stories were the created before the first iteration based on our first meeting with Samsung. Although we used developer user stories during other iterations, these were our initial guidelines for planning our designs.

## Client User Stories

1. As a client, I want a rules engine, so that users can automate phone actions.
2. As a client, I want the rule creation process to be as easy as possible.
3. As a client, I want a fluid user interface, so that all users can use our application with ease.
4. As a client, I want a simple way to add new causes and effects, so that third parties can add to the repository Extensible to Third Parties
5. As a client, I want rules to work with Boolean algebra, so that users can create complex rules to tailor to their needs.
6. As a client, I want rules to be sharable, so that users can easily distribute their rule frameworks to others
7. As a client, I want the application to require a minimum of Android version 4.0 (Ice Cream Sandwich), so that we can utilize the next generation of Android devices.
8. As a client, I want the application to run on all Android v4.0+ devices, so that we can have as much user compatibility as possible
9. As a client, I want to include NFC based rules and sharing, so that the app can make use of Samsung's latest hardware sharing technology.
10. As a client, I want rules that react to the user's location, so that users can create intricate rules using any given location.

11. As a client, I want to include Wi-Fi based rules and sharing, so that the app can make use of local network capabilities.
12. As a client, I want to be able to put many rules in (stress test) and have it work as quickly as with very few rules.
13. As a client, I want the application to be optimized for battery efficient, so that the application will not drain the phone's battery life.
14. As a client, I want an additional drag and drop rule creation interface, so that the user can choose to have a fluid look and feel while creating rules.
15. As a client, I want the application to run quickly and not to slow down my phone's other processes.
16. As a client, I want the application to not take up much space on my phone.
17. As a client, I want to know the application is working correctly when closed.

## Developer User Stories

1. As a developer, I want to be able to also share using email, MMS, Facebook, and Dropbox, so that users can have options on the sharing medium.
2. As a developer, I want to create user notifications, so that users are aware of the application's activity and progress.
3. As a developer, I want to include a user preferences activity, so that users can tweak and enable settings for the application.
4. As a developer, I want to have text message based rules, so that I can debug rules using the phone's mobile network.
5. As a developer, I want to have time-based rules, so that I can debug the application easily.
6. As a developer, I want to have vibrate-based rules, so that I can debug the application easily.
7. As a developer, I want to have "Toast" based rules, so that I can debug the application easily.
8. As a developer, I want to have Facebook-based rules, so that I can test the account sync rules.
9. As a developer, I want to have Google-based rules, so that I can test the account sync rules.
10. As a developer, I want to have Dropbox-based rules, so that I can test the account sync rules.
11. As a developer, I want an easily accessible database (such as SQLite).
12. As a developer, I want to be able to decipher and encode rules with Boolean algebra.

13. As a developer, I want to have screen animations, so that the application is fluid and aesthetically pleasing.
14. As a developer, I want a cool logo, so that our application is memorable and aesthetically pleasing.
15. As a developer, I want a cool application icon, so that our application is easy to recognize on the Android home screen.

## Iteration User Stories

Each iteration consisted of a set of user stories that were accomplished during that time. Some iterations had more user stories than others. This depended on the difficulty of certain user stories, and the amount of time that our group could put towards each iteration. Some user stories were entirely based on code, where others may be based on design or testing.

### Iteration 1

1. As a developer, I want a Login screen design, so that we can implement a mock graphical user interface.
2. As a developer, I want a Main Menu screen design, so that we can implement a mock graphical user interface.
3. As a developer, I want a Rule Creation screen design, so that we can implement a mock graphical user interface.
4. As a developer, I want a My Rules screen design, so that we can implement a mock graphical user interface.
5. As a developer, I want a Rule Description screen design, so that we can implement a mock graphical user interface.
6. As a developer, I want a Cause screen design, so that we can implement a mock graphical user interface.
7. As a developer, I want an Effect screen design, so that we can implement a mock graphical user interface.
8. As a developer, I want an Account Sync screen design, so that we can implement a mock graphical user interface.
9. As a developer, I want a Cause and Effect Option screen design, so that we can implement a mock graphical user interface.
10. As a developer, I want a Preferences screen design, so that we can implement a mock graphical user interface.
11. As a developer, I want a General Preferences sub screen design, so that we can implement a mock graphical user interface.

12. As a developer, I want an Account Sync Preferences sub screen design, so that we can implement a mock graphical user interface.
13. As a developer, I want a Security Preferences sub screen design, so that we can implement a mock graphical user interface.
14. As a developer, I want an About Page sub screen design, so that we can implement a mock graphical user interface.
15. As a developer, I want a flow diagram, so that the activity screens can organize and have basic flow.
16. As a developer, I want all of these designs to be implemented into a Mock GUI Application, so that we have a release our first iteration and can get feedback from Samsung.

## Iteration 2

17. As a developer, I want an alphabetical list view for rules, causes, and effects, so that users can view rules in whichever method they prefer.
18. As a developer, I want a category list view for rules, causes, and effects, so that users can view rules in whichever method they prefer.
19. As a developer, I want a grid list view for rules, causes, and effects, so that users can view rules in whichever method they prefer.
20. As a developer, I want buttons that allow the user to switch between different views types on any given list.
21. As a developer, I want an action to occur when any item in a list is selected, so that the flow of the application can be improved and items can be selected.
22. As a developer, I want a grid view for rules, causes, and effects, so that users can view rules in whichever method they prefer.
23. As a developer, I want the help items menu to contain fields, so that we can simulate the future functionality for this activity.
24. As a developer, I want menus on the Main Menu screen, so that users can navigate the application with ease.
25. As a developer, I want menus on the New Rules screen, so that users can navigate the application with ease.
26. As a developer, I want menus on the Help screen, so that users can navigate the application with ease.
27. As a developer, I want menus on the Settings screen, so that users can navigate the application with ease.
28. As a developer, I want divider lines for each list, so that the user can clearly see the distinction between list items.

29. As a developer, I want to hide the password characters on the Account Sync screen, so that users can hide their personal login information.
30. As a developer, I want a confirmation toast on the Account Sync screen, so that users know when the account was synched successfully.

## Iteration 3

31. As a client, I want a Rules Engine design document, so that we can implement the Rules Engine backend in the next iteration.
32. As a developer, I want a design for a Broadcast Receiver class, so that we can receive system calls from Android OS.
33. As a developer, I want a design for an Event Handler class, so that other third party causes can be invoked.
34. As a developer, I want a design for a Rules Database class, so that user data can be stored and accessed through simple and complex queries.
35. As a developer, I want a design for a Rules Engine class, so that the events triggered by the Broadcast Receiver and the Event Handler can be evaluated as true or false.
36. As a developer, I want a design for an Action Executer class, so that this can be called from the Rules Engine when the cause triggers are evaluated as true.
37. As a developer, I want these rules features to be designed to run on separate threads, so that the main user thread is not slowed down.
38. As a developer, I want the Rules Database class to consist of either SQLite, a B+ Tree, a Hash Table, or an XML/JSON structure, so that there can be structure to the rule data.
39. As a developer, I want the Rules Engine class to contain a recursive logic tree function, so that rules can be evaluated from expressions including ANDs and ORs.
40. As a developer, I want the Rules Engine class to contain a check cause function, so that the causes can be evaluated when the tree function reaches an individual cause.

## Iteration 4

41. As a developer, I want an implementation of the Broadcast Receiver, so that we can start the rules engine flow.
42. As a developer, I want an implementation of the Rules Database, so that we can manage rule data.
43. As a developer, I want an implementation of the Rules Engine, so that we can process causes using the database.

44. As a developer, I want an implementation of the Action Executer, so that we can display results of the Rules Engine.
45. As a developer, I want styling on the buttons, so that the user will have a more aesthetically pleasing experience.
46. As a developer, I want styling on the activity screens, so that the user will have a nice contrast that is clear and easy to look at it.
47. As a developer, I want to use our application logo on the Main Menu, so that the user can remember our application.
48. As a developer, I want to use our application icon on the application, so that the user can identify our application from the application menu on the Android OS.
49. As a developer, I want to redesign the Edit Rule screen, so that it can allow for new causes and effects to be added to a rule.
50. As a developer, I want three cause types, so that we can test the rules engine evaluation.
51. As a developer, I want three effect types, so that we can test the rules engine results.

## Iteration 5

52. As a developer, I want to connect the Broadcast Receiver and the Rules Engine classes, so that the causes are sent to be evaluated once they are triggered.
53. As a developer, I want to connect the Rules Engine and the Rules Database classes, so that the Rules Engine can evaluate cause data stored in the database.
54. As a developer, I want to connect the Rules Engine and the Action Executer classes, so that a cause tree evaluated to true will produce results that the user can see.
55. As a developer, I want a test database function, so that the user will have a few sample rules available when they first install the application.
56. As a developer, I want to reskin the Edit Rule screen, so that the user can see clear definition between different sections with clarity.
57. As a developer, I want an Update Rule button, so that the user can save a new rule in the database.
58. As a developer, I want the Update Rule button to handle rules that are already created, so that they can be updated in the database instead of being recreated.
59. As a developer, I want a phone call Cause dialog, so that the user can choose a contact to be used in their rule.
60. As a developer, I want a text message Cause dialog, so that the user can choose a contact to be used in their rule.
61. As a developer, I want a time Cause dialog, so that the user can choose a time that they want the rule to be triggered.

62. As a developer, I want a notification Effect dialog, so that the user can create a custom notification to be displayed in the notification bar.
63. As a developer, I want a toast Effect dialog, so that the user can create a custom message to be displayed briefly.
64. As a developer, I want a vibrate Effect dialog, so that the user can create custom vibrate tones and store this tone as an effect.
65. As a developer, I want to update the dividers between items in the lists, so that they are more aesthetically pleasing.
66. As a developer, I want to create a Preliminary Design Document, so that we can present documentation at the presentation at the end of the iteration.
67. As a developer, I want to create a Preliminary Design Slideshow, so that we can have a basis for the presentation at the end of the week.

## Iteration 6

68. As a developer, I want to be able to search causes by type, so that I can evaluate more than one rule at once if they trigger at the same time.
69. As a developer, I want to redesign the cause tree, so that it will evaluate multiple causes properly.
70. As a developer, I want to allow rules to be turned on and off, so that the user can decide which rules they want to be evaluated at a given time.
71. As a developer, I want to provide a slide presentation for Samsung, so that we can update our clients on our progress since before winter break.
72. As a developer, I want to add a flow mini-map to our presentation slides, so that the client and professors can follow along with our backend code design.
73. As a developer, I want to update the contrast in our slides, so that the user can see our slides in different lighting conditions.
74. As a developer, I want to design an AND and OR interface, so that the user can add their own ANDs and ORs to the causes.
75. As a developer, I want to turn off screen rotation, so that the user doesn't get confused due to phone orientation.
76. As a developer, I want to add "Not Implemented" toast messages, so that the user can see what features are completed, and which are not yet implemented.
77. As a developer, I want to remove the grid view, so that the user can have a clear and simple interface for choosing between text items.
78. As a developer, I want to be able to save a rule after any fields are updated, so that the user doesn't have to manually update the rule and the information is retained properly.

79. As a developer, I want to fix the back stack, so that there aren't extraneous activities being created throughout the rule creation process.
80. As a developer, I want a vertical separation of causes and effects on the Edit Rule screen, so that the user won't cross the divider lines for each list.
81. As a developer, I want to limit ANDs and ORs to expressions with at most one level of nesting, so that users won't get confused by the complexity of the expressions for rules.
82. As a developer, I want to create a cause tree structure that generates from a string of leaves, so that the algorithm is more efficient.
83. As a developer, I want to start designing a system to implement the rules engine under a separate service so that the rules are evaluated in all of the edge cases.
84. As a developer, I want to check causes and effects for duplicates, so that the interface is clear of clutter.

## Iteration 7

85. As a developer, I want a function that will display a user friendly version of the cause tree, so that the user can see what the rule is supposed to do.
86. As a developer, I want to be able to edit causes, so that the user can make quick changes or tweak parameter values.
87. As a developer, I want to be able to edit effects, so that the user can tweak parameter values for effect messages.
88. As a developer, I want to be able to delete causes, so that the user can remove unwanted causes from the cause list.
89. As a developer, I want to be able to delete effects, so that the user can remove unwanted effects from the effect list.
90. As a developer, I want to run the rules engine from a service, so that the rules evaluate in edge cases and the evaluation is more efficient and independent from the application.
91. As a developer, I want a sound Effect type, so that the user can play sounds as an effect.
92. As a developer, I want a ring mode Effect type, so that the user can change their ringer settings as an effect.
93. As a developer, I want a location Cause type, so that the user can creates rules that are dependent on the phone's location.
94. As a developer, I want the location type to account for multiple kinds of services, so the user can decide how accurate they want the location updates to be and save battery as need be.
95. As a developer, I want to change the vibrate Effect so that it is a simple tone, so that the user doesn't get confused with the vibrate tone integer string during effect creation.

96. As a developer, I want a Wi-Fi Cause type, so that the user can create rules based on current Wi-Fi settings.
97. As a developer, I want to be able to share rules with another phone, so that users can share their created content as they see fit.
98. As a developer, I want to be able to share rules using NFC, so that users can use this useful and cutting edge feature on Samsung devices.
99. As a developer, I want to remove the account tables, so that we can provide a much more secure OAuth service to the user.

## Iteration 8

100. As a developer, I want to add ANDs and ORs to the cause list, so that the user can see the expressions as they are being created.
101. As a developer, I want to migrate all unresolved bugs to our Bitbucket tracker, so that our clients can view our progress on fixing errors and we can document them in a formal fashion.
102. As a developer, I want to fix the stack on the Edit Rule screen, so that new causes, edited causes, and deleted causes don't create new activities that are not needed.
103. As a developer, I want to redesign the background service, so that it will properly trigger all the causes from the Android OS and interact with application when necessary.
104. As a developer, I want to implement actions on the action bar, so that the user can access different functionality at the touch of a button.
105. As a developer, I want to implement contextual action bars, so that the user can have different action bar items available depending on the current item selection and screen.
106. As a developer, I want to be able to handle duplicate rules, so that sharing will not create redundant instances of rules that are identical in every way.
107. As a developer, I want a listener for Wi-Fi, so that Wi-Fi causes will be triggered whenever a change in Wi-Fi settings has occurred.
108. As a developer, I want a listener for location, so that the location causes will be triggered whenever the phone receives an update that is better than the old location.
109. As a developer, I want to update our slides for the Midterm Design Review, so that we can have an updated presentation representing our work since the PDR.
110. As a developer, I want to update our Midterm Design Review document, so that we can update our documentation for our work since the PDR.
111. As a developer, I want to spend additional time debugging, so that we have a clean iteration release, and we have a robust interface for user testing.

112. As a developer, I want to format our code into one standard form, so that our code is legible, clear, and uniform.
113. As a developer, I want to create Javadoc comments, so that our code is documented better internally.
114. As a developer, I want to generate a Javadoc file, so that external developers (such as our clients after we hand it off) can understand all of the classes, functions, and members through code documentation.

## Iteration 9

115. As a developer, I want to restructure the back end, so that an API can be created that can be used by 3rd party developers.
116. As a developer, I want to move all cause files to their own folder, so that all causes are listed in one location for the API.
117. As a developer, I want to move all effects files to their own folder, so that all effects are listed in one location for the API.
118. As a developer, I want to build an API out of our objects, so that 3rd party developers can build Cause and Effect plug-ins for the application.

## Iteration 10

119. As a developer, I want to build tests for the acceptance test plan that are easy to follow, so that the testing team can understand our application.
120. As a developer, I want to build tests for the acceptance test plan that are thorough, so that the testing team can stress test our application.
121. As a developer, I want a mid iteration release, so that the testing team can use our phones with clean application installations for the acceptance testing.

## Iteration 11

122. As a developer, I want to remove any debugging features, so that the rules are permanent, and the application is polished as a release.
123. As a developer, I want to clean up the preferences page, so that the information on these pages represents our final design.
124. As a developer, I want to finalize the help screen, so that users can understand our application using a tutorial of the final design.
125. As a developer, I want to clear all bugs from our application, so that our final release is bug free.

126. As a developer, I want to create the FDR documentation, so that our final report will include all of our finalized designs.
127. As a developer, I want create slides for the FDR, so that our presentation will be clear and summarize our work on this project.
128. As a developer, I want to update the Javadoc file, so that external developers (such as our clients after we hand it off) have code documentation for the final version of our project.
129. As a developer, I want to standardize the code formatting, so that the code is easy to read and understand, and is all in one form.

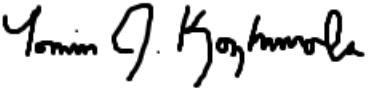
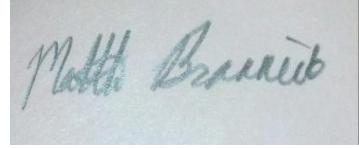
## Phone/Physical Rules

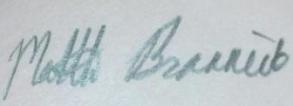
There are many phone features that are not currently included in our application. Some of the rule types are considered more important, and are listed above. Others were not included in this list. For the sake of space and organization, our list of phone-based rules that could be included is the Backlog Appendix towards the end of this document.

## 3rd Party API Based Rules

There are also numerous account types that are not currently included in our application. The plethora of accounts and rule types even prompted the creation of a "Category View" that could be used as the default view for browsing rule options in a future release. For the sake of space and organization, the list of account types is also included in the Backlog Appendix towards the end of this document.

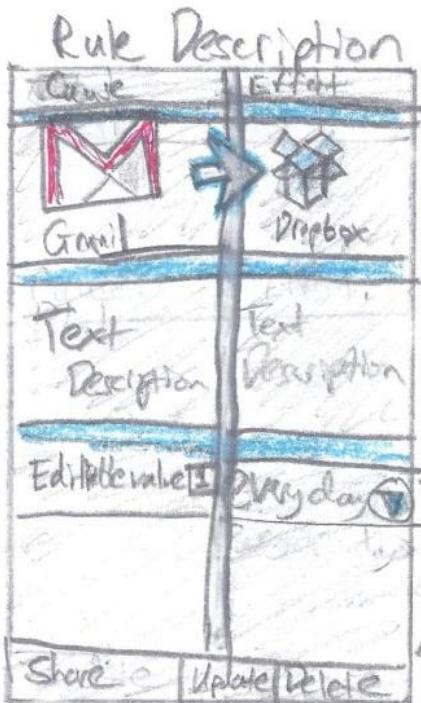
# Change Request Forms

Project: Cause & Effect for Android	Document: Final Design Report
Date: February 1st, 2013	Version #: 2
Changes	
<p>The Account Tables in the database schema were removed because our client asked us not to store any user data. Instead, the application can use OAuth 2.0 to integrate 3rd party applications.</p>	
Signatures	
   	

Project: Cause & Effect for Android	Document: Final Design Report
Date: March 9th, 2013	Version #: 3
Changes	
<u>User Stories Removed from the Application</u>	
1, 8, 24, 29, 30, 57, 58, 106	
These user stories were removed from our application in later iterations because our client told us to remove these features.	
Signatures	
   	

# Initial Design Ideas

## Rule View Activity

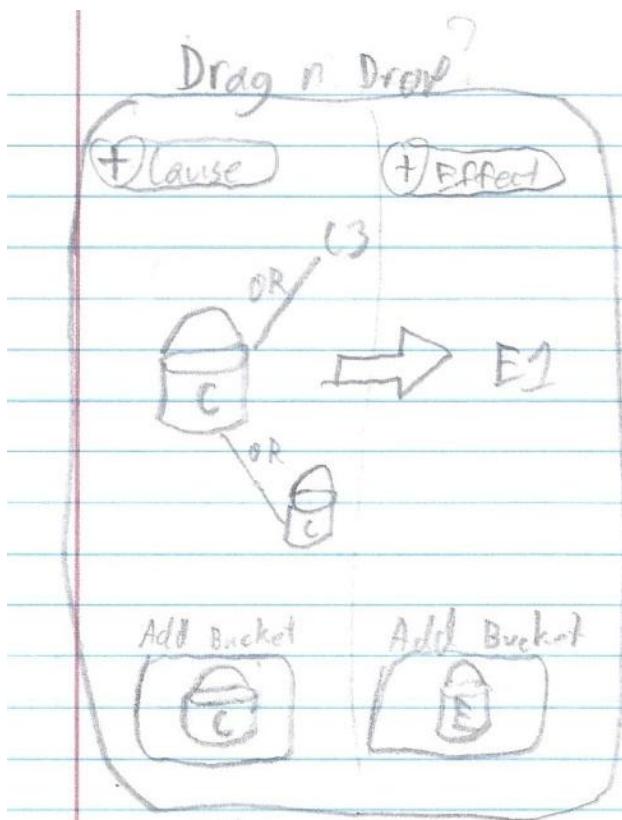


### Rule View

- Views one rule
- easy to read view
- Share/update/delete the rule
- lower section for editable fields

Here, the user will follow the rule description page as they create the rule. They click on the cause size to choose a cause, and the effect page to choose an effect. As the user adds causes and effects, they will return to this page to see how their rule has progressed as they create it.

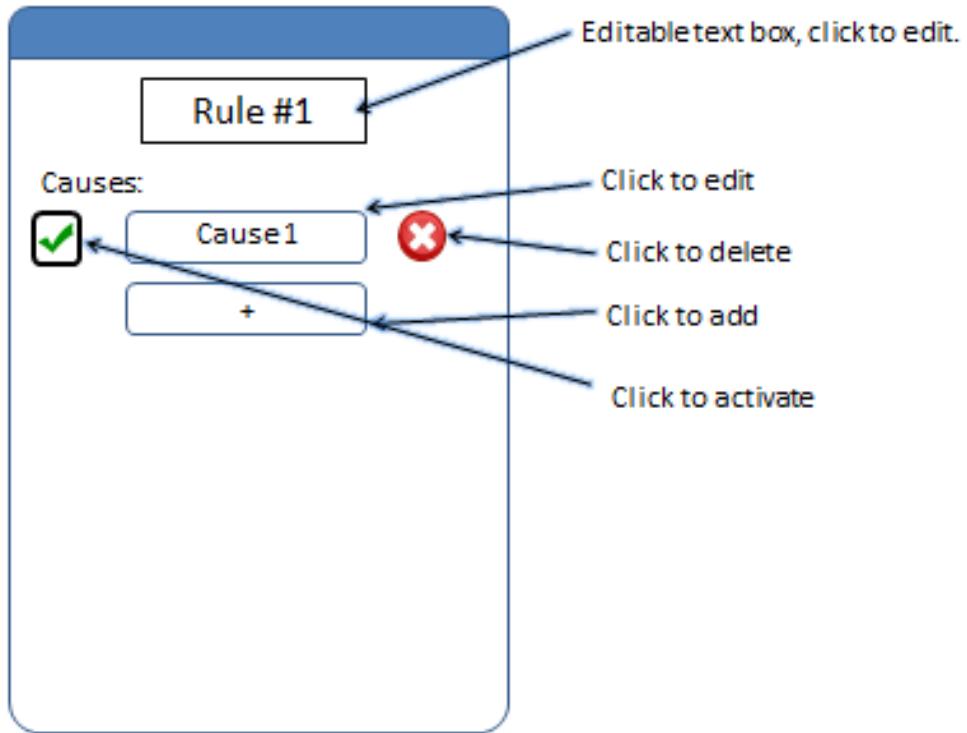
## Drag and Drop View



- Buckets represent AND groups
- User can add causes/effects to the bucket
- Lines represent ORs
- Can OR cause buckets with causes or cause buckets:

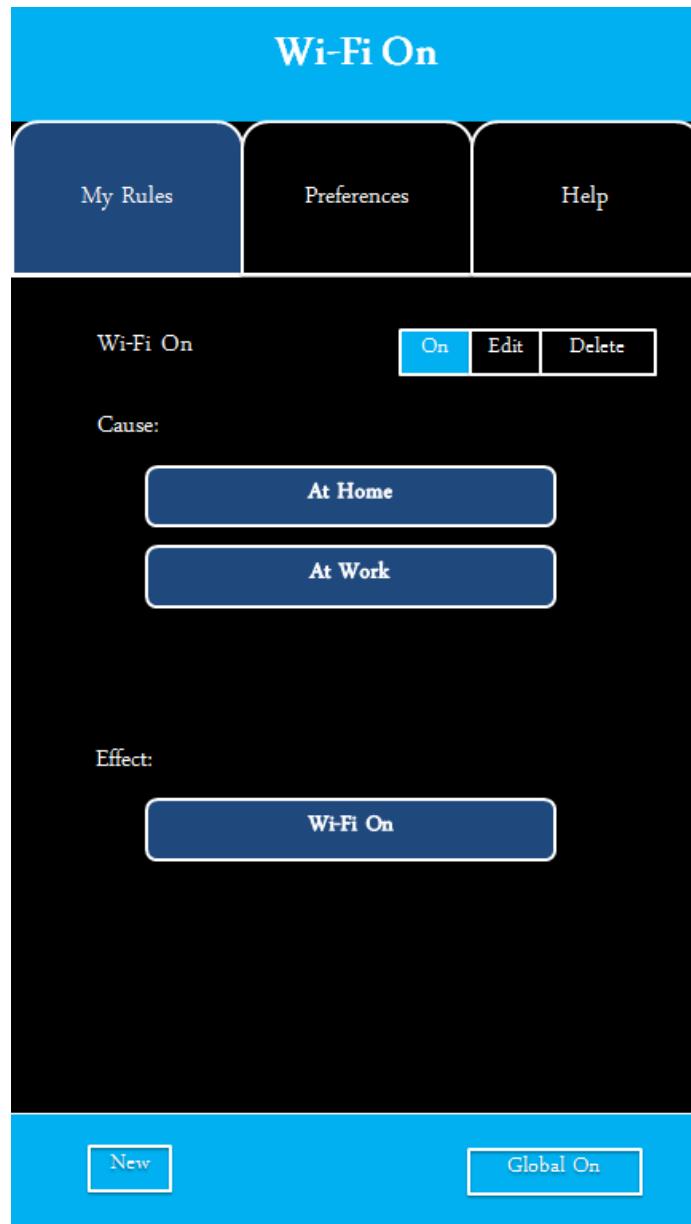
Here is an idea for a drag and drop implementation of the rule creation page, in which users place selections in “buckets” to represent groups of causes or effects, and link them to other buckets or individual items by an OR line. This would create a more abstract view for rule creation and could benefit visual-based users.

## Rule Creation Activity



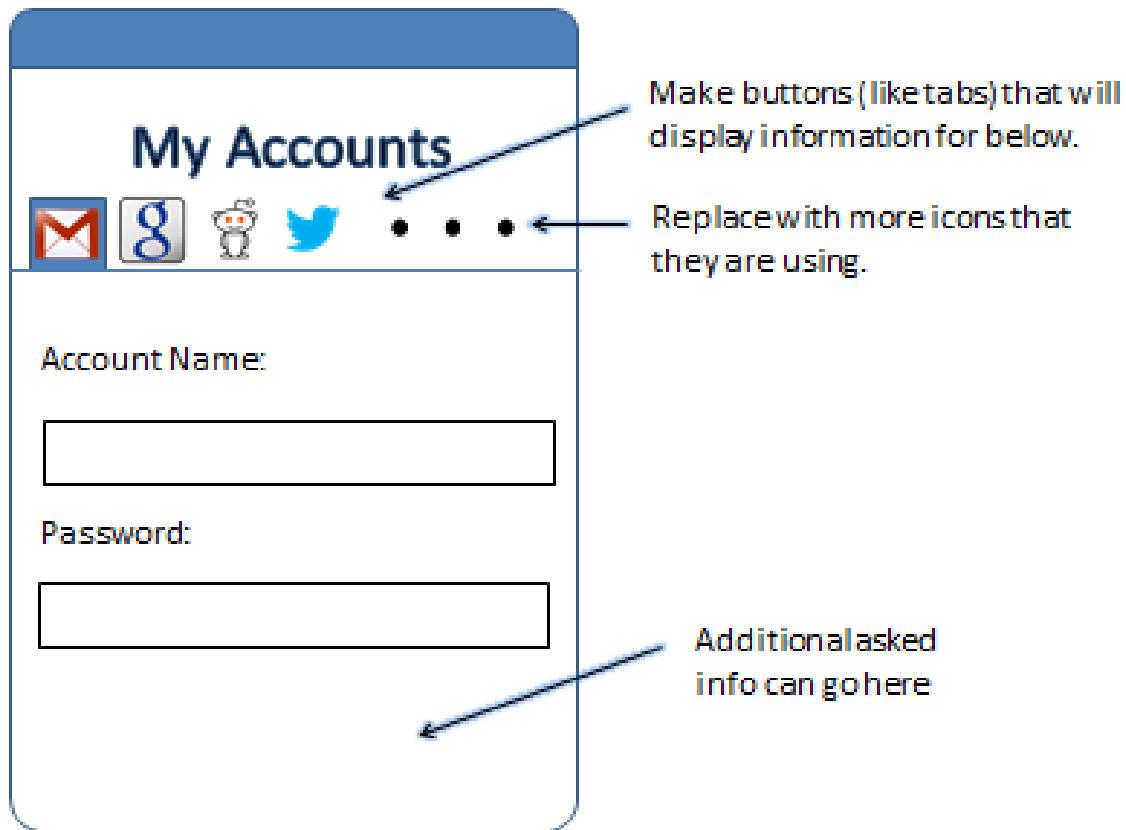
Here is a rule creation page in which causes and effects are separated by different pages and takes a more list view approach.

## My Rules Activity (Version 2)



Another conceptual rule view where causes and effects are listed sequentially in a list format.

## Account Sync Activity



This is an accounts page in which the user syncs their Google, Facebook, etc. accounts for use with rule creation. Another idea was moving the list of accounts to a separate page accessed from preferences, rather than a slide at the top.

## My Rules Activity



This was a concept screen for the viewing of existing rules. From this screen, users can turn a rule on or off, go to the edit menu, or delete the rule.

## Help Activity



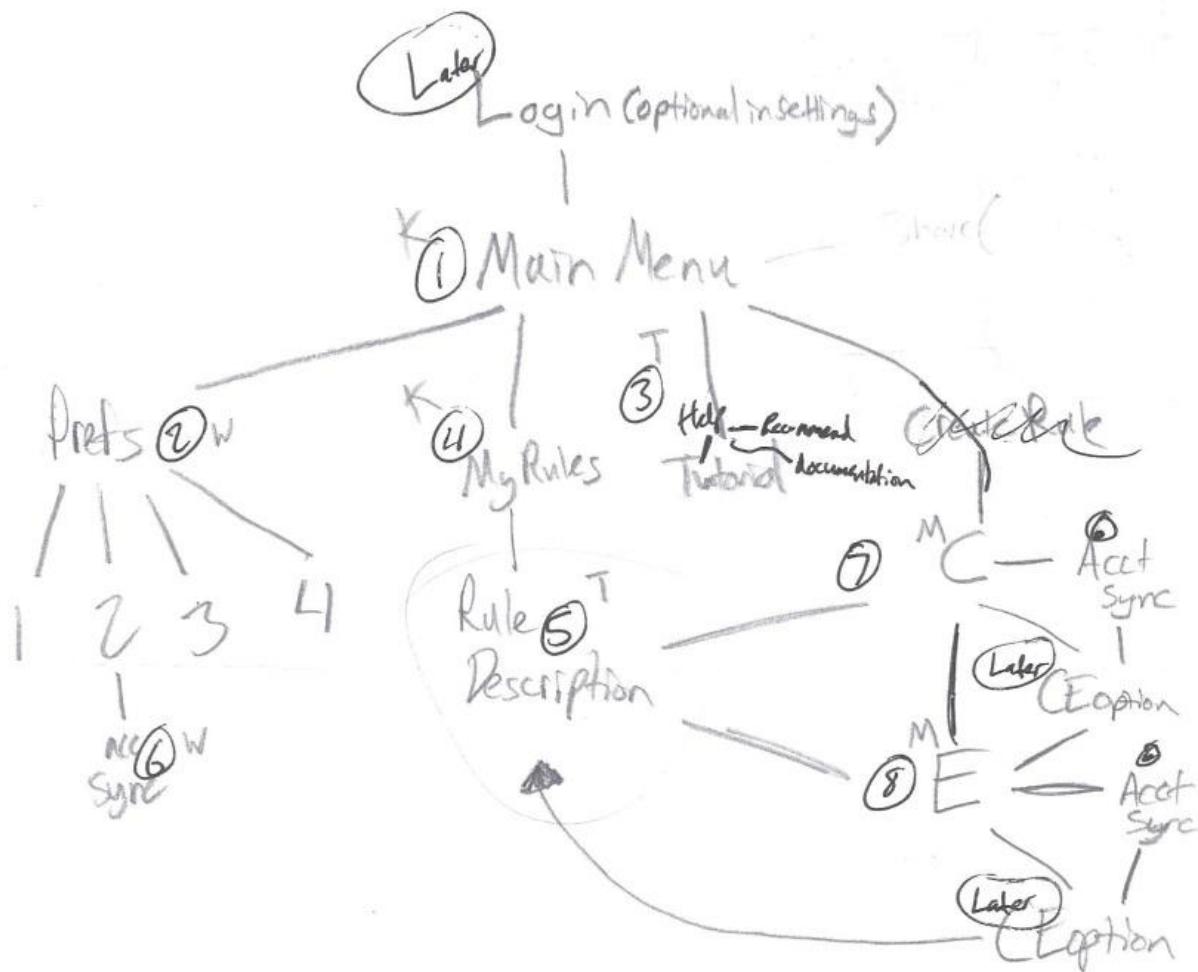
### Tutorial

- A "So easy your grandma could use it" guide for the app.
- Page turning per step/direction set
- ★ editable page selection? I.E: Page 1/6
- has pictureguide
- ★ video guide? link? Embed? YouTube integration?

The tutorial page is an idea we had that would help first time users understand how to operate the app. There was also an idea for a tutorial overlay for the app that would direct users towards options and functions for app operation.

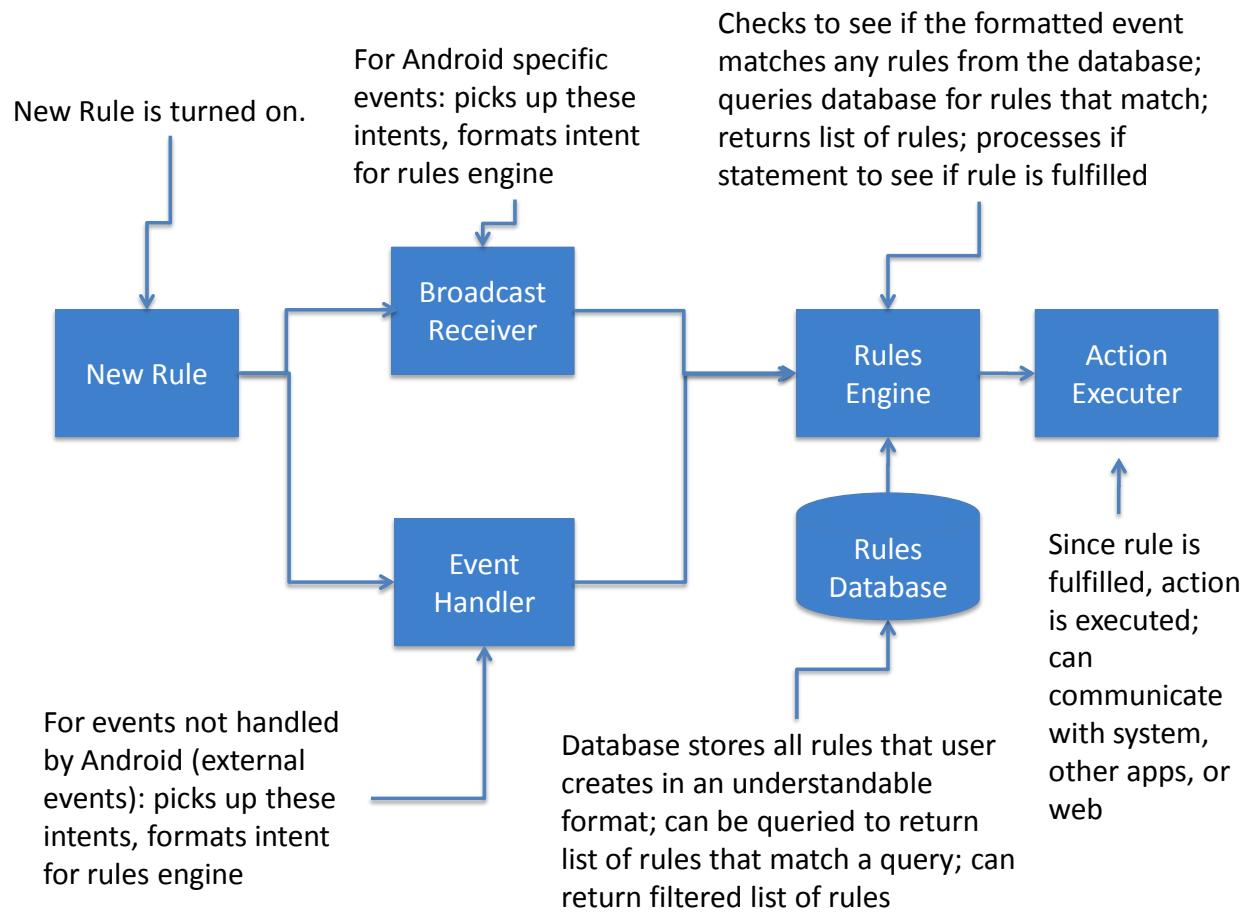
## Android Screen Flow Diagram

### Flow Diagram (Activity Level View)



This Original flow diagram represents the screen flow for users operating the app

## Rule Handling Workflow (Backend)



This flow diagram represents the initial backend design for the app.

# Final Design

## User Interface

The user interface (UI) of the application is the portion of the application that the user directly interacts with. This is really the only portion of the app that the user will ever see. This portion of the application is critical, as it molds the application experience and determines the users' overall satisfaction with it. The interface is constantly changing from version to version, since it must cater to the user's need. The interface must be user centric, and a bad interface is one which is difficult for the user to complete tasks and acceptance tests. We specifically chose to leave out any sort of tutorial for the first release of this senior design project since the user interface will constantly be changing in each revision during our proof of concept. In the long term, our client wishes to avoid the need for a tutorial to teach the user how to use the application, because we feel that the interface must be intuitive and centered on the user. If the user cannot figure out how to complete user stories without instruction, we must redesign to fix this rather than train the user to use our interface. We do, however, now include a help tutorial on Wordpress that is private to application users.

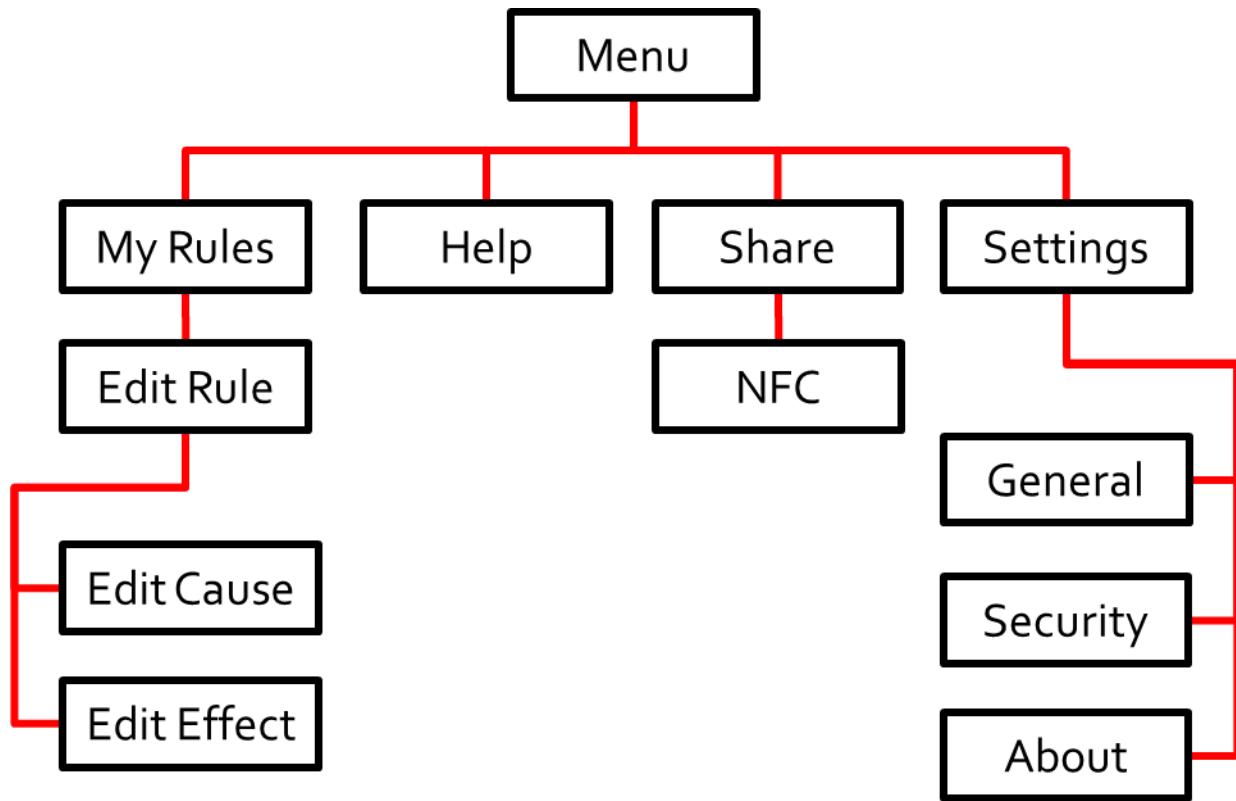
As new causes, effects, rules, and features are added to the application, the UI was updated to reflect these changes. The initial screen shown to the user is the main menu. This menu is the first navigation to be interacted with, and lets the user continue to view a list of the rules, create a new rule, view help documents, change user preferences, and share rules with other devices.

Each screen was designed to be intuitive and have a natural flow while navigating from screen to screen. These activity screens were also designed to have a vertical orientation, to avoid the confusion of screen rotation. Certain features were important, so we used color contrast, size, font, and location to emphasize these elements of the design. Like many other Android features, we designed our application update each change while the user makes changes to rules. As an example, the user can press once in the Wi-Fi settings to turn Wi-Fi on. Our clients expressed interest in having our application have a similar instant gratification feel. All changes to be effective immediately could include all CRUD operations: Create, Read, Update, and Delete. As soon as a user makes a change, the changes are reflected on the GUI and in the database.

Continuing into the app, each new screen shown to the user is known as an activity in the Android operating system. The operating system keeps track of background activities in memory in a data structure known as the “back stack.” The back stack is a LIFO collection of all activities created by the application. Each time the application creates a new activity, it is added to the top of the stack. Each time the user presses the back button on the device, the current activity is “popped” from the top of the stack, destroyed, and the next most recent activity is brought into the foreground. It is important to ensure that the application does not create too many activities, because it can result in slower performance over an extended period of time. To combat this, each time a new instance of an activity is requesting to be created, a flag is set on the request to ensure that an instance of activity does not already exist in the back stack. If the activity does not already exist, a new one is created. If the activity already exists and buried in the back stack, the application will locate the existing instance, pop all newer activities off it from the stack, and bring the instance into the foreground. This limits the stack depth to at most 3 or 4 activities.

Navigation within the application is one of the most important aspects of the interface. A bad navigation scheme can result in confusion to the user and a bad overall user experience. As shown in Figure 1 (below), our application takes advantage of a hierarchy structure to center the navigation around. The user is able to organically navigate through typical workflows from within each activity, and go deeper into the hierarchy to complete tasks. The “back” hardware button on the Android device will take the user back an activity based on temporal navigation, meaning the last activity seen. This can be analogous to the back button on an internet browser. The “up” button is located on the top left corner of the application and consists of the application logo and a back arrow. This button is used to ancestral navigation and will take the user up one level in the hierarchy regardless of the path the user took to get to the current activity. This is similar to navigating up one directory in a folder structure.

The application also takes advantage of the Android SDK’s “action bar” for further navigation. Except for home screen, each activity has a menu bar at the top of the screen. This contains the “up” button, in addition to a variety of additional button specific to the activity. Each activity will have a different workflow, so it is important for us to make sure this is tailored for the optimal experience for the user, and does not get confusing. Most typical activities contain a new rule button, and a share rule button. These buttons navigate the user to the respective activities. The menu overflow button is includes in almost every activity, and will contain navigation to tertiary actions, such as the help page or the settings page.

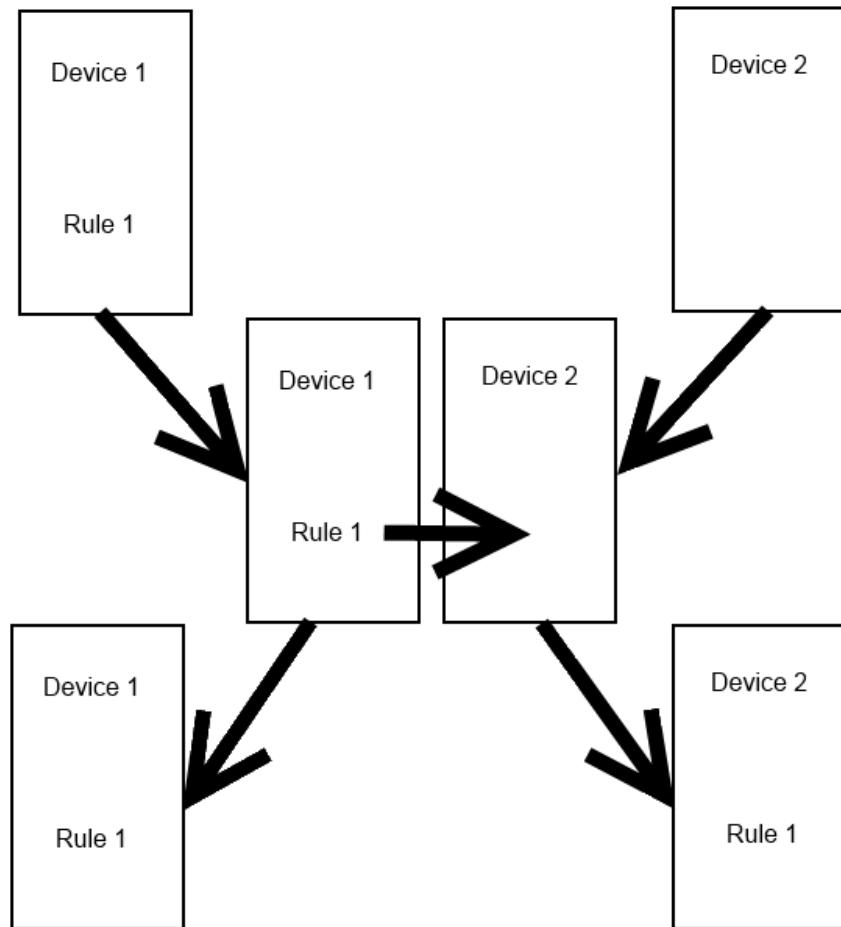


**Figure 1.**The Application Hierarchy

Future designs of the application will remove the initial menu screen from the application and jump straight into the “My Rules” activity. We felt that it was important to push the user directly to the content as efficiently as possible, and the menu screen adds an additional level of confusion. Although the application and client logo were initially a benefit of this “splash screen,” they were mostly for branding purposes, and we were able to emulate good branding by following our client’s color scheme, and using our logo on the action bar. Furthermore, the future designs of the application will also take advantage of the contextual action bar. This is a modification of the existing action bar and will come into play when the user is executing specific tasks. For instance, the user will be able to select one or many rules by touching the checkboxes next to them (or a popup menu could be added). The original action bar will be temporarily replaced with the contextual action bar containing task-specific action buttons, such as edit, share, and delete.

## NFC Sharing

Users have the ability to share their rules with other users via NFC. To access this feature, the user must choose the “Share” button from the main page, choose a rule to share, and press their phone/android device to another, and push the screen to share the rule with the other device. The sending device sends the selected rule in plain text form. The receiving device will parse the string and create a new rule in the database. Both phones will need the application installed, but the receiving phone does not need the application to be open to receive a rule.



## The Application Service

The entire Rules Engine, consists of the Broadcast Receiver, Rules Engine, and the Action Executer. The Broadcast Receiver moves into the Rules Engine class, and then finally fires the Action Executer. The Action Executer is only fired if the causes are evaluated to be true. This entire engine is actually stored within a separate service that handles the application's backbone. The service is created when the application is first opened and has longevity compared to a simple Broadcast Receiver. The key feature of the service is that the Broadcast Receiver is registered to the service instead of the application itself, and therefore will run smoothly regardless of the application's current state. The user can end the process or "kill" the application and the rules will still be stored securely. This fixed many problems with false positive causes and random data being evaluated for location and Wi-Fi based causes. Simply, the entire backbone is run in its own process, is much more permanent, and evaluates data from all types in the correct manner.

## Broadcast Receiver

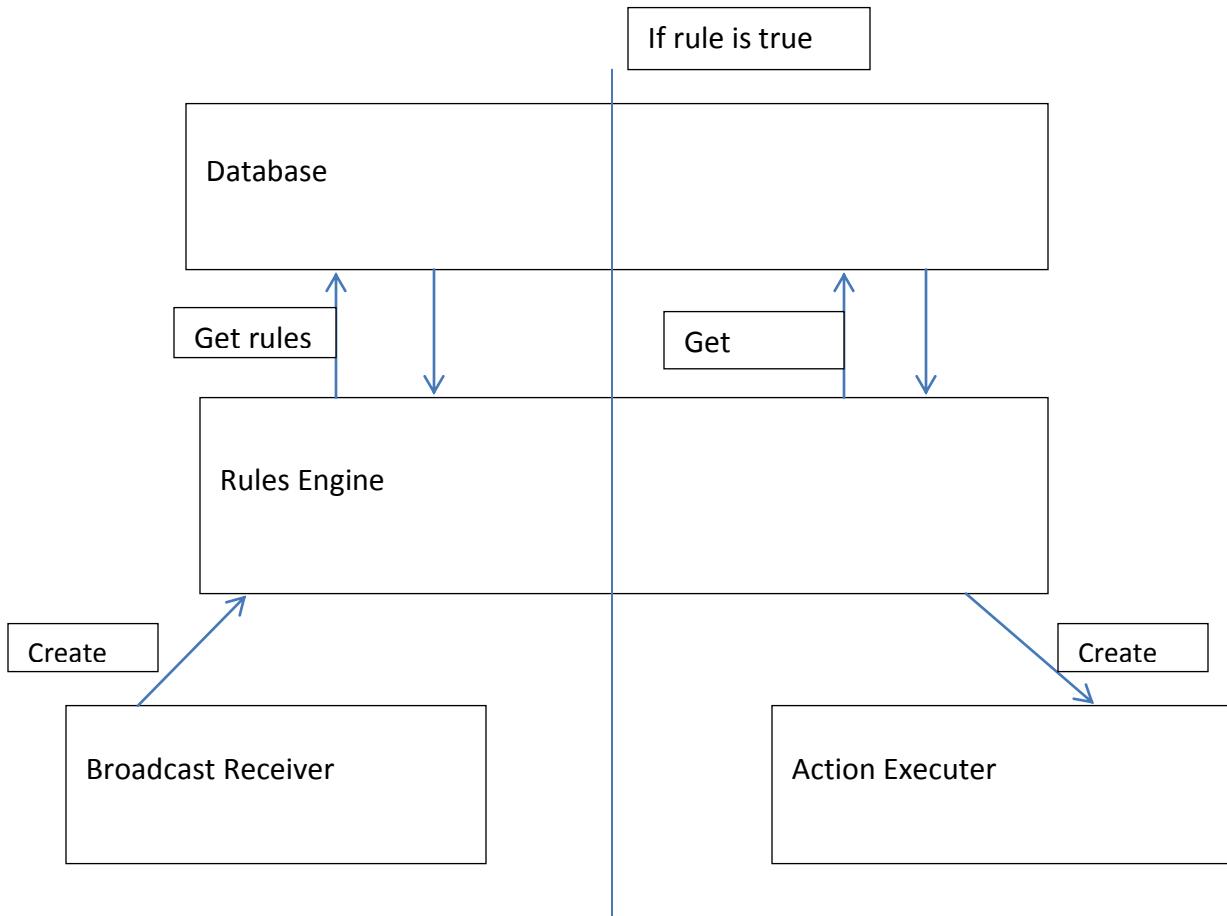
The first part of the rules engine is the broadcast receiver. The broadcaster receiver is a class built into the Android API that either sends or receives messages to/from the Android operation system. In our case, the broadcast receiver receives the different cause types and sends the data to the Rules Engine to process the causes if they exist in the database. The reason why the broadcast receiver is so light is because the broadcast receiver is only given 10 seconds to operate after it receives a broadcast. If the broadcaster receiver is still active after 10 seconds, the Android operating system will kill the process and the intent is lost. Therefore, we kept the broadcast receiver light and commands are sent off with the minimal amount of processing. The broadcast receiver is also designed to work in the background so that it will be active even when the application is no longer focused (on the service).

## Rules Engine

The main function of the rules engine is the evaluation of rules. The rules engine object is what determines whether a rule is true or false. A rules engine object is created by the broadcast receiver once an event has been triggered. The broadcast receiver passes in the cause type, parameter and the context to the rules engine. The rules engine uses the type to send a request to the database for all rules of that type. Once the list of rules has been received, the rules engine evaluates each rule in separate threads using Android's AsyncTask.

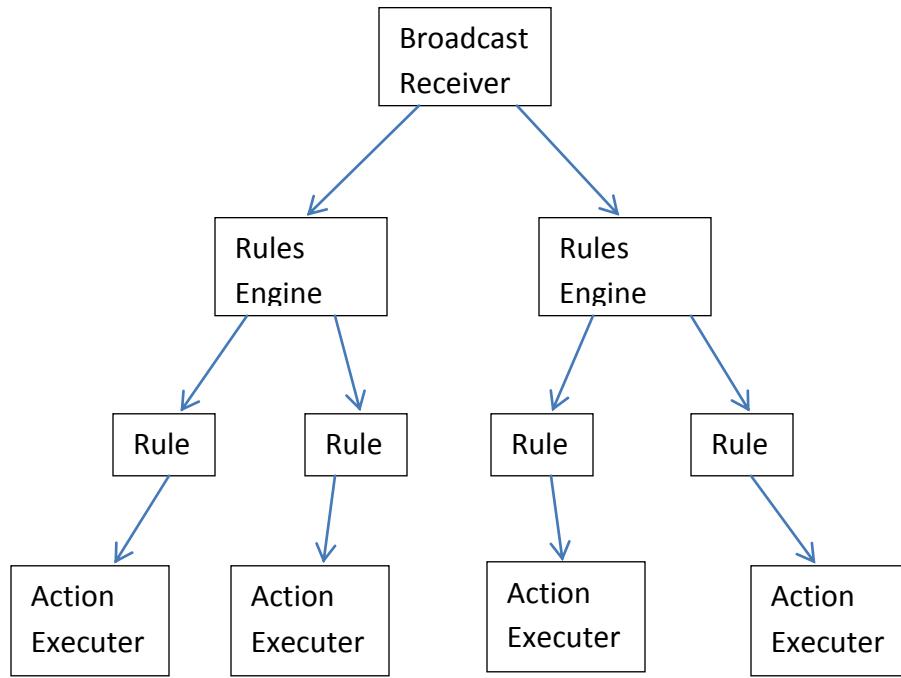
In each thread, an expression tree of rule causes is built using a stored character string in each rule. This expression tree is then evaluated recursively in a depth-first fashion. Each individual cause object within the tree has its own `isTrue()` method that returns either true or false depending on what value is evaluated for the cause. Using these returned values, the expression tree will return a final value, true or false, to the AsyncTask. If the tree of the rule returns true, then the task will request the effect list for the rule from the database, should it exist. This effect list is then sent to an ActionExecuter object for execution, along with the context initially passed from the broadcast receiver. If there are no effects, the rules engine will not create an ActionExecuter, since there are no actions.

## Structural View



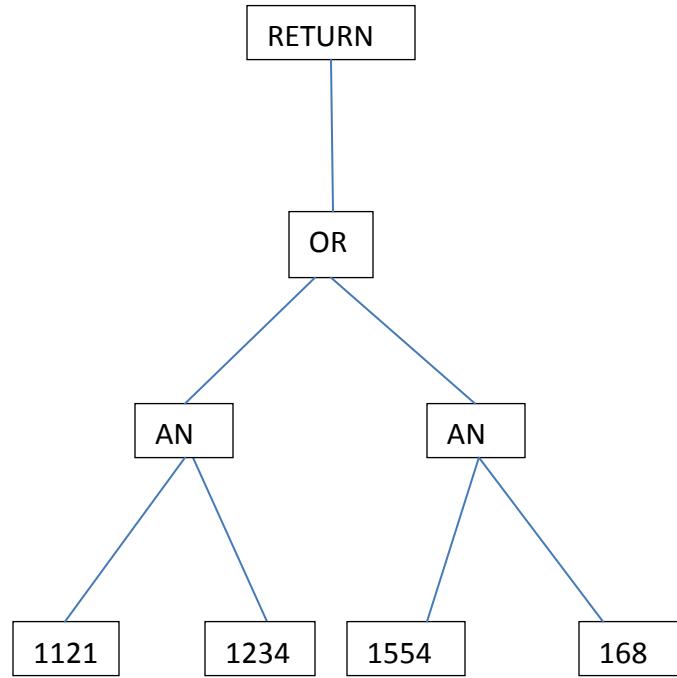
The rules engine interacts with the broadcast receiver, the database, and the action executer depending on if the rule cause tree returns true or not. If the rule evaluates to be false, then the right half of the view does not occur. This all occurs on the service.

## Thread View



Each rule stemming from a rules engine object is evaluated in its own thread. Should the rule return true, then an effect list is fetched from the database and an action executer object is created. This takes in the effect list and executes each effect.

## Expression Tree



This is a view of the expression tree used to evaluate rule causes. This particular tree is the tree for a Boolean algebra equivalent of  $(1121 \& 1234) + (1554 \& 1685)$ . The character string stored in the rule for this tree would be "1121,1234,&,1554,1685,&,\$". '+' indicates an OR node. '&' indicates an AND node. The expression tree string is stored in such a way that the tree is built from the leaves using a stack. Every '&' and '+' pushed to the stack causes two nodes to be popped as its children in the tree from the stack. The '\$' symbol represents an end of line.

## Individual Cause Evaluation

Causes are evaluated using a function containing a switch statement that chooses evaluation methods based on the rule type.

Pseudo code for cause evaluation:

```
Bool isTrue(eventType, param)
{
    // event type is given to use by the broadcast receiver
    Switch(eventType)
    {
        // each case returns true or false depending on specific
        //conditions of each case

        Case wifi: // do something
            Break;

        Case phoneCall: // do something
            Break;

        Case time: // do something
            Break;

        Default: return false;
    }
}
```

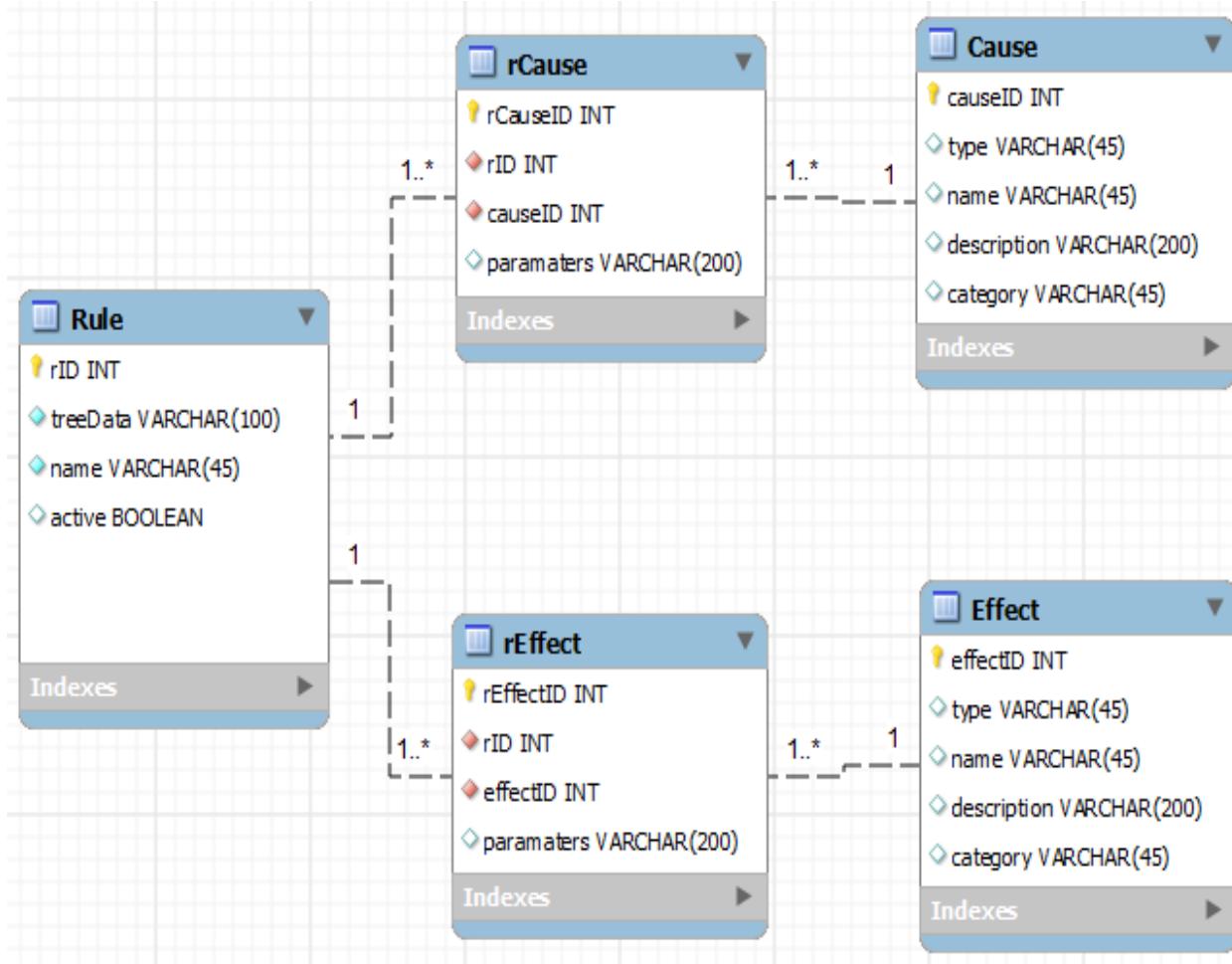
## Database

While developing the structure of the android application we decided that we needed a full database to hold all of the relevant information that we would need. This database would keep track of all causes, effects, and rules information so we needed it not only to be large, but secure as well. We inquired with Samsung to see if they had any preferences on what we should use to accomplish this task, and were presented with TouchDB (<https://github.com/couchbaselabs/TouchDB-Android>). We started playing around with it to try to get used to it, but found the documentation for it to be subpar. After this setback we decided to go with SQLite since we needed to keep the database lightweight to keep our app from taking up too much memory on the phone. The database has to be constantly running in the background, but still contain all the search power of an SQL implementation.

We had to set up a structure for how we wanted the database to look. This included how we would like to phrase the queries of the database. We started creating the objects that we would be using in our program to decide how many objects we would need to store in the database. We had to keep the database lean while also meeting any parameter requirements for each object. We also had to keep track of links between the objects, so that we could pull out an entire rule from the database when needed. There was a bit of a steep learning curve between not only learning SQL, but then learning the limitations that were present when using SQLite on a mobile device, which was a challenge due to the large number of tables and foreign keys required for our schema.

We have to first start by looking at how the objects were made and how that information would be stored in the tables. The cause object is one of the important objects, so it needed to hold a lot of information: cause id, type, name, description, and category. The cause id would of course be the primary key to keep track of which cause you were referencing in the foreign key in the rCause table. The auto-increment functionality was often used, as we discovered that it could be set on table creation, and assigning a primary key actually renames the column to whatever you called it. The cause table would be kept separate from the rCause table, since the causes store information on the different triggers, and the rCause table stores information on a cause that is associated with a rule. The effect tables were basically mirror images of the cause tables.

When analyzing the rule objects, one can see that the tree structure of the rule is stored as a string. This string structure was designed to be in a format that the database could store (string). The Rule table also has its own primary key ID. This rule ID is referenced by the rCause/rEffect tables. Below, you will see a display of how the tables are linked together:



After that whole database is created, then there was a need to be able to call create, read, update, and delete functions from Java. To accommodate for this, the database handler was created to store all of the database commands in one class. Some of the most obvious commands were to insert causes and effects into the corresponding tables. Whenever you start up the application for the first time, the application will insert all of the causes and effects from either a text file (to possibly be implemented later) or just hard coded (current implementation). For inserting the rule, you also need to be insert the specific rCause and rEffect information since these are important parts of the rule. The application is designed to keep all of this information so that the user doesn't have to create new rules every time they open the application. This created a need for the database to be kept permanent, so a file is made that keeps track of all the information in the database. If for some reason the app crashes, or you

stop the application for any reason (shutting off the phone, ending the process), all of the information is stored securely. This file is read on application start-up and loaded for the next database initialization.

After creating tables, and inserting all of the information into the table rows, the application had several queries that needed to be implemented to handle basic application interaction. The original implementation included returns for all cause, effect, and rule objects (separately). This was quite simple, as an SQL statement would pull all of the information out of a certain table, and put the information into an ArrayList (Java). After some further development, the application got more complicated, and more complex calls were required from the database. The first method was to return all cause objects where the category of that cause is equal to a given category. With a string parameter, the category was used in the SQL query to create a result table that contained all of the needed information. This data was pulled from the table using a database cursor. The same query also needed to be made for effects and rules. The rules query was much more complicated since the rule object also contained the rCause and rEffect tables along with external columns from other tables using joins (since there is no category column in the rules table or the rCause/rEffect).

There also needed to be a function to return a full cause object given a cause id. We also needed a quick function that received all rule objects where the type of that rule is equal to the given type, only requesting the tree string information. This was most important for early parts of the application when you are pulling up some information quickly to display to the user, and this shouldn't take up much processing power. Another method was made to return an effect list given the rule id, which made sure that we had all the method calls for the other parts of the rules engine.

As we added more application functionality, more database functionality was needed including the deletion of rules, rCauses, and rEffects. As we got larger sets of rules, we needed to make sure there were no duplication of causes or effects in a rule. This quality of life update came as a change that made the program more efficient by not allowing the user to create nearly empty rules with illogical information.

## Action Executer

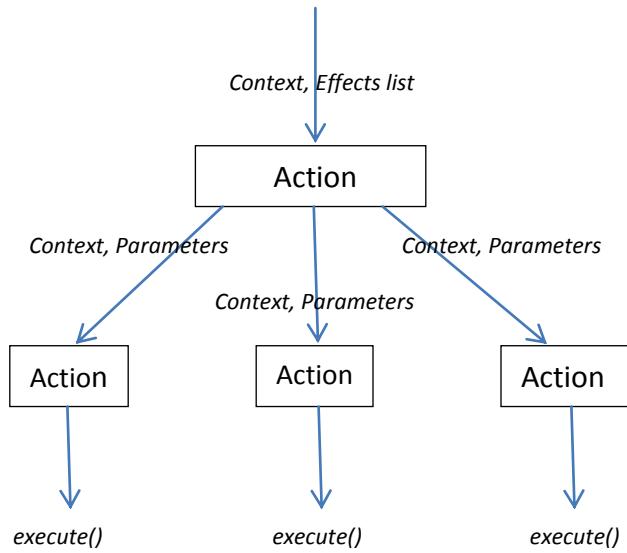
The main function of the action executer is to carry out the effects of a positively evaluated rule. The action executer is what carries out the actual actions that are associated with a rule. The action executer object is created by the rules engine each time a rule has been evaluated as true. Many action executers can exist at one time since multiple rules can be triggered, and each can trigger multiple effects. The rules engine passes the context of the current state of the application as well as an array list of the rEffect objects from the rule. Once the object has been constructed, the action executer will loop through all the effects in the array list. In each loop, the “type” of the effect is enumerated and evaluated as the argument in a switch statement. This is more efficient and organized than using many if statements. Each individual Effect that is evaluated would create a new action object and execute the action.

Each rule type has a unique action class that inherits from AsyncTask. AsyncTasks are Android’s native method of threading and allows for the parallelization of actions being executed. Each thread will be made for an action that needs to be executed. For instance, a separate thread is created to execute a “toast” (Android’s most simple notification to the user). The individual action classes are passed both the context of the action, and the parameters needed to execute that action in the form of a string. The constructor of the action will parse the parameters to make sense of one long string before executing the action. Each action type has a unique method for parsing the parameters because different rules require different parameters (some effects do not have any parameters). The “execute” function of the action class contains source code that will carry out the individual task, using both the application context and the parsed parameters.

The specifications of the action executer require it to be very lightweight and adaptable. The possibility of several action executers existing in memory and executing effects at one time is very probable. The action executer must also be able to execute any type of action. It must be universal enough to create an action object of any type needed and pass the necessary application context and parameters.

Our current version of the action executer remains relatively unchanged from the initial design. It has been stable and has not caused any problems at all with the application. Any changes to the action executer have been adding new effect types for execution.

## Thread Diagram



Each action object stemming from the action executer is executed on its own thread.

## Cause and Effect API

During the last three iterations, one of the additional challenges by Samsung was to create an API for Cause and Effect for Android. This API should allow 3<sup>rd</sup> party developers to develop their own Cause and Effect types that could be added to the application.

After reviewing the current state of our code, we came to the conclusion that it functionally worked well, and could efficiently deliver all of the original functionality that Samsung requested. However, the application code was not in a state that would allow for an API. In order to allow for this, we split the AlphabeticalList class into three sections, one for Rules, Causes, and Effects. This design split up the functionality of these three Activity classes, which originally were all contained in the same FragmentActivity, AlphabeticalList. We then created a Cause folder, and an Effect folder. These folders hold files for causes and effects, respectively. In order to create a new cause or effect, the cause/effect must have a front end for the user to choose parameters for that object. It must also have a back end to display and handle this front end. When the user submits arguments to this cause/effect edit screen, the information is stored in a string called “parameters”. This string can be in any format that the developer needs, but the developer must keep in mind that they will need this information later when the cause is evaluated, or the effect is activated.

For cause objects, there must be a function to evaluate the cause. Since we needed to make the causes “pluggable,” it was necessary to ask for sample rCause object that contains said function, so that a Cause of that type could be added to the system. The parameters in the rCause object are used to store information needed to evaluate the rCause. As you will see below, the rCause file is included for this reason.

For rEffect objects, there must be a class that extends AsyncTask so that the rules engine can create a thread that will do an action that corresponds to the rEffect object. The parameters in the rEffect are used to execute the user defined action when triggered (e.g. Text for a Toast message effect). Samsung could decide to provide a sample class for 3<sup>rd</sup> party developers who wish to use this API. Our examples of this are contained in the effect folder, and are prefixed with the word “Action.”

**Therefore, the API contains 4 files, each corresponding to our objects:**

Cause.java  
Effect.java  
rCause.java  
rEffect.java

**Using this structure, a 3<sup>rd</sup> party developer could create the following files for a Cause:**

**CauseEdit.java** – Cause Edit Screen: Back End  
**causeEdit.xml** – Cause Edit Screen: Front End  
**rCauseExample.java** – rCause Object Example: Fill in the isTrue() function!  
**CauseExample.java** – Cause Object Example: Include information about the cause (Name, Description, etc.)  
NOTE: This needs to include triggers needed for this cause in the Broadcast Receiver.

**A 3<sup>rd</sup> party developer could also create the following files for an Effect:**

**EffectEdit.java** – Effect Edit Screen: Back End  
**effectEdit.xml** – Effect Edit Screen: Front End  
**EffectExample.java** – Effect Object Example: Include information about the effect (Name, Description, etc.)  
**ActionEffect .java** – Create an AsyncTask that will execute the effect

For both Causes and Effects, there are multiple files that are needed for a new Cause or Effect type. However, these files are simple and straight forward, and allow the 3<sup>rd</sup> party developers to be creative, but still restricting the access to within the bounds of the application. Our idea was that these groupings of files could be packaged as “Plugins”. These plugins are what could be used to include new Cause and Effect types in the application from 3<sup>rd</sup> party developers. The plugins can also speed up the internal creation of Causes and Effects. Plugins would be stored in the corresponding Cause and Effect folders within the project.

Our other client influenced goals during the last two iterations were to have a bug free application, and to avoid feature creep. Instead of trying to build an engine to integrate the plugins, developed the API, and built a system that could integrate with many plugin ideas.

Since we didn't want to start with features so late in development, we came up with a handful of ideas that could all work for handling plugins. These plugin integrations would be considered a "Version 2.0" release (this FDR is the "Version 1.0").

Here are the plugin ideas that we came up with:

1. Submitting Cause/Effect plugins to the primary development team
2. Plugin Market
3. Local Storage
4. Cause/Effect Plugin Creation with the Application

First, we should analyze the idea of submitting to the development team. With the plugin structure, it would be the most secure for users to have a screening process by Samsung. Samsung could also optimize or tweak code if need be, and then drop the plugin into the application. They only need to include one line of code to add the Cause or Effect into the database (which would be really simple). The development team at Samsung could then push an update on the Google Play Market. This update would include the new cause or effect types that everyone could then use within their applications. This process would be the same as if a Samsung developer made a plugin.

The second option is the plugin market. This option allows users to upload their own plugins to a market in which other users could search through this market (just like the Google Play idea) and choose Causes and Effects that they want to include. Using this strategy, the lists of Causes and Effects would remain small by default, and would only include additional types that the user specifically wants. There could still be a screening process by Samsung, and then files can be placed on this market instead of directly into the application. This is our favorite option, but would require the most work/maintenance.

The third option is to store plugins in local storage. Users can create their plugins and store them in a local data folder on the phone (or the cloud), and would have to share Causes and Effects more manually via email, or NFC. If plugins were stored on the cloud (for this idea or others), there could also be an online plugin management site. This idea is the least secure since plugins would avoid a screening, and could contain malicious code.

The fourth option is to create a similar flow to the current Rule creation process, but instead for Causes and Effects. These plugins could be created and stored internally. This could also include an option for direct code input, or options to look up plugin files already stored on the phone, and then package them all together into a plugin. This idea is safer than the local storage, since the application itself could check each step for malicious and invalid code. However, this would require a lot of development to be done on the application itself. As mentioned, the flow wouldn't have to be too much different from the rule creation flow, so there could be some code reuse to speed up this development process.

All of these ideas could potentially be implemented in a 2.0 release. However, this was out of our project scope we have not made any attempts to implement this plugin engine. We did design the code to allow for this plugin engine to be completed by Samsung at a later date, as well as other future works (see Future Works).

# Risk Assessment and Management Plan

Risk	Type	Probability	Consequence	Strategy
The application does not adapt to all targeted Android devices.	Technology	High probability	Tolerable	We will program our application keeping in mind that our application must be dynamic and usable across a variety of devices, regardless of hardware, API level, and screen resolution.
The application leaks personal information to hackers.	Technology	Moderate probability	Serious	We will be using the highest available levels of security (OAuth 2.0) to encrypt user data and account information.
Team members are frequently sick or unavailable to work on the project or meet.	People	High probability	Tolerable	We will communicate frequently about our school work and upcoming assignments when assigning project tasks. We will set times and importance for our meetings, so that all team members will be available and accountable for each other.
Changes to requirements that require major design rework are proposed.	Requirements	Low probability	Serious	We will frequently be communicating with our client to be sure they are satisfied with our progress and results. Any changes to requirements will be up for discussion and source code will be distributed and dynamic enough to adapt.

<b>Risk</b>	<b>Type</b>	<b>Probability</b>	<b>Consequence</b>	<b>Strategy</b>
The time required to develop is underestimated.	Estimation	Moderate probability	Serious	Team members will regularly meet to discuss current status and estimate deadlines. As different tasks and components are completed, team members will take on new tasks to complete.
The source code is inefficient and the application is slow when executed.	Technology	Moderate probability	Insignificant	Our team will program efficiently and constantly review each other's source code to look for areas of improvement.
The test devices are not sufficient for testing.	Tools	Low probability	Insignificant	Members of the team have personal devices to test on. New devices may be purchased for the purpose of testing.

# Agile Development Schedule

Agile Development has been quite an interesting trip so far. Agile development is essentially short sprints called “iterations.” The entire requirements, design, implementation, testing/verification, and release process is inside of each sprint of agile development. The requirements are broken into development needs called “User Stories.” User stories one of the most unique parts about agile development. When a new iteration is started, user stories are chosen and ranked in a game called “Playing Poker”. This allows developers to get together as a team, and coordinate which tasks will be achieved during any given iteration by group decision and analysis of user story difficulty. The group ideally takes on an appropriate amount of user stories and begins the iteration with these user stories defined. Each user story is very similar in concept to the change request forms idea from other development cycles. Each user story is its own change to the last release that has been decided upon, and ranked by the group. Typically (and ironically), there is little or no documentation for agile development projects. One feature of agile development is the SCRUM architecture. The typical agile development team meets every day in meetings called “Standup Meetings.” The daily standup meeting is critical to the success of the team. Since agile is such a fast moving development structure, these meetings are used to both check on the progress of development and discuss development issues, clarify user stories, clarify client’s needs, coordinate the group’s efforts, and unite the team members.

The SCRUM master (our team leader, Keith), is in charge of running the scrum meetings for our team. Normally, the SCRUM master is not a developer. However, all 4 members of our team are developers and are responsible for code. This has led to an interesting group dynamic. To spread out duties from the typical agile development team, the acceptance of user stories is handled by Matthew and William. The note taking for all of our meetings internally and with the client are handled by Tomin. Our team meets 4 times a week, at 12:30PM on Tuesday and Thursday, and at 1:00PM Monday and Friday.

Our iterations are 2 weeks in length. Iterations in agile development vary from 2-4 weeks, but each team decides on their own iteration length and sticks to it. Our iterations start on Saturday and go through the Friday that is 13 days later. The finished code with all the iteration’s user stories is due by the Wednesday on day 12 of the iteration. Between day 12 and day 13, all unfinished acceptance testing is handled. If a user story is not accepted, the story is either patch fixed (if possible) or pushed into the next iteration’s user stories as a high

priority user story. Between day 13 and day 14, the team members submit documentation on each user story they completed and on day 14, we plan the next iteration's user stories and rank them using Playing Poker. The next iteration then starts immediately on the following Saturday.

Our iteration plan was to have 5 iterations in fall and 6 in spring. This gave us a total of 11 iterations, so 22 weeks of formal development. During our first iteration, we defined most of our user stories and developed a mock GUI so we could begin testing functionality. With some client feedback, we improved our GUI during iteration two, and added backend functionality on most of the application (outside of rule creation). After our second iteration, we were advised by our client to spend additional time to design the rules engine structure. Our release for iteration 3 was only a rules engine design, and a few cleaned up features on the mock GUI. This structure is the foundation of the functionality of our application, and was complex to parallelize and integrate. To account for this, iterations three and four were designed to be the first and second halves of the rules engine. With school ramping up, iteration 5 was designed to merge, clean, test, and optimize the application before the PDR. Since we did agile development, this presentation also included a demo of our iteration 5 prototype release. It just so happens (thanks to signing up early) that iteration 5 ends on the same day as the PDR.

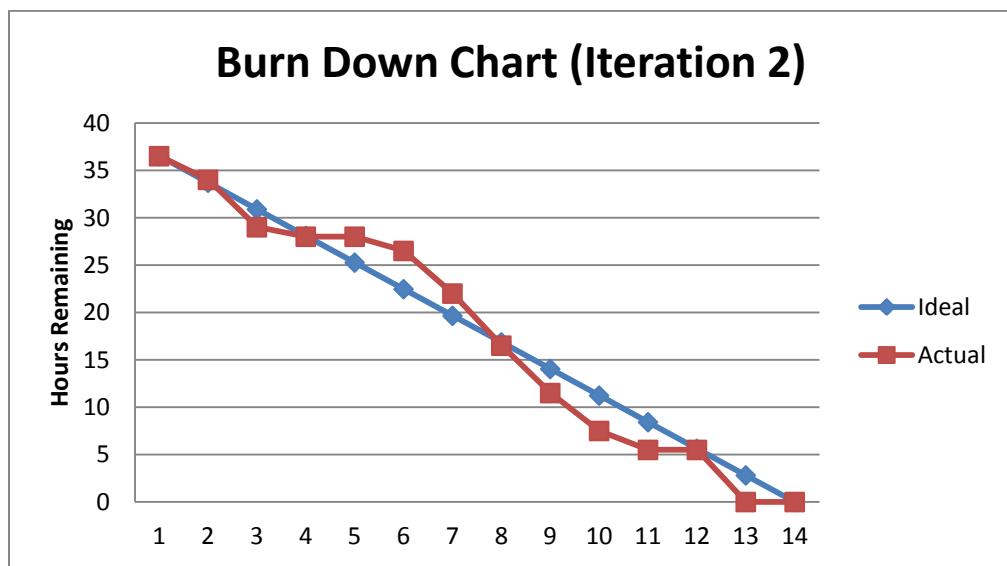
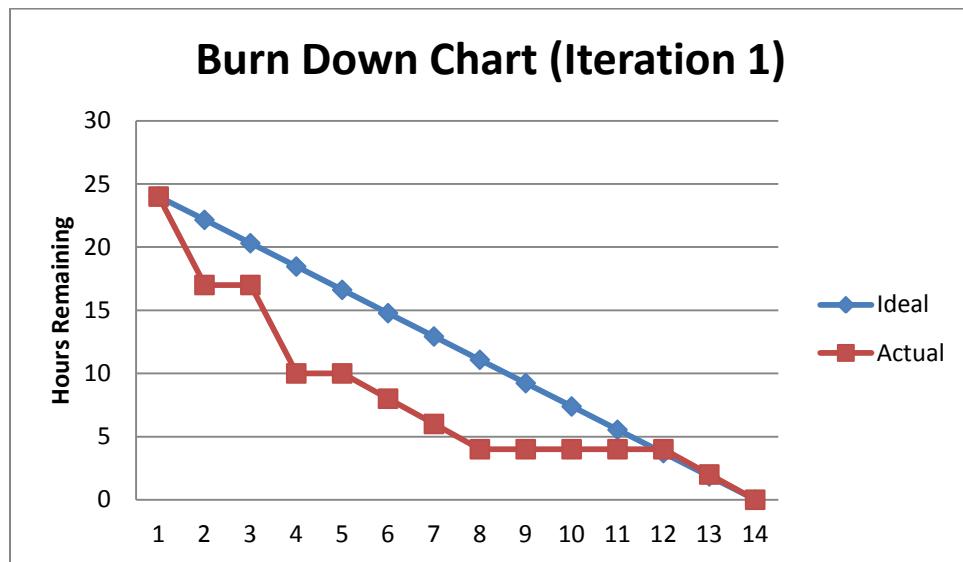
While planning around winter and spring break, we decided to assign a basic list of stories to be taken on during Winter Break, and to make the iteration over Spring Break a 3 week iteration. The great thing about agile development is that all of the user stories that have not been assigned to an iteration live in the backlog. The backlog currently consists of a few functional user stories and the entire list of appendices at the end of this document. The benefit to this is that winter break doesn't have a formal structure encapsulating it (to respect the holidays and difficulty of meeting for SCRUM stand-ups), but instead the backlog will be updated with priorities so team members can work on user stories during the break and clear up that list. Our intention is to set a minimum amount of stories for each person to complete over break and let everyone develop when they have time.

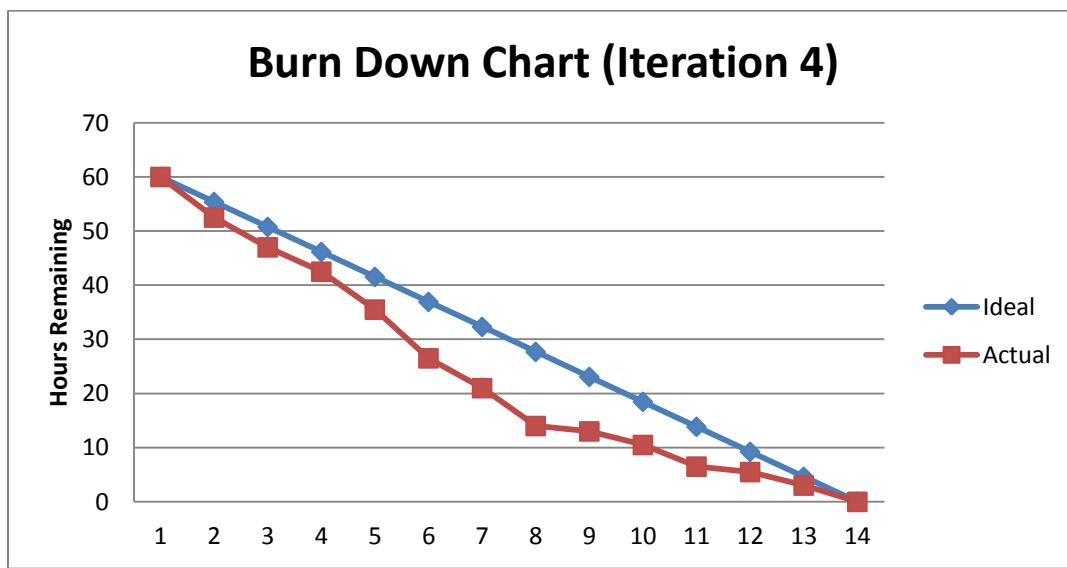
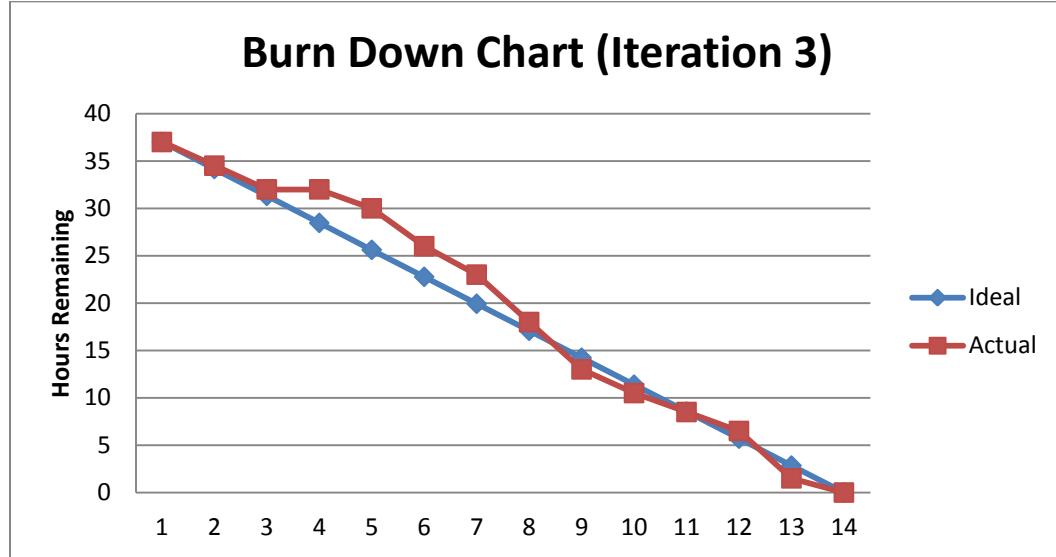
This spring, we have worked really hard. We started off the semester with a presentation to Samsung the first week of Iteration 6. After receiving critical feedback, we planned the structure of our iteration schedule until the MDR (Midterm Design Review). The first 6 weeks (3 iterations) of Spring have not allowed for a lot of sleep, but we have implemented all of the original feature requirements from Samsung. We then presented our MDR to the professors,

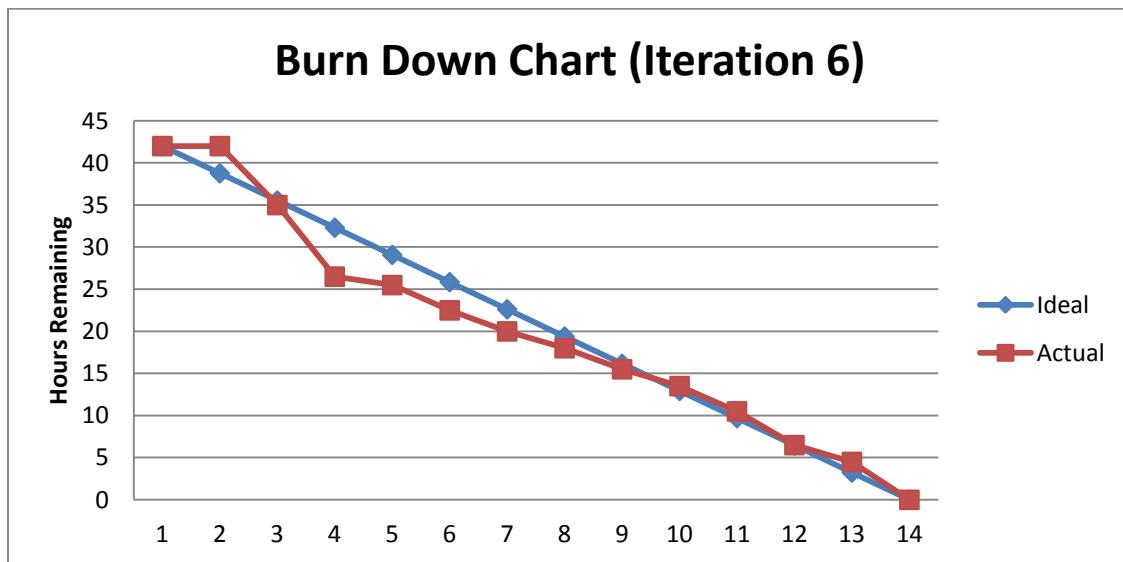
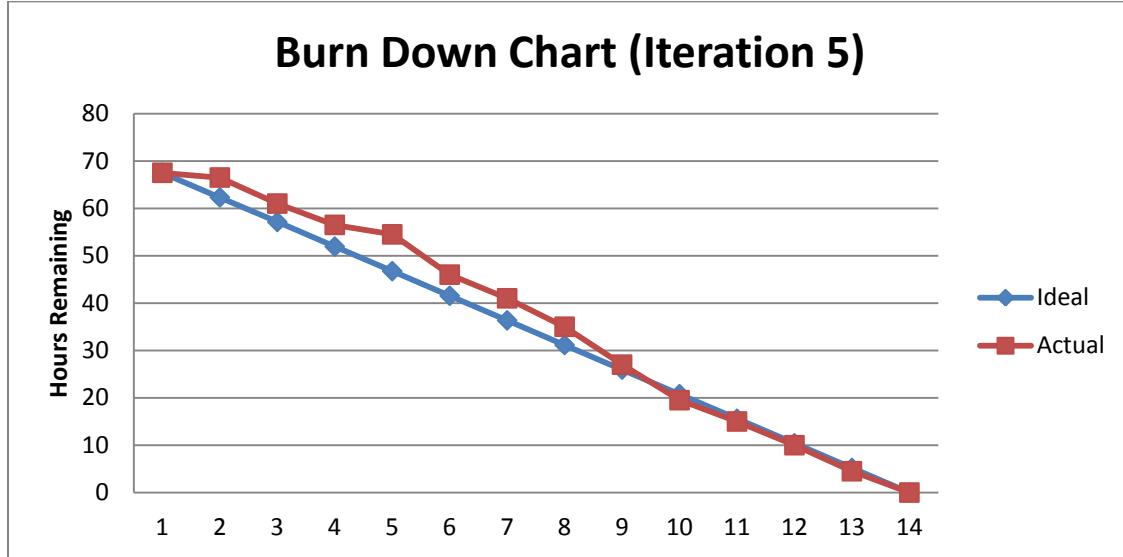
which included our first draft of the Acceptance Test Plan. After the MDR, Samsung requested that we spend more time debugging the application, and try to avoid feature creep. They also challenged us to come up with an API that could allow third party developers to create cause and effect types. During Iteration 9, we created the strucuture needed for the API. Iteration 10 included the Acceptance Test Plan, which allowed us only the first week for development, and the second for testing another team's project. This restraint required us to have a mid iteration release, so all stories were completed before the ATP. For iteration 11, we spent time finalizing, debugging, documenting, and formatting the code. As the last iteration winded down, we kept testing the application and used the last week to create and update documentation for this FDR.

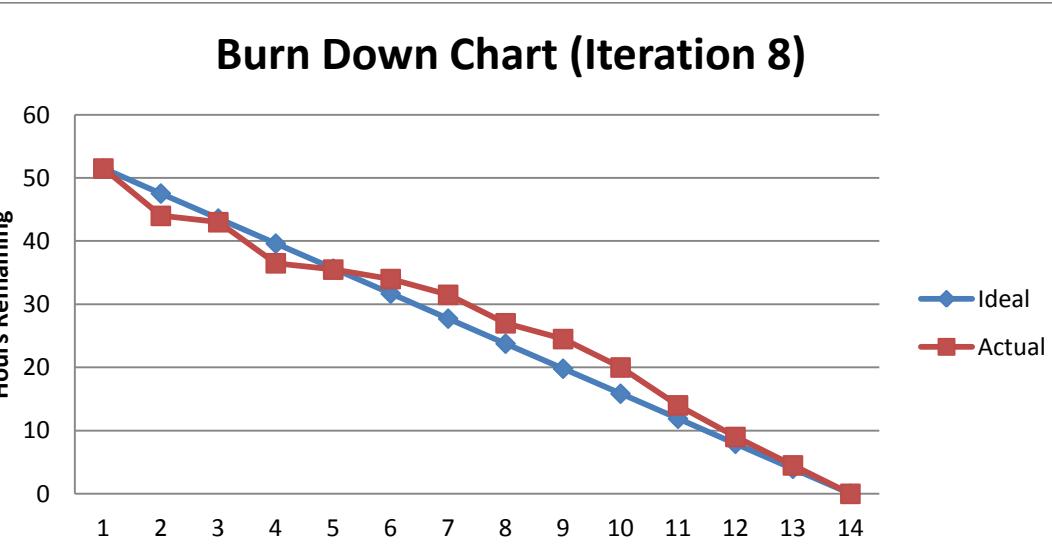
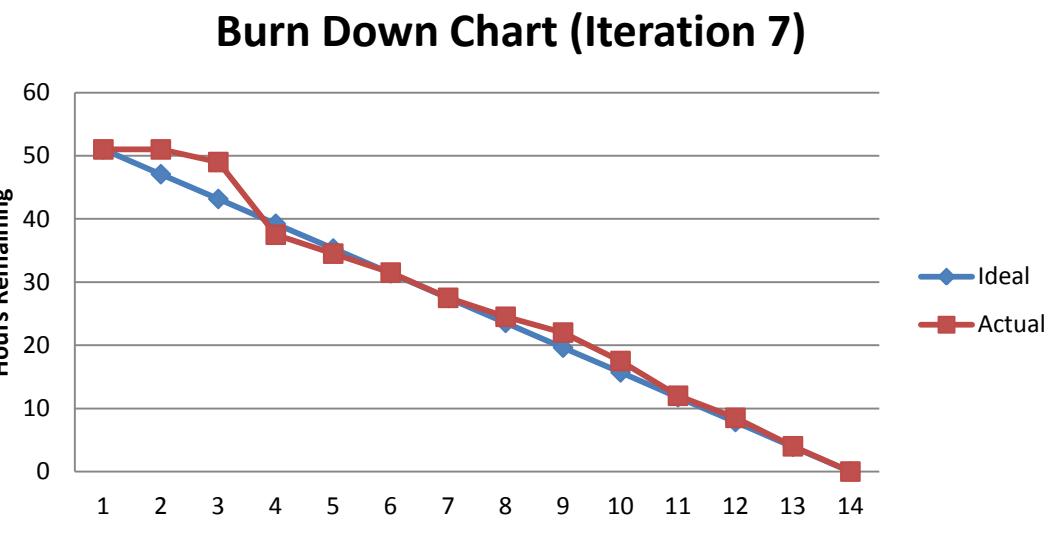
# Iteration Burn Down Charts

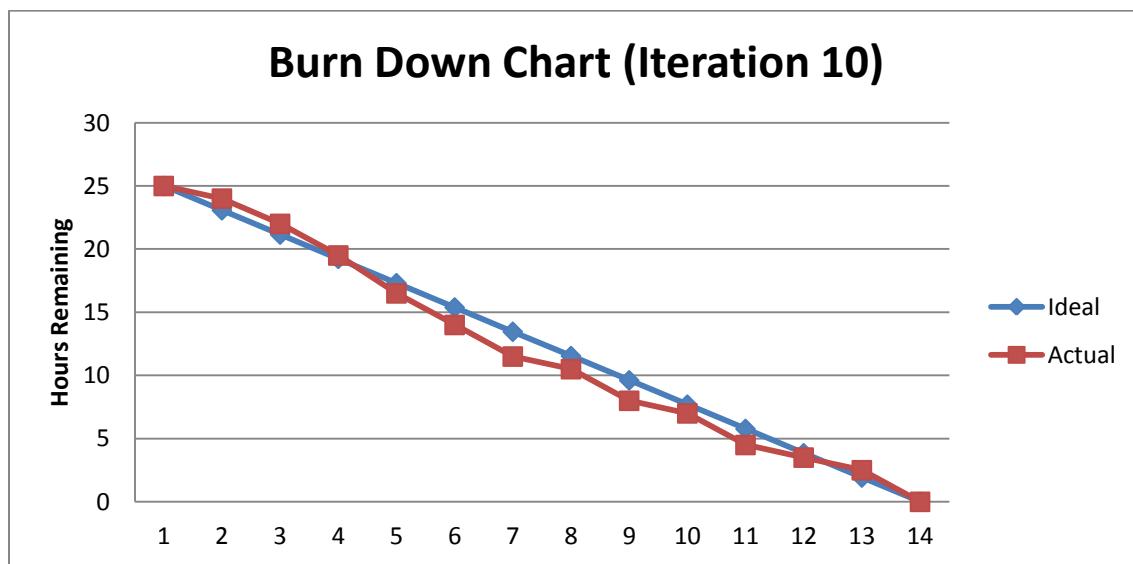
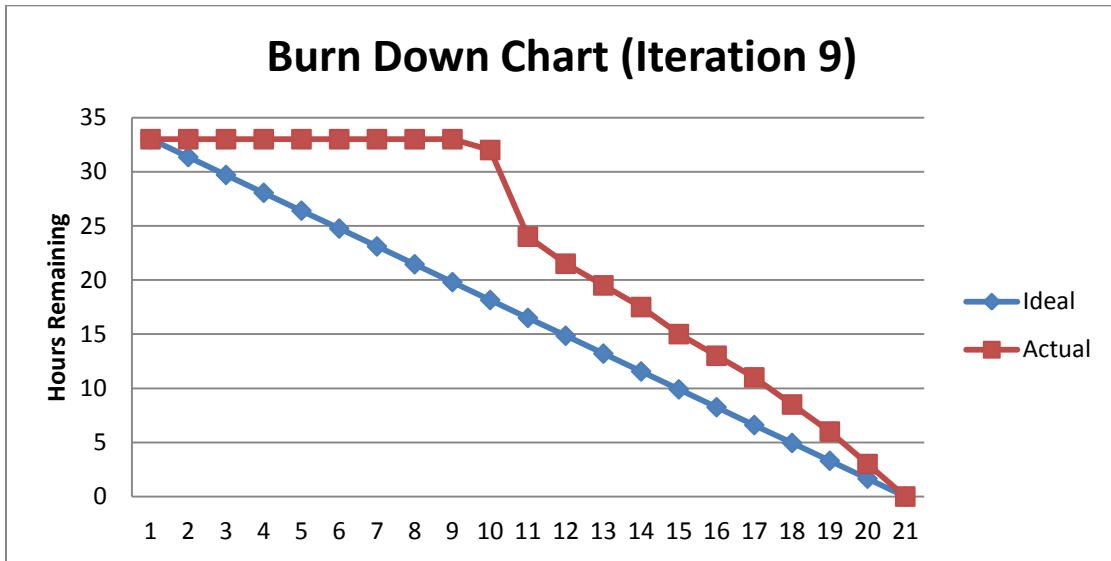
These Burn Down Charts are representative of our work throughout each iteration. They can be used to track our progress and see if the team was either ahead or behind schedule. During iterations, the team's progress can be monitored carefully so everything is completed on time. To create a smoother graph, user stories (chunks of hours left to finish) can be broken into tasks so updates on work are more frequent.



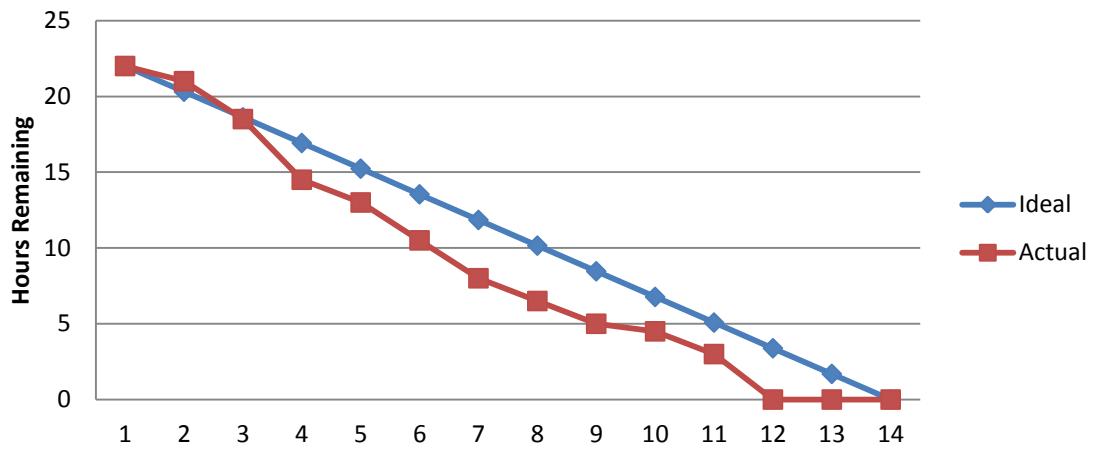








## Burn Down Chart (Iteration 11)



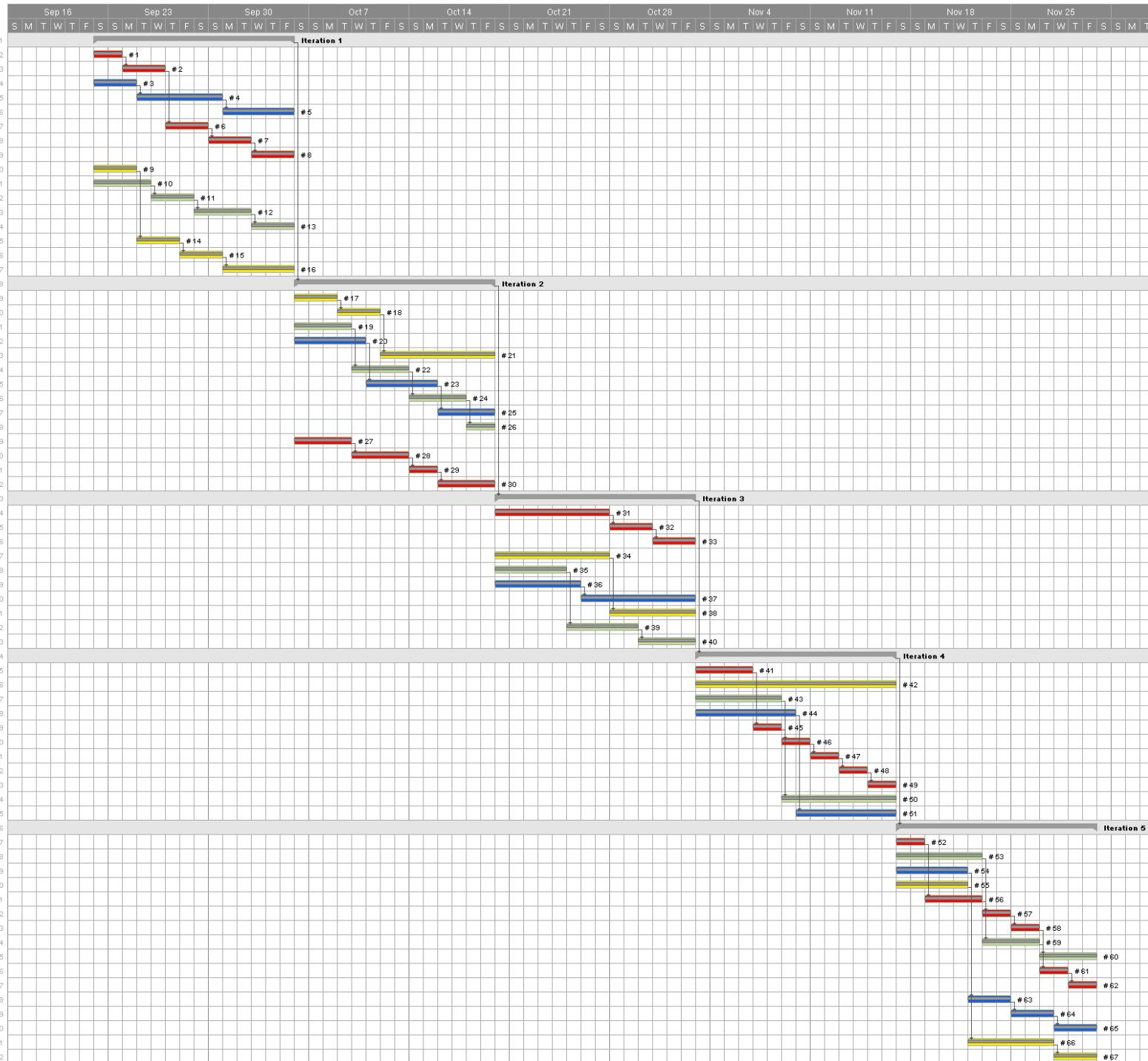
# Gantt Charts

These Gantt Charts represent the entire project, divided by semester, and divided again by iteration.

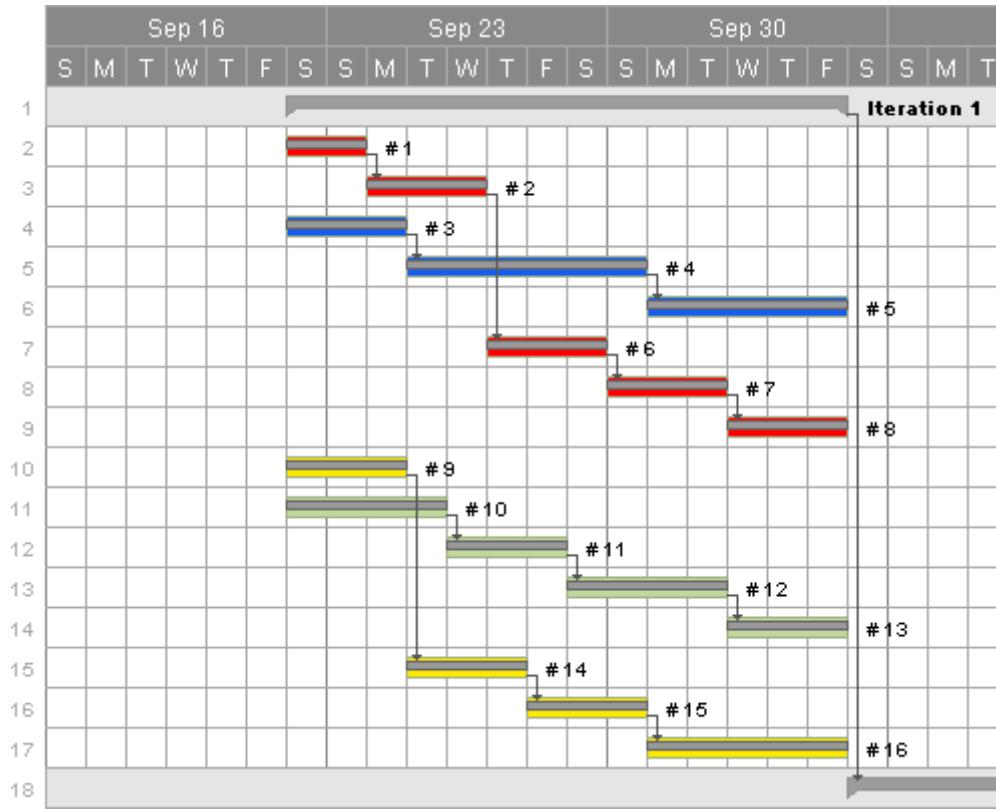
The semester charts can be used to see the overall progress over time, and the iteration charts can be used to identify specific progress on user stories by individuals.

**KEY:** Keith = **RED**, Matt = **YELLOW**, Tomin = **BLUE**, Will = **GREEN**, and Holidays = **PURPLE**

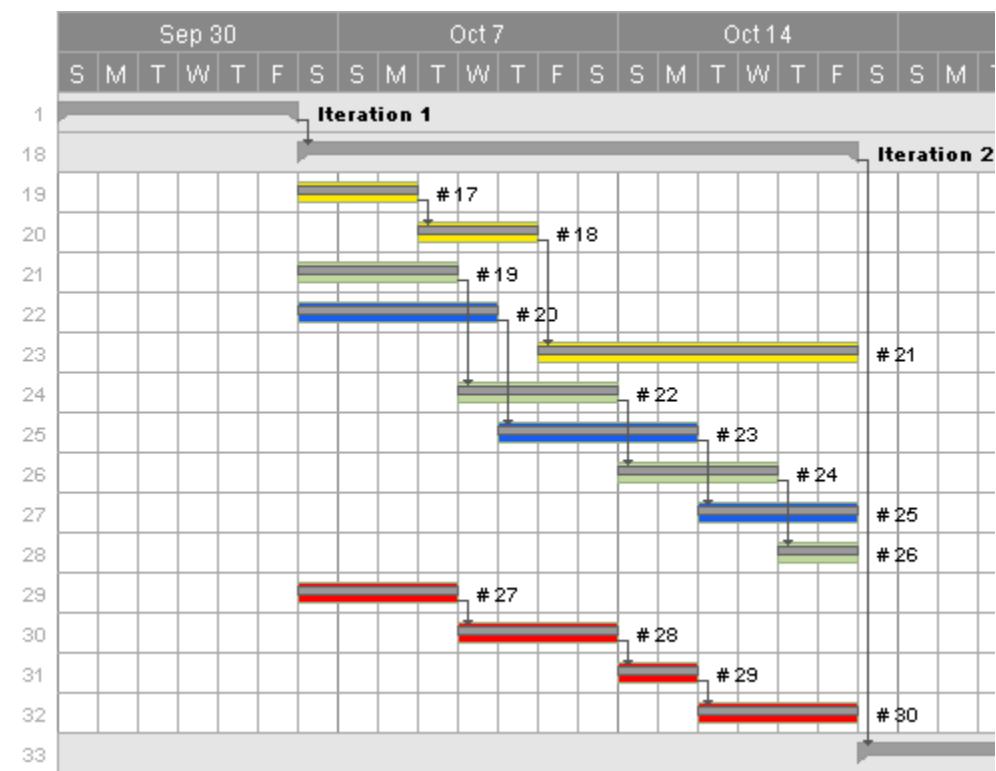
## Fall



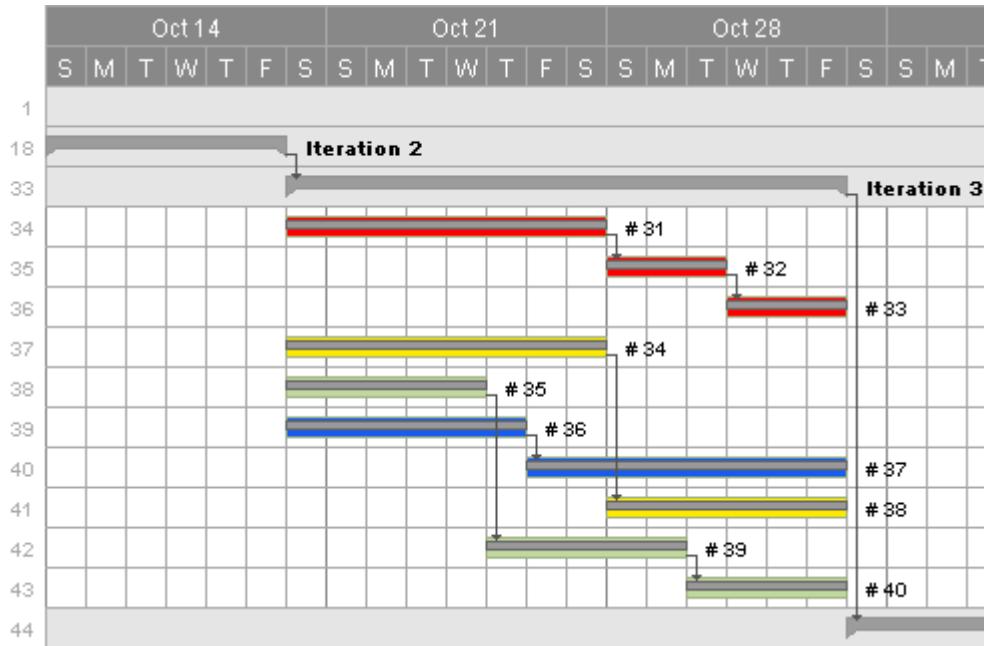
## Iteration 1



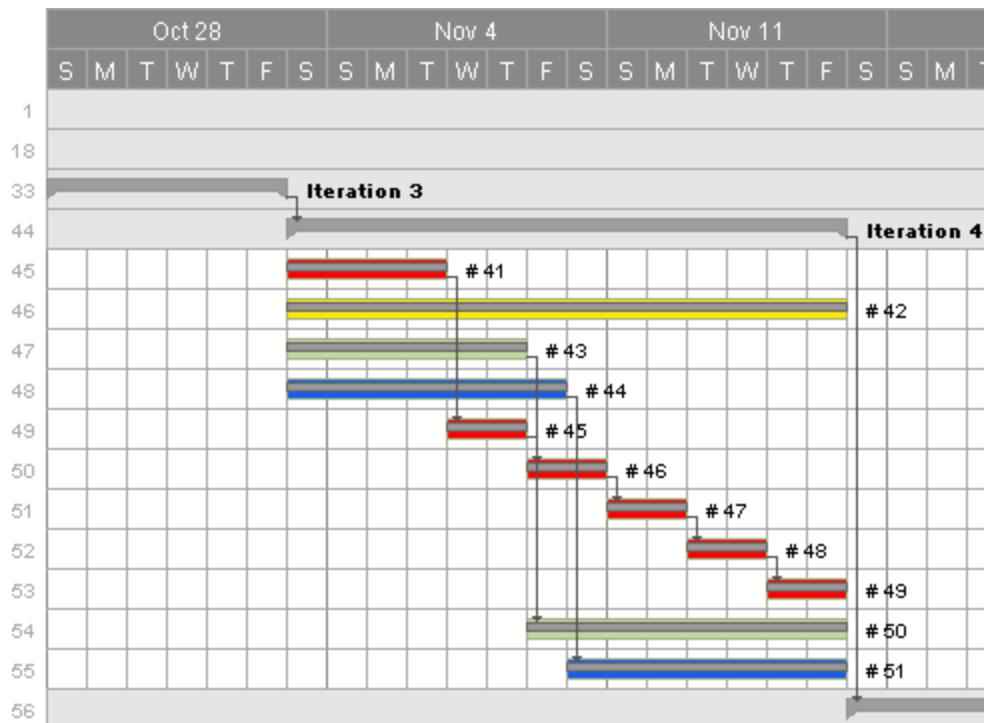
## Iteration 2



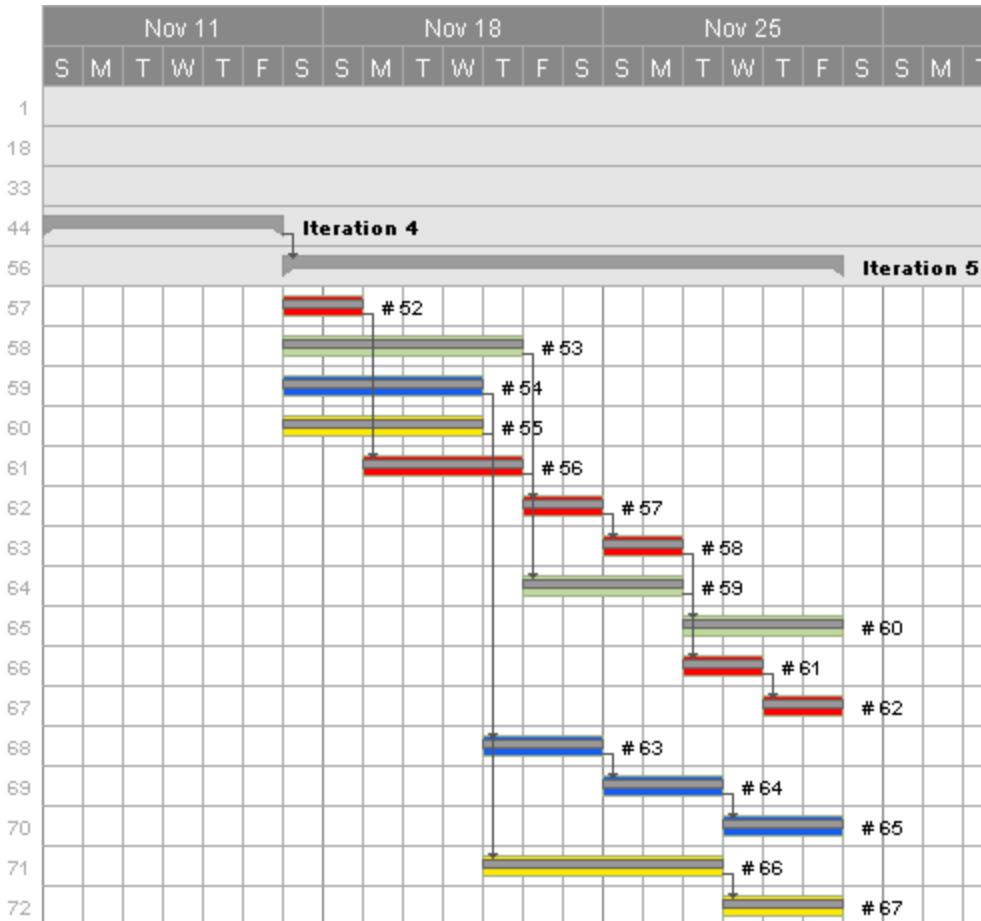
## Iteration 3



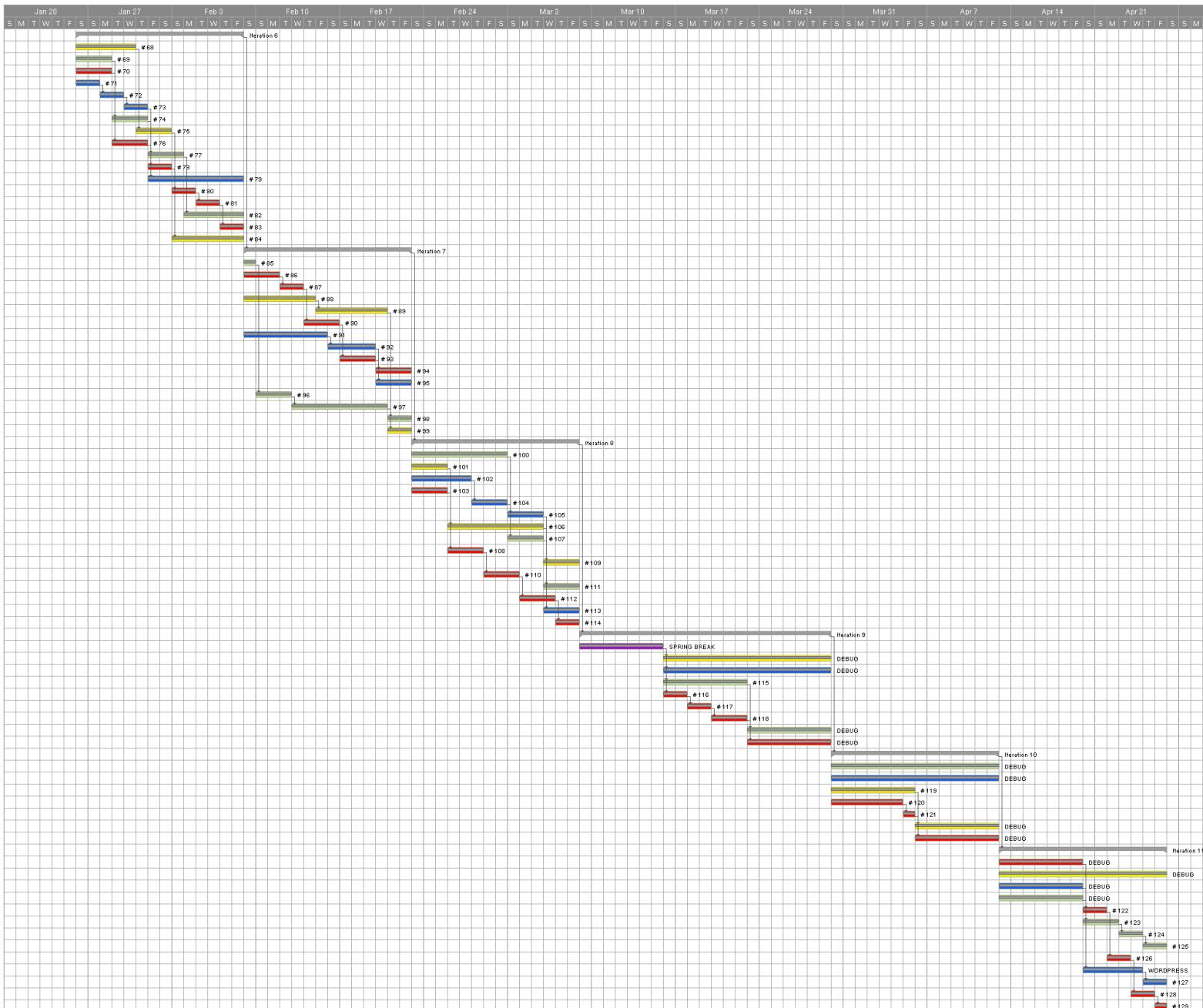
## Iteration 4



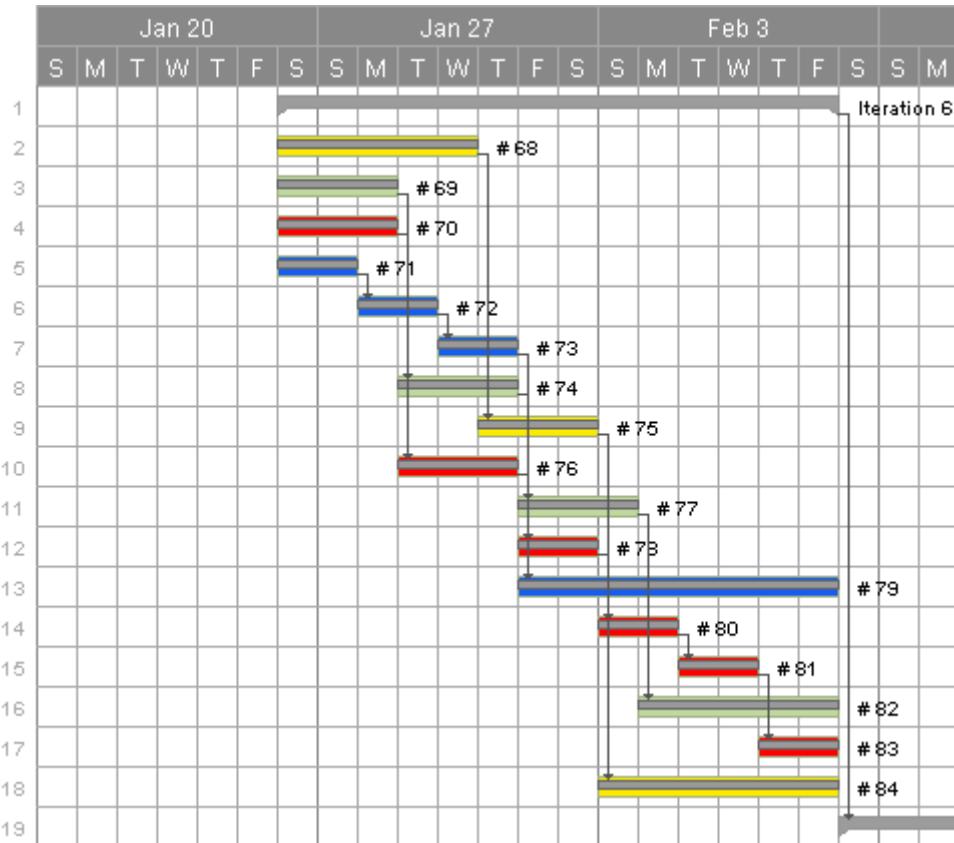
## Iteration 5



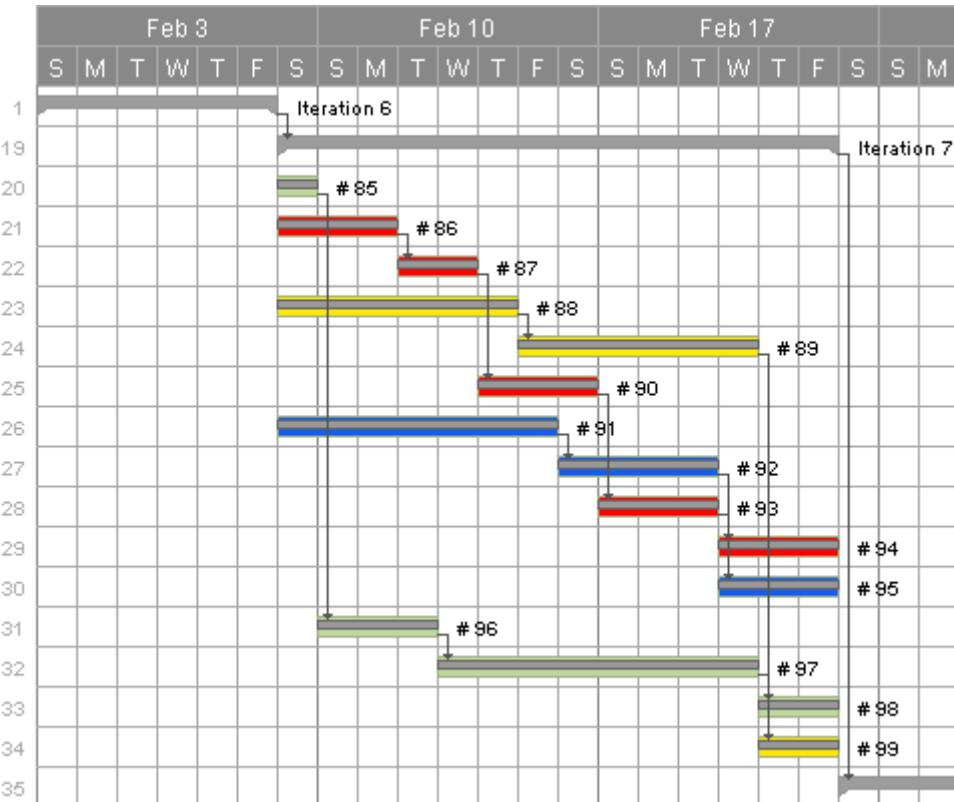
# Spring



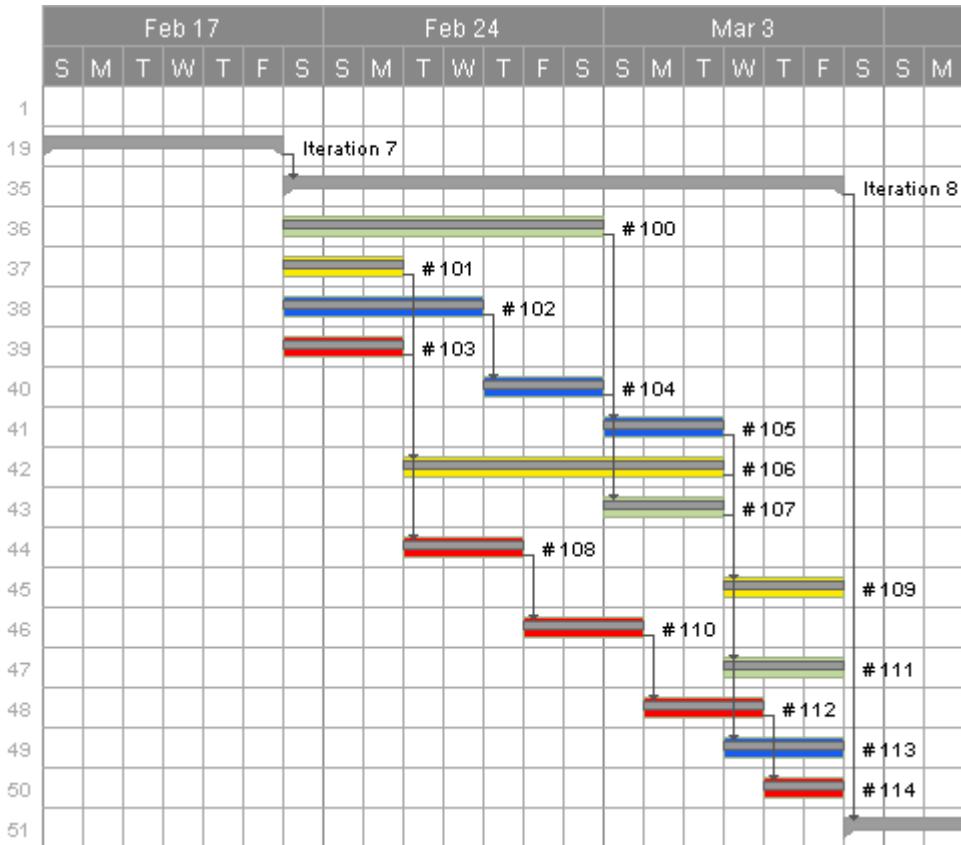
## Iteration 6



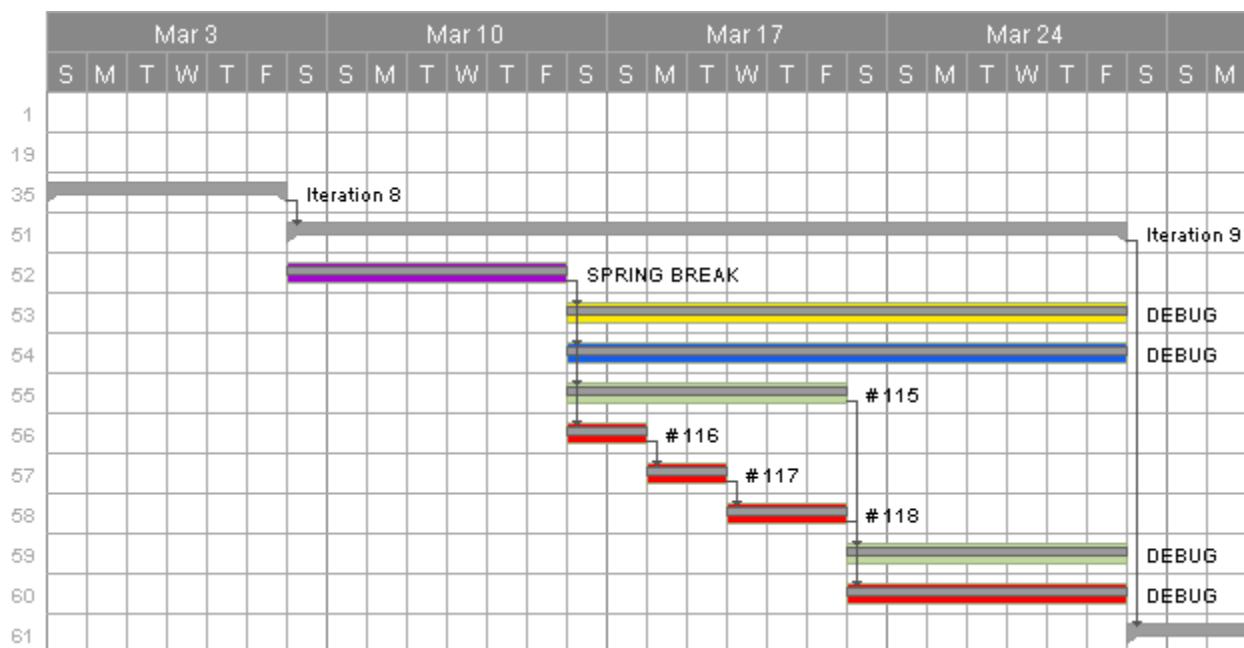
## Iteration 7



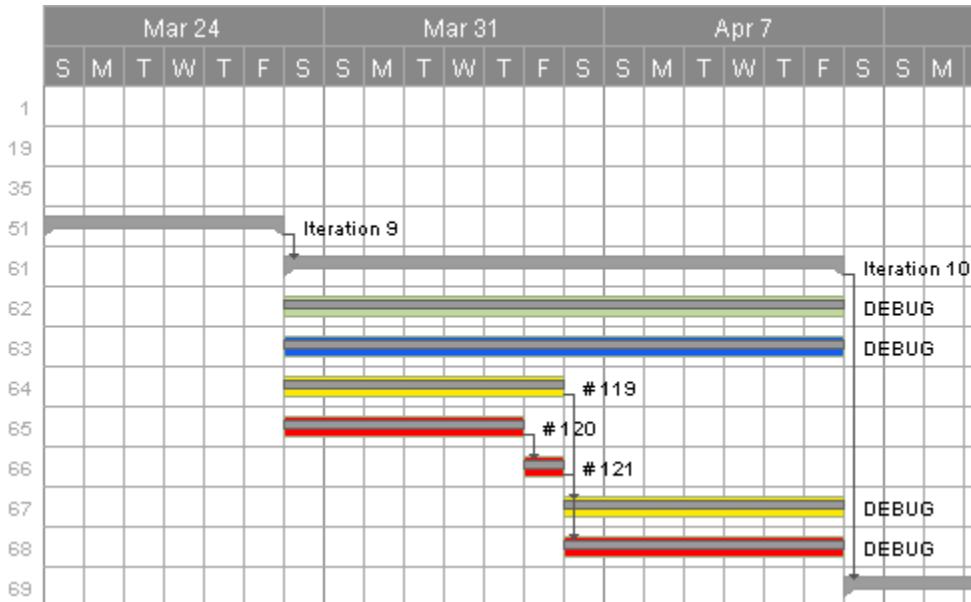
## Iteration 8



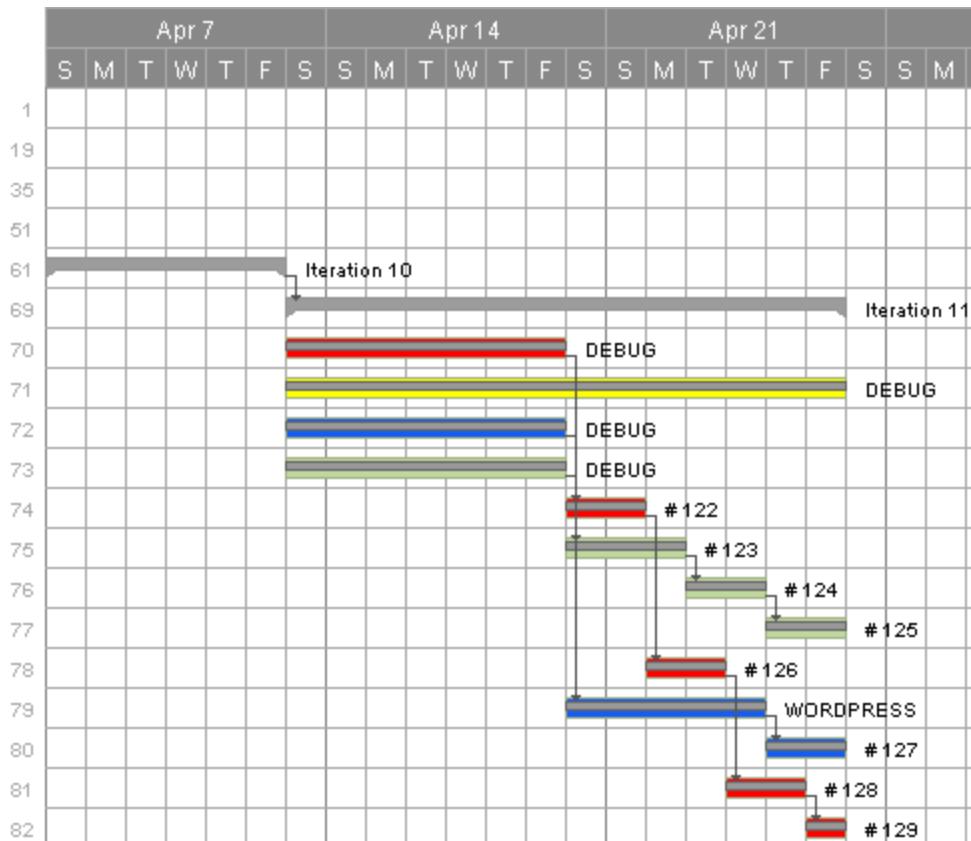
## Iteration 9



## Iteration 10



## Iteration 11



# Final Budget

## Proposal Budget

Project Name: Cause & Effect

**Period #1: 9/22/2012 - 4/20/2013**

Description	Rate	Hours	Subtotal	Total
<b>Employee Salary</b>				
Keith Adler	550	\$13,750.00	\$13,750.00	
Matthew Brannick	440	\$11,000.00	\$11,000.00	
Tomin Kozhimala	440	\$11,000.00	\$11,000.00	
William Vennes	440	\$11,000.00	\$11,000.00	
Employee Salaries Subtotal	1,870	\$46,750.00	\$46,750.00	
<b>Employee Benefits</b>				
Emp Ben, FT Exempt Staff (42%)		\$19,635.00	\$19,635.00	
Emp Ben, Non-Exempt Staff (42%)		\$0.00	\$0.00	
Employee Benefits Subtotal		\$19,635.00	\$19,635.00	
<b>Employee Salaries and Benefits Subtotal</b>		<b>1,870</b>		<b>\$66,385.00</b>
<b>Equipment</b>				
Samsung Nexus S (on Loan)	\$0.00	2	\$0.00	\$0.00
Samsung Epic 4G (on Loan)	\$0.00	1	\$0.00	\$0.00
Samsung Galaxy Tab 10.1 (on Loan)	\$0.00	1	\$0.00	\$0.00
Equipment Costs Subtotal		4	\$0.00	\$0.00

<b>Software</b>				
Eclipse IDE	\$0.00	4	\$0.00	\$0.00
Android SDK	\$0.00	4	\$0.00	\$0.00
Bitbucket Repository	\$0.00	4	\$0.00	\$0.00
Mercurial Source Control	\$0.00	4	\$0.00	\$0.00
Android Open Source Project Code	\$0.00	4	\$0.00	\$0.00
<b>Software Costs Subtotal</b>			<b>\$0.00</b>	<b>\$0.00</b>
 <b>Printing</b>				
PDR Printing	\$69.33	1	\$69.33	\$69.33
FDR Printing	\$9.08	1	\$9.08	\$9.08
<b>Printing Costs Subtotal</b>			<b>\$79.41</b>	<b>\$78.41</b>
 <b>Total Equipment, Software, and Printing Costs</b>				<b>\$78.41</b>
 Contingency Costs (20%)				\$0.00
 <b>Project Costs Total</b>				<b>\$78.41</b>

Our current proposed budget is \$78.41. The budget above represents our total projected budget for the duration of the Cause & Effect Project. These figures for employee salary and benefits were estimated using an hourly wage of \$25.00 per hour. This is the approximate starting salary for a Software Engineer at Samsung Mobile (Glassdoor). The group leader is estimated to work 25 hours per week, while the remaining group members are estimated to work 20 hours per week. Employee benefits were estimated to be close to 42% of the rate of wages and salaries (Bureau of Labor Statistics).

Our equipment consists of four devices given on loan from Samsung Mobile to assist us in testing our application. These include two Samsung Nexus S developer phones, a Samsung Epic 4G, and a Samsung Galaxy Tab 10.1.

In order to construct a software environment that was both robust and economical, our team took advantage of many free and open source software programs. Android Developers provide the necessary Android Software Development Kit (SDK). We are currently using Eclipse as our Integrated Development Environment (IDE) with the Android Development Tools (ADT) plug-in, which provides us with a powerful integrated environment to develop our application, as well as build an app UI, debug, and export app packages (APK) to distribute and test. Our environment also includes Bitbucket hosting service and our team was able to take advantage of using our .edu email accounts to register with Bitbucket and create a free private repository to host our source code. We chose to use the free Mercurial distributed source control to manage our software code at our client's recommendation and our own interest. Finally, the Android Open Source Project provides us with a free developer's website with many resources for reference about the operating system ([developer.android.com](http://developer.android.com)).

Finally, two of our project's review checkpoints will be accompanied by hard copy reports. The PDR have five copies of the final report which will be double-sided printed and bound. The PDR reports each had 61 pages and cost approximately \$70. The MDR reports did not require any special printing and binding, since it was informal, but we spent approximately \$9 on a binder to contain our updated report. Upon final updates from our professors, we will no longer require printing and binding for our FDR final report as it will be submitted electronically.

# Assessment of Context

If this project were to be picked up by Samsung Mobile, turned into a major product that would be featured on the Google Play market, and possibly come as a standard application on all Samsung Galaxy devices, we predict that it would be a success. Since Samsung Mobile is such an enormous company that distributes devices globally, the opportunity for expansion is great. Samsung has sold over 100 million of their flagship Galaxy line devices worldwide, which demonstrates its popularity. This application could possibly come in different languages for different regions, but function the same universally. If this application were to come standard on each device as part of the TouchWiz suite of applications, the potential for exposure would increase drastically, rather than with a more traditional channel of distribution such as the Google Play market or the Amazon AppStore. The utility of the application is universal. Categorized as a “productivity app,” Cause and Effect would be useful for any user who wants to get more out of their device and utilize all of the possible features, rather than the ones that are apparent at the surface.

For Samsung Mobile, the success of the application could drive more device sales. Specifically, if the application were to become more polished and intuitive than any other task automation software, it would be a strength to the devices and the company. Samsung Mobile marketers could highlight the application as one of the Galaxy line’s features during advertising to draw in more customers, comparably to the S Pen or the Samsung Wallet. Moreover, there is potential for a version of the application to hit the Google Play market, which could have advanced features and could generate revenue through ads, or pay for downloading. In addition, the Samsung developer community would also grow, as the Cause and Effect API would potentially be able to grow into an SDK for third party developers to create on top of. In the past, Motorola was able to use this strategy to include its Smart Actions application on all newly purchased devices from their flagship Droid line. Smart Actions functions in a manner similar to Cause and Effect, automating tasks, features, and regulating battery life in the background, so that the user can keep use the device for other purposes. This application is Motorola device exclusive and does not have the large potential for growth as Cause and Effect would.

Since the application itself is not physical, but rather acts as software on a mobile device, there does not seem to be any apparent environmental impact. As the application progresses and evolves, this could change, as more efficient use of the device by user created rules could

potentially make the device use less battery power overall. As a result, the device would act more efficiently and draw less electricity from the battery and not need to be charged quite as often. Consequently, there is a possibility that the application may allow the user to make their device behave in ways that are not efficient to power use and may draw more power than usual, either intentionally or unintentionally. This can be detrimental to the environment as more power may be consumed by a single individual. When this is scaled to many individuals, the effects may cause a substantial draw of power or electricity from the external environment. Since this would not be ideal, we would push users to create efficient rules with best practices to encourage efficient use of the limited power supply of the mobile device.

Finally, the application itself may have some beneficial and detrimental effects on society. When the application finds its way to a majority of user's mobile devices, several unforeseen results may occur. Task automation applications allow the user to convert their mobile device into a personal assistant. Moreover, task automation allows the user to set a rule once on the device and forget that it even exists, allow the device take care of the details in the background, while the user is able to utilize the foreground for other, more practical purposes. Task automation created with artificial intelligence may allow the device to learn from the user and how users interact with their devices. The application may be able to offer suggestions on useful rules that may make the user's life a little easier each day or save time in the long run, as the device and application would take on the mundane and tedious tasks that users dislike about their devices. Issues may occur when users grow too dependent on task automation. If users were to think all devices to take care of mundane tasks in the background, these ideas may lead to unrealistic expectations in the real world. These tedious and boring tasks make up most of our daily lives and are part of a routine that may live by in a moderate lifestyle. To push this to the extreme may be unreasonable, but not out of sight. If users were to expect more and more tasks to become automated, it may project into their daily lives and effect how they see the world.

# Lessons Learned

The purpose of the senior design project is to take all of our conceptual knowledge learned during our undergraduate coursework, and apply them to a real world problems. We were given real projects by real clients, and asked to execute on a timely schedule. One lesson learned by the group has been cooperation through the software engineering process. The engineering method we were given to utilize was the SCRUM/Agile development process. Although it was challenging at first to adapt to these fast-paced iteration schedules, ultimately it became such an important part of the project that we learned to live by the procedure. Daily standup meetings with the team became a part of our regular routine to keep each other up to date on the different aspects of the overall application. We utilized the schedule to set short term deadlines and goals for ourselves and each other. Throughout this project, we have realized the importance of iteration planning. Without the Agile schedule to push us forward, we may have not been able to meet the deliverables as efficiently and timely as we have. We can see why Agile is used so universally in the workplace, as it is an efficient and quick means of getting a project finished thoroughly and on time. This is something that you cannot just read about in some textbook, but is something you have to go out and experience first-hand. Making sure that your code is both effective and efficient at any point in time is an important part of agile programming. Once you have one part of the project done, you want to be sure that it works so that you have a usable product at all times (even if it's not fully complete). Most computer science majors will end up having some kind of programming job at some point in their career that uses this architecture, so having experience already is a huge benefit. In addition, we learned the importance of staying in contact with a customer. Customer contact is necessary for the definition and clarification of user stories and requirements. We were able to clarify many requirements through meetings with Samsung, and as such, were able to implement exactly what they were looking for.

Another useful skill that we learned while working with on this project was the importance of teamwork and coordination when handling source code. At the recommendation of our client, Samsung Mobile, we utilized Mercurial for our source control management. Source control manages the changes made to the source code. Each committed change is a version of the application, allowing our team to upgrade and downgrade between various revisions of the application as needed. This main source is stored online on our repository in Bitbucket. This cloud based repository is safe and reliable in handling our code. Each member of the team is able to “push” and “pull” from the repository in order to finalize changes or add features to the application. We learned the difficult way that when two developers are working on the same

part of the code at the same time, and merge conflicts occur when changes are uploaded to the repository. Initially, we had to manually approve of each change to fix these conflicts, but we learned over time a good workflow to ensure that merge conflicts will no longer happen. This required coordination between team members to ensure that each developer locally tests all changes before pushing to the repository. Source control taught us the importance of having a logical manner to organize our large amount of code among an entire team.

During this project, we were able to achieve broad goals by setting high standards from the beginning that we would eventually have to fulfill. Through careful time management and hard work, we were able to achieve a great result. Since we did not all have experience with databases during college, we also learned a lot about how SQL works, and more specifically SQLite on Android. Learning how databases work and interact should be part of every computer science major's degree since so many projects could be dealing with a database system. Knowing how to search databases and develop complex queries stretching across different tables is of critical importance when searching for specific information in the database. Also, several open-sourced software packages were experimented with to develop our design. One example would be CouchDB (an open-source database API for Android and iPhone), even though we decided not to use it because of its complexity and short-comings.

This project has also been a great learning experience with regards to Android and new technologies. As a group, we did not all come into this project with prior knowledge or experience with Android applications. We learned the importance of continuous learning when it comes to projects and technical skills. Even in the workplace, one must continue to deal with new, emerging technologies in order to stay competitive in an industry. It is important to realize that learning does not cease after graduation from college: you must continue to learn from the future projects and technologies in your career. Having to learn much more about Android for this project has left us confident in our ability to create future applications in Android, and has given me a new set of skills to utilize in the future. Overall, senior design made for an invaluable learning experience.

# Future Work

Although our application ended up fulfilling the initial requirements set up at the beginning of the project, there were several features which we did not get a chance to implement. As our project naturally evolved, we were able to introduce new ideas for user stories that we felt would be valuable in the second version of the application.

First, the future of this application lies in the robustness of its code features: the causes and the effects. We would push for the next version of this application to grow exponentially in terms of the number and variety of cause and effect types. Moreover, we also hope to expand in the future to lead the application to interact with external services such as Facebook and Google. For instance, many have responded positively to the idea of causes and effects based on the Google Calendar web application, as it would be personalized and practical for fast paced users. This external integration would require our application to be authorized by third party server's resources and data. Specifically, we would implement OAuth 2.0, one of the most secure and widely used standards for authentication. This open standard allows for authorization without the need for storing or passing user credentials.

The other aspect of the application that we hope to refine would be the overall user interface and experience. Application design is important to the overall experience because it lures the user in for engagement, and adds to the experience. As our application grows in terms of new causes and effects, we hope to create a sort of category system, and view in order for the user to easily search and access different types of causes and effects. This is important because as the application grows, it will naturally become disorganized, and we must use the application as different users would in order to find the best fit.

Another feature of the interface that we could potentially improve is the navigation. Again, as the application grows, we must anticipate the navigation to grow as well with many different activities. In order to keep this intuitive and not confusing for the user, we must implement the best practices of Android to keep the user directed in completing various user stories. The action bar is the main method of navigation currently that we may expand in the future using the Android SDK, or eventually replace if it no longer serves its purpose. Although our current interface for adding and editing rules has been easy enough for users, we hope to create a drag and drop UI in the future for users to be able to manipulate different aspects of the rules for a better understanding of how it works.

Finally, the third-party API and plugin options should allow for an interface to interact with the application, and maybe even directly with the user. We anticipate the API to be a popular feature that would allow external developers to expand on our original ideas. These ideas for future development are mentioned in the above API section of the design.

Although we were not able to fully implement these features in the proof of concept for this application, we see a large potential for growth within it for the future. As we hand the source code over to Samsung Mobile at the end of the project, it is ultimately up to them to see what features are fit for the second version of the application.

## Source Code Repository

Our Repository can be found at:

<https://Talador12@bitbucket.org/Talador12/ceandroid>

Feel free to request read privileges and/or follow our project.

## Javadocs

Our Javadocs file can be viewed by the private audience of this paper.

<https://docs.google.com/file/d/0B0V8MFEgBSOdSWdzWFpZLTg2cG8/edit?usp=sharing>

This link is *private*.

We recommend downloading the zip file and extracting it.

Then, open index.html in your favorite web browser for viewing.

## Wordpress

Our developer test plan has been uploaded online on a protect Wordpress blog. This may act as a tutorial for the application if needed. The document may be found at:

<http://ceandroid.wordpress.com/>

Password: “dmr”

# References

[Glassdoor]

[http://www.glassdoor.com/Salary/Samsung-Group-Software-Engineer-Salaries-E3363\\_D\\_KO14,31.htm](http://www.glassdoor.com/Salary/Samsung-Group-Software-Engineer-Salaries-E3363_D_KO14,31.htm)

[Bureau of Labor Statistics]

<http://www.bls.gov/news.release/ecec.nr0.htm>

[Android Developers]

<http://developer.android.com/guide/components/index.html>

[TouchDB-Android]

<https://github.com/couchbaselabs/TouchDB-Android>

[Mercurial Source Control]

<http://mercurial.selenic.com/>

[Bitbucket]

<https://bitbucket.org/>

# Backlog

This is a list of all of the low priority user stories on our backlog. It includes all of the rule types that we came up with through hours of research and brainstorming.

## Phone/Physical Rule User Stories

- As a developer, I want to be able to integrate Bluetooth functionality, so that users can use rules and sharing that utilizes the phone's Bluetooth capabilities.
- As a developer, I want the app to feature media message based rules, so that the user can handle complex message types.
- As a developer, I want the application to access the display features, so that the display can be turned on or off as desired by the user.
- As a developer, I want the application to recognize when it is docked, so that the user can create rules based on docking status.
- As a developer, I want the application to be able to send fax messages, so that our users can use this functionality.
- As a developer, I want the application to recognize when headphones are plugged in, so that the user can create rules based on headphones.
- As a developer, I want the application to recognize when an external display is plugged in, so that the user can create rules that interact with the external display.
- As a developer, I want to include rules associated with the calendar, so that users can interact with their schedules using our application.
- As a developer, I want to take advantage of Android (and especially Android 4.1)'s voice commands, so that the user can interact with rules using their voice.
- As a developer, I want the application to be able to open or terminate other applications, so that the user can optimize application flow on their phone.
- As a developer, I want the application to be able to access the phone's memory and storage, so that the user can interact with the phone's data hardware.
- As a developer, I want rules that affect the battery usage, so that the user can optimize battery life using our application.
- As a developer, I want to access the phone's alarm system, so that the user can edit their alarm settings using rules.
- As a developer, I want to be able to toggle airplane mode, so that the user can access this feature using rules.
- As a developer, I want to be able to change the wallpaper using our app, so that users can have dynamically changing or updating wallpapers.
- As a developer, I want to be able to manage files on the device use our app, so that users can manage their file structure using rules.

- As a developer, I want to access the camera, so that users can create rules that can utilize the camera's features (new photo, flash, etc).
- As a developer, I want rules that can access the current time zone, so that the user can utilize different time zones as they change.
- As a developer, I want rules that access email, so that the user can manage their email using our rules engine.
- As a developer, I want rules that can access the web, so that the user can access web content using our rule system.
- As a developer, I want to access the state of the USB tethering, so that the user can build rules from this.
- As a developer, I want to be able to build rules for a wireless hotspot, so that the user can utilize their over expensive data plan or rooted phone.
- As a developer, I want to be able to access the phone's Radio, so that there can be rules that use the radio.
- As a developer, I want to be able to access different profiles, so that the user can enable or disable sets of rules at any time (even with rules).
- As a developer, I want to integrate rule types for the lock screen, so that the user can create rules using the lock functionality.
- As a developer, I want rules that access the motion sensors/accelerometers, so that the user can create rules based on the motion of the phone.
- As a developer, I want rules that access the gyroscope, so that the user can create rules based on the device's orientation.
- As a developer, I want to access when the phone is shutdown or restarted, so that the user can create rules based on these actions/events.
- As a developer, I want to know when the first boots up, so the user can respond using a set of rules.
- As a developer, I want language based rules, so that the application can be used internationally and for translation.
- As a developer, I want debug based rules, so that the user can create sets of rules for debugging their applications (including this one).
- As a developer, I want to be able to interact with a printer, so that the user can create rules that interact with the printer.
- As a developer, I want to allow the user to directly input JavaScript, so that a technical user can use the rules engine to its fullest.
- As a developer, I want to access the weather status, so that the user can use rules based on this.

- As a developer, I want to access emergency messages, so that the user is quickly informed of hazards.
- As a developer, I want to access the news, so that the user can create rules based on this.
- As a developer, I want to create rules based on traffic, so that the user can create rules for this.
- As a developer, I want to access Stocks, so that the user can integrate with any stock service.
- As a developer, I want to access Banking, so that the user can manage their banking using rules.
- As a developer, I want to store VPN settings, so that the user can manage VPN connections.
- As a developer, I want to react to Data roaming, so the user can include this state in rules.
- As a developer, I want to access Voicemail, so that the user can manage voicemail using rules.
- As a developer, I want to know when something is downloaded and uploaded, so the user can create rules for this.
- As a developer, I want to be able to access movie times, so that the user can create rules for this.

## API Based User Stories

- As a developer, I want to access Dropbox, so that the user can manage their files stored on the cloud.
- As a developer, I want to access Evernote, so that the user can manage their notes using this rules engine.
- As a developer, I want to access Flickr, so that the user can manage photos using this application.
- As a developer, I want to access Craigslist, so that the user can interact with this service using our application.
- As a developer, I want to access Blogger, so that the user can interact with blogs using rules.
- As a developer, I want to access RSS feeds, so that users can integrate their rules with the feeds.
- As a developer, I want to access Gmail, so that the user can use Google email as an email source for rules.
- As a developer, I want to access eBay, so that the user can interact with this service using our rules.
- As a developer, I want to access Instagram, so that the user can manage their sepia stained photos.
- As a developer, I want to access Twitter, so that the user can tweet using our rule system.
- As a developer, I want to access Steam, so that users can interact with their gaming communities using this application.
- As a developer, I want to access Foursquare, so that users can interact with this service and sync the user's location using this service.
- As a developer, I want to access Google+, so that users can interact with social media.
- As a developer, I want to access Pinterest, so users can manage their boards using our app.
- As a developer, I want to access Path, so users can stay connected with family & close friends.
- As a developer, I want to access last.fm, so users can manage music using our application.
- As a developer, I want to access LinkedIn, so users can manage their professional identities.
- As a developer, I want to access Google and Yahoo Finance, so users can manage their financial interests using our app.
- As a developer, I want to access YouTube, so they can manage their video accounts.
- As a developer, I want to access WordPress, so they can manage their blogs.
- As a developer, I want to access Tumblr, so that users can manage their Tumblr content.
- As a developer, I want to access StumbleUpon, so that users can stumble using our application.
- As a developer, I want to access Yahoo, so that users can manage their Yahoo accounts.

- As a developer, I want to access Windows live, so that users can integrate with their Window's live features.
- As a developer, I want to access ESPN, so that users can manage their sports.
- As a developer, I want to access GroupMe, so that users can manage group text messaging using our app.
- As a developer, I want to access Samsung Kies, so that users can update their hardware using this application.
- As a developer, I want to access Amazon, so that users access this service.
- As a developer, I want to access Newegg, so that users can interact with this online store.
- As a developer, I want to access Google play store, so that the user can create rules involving the Android marketplace.
- As a developer, I want to access Google voice, so that users can integrate with voice accounts.
- As a developer, I want to access AIM, so that users can interact with this messaging service.
- As a developer, I want to access Google Talk, so that users can interact with this service.
- As a developer, I want to access Skype, so that users can text/audio/video with other users.
- As a developer, I want to access Netflix, so that users can build rules for this type.
- As a developer, I want to access Hulu Plus, so that users can build rules for this type.
- As a developer, I want to access Kindles, so that users can interact with these accounts and devices.
- As a developer, I want to access Wikipedia, so that users can be well informed.
- As a developer, I want to access App.net, so that other developers can create rules based on this wonderful service.
- As a developer, I want to access Bit.ly, so that users can save, share, & discover links.
- As a developer, I want to access Buffer, so that users can have a smart way to share on social media.
- As a developer, I want to access Delicious.com, so that users can access cooking and recipe features.
- As a developer, I want to access Diigo, so that users can collect and organize using our app.
- As a developer, I want users to be able to ordering food (Example: Pizza Hut), so that users can automate this process.
- As a developer, I want to access Readability, so users can read comfortably if they prefer to.
- As a developer, I want to access Storify, so that users can create custom scenarios using social media.
- As a developer, I want to access Spotify, so that users can manage their media libraries.
- As a developer, I want to access Rdio, so that users can listen to music whenever they want.

- As a developer, I want to access Svpply, so that users can shop a wide variety of products.
- As a developer, I want to access Vimeo, so that users can have multiple options for viewing media.
- As a developer, I want to access ZooTool, so that users can bookmark and share content with ease.
- As a developer, I want to access Myspace, so that users can access music and video based social media.
- As a developer, I want to access Friendster, so that users can access this social media source.
- As a developer, I want to access Classmates.com, so that users can communicate with old classmates using our app.
- As a developer, I want to access Deviant art, so that users can manage their art portfolios using our application.
- As a developer, I want to access CrunchyRoll, so that users can be notified of new release dates.
- As a developer, I want to access Blogster, so that users can manage their blogs.
- As a developer, I want to access IGN, so that users can use this game database functionality.
- As a developer, I want to access GameFly, so that users can order games using our application.
- As a developer, I want to access IMDB, so that users can use this movie database functionality.
- As a developer, I want to access Live journal, so that users can manage their personal publishing.
- As a developer, I want to access Xanga, so that users can manage their own blogs.
- As a developer, I want to access Yelp, so that users can access local reviews and information.
- As a developer, I want to access Bebo, so that users can use this social media outlet.
- As a developer, I want to access Badoo, so that users can communicate with local people.
- As a developer, I want to access Stackoverflow, so that users can communicate on these developer forums.
- As a developer, I want to access XDA developers, so that developers can communicate on this medium.
- As a developer, I want to access the MSDN, so that developers can use this means of communication.
- As a developer, I want to access Digg, so that users can stay up to date on current events.
- As a developer, I want to access Orkut, so that users can social network and discuss using Google's engine.
- As a developer, I want to access PlayStation Network, so that we can accompany PlayStation users.

- As a developer, I want to access MSN, so that users can use access this functionality.
- As a developer, I want to access Bing, so that users can choose which search engine they wish to use.
- As a developer, I want to access AOL, so that we can address all user types.
- As a developer, I want to access Ask.com, so that users can have their questions answered.
- As a developer, I want to access Google's basic search, so that users can use this search engine.
- As a developer, I want to access CNN, so that users can receive updates about news.
- As a developer, I want to access FOX, so that users can receive updates about entertainment.
- As a developer, I want to access ABC, so that users can receive updates about news.
- As a developer, I want to access BBC, so that users can integrate with an international news source.
- As a developer, I want to access Wolfram Alpha, so that users can do complex math using our application.
- As a developer, I want to access Whitepages, so that users can quickly search this database.
- As a developer, I want to access Yellowpages, so that users can quickly search this database.
- As a developer, I want to access RememberTheMilk.com, so that users can update and receive reminder notifications.
- As a developer, I want to access iTunes, so that users can integrate their phone with this music application.
- As a developer, I want to access Apple accounts, so that users can integrate their Android device with Apple hardware.
- As a developer, I want to access PayPal, so that users can manage purchases using this application.
- As a developer, I want to access GoDaddy, so that users can manage website domains using our application.
- As a developer, I want to access 4shared, so that users can manage files on this service.
- As a developer, I want to access The Weather Channel, so that users can have more accurate weather on demand.
- As a developer, I want to access SourceForge, so that developers can manage their projects from their phone.
- As a developer, I want to access GitHub, so that developers can manage their code repositories.
- As a developer, I want to access Bitbucket, so that developers can manage their code repositories.
- As a developer, I want to access Walmart, so that users can be aware of updates and make purchases.
- As a developer, I want to access Target, so that users can utilize this department store's API.

- As a developer, I want to access Fedex, so that users can monitor shipping status.
- As a developer, I want to access UPS, so that users can monitor shipping status.
- As a developer, I want to access Dictionary.com, so that users can define words.
- As a developer, I want to access Urban dictionary, so that users can understand local colloquialisms.
- As a developer, I want to access Thesaurus.com, so that users can use this service.
- As a developer, I want to access Google Documents, so that users can manage their documents from our app.
- As a developer, I want to access Google Drive, so that users can manage their files on this service.
- As a developer, I want to access GroupOn, so that users can get deals from local businesses.
- As a developer, I want to access Living Social, so that users can receive deals on local services.
- As a developer, I want to access Urban Spoon, so that users can have information on cooking at hand.
- As a developer, I want to access Rhapsody, so that users can use this music player.
- As a developer, I want to access Imugr, so that users can manage their image files.
- As a developer, I want to access Heroku, so that users can manage their projects using this service.
- As a developer, I want to access Speedtest.net, so that users can check their current connection status with detail.
- As a developer, I want to access Photobucket, so that users can manage their image and video files.
- As a developer, I want to access Pandora, so that users can use this music service.
- As a developer, I want to access CNet, so that users can view technical reviews on products.
- As a developer, I want to access Battle.net, so that these users can manage their accounts.
- As a developer, I want to access Grooveshark, so that users can use this music service.
- As a developer, I want to access Monster, so that users can find the job that's right for them.
- As a developer, I want to access the NFL, so that users can receive updates from this feed.
- As a developer, I want to access Expedia, so that users can manage travel details.
- As a developer, I want to access Hotels.com, so that users can manage travel details.
- As a developer, I want to access Kayak, so that users can manage travel details.
- As a developer, I want to access Travelpedia, so that users can manage travel details.
- As a developer, I want to access Engadget, so that users can be up to date on technology news.
- As a developer, I want to access Slashdot, so that users can be up to date on technology news.
- As a developer, I want to access Fandango, so that users can manage movie tickets.

- As a developer, I want to access Ticketmaster, so that users can book and manage tickets.
- As a developer, I want to access StubHub, so that users can book and manage tickets.
- As a developer, I want to access Match.com, so that users can communicate with local individuals.
- As a developer, I want to access Rotten Tomatoes, so that users can use this movie review service.
- As a developer, I want to access Twilio, so that users can manage VoIP, Voice, and Text applications over the web.
- As a developer, I want to access Metro Lyrics, so that users can search for song lyrics.
- As a developer, I want to access Shazam, so that users can recognize songs using this service.
- As a developer, I want to access the NBA, so that users can receive updates from this service.
- As a developer, I want to access Dailymotion, so that users can upload, share, and embed their own videos.
- As a developer, I want to access Chacha, so that they can have their questions answered.
- As a developer, I want to access Best Buy, so that users can access this API service.
- As a developer, I want to access Quora, so that users can connect and share content using this service.
- As a developer, I want to access Voxeo, so that users can manage their cloud computing profiles.
- As a developer, I want to access "Let me Google that for you," so that users can share knowledge with peers.
- As a developer, I want to access About.com, so that users can solve their needs.
- As a developer, I want to access eHow, so they can use this service for information.
- As a developer, I want to access Mustang TRAK, so that users can manage their SMU job profiles.
- As a developer, I want to access to Access.smu.edu, so that users can manage their student accounts using our application.
- As a developer, I want to access Blackboard, so that users can manage their classes using this rules engine.

# Glossary of Terms and Acronyms

## .apk file [1]

Android application package file. Each Android application is compiled and packaged in a single file that includes all of the application's code (.dex files), resources, assets, and manifest file. The application package file can have any name but *must* use the .apk extension.

## Action [1]

A description of something that an Intent sender wants done. An action is a string value assigned to an Intent. Action strings can be defined by Android or by a third-party developer. For example, android.intent.action.VIEW for a Web URL, or com.example.rumbler.SHAKE\_PHONE for a custom application to vibrate the phone.

## Activity [1]

A single screen in an application, with supporting Java code, derived from the [Activity](#) class. Most commonly, an activity is visibly represented by a full screen window that can receive and handle UI events and perform complex tasks, because of the Window it uses to render its window. Though an Activity is typically full screen, it can also be floating or transparent.

## Android [2]

Google's open-source mobile operating system. It's used primarily in smartphones but also can be found on tablets, Mobile Internet Devices (MIDs) or even in kitchen appliances and automobile navigation.

## API [3]

API, an abbreviation of *application program interface*, is a set of [routines](#), [protocols](#), and tools for building [software applications](#). A good API makes it easier to develop a [program](#) by providing all the building blocks. A [programmer](#) then puts the blocks together.

## **Application [1]**

From a component perspective, an Android application consists of one or more activities, services, listeners, and intent receivers. From a source file perspective, an Android application consists of code, resources, assets, and a single manifest. During compilation, these files are packaged in a single file called an application package file (.apk).

## **Bluetooth [2]**

A short-range radio build into smartphones that lets you connect headsets, speakerphones or even computers to your smartphone.

## **Broadcast Receiver [1]**

An application class that listens for Intents that are broadcast, rather than being sent to a single target application/activity. The system delivers a broadcast Intent to all interested broadcast receivers, which handle the Intent sequentially.

## **Dialog [1]**

A floating window that acts as a lightweight form. A dialog can have button controls only and is intended to perform a simple action (such as button choice) and perhaps return a value. A dialog is not intended to persist in the history stack, contain complex layout, or perform complex actions. Android provides a default simple dialog for you with optional buttons, though you can define your own dialog layout. The base class for dialogs is [Dialog](#).

## **Fragment [2]**

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. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

## **Google™ [2]**

Our benevolent overlord, and owner of Android.

*Note:* This is the ACTUAL definition, provided by Google.

## **Intent [1]**

A message object that you can use to launch or communicate with other applications/activities asynchronously. An Intent object is an instance of [Intent](#). It includes several criteria fields that you can supply, to determine what application/activity receives the Intent and what the receiver does when handling the Intent. Available criteria include the desired action, a category, a data string, the MIME type of the data, a handling class, and others. An application sends an Intent to the Android system, rather than sending it directly to another application/activity. The application can send the Intent to a single target application or it can send it as a broadcast, which can in turn be handled by multiple applications sequentially. The Android system is responsible for resolving the best-available receiver for each Intent, based on the criteria supplied in the Intent and the Intent Filters defined by other applications. For more information, see [Intents and Intent Filters](#).

## **Intuitive [4]**

An adjective meaning: perceived by, resulting from, or involving [intuition](#).

## **Layout Resource [1]**

An XML file that describes the layout of an Activity screen.

## **Manifest File [1]**

An XML file that each application must define, to describe the application's package name, version, components (activities, intent filters, services), imported libraries, and describes the various activities, and so on. See [The AndroidManifest.xml File](#) for complete information.

## **NFC [2]**

Near-field communication. Short-range communication between your phone and something else -- another phone, a cash register, etc. Used by some credit cards as a method of quick payment.

## **Open Source [2]**

Software which is liberally licensed to grant the right of users to study, change, and improve its design through the availability of its source code.

## **SDK [2]**

Stands for Software Development Kit. Generally, a set of tools used to create software for a certain platform following guidelines provided in the kit. For Android, the SDK provides tools to create applications that run on Android devices.

## **Service [1]**

An object of class [Service](#) that runs in the background (without any UI presence) to perform various persistent actions, such as playing music or monitoring network activity.

## **View [1]**

An object that draws to a rectangular area on the screen and handles click, keystroke, and other interaction events. A View is a base class for most layout components of an Activity or Dialog screen (text boxes, windows, and so on). It receives calls from its parent object (see viewgroup, below) to draw itself, and informs its parent object about where and how big it would like to be (which may or may not be respected by the parent). For more information, see [View](#).

## **Widget [1]**

One of a set of fully implemented View subclasses that render form elements and other UI components, such as a text box or popup menu. Because a widget is fully implemented, it handles measuring and drawing itself and responding to screen events. Widgets are all in the [android.widget](#) package.

## **Glossary References**

1. <http://developer.android.com/guide/appendix/glossary.html>
2. <http://www.androidcentral.com/dictionary>
3. <http://www.webopedia.com/TERM/A/API.html>
4. <http://dictionary.reference.com/browse/intuitive>

# Acceptance Test Plan

This is our acceptance test plan that we created by looking at what the user can see while using the application to fully test the both the front and back end of the program. We decided to order the tests by the flow of the application, not by the user story list to give a more user-friendly test plan that can be completed by simply by following the tests step-by-step. For the verification of the user stories, we used the iteration story list for the numbering as this is a more concrete list of what we had accomplished during the production of this application instead of giving the broad overview of all of what we could have done. So like I said before, follow these tests in order and you should have no difficulties while understanding what any of these tests are asking (even though the tests could be run in any order).

The following user stories were not included in the ATP. Reasons for their exclusion are as follows:

## Removed from the Application

1, 8, 24, 29, 30, 57, 58, 106

## Never in the Application or Not Testable

66, 67, 71, 72, 73, 101, 109, 110, 111, 112, 113, 115, 116, 117, 118, 119, 120, 121, 122, 123, 125, 126, 127, 128, and 129

## **Test 1: The app starts up correctly.**

Verifies user story: 2,15,16,45,46,47,48,75,79

Check to see phone is on. To do so, turn on the screen by pressing one of the side keys.

Open up the phone by sliding the lock to the right (put in password if needed, if not known, ask the phone's owner). Then, open the apps menu by pressing the button on the bottom of the screen. Find and open Cause & Effect.

### Process

Press the logo from the apps menu, see if it starts up.

### Results

Able to start up (yes/no): \_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---



Lock Screen

Unlocking Screen

Home Screen

All Apps Menu

App Home Screen

## **Test 2: Able to view all rules.**

Verifies user story: 4,17,18,19,20,21,22,65,77,104,105

Test by asking the user to tap My Rules from the home page.

### Process

Press the My Rules button from home page button. Verify that all current rules show up (if there are no rules, the New Rule option on the Main Menu can be used to populate this screen, see Test 54).

### Results

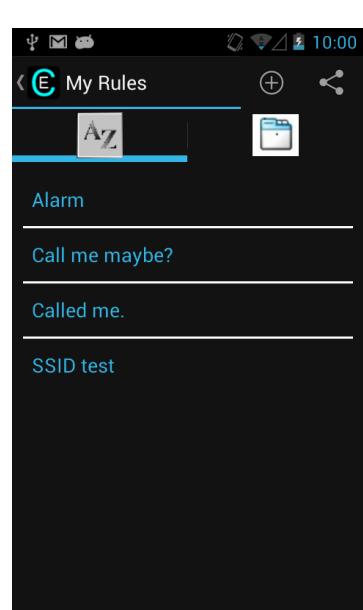
Able to view rules (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]



My Rules screen

### Test 3: Able to view details of a rule.

Verifies user story: 5,28,80,81,85,102

Test by viewing the details of a rule.

#### Process

After following Test 2, tap on one of the rules (If there are no rules, go to the New Rule page from the main menu). Verify that the name is correct at the top, and that the rule's causes and effects appear. If there are multiple causes, an AND or OR should be between them.

#### Results

Able to bring up details of rule (yes/no):\_\_\_\_\_

Rule's name appears and is correct (yes/no):\_\_\_\_\_

"+" appears at bottom of both causes and effects (yes/no):\_\_\_\_\_

(Possible) AND or OR shows up between multiple causes (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

Rule View	
Alarm	On
Causes	Call me maybe?
Time 12:30	Time 12:30
+	OR
	Phone Call Nathan R Huntoon
Effects	
Notification Wake up! Go to school.'	Vibrate Tone:0 200 200 200 -1
+	Notification Call from Nathan'Hangup Quick!'
	+

[Edit Rule page](#)

[Edit Rule page with OR](#)

## Test 4: Arrive map location (GPS off).

Verifies user story: 6,49,50,56,78,93,94

Test by having the user try to add the Cause: Arrival at a Destination, and view the first screen.

### Process

Before doing this test, make sure the GPS is off (in your phone settings). At the top of your screen, there is a status bar. Touch and drag this bar down to see the entire status screen. At the top of the status screen, there is a settings icon (which is also located in the Apps Menu). Tap the settings icon to view your settings, and then scroll down and tap "Location Access." Turn off the GPS (location services) but leave the Wi-Fi and mobile networks on. After getting past Test 3, tap the "+" below the causes list to create a new cause. Tap "Arriving at a Location". The Google maps should show up with the last known location. If you can't get your location, move to a different location with better signal strength. If this still doesn't work, restart your Wi-Fi and mobile networks, and try again.

### Results

"Arriving at a Location" is a cause listed (yes/no): \_\_\_\_\_

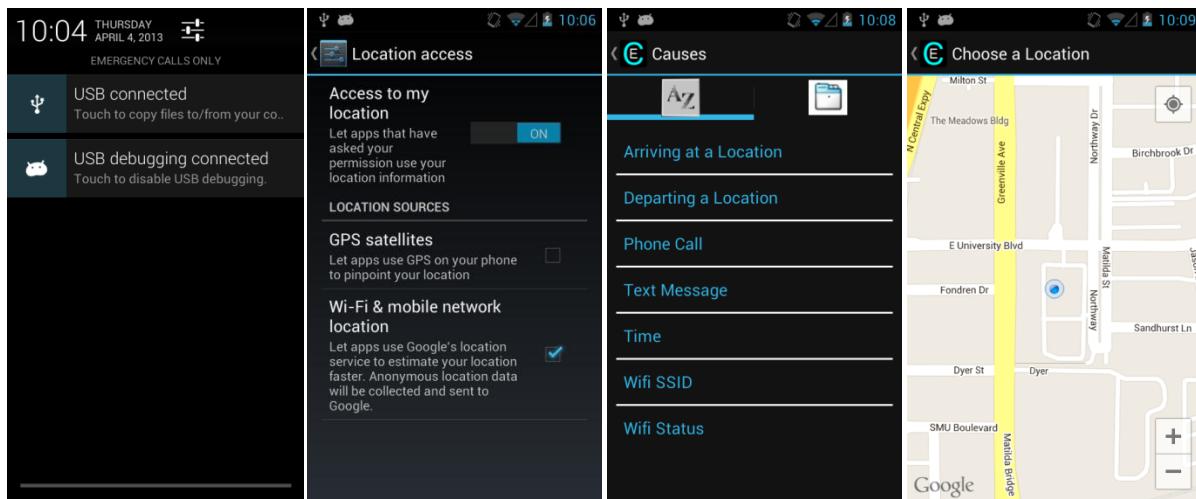
Last location known shows up on the screen (yes/no): \_\_\_\_\_

Comments:

---

---

[Passed/Failed]



Pull-down screen

Location menu

Causes menu

Google maps

## Test 5: Pan the Google map.

Verifies user story: 93

Test by having the user try to pan the map pulled up by Test 4.

### Process

As with most touch screens, attempt tap and drag the screen to pan the view of the Google map that is pulled up from Test 4.

### Results

Able to pan map (yes/no): \_\_\_\_\_

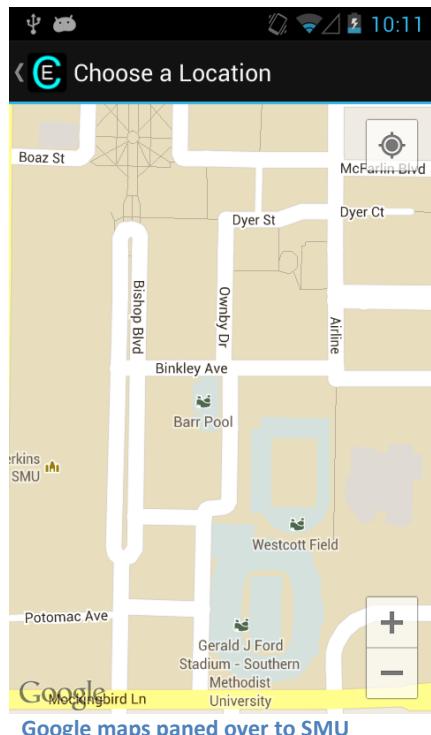
Comments:

---

---

[Passed/Failed]

---



Google maps panned over to SMU

## **Test 6: Zoom in on the Google map.**

Verifies user story: 93

Test by having the user try to zoom in all the way on the map pulled up by Test 4.

### Process

As with most touch screens, put your two fingers together (preferably the thumb and index finger) on the screen. While pressing on the screen, slowly separate the two fingers to zoom in on the view of the Google map that is pulled up from Test 4 until you can no longer. Verify that app does not crash while doing so.

### Results

Able to zoom in on map (yes/no): \_\_\_\_\_

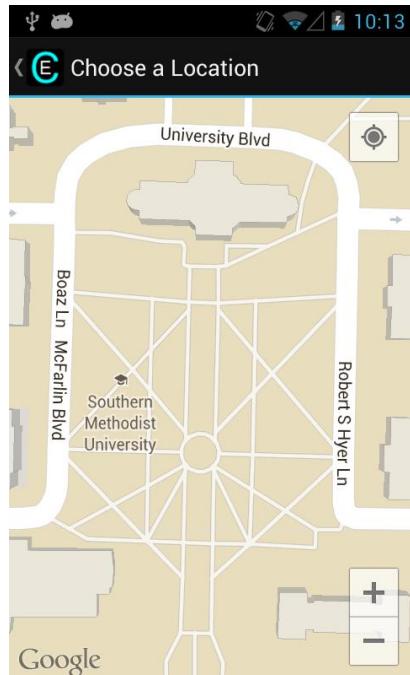
Comments:

---

---

[Passed/Failed]

---



**Zoomed-in view of SMU**

## **Test 7: Zoom out on the Google map.**

Verifies user story: 93

Test by having the user try to zoom out all the way on the map (pulled up by Test 4).

### Process

As with most touch screens, put your two fingers apart (preferably the thumb and index finger) on the screen. While pressing on the screen, slowly close the two fingers to zoom out on the view of the Google map that is pulled up from Test 4 until you can no longer. Verify that app does not crash while doing so.

### Results

Able to zoom out on map (yes/no):\_\_\_\_\_

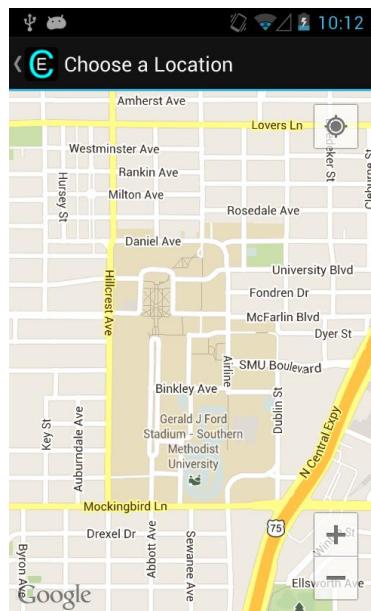
Comments:

---

---

[Passed/Failed]

---



[Zoomed-out Google maps](#)

## **Test 8: My location button (GPS off).**

### Verifies user story: 93

Test by having the user try to use the My Location button on Google maps (top right of screen).

### Process

Before doing this test, make sure the GPS is off (in your phone settings). Test 4 has an explanation on how to turn off your GPS. Press the My Location button on the top right of the screen. The screen should show a circle around your current location while also panning to your location.

### Results

Circle shows up around a blue dot that includes your current location (yes/no):\_\_\_\_\_

Dot shows up in the middle of the screen (yes/no):\_\_\_\_\_

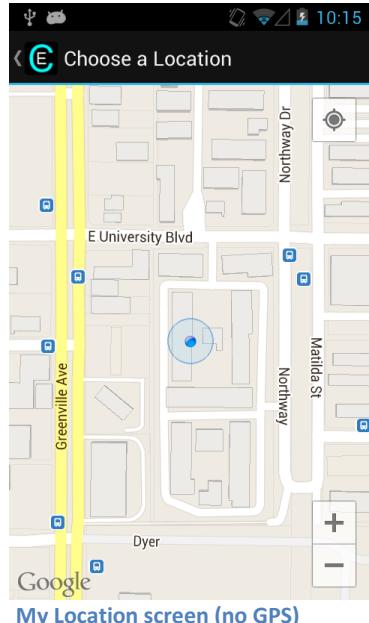
Comments:

---

---

[Passed/Failed]

---



## Test 9: Arrive map location (GPS on).

### Verifies user story: 93,94

Test by having the user try to add the Cause: Arrival at a Destination and view the map screen.

### Process

Before doing this test, make sure the GPS is on (in your phone settings). See Test 4 for details on how to get to the GPS in the Settings. After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Arriving at a Location”. The Google maps should show up with the last known location. This will be a lot more accurate than with the GPS off. If you can't get GPS signal, move to a different location that can receive GPS signal (outside, clear skies).

### Results

Last location known shows up on the screen (yes/no):\_\_\_\_\_

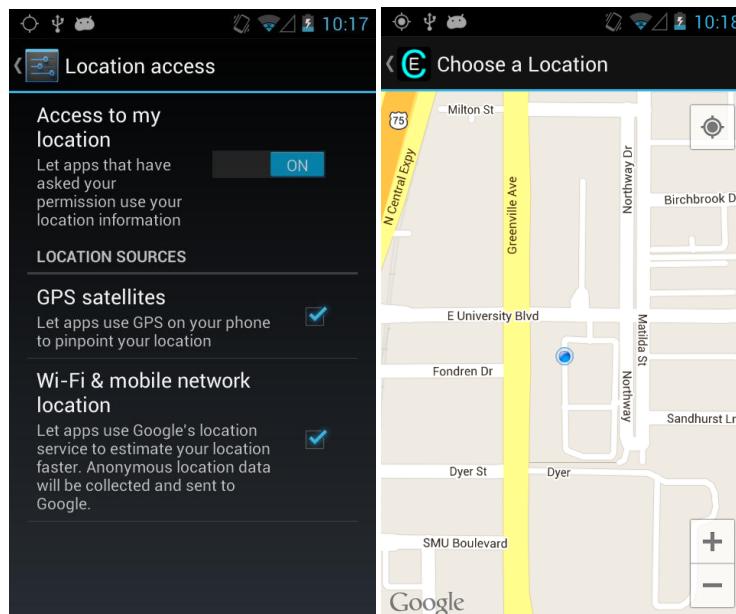
Comments:

---

---

[Passed/Failed]

---



Turning GPS on

Shown Location (GPS on)

## Test 10: My location button (GPS on).

Verifies user story: 93

Test by having the user try to use the My Location button on Google maps (top right of screen).

### Process

Before doing this test, make sure the GPS is on (in your phone settings). Press the My Location button in the top right of the screen. The screen should show a blue dot (or arrow) indicating your current location while also panning to your location (if you are zoomed in a small circle will appear).

### Results

Blue dot shows your current location (yes/no):\_\_\_\_\_

Dot shows up in the middle of the screen (yes/no):\_\_\_\_\_

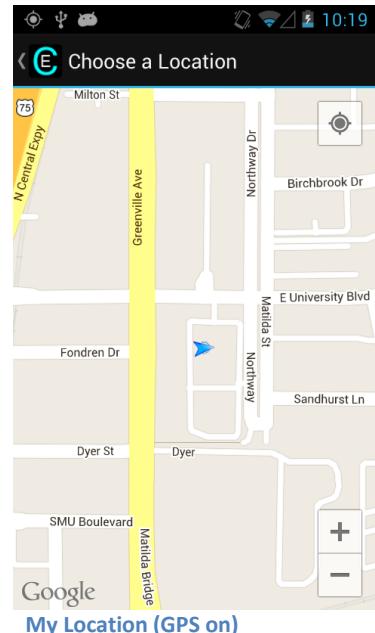
Comments:

---

---

[Passed/Failed]

---



### **Test 11: Tap to add pin to Google custom map.**

Verifies user story: 93

Test by having the user tap the screen on the Google map to add a pin.

#### Process

While inside of Arriving at a Location, tap on the screen to add a pin. A dialog should appear that asks for a name for the location. Leave the name blank and tap cancel. Some phones may not have a cancel button, instead use the back button. The screen should clear, and no pins should be visible anymore.

#### Results

Pin appears on screen after tapping (yes/no):\_\_\_\_\_

Pin disappears on screen after tapping Cancel (yes/no):\_\_\_\_\_

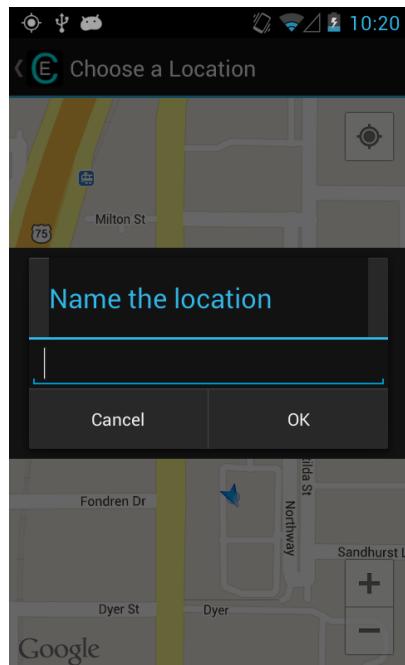
Comments:

---

---

[Passed/Failed]

---



[Adding a pin](#)

## Test 12: Tap to add multiple pins to Google custom map.

Verifies user story: 93

Test by having the user tap the screen on the Google map to add a pin.

### Process

While inside of Arriving at Location, tap on the screen to add a pin. A dialog should appear that asks for a name of the location. Leave the name blank and tap OK. The pin should still be visible. A blue overlay will appear to show the location that the rule is in effect. Repeat this process a couple times to keep adding more pins. Multiple pins should be visible. If you were to approve of this choice, the most recent pin would be used for the cause. Now add a new pin and instead of tapping OK, tap cancel. Some phones may not have a cancel button, instead use the back button. All pins should now be gone.

### Results

Pin appears on screen after tapping (yes/no):\_\_\_\_\_

Multiple pins appear after creating multiple (yes/no):\_\_\_\_\_

All pins disappear on screen after tapping Cancel (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---



Multiple pins shown on map

### **Test 13: Tap to add pin to Google custom map (blank name).**

Verifies user story: 31,32,33,34,35,36,37,38,39,40,41,42,43,44,52,53,54,55,  
68,69,82,93,99,108

Test by having the user tap the screen on the Google map to add a pin.

**Note:** This is the first time the back-end is used, so all relevant backend user stories are listed only on this test.

#### Process

While inside of Arriving at Location, tap on the screen to add a pin. A dialog should appear that asks for a name for the location. Leave the name blank and tap OK. Tap and hold inside of the square to accept the location. Verify that this rule now appears in the cause list. Verify that the rule works by arriving at this location specified.

#### Results

Pin appears on screen after tapping (yes/no):\_\_\_\_\_

Able to add pin with blank name (yes/no):\_\_\_\_\_

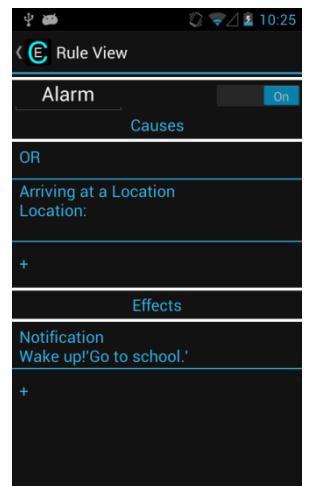
Holding accepts the rule and stops the editing of the cause (yes/no):\_\_\_\_\_

New cause appears at the bottom of the cause list (yes/no):\_\_\_\_\_

Cause is triggered when arriving at the location (yes/no):\_\_\_\_\_

Comments:  
\_\_\_\_\_  
\_\_\_\_\_

#### [Passed/Failed]



**Test 14: Tap to add pin to Google custom map (actual name).**

Verifies user story: 93

Test by having the user tap the screen on the Google map to add a pin.

Process

While inside of Arriving at Location, tap on the screen to add a pin. A dialog should appear that asks for a name of the location. Fill in the name and tap OK. The screen should show the pin. Tap and hold inside of the square to accept the location. Verify that this rule now appears in the cause list with the name visible after “Location:” in the cause. Verify that the cause works by arriving at the specified location.

Results

Pin appears on screen after tapping (yes/no):\_\_\_\_\_

Able to add pin with blank name (yes/no):\_\_\_\_\_

Holding accepts the rule and stops the editing of the cause (yes/no):\_\_\_\_\_

New cause appears at the bottom of the cause list with name (yes/no):\_\_\_\_\_

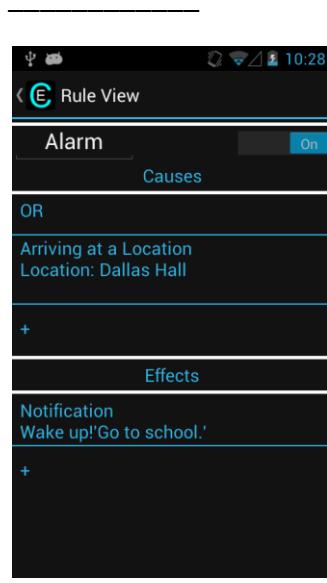
New cause triggers the rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]



## **Test 15: Tap to add pin to Google custom map (blank name).**

### Verifies user story: 93

Test by having the user try to add the Cause: Departing a Location, and follow the steps to create the cause.

### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Departing a Location”. The Google maps should show up with the last known location. Tap on the screen to add a pin. Leave the name blank and tap OK. Tap and hold inside the square to finalize the location. The new cause should show up at the bottom of the cause list. Finally, verify that leaving the designated location triggers the rule effects.

### Results

“Departing at a Location” is a cause listed (yes/no):\_\_\_\_\_

Last location known shows up on the screen (yes/no):\_\_\_\_\_

Able to add blank pin (yes/no):\_\_\_\_\_

Holding accepts the rule and stops the editing of the cause (yes/no):\_\_\_\_\_

New cause appears at the bottom of the cause list (yes/no):\_\_\_\_\_

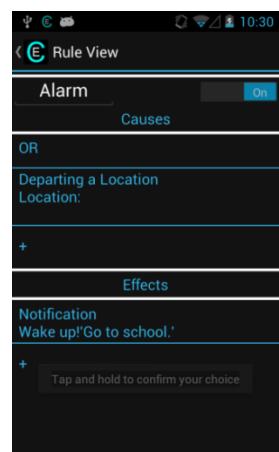
New cause triggers the rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]



Rule with blank Departing cause

## **Test 16: Tap to add pin to Google custom map (actual name).**

### Verifies user story: 93

Test by having the user try to add the Cause: Departing a Location, and follow the steps to create the cause.

### Process

While inside of departing a location, tap on the screen to add a pin. Fill in the name and tap OK. Tap and hold inside the square to finalize the location. The new cause should show up at the bottom of the cause list. Verify that this rule now appears in the cause list with the name visible after “Location: ” in the cause. Finally, verify that leaving the designated location triggers the rule effects.

### Results

Able to add actual name pin (yes/no):\_\_\_\_\_

Holding accepts the rule and stops the editing of the cause (yes/no):\_\_\_\_\_

New cause appears at the bottom of the cause list with name (yes/no):\_\_\_\_\_

New cause triggers the rule effects (yes/no):\_\_\_\_\_

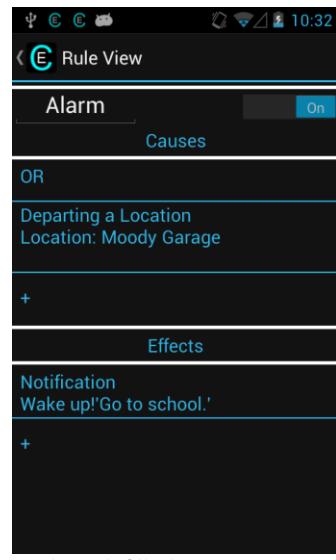
Comments:

---

---

[Passed/Failed]

---



Rule with filled Departing cause

## Test 17: Adding Phone Call cause.

Verifies user story: 51,59

Test by having the user try to add the cause of phone call.

### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Phone Call”. Your contact list should be displayed. Tap on one of the contacts (if you have no contacts, add one and restart the test). The new cause should now be displayed at the bottom of the list with the correct name. Verify that the rule works by having that contact call the phone.

### Results

“Phone Call” is a cause listed (yes/no):\_\_\_\_\_

Contact list displayed (yes/no):\_\_\_\_\_

New cause added to list with correct contact name (yes/no):\_\_\_\_\_

Rule effects triggered by new cause (yes/no):\_\_\_\_\_

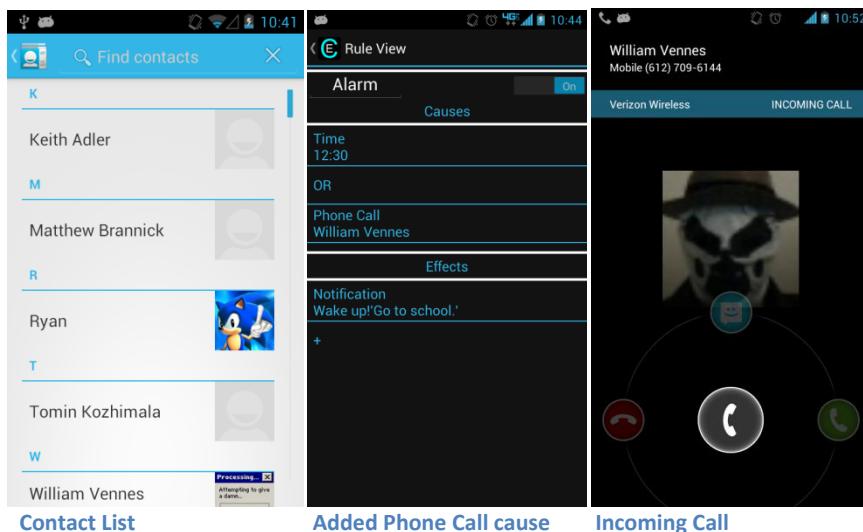
Comments:

---

---

[Passed/Failed]

---



## **Test 18: Adding Text message cause.**

### Verifies user story: 60

Test by having the user try to add the cause of text message.

### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Text Message”. Your contact list should be displayed. Tap on one of the contacts (if you have no contacts, add one and restart test). The new cause should now be displayed at the bottom of the list with the correct name. Verify that the rule works by having that contact text the phone.

### Results

“Text Message” is a cause listed (yes/no):\_\_\_\_\_

Contact list displayed (yes/no):\_\_\_\_\_

New cause added to list with correct contact name (yes/no):\_\_\_\_\_

Rule effects triggered by new cause (yes/no):\_\_\_\_\_

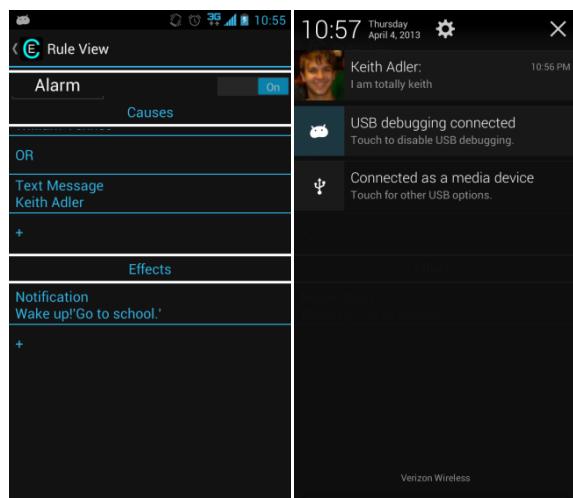
Comments:

---

---

[Passed/Failed]

---



Added Text Message Cause Incoming Text Message

## Test 19: Adding Time cause.

### Verifies user story: 61

Test by having the user try to add the cause of time.

### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Time”. Input a time and tap OK. The new cause should now be displayed at the bottom of the list with the correct time. Verify that the rule works by waiting for that time.

### Results

“Time” is a cause listed (yes/no):\_\_\_\_\_

Time input dialog displayed (yes/no):\_\_\_\_\_

New cause added to list with correct time (military time) (yes/no):\_\_\_\_\_

Rule effects triggered by new cause (yes/no):\_\_\_\_\_

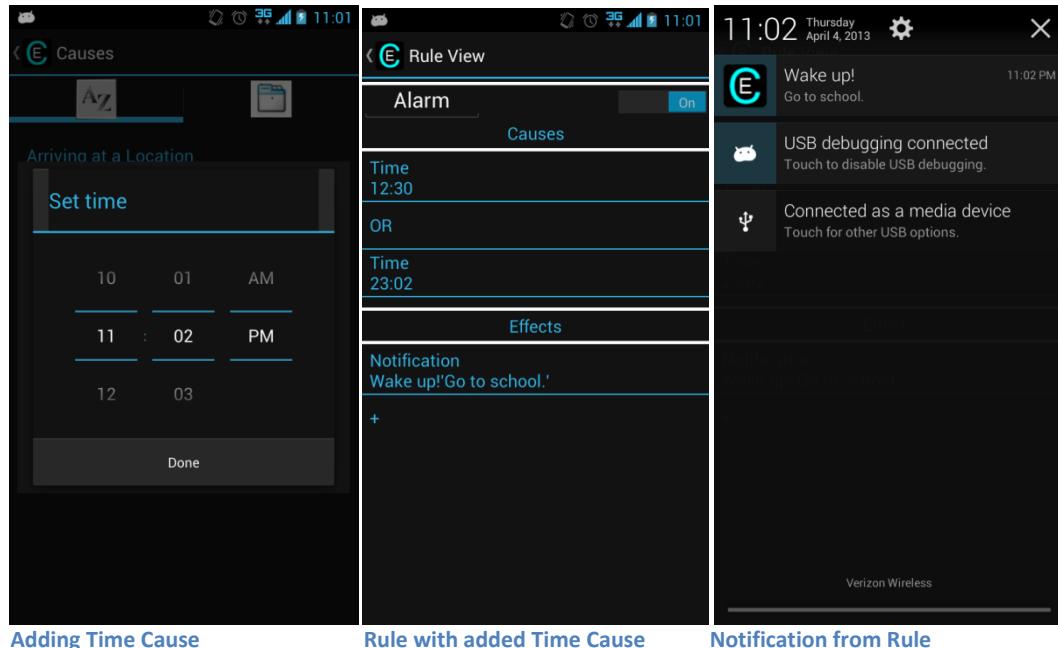
Comments:

---

---

[Passed/Failed]

---



**Test 20: Adding Time cause (cancel).**

Verifies user story: 61

Test by having the user try to add the cause of time.

Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Time”. Input a time and tap cancel. No new cause should be added.

Results

Time input dialog displayed (yes/no):\_\_\_\_\_

No new cause added (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

## Test 21: Adding Wi-Fi SSID cause.

Verifies user story: 96,107

Test by having the user try to add the cause of Wi-Fi SSID.

### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Wi-Fi SSID”. A create dialog should appear asking for text input. Input a name for the SSID (wireless network name) and tap Submit. Verify that the new cause appears at the bottom of the cause list, and that the rule effects are triggered by entering the range of the SSID.

### Results

“Wi-Fi SSID” is a cause listed (yes/no):\_\_\_\_\_

Create dialog appears (yes/no):\_\_\_\_\_

Able to create new cause with correct SSID name (yes/no):\_\_\_\_\_

Rule activates correctly (yes/no):\_\_\_\_\_

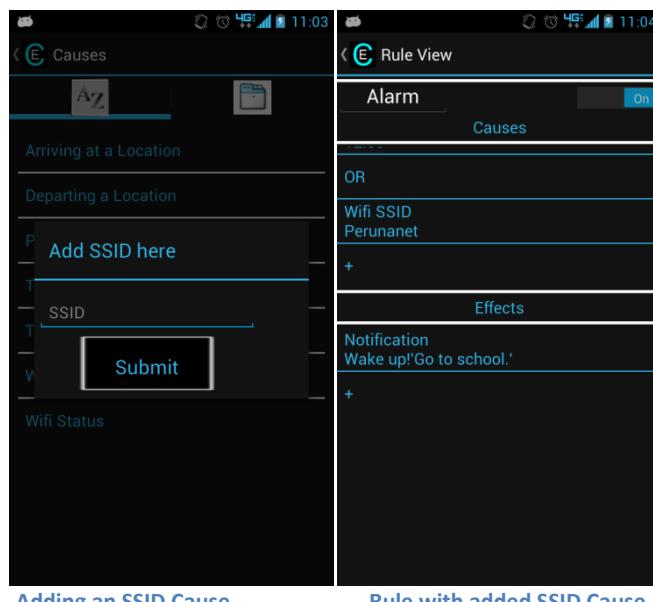
Comments:

---

---

[Passed/Failed]

---



Adding an SSID Cause

Rule with added SSID Cause

## **Test 22: Adding Wi-Fi SSID cause (blank).**

Verifies user story: 96,107

Test by having the user try to add the cause of Wi-Fi SSID.

### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Wi-Fi SSID”. A create dialog should appear asking for text input. Leave the name blank and tap Submit. Verify that the new cause appears at the bottom of the cause list.

### Results

“Wi-Fi SSID” is a cause listed (yes/no):\_\_\_\_\_

Create dialog appears (yes/no):\_\_\_\_\_

Able to create new cause with blank SSID name (yes/no):\_\_\_\_\_

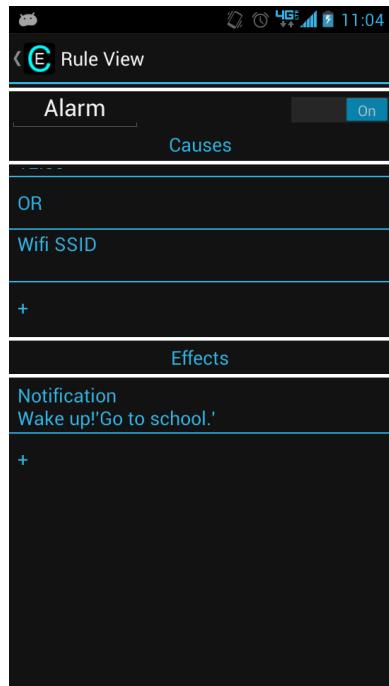
Comments:

---

---

[Passed/Failed]

---



Rule with blank SSID

### Test 23: Adding Wi-Fi Status cause (on).

Verifies user story: 96,107

Test by having the user try to add the cause of Wi-Fi Status.

#### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Wi-Fi Status”. A create dialog should appear with a slider (default in off position). Slide the slider into the one position and tap Submit. Verify that the new cause appears at the bottom of the cause list, and that the rule effects are triggered by turning on the Wi-Fi.

#### Results

“Wi-Fi Status” is a cause listed (yes/no):\_\_\_\_\_

Create dialog appears with slider (yes/no):\_\_\_\_\_

Able to create new cause with correct status (yes/no):\_\_\_\_\_

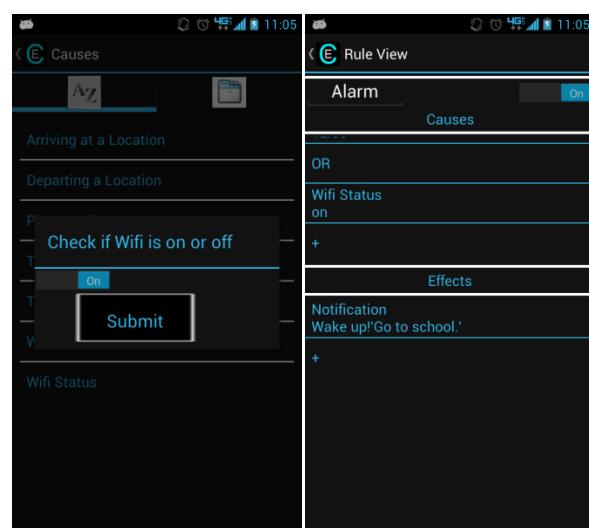
Rule activates correctly (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]



Added Wifi on Cause

Rule with added Wifi cause (on)

## Test 24: Adding Wi-Fi Status cause (off).

Verifies user story: 96,107

Test by having the user try to add the cause of Wi-Fi Status.

### Process

After getting past Test 3, tap the “+” below the causes list to create a new cause. Tap “Wi-Fi Status”. A create dialog should appear with a slider (default in off position). Leave it in the off position and tap Submit. Verify that the new cause appears at the bottom of the cause list, and that the rule effects are triggered by turning off the Wi-Fi.

### Results

Able to create new cause with correct status (yes/no):\_\_\_\_\_

Rule activates correctly (yes/no):\_\_\_\_\_

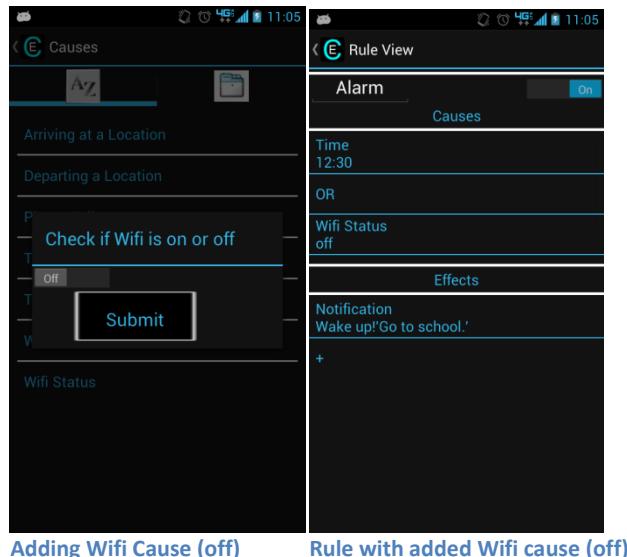
Comments:

---

---

[Passed/Failed]

---



## Test 25: Adding Notification effect (blank).

Verifies user story:7,62

Test by having the user try to add the effect of notification.

### Process

After getting past Test 3, tap the “+” below the effects list to create a new effect. Tap “Notification”. A create dialog should appear waiting for input of text for Title and Subtext. Leave both blank and tap Submit. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

### Results

“Notification” is an effect listed (yes/no):\_\_\_\_\_

Create dialog appears with empty text slots (yes/no):\_\_\_\_\_

Able to create new effect with correct text (blank) (yes/no):\_\_\_\_\_

Notification effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

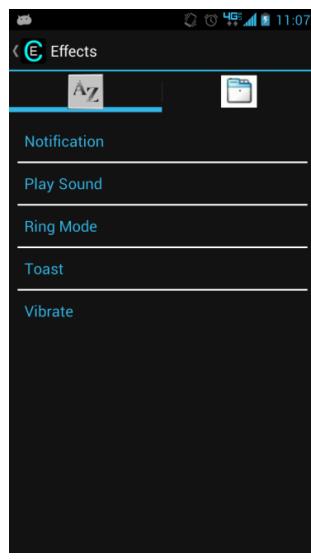
Comments:

---

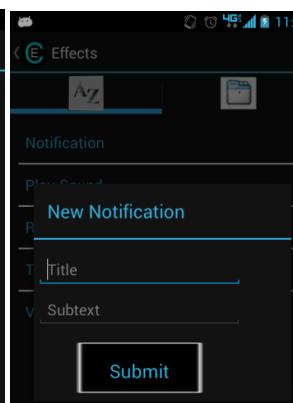
---

[Passed/Failed]

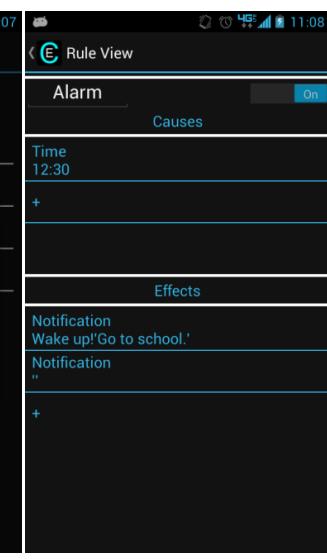
---



Effects Screen



Adding new Notification effect



Rule with new Notification effect

## Test 26: Adding Notification effect (Title only).

### Verifies user story: 62

Test by having the user try to add the effect of notification.

### Process

While inside creating a notification (Test 25), fill in the Title box but leave Subtext blank and tap Submit. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

### Results

Able to create new effect with correct text (yes/no):\_\_\_\_\_

Notification effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

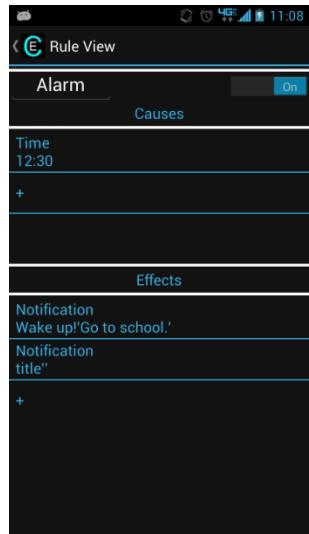
Comments:

---

---

[Passed/Failed]

---



Rule with Notification with only Title

## Test 27: Adding Notification effect (Message only).

### Verifies user story: 62

Test by having the user try to add the effect of notification.

### Process

While inside creating a notification (Test 25), fill in only the Subtext box, and tap Submit. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

### Results

Able to create new effect with correct text (yes/no):\_\_\_\_\_

Notification effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

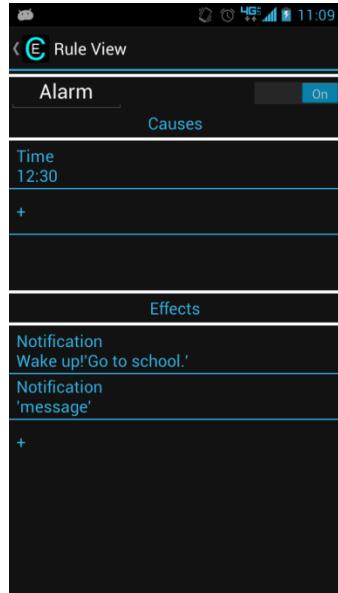
Comments:

---

---

[Passed/Failed]

---



Rule with Notification Effect with Message only

## Test 28: Adding Notification effect (Full notification).

### Verifies user story: 62

Test by having the user try to add the effect of notification.

### Process

While inside creating a notification (Test 25), fill in the Subtext box and the Title box, then tap Submit. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

### Results

Able to create new effect with correct text (yes/no):\_\_\_\_\_

Notification effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

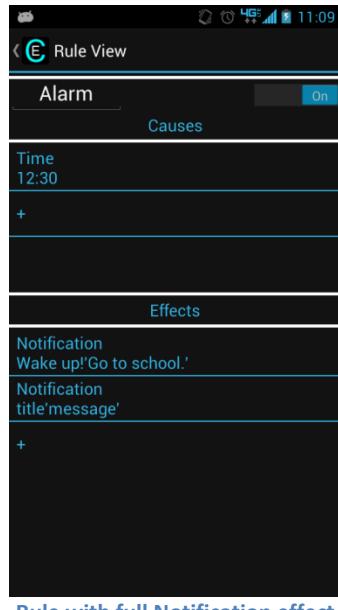
Comments:

---

---

[Passed/Failed]

---



Rule with full Notification effect

## Test 29: Adding Play Sound effect.

### Verifies user story: 91

Test by having the user try to add the effect of sound.

### Process

After getting past Test 3, tap the “+” below the effects list to create a new effect. Tap “Play Sound”. A “Select a File to Play” dialog should appear and tell you to pick a way to choose your sound. Select the option of your choice, and choose a sound to play. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

### Results

“Play Sound” is an effect listed (yes/no):\_\_\_\_\_

“Select a File to Play” dialog appears (yes/no):\_\_\_\_\_

Able to create new effect with correct sound file displayed (yes/no):\_\_\_\_\_

Correct sound is played (yes/no):\_\_\_\_\_

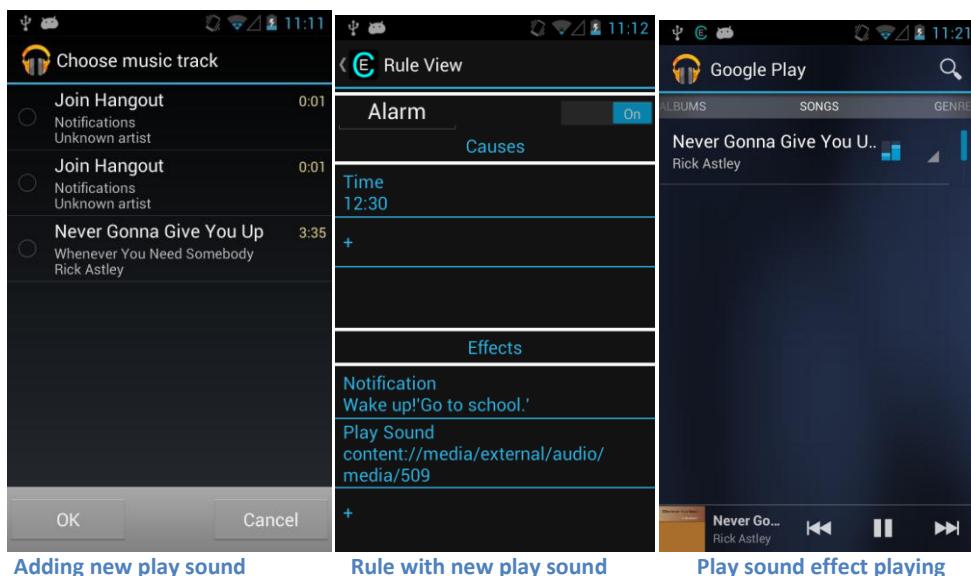
Comments:

---

---

### [Passed/Failed]

---



### Test 30: Adding Ring Mode effect (Normal).

Verifies user story: 92

Test by having the user try to add the effect of ring mode.

#### Process

After getting past Test 3, tap the “+” below the effects list to create a new effect. Tap “Ring Mode”. A “Ring Mode” dialog should appear with 3 options. Tap on “Normal”. Verify that the new effect appears at the bottom of the effect list and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

#### Results

“Ring Mode” is an effect listed (yes/no):\_\_\_\_\_

“Ring Mode” dialog appears with 3 options (yes/no):\_\_\_\_\_

Able to create new effect with “Normal” for ring mode (yes/no):\_\_\_\_\_

Ring mode effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

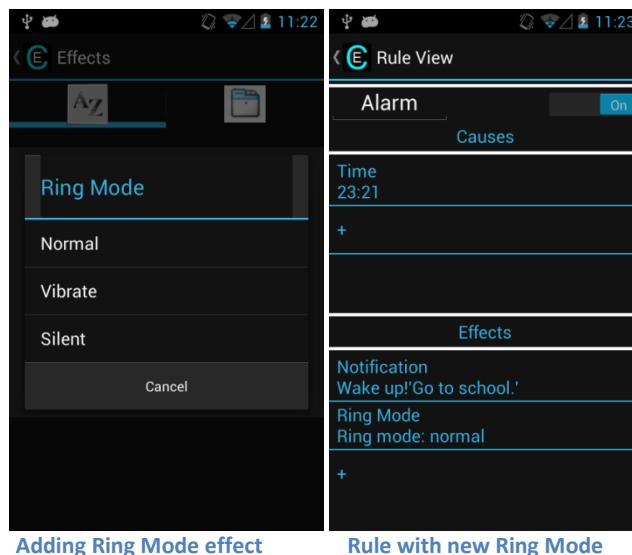
Comments:

---

---

[Passed/Failed]

---



### **Test 31: Adding Ring Mode effect (Vibrate).**

Verifies user story: 92

Test by having the user try to add the effect of ring mode.

#### Process

Starting with dialog box from Test 30, tap on “Vibrate”. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

#### Results

Able to create new effect with “Vibrate” for ring mode (yes/no):\_\_\_\_\_

Ring mode effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

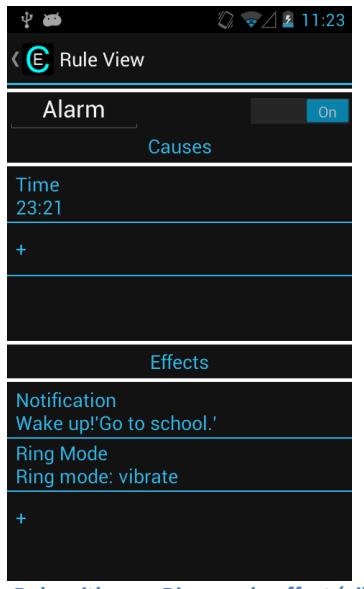
Comments:

---

---

[Passed/Failed]

---



### **Test 32: Adding Ring Mode effect (Silent).**

Verifies user story: 92

Test by having the user try to add the effect of ring mode.

#### Process

Starting with dialog box from Test 30, tap on “Silent”. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

#### Results

Able to create new effect with “Silent” for ring mode (yes/no):\_\_\_\_\_

Ring mode effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

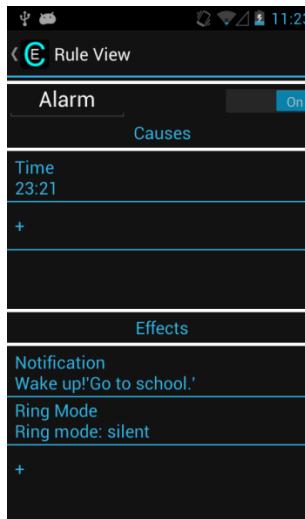
Comments:

---

---

[Passed/Failed]

---



Rule with new Ring Mode effect (silent)

### **Test 33: Adding Toast effect (blank).**

Verifies user story: 63

Test by having the user try to add the effect of Toast.

#### Process

After getting past Test 3, tap the “+” below the effects list to create a new effect. Tap “Toast”. A “New Toast” dialog should appear with an empty Message text input. Leave the message blank and tap Submit. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

#### Results

“Toast” is an effect listed (yes/no):\_\_\_\_\_

Able to create new effect with blank message (yes/no):\_\_\_\_\_

Toast effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

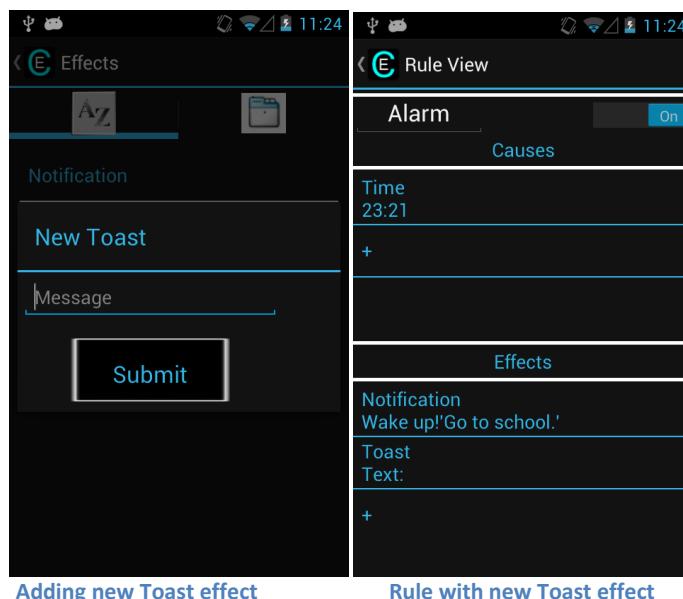
Comments:

---

---

[Passed/Failed]

---



### **Test 34: Adding Toast effect.**

Verifies user story: 63

Test by having the user try to add the effect of Toast.

#### Process

From the new Toast dialog, enter a message and tap Submit. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

#### Results

Able to create new effect with correct message (yes/no):\_\_\_\_\_

Toast effect added to list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

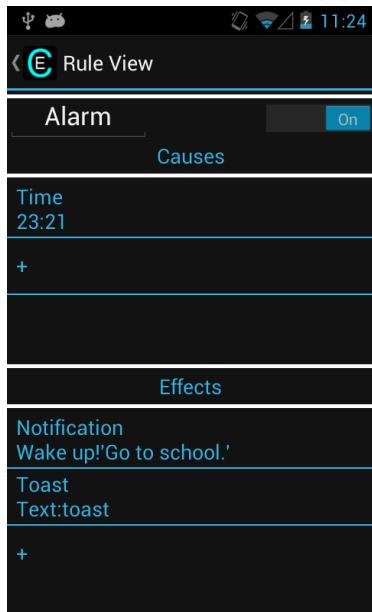
Comments:

---

---

[Passed/Failed]

---



### **Test 35: Adding Vibrate effect.**

Verifies user story: 64,95

Test by having the user try to add the effect of Vibrate.

#### Process

After getting past Test 3, tap the “+” below the effects list to create a new effect. Tap “Vibrate”. Verify that the new effect appears at the bottom of the effect list, and that the rule effect is triggered by activating the rule (making the causes true, whatever they may be).

#### Results

“Vibrate” is an effect listed (yes/no):\_\_\_\_\_

Vibrate is added to effect list (yes/no):\_\_\_\_\_

Effect is displayed correctly (yes/no):\_\_\_\_\_

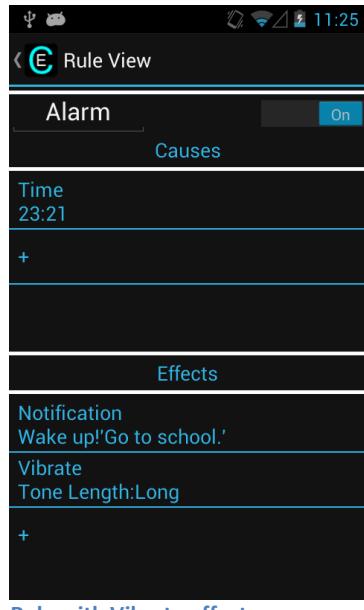
Comments:

---

---

[Passed/Failed]

---



Rule with Vibrate effect

### **Test 36: Editing Arrive cause.**

#### Verifies user story: 86

Test by having the user try to edit the Arrive cause.

#### Process

After adding an arrive cause (see earlier tests), tap the cause. Verify that the map appears and allows you to set a new location. Input a new location (following the Test 7 for adding an arrive cause). The updated information should now appear in the cause list. Finally, verify that the rule now activates when this new cause is true.

#### Results

Map appears when you tap on arrival cause (yes/no):\_\_\_\_\_

Able to make the new cause (yes/no):\_\_\_\_\_

Updated cause is displayed correctly (yes/no):\_\_\_\_\_

New cause triggers rule effects (yes/no):\_\_\_\_\_

Old cause does not trigger rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

### **Test 37: Editing Depart cause.**

#### Verifies user story: 86

Test by having the user try to edit the Depart cause.

#### Process

After adding a depart cause (see earlier tests), tap the cause. Verify that the map appears and allows you to set a new location. Input a new location (following the Test 7 or adding an arrive cause). The updated information should now appear in the cause list. Finally, verify that the rule now activates when this new cause is true.

#### Results

Map appears when you tap on depart cause (yes/no):\_\_\_\_\_

Able to make the new cause (yes/no):\_\_\_\_\_

Updated cause is displayed correctly (yes/no):\_\_\_\_\_

New cause triggers rule effects (yes/no):\_\_\_\_\_

Old cause does not trigger rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

### **Test 38: Editing Phone Call cause.**

#### Verifies user story: 86

Test by having the user try to edit the Phone Call cause.

#### Process

After adding a phone call cause (see earlier tests), tap the cause. Verify that the contact list appears and allows you to set a new contact. Input a new contact (following the Test 7or adding a phone call cause). The updated information should now appear in the cause list. Finally, verify that the rule now activates when this new cause is true.

#### Results

Contact list appears when you tap on phone call cause (yes/no):\_\_\_\_\_

Able to make the new cause (yes/no):\_\_\_\_\_

Updated cause is displayed correctly (yes/no):\_\_\_\_\_

Old cause does not trigger rule effects (yes/no):\_\_\_\_\_

New cause triggers rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

### **Test 39: Editing Text Message cause.**

#### Verifies user story: 86

Test by having the user try to edit the Text Message cause.

#### Process

After adding a text message cause (see earlier tests), tap the cause. Verify that the contact list appears and allows you to set a new contact. Input a new contact (following the Test 7 or adding a text message cause). The updated information should now appear in the cause list. Finally, verify that the rule now activates when this new cause is true.

#### Results

Contact list appears when you tap on text message cause (yes/no):\_\_\_\_\_

Able to make the new cause (yes/no):\_\_\_\_\_

Updated cause is displayed correctly (yes/no):\_\_\_\_\_

New cause triggers rule effects (yes/no):\_\_\_\_\_

Old cause does not trigger rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

#### **Test 40: Editing Time cause.**

##### Verifies user story: 86

Test by having the user try to edit the Time cause.

##### Process

After adding a time cause (see earlier tests), tap the cause. Verify that the same dialog appears asking for a time. Input a new time and tap OK. The updated information should now appear in the cause list. Finally, verify that the rule now activates when this new cause is true.

##### Results

Time dialog appears when you tap on time cause (yes/no):\_\_\_\_\_

Updated cause is displayed correctly (yes/no):\_\_\_\_\_

New cause triggers rule effects (yes/no):\_\_\_\_\_

Old cause does not trigger rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

#### **Test 41: Editing Wi-Fi SSID cause.**

##### Verifies user story: 86

Test by having the user try to edit the Wi-Fi SSID cause.

##### Process

After adding a Wi-Fi ssid cause (see earlier tests), tap the cause. Verify that the same dialog appears asking for a Wi-Fi ssid. Input a new ssid and tap Submit. The updated information should now appear in the cause list. Finally, verify that the rule now activates when this new cause is true.

##### Results

Wi-Fi SSID dialog appears when you tap on Wi-Fi SSID cause (yes/no):\_\_\_\_\_

Updated cause is displayed correctly (yes/no):\_\_\_\_\_

New cause triggers rule effects (yes/no):\_\_\_\_\_

Old cause does not trigger rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

#### **Test 42: Editing Wi-Fi Status cause.**

##### Verifies user story: 86

Test by having the user try to edit the Wi-Fi Status cause.

##### Process

After adding a Wi-Fi status cause (see earlier tests), tap the cause. Verify that the same dialog appears with the slider. Switch the slider to the opposite of what is set, and tap Submit. The updated information should now appear in the cause list. Finally, verify that the rule now activates when this new cause is true.

##### Results

Wi-Fi Status dialog appears when you tap on Wi-Fi Status cause (yes/no):\_\_\_\_\_

Updated cause is displayed correctly (yes/no):\_\_\_\_\_

New cause triggers rule effects (yes/no):\_\_\_\_\_

Old cause does not trigger rule effects (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

### **Test 43: Editing Notification effect.**

Verifies user story: 87

Test by having the user try to edit the Notification effect.

#### Process

After adding a notification effect (see earlier tests), tap the effect. Verify that the same dialog appears asking for text for the notification. Input new text and tap Submit. The updated information should now appear in the effect list. Finally, verify that when the rule is activated the new effect appears.

#### Results

Notification dialog appears when you tap on Notification effect (yes/no):\_\_\_\_\_

Updated effect is displayed correctly (yes/no):\_\_\_\_\_

New effect is displayed (yes/no):\_\_\_\_\_

Old effect is not displayed (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

#### **Test 44: Editing Sound effect.**

##### Verifies user story: 87

Test by having the user try to edit the Sound effect.

##### Process

After adding a sound effect (see earlier tests), tap the effect. Verify that the same dialog appears asking for which program to use to select the new sound. Use any input, and select a new sound. The updated information should now appear in the effect list. Finally, verify that when the rule is activated the new effect appears.

##### Results

Sound dialog appears when you tap on Sound effect (yes/no):\_\_\_\_\_

Updated effect is displayed correctly (yes/no):\_\_\_\_\_

New effect is played (yes/no):\_\_\_\_\_

Old effect is not played (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

#### **Test 45: Editing Ring Mode effect.**

Verifies user story: 87

Test by having the user try to edit the Ring Mode effect.

#### Process

After adding a ring mode effect (see earlier tests), tap the effect. Verify that the same dialog appears asking for which ring mode to use. Select a different ring mode. The updated information should now appear in the effect list. Finally, verify that when the rule is activated the new effect appears.

#### Results

Ring Mode dialog appears when you tap on Ring Mode effect (yes/no):\_\_\_\_\_

Updated effect is displayed correctly (yes/no):\_\_\_\_\_

New effect is used (yes/no):\_\_\_\_\_

Old effect is not used (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

#### **Test 46: Editing Toast effect.**

##### Verifies user story: 87

Test by having the user try to edit the Toast effect.

##### Process

After adding a toast effect (see earlier tests), tap the effect. Verify that the same dialog appears asking for text for the toast. Input new text, and tap Submit. The updated information should now appear in the effect list. Finally, verify that when the rule is activated the new effect appears.

##### Results

Toast dialog appears when you tap on Toast effect (yes/no):\_\_\_\_\_

Updated effect is displayed correctly (yes/no):\_\_\_\_\_

New effect is displayed (yes/no):\_\_\_\_\_

Old effect is not displayed (yes/no):\_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

### **Test 47: Editing Vibrate effect.**

Verifies user story: 87

Test by having the user try to edit the Notification effect.

#### Process

After adding a vibrate effect (see earlier tests), tap the effect. Verify that no dialog appears.

#### Results

No dialog appears (yes/no): \_\_\_\_\_

Comments:

---

---

[Passed/Failed]

---

#### **Test 48: Able to delete causes.**

Verifies user story: 88

Test by having user delete a cause.

#### Process

After adding a cause, long tap (tap and hold) on a cause to verify that the delete dialog appears. Test the "no" or "cancel" option on this dialog (nothing should happen). Then test the "Delete" option, which should delete the cause from the list.

#### Results

Delete dialog appears for causes (yes/no):\_\_\_\_\_

Nothing happens if you tap no (yes/no):\_\_\_\_\_

Deleted cause if you tap yes (for cause) (yes/no):\_\_\_\_\_

Rule activates correctly after deletion of cause (yes/no):\_\_\_\_\_

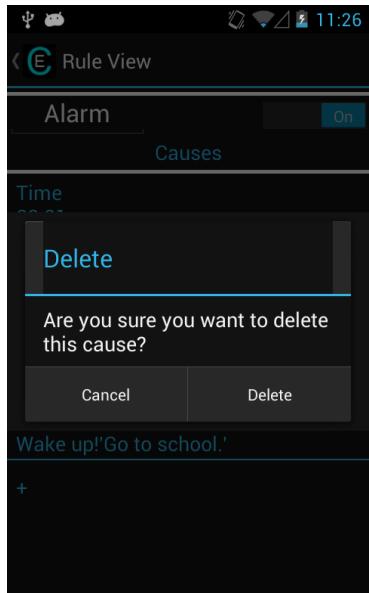
Comments:

---

---

[Passed/Failed]

---



**Delete cause dialog**

### **Test 49: Able to delete effects.**

Verifies user story: 89

Test by having user delete an effect.

#### Process

After adding an effect, long tap on an effect to verify that delete dialog appears. Test the "no" or "cancel" option on this dialog (nothing should happen). Then test the "Delete" option, which should delete the effect from the list.

#### Results

Delete dialog appears for effects (yes/no):\_\_\_\_\_

Nothing happens if you tap no (yes/no):\_\_\_\_\_

Deleted effect if you tap yes (for effect) (yes/no):\_\_\_\_\_

Rule activates correctly after deletion of effect (yes/no):\_\_\_\_\_

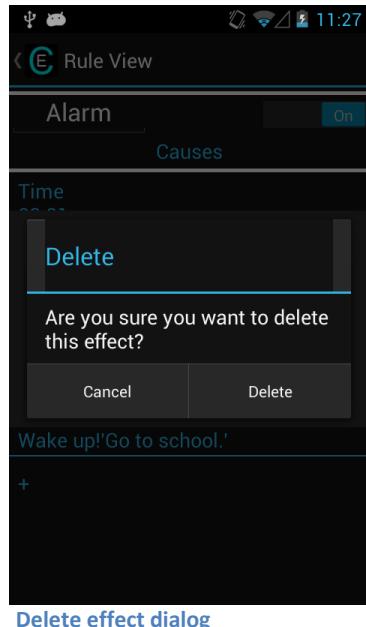
Comments:

---

---

[Passed/Failed]

---



## Test 50: Able to delete rules.

Verifies user story: 25

Test by having user delete a rule.

### Process

While looking at the rule list, long tap on a rule to verify that delete dialog appears. Test the "no" or "cancel" option on this dialog (nothing should happen). Then test the "Delete" option, which should delete the rule from the list.

### Results

Delete dialog appears for rules (yes/no):\_\_\_\_\_

Nothing happens if you tap no (yes/no):\_\_\_\_\_

Deleted rule if you tap yes (yes/no):\_\_\_\_\_

Rule no longer appears and does not activate (yes/no):\_\_\_\_\_

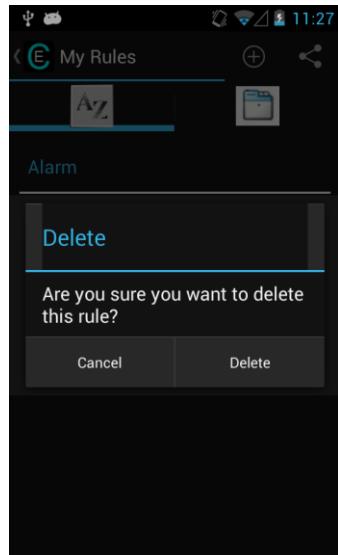
Comments:

---

---

[Passed/Failed]

---



Delete rule dialog

### **Test 51: Up button functionality works.**

Verifies user story: 79,104,105

Test by using integrated up button at top of screen while inside of app.

#### Process

While inside of an activity (any screen other than main menu), try using the integrated up button to return to the previous major screen. This should eventually lead back to the main screen without going back through your screen history (e.g. edit rule -> rule list, rule list -> main menu, new rule -> rule list).

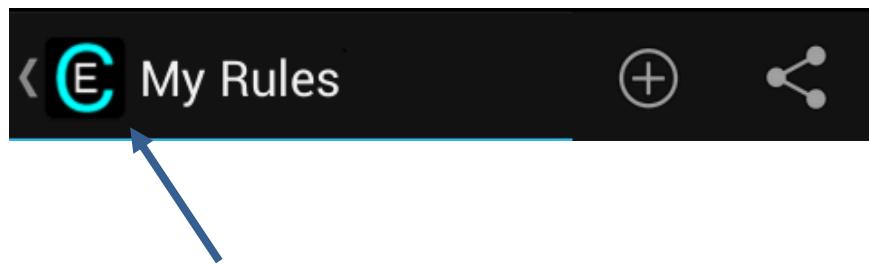
Note: the back button on phones is more of an undo, so using that is not applicable for this test, but can still be used.

#### Results

Up button is visible on all screens but Main Menu (yes/no):\_\_\_\_\_

Up button works as intended (listed in process) (yes/no):\_\_\_\_\_

[Passed/Failed]  
\_\_\_\_\_



## Test 52: Able to change name of rule.

### Verifies user story: 25

Test by changing the name of a rule and verifying it has updated.

### Process

While in the editing page of a rule, tap on the rule name, and edit it (does not matter what you name it). Then use the back button (from Test 51) to go back to the rule list. The rule name should be updated in this list. Tap on the rule again to make sure the correct name appears in the edit rule page again.

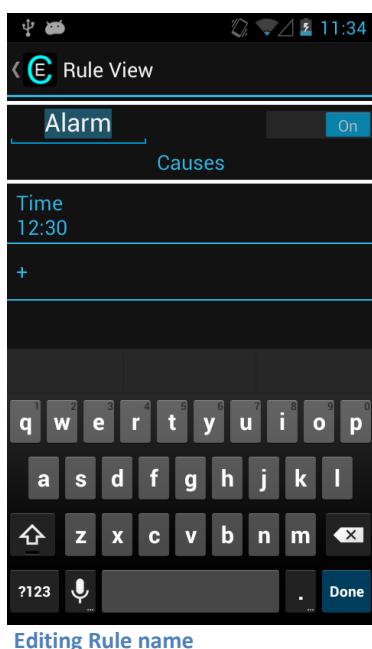
### Results

Able to edit name of rule (able to change text) (yes/no): \_\_\_\_\_

New name appears in rule list (yes/no): \_\_\_\_\_

New name appears again in edit rule page (yes/no): \_\_\_\_\_

[Passed/Failed]



### Test 53: Able to change rule active.

Verifies user story: 70

Test by changing the rule active icon in edit rule page.

#### Process

While in edit rule page, change the slider at the top of the rule (either slide or tap) to off (default is on usually, change to on if already off). Verify that the rule works now. Then, repeat after changing it back to the original state.

#### Results

Slider is visible (yes/no):\_\_\_\_\_

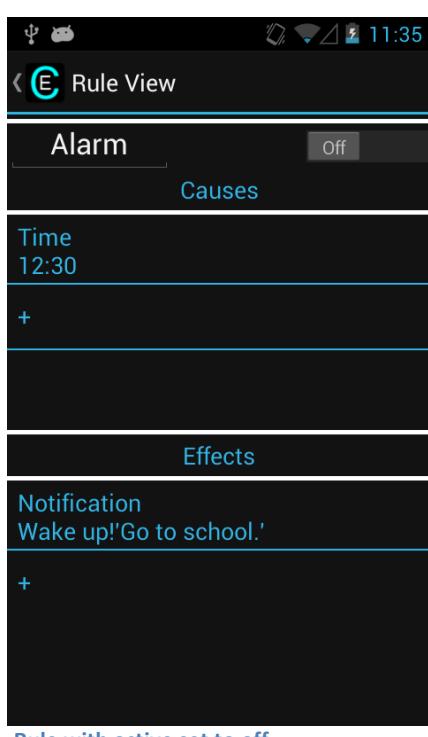
Able to change slider using slide (yes/no):\_\_\_\_\_

Able to change slider using tap (yes/no):\_\_\_\_\_

Rule does not activate while in off position (yes/no):\_\_\_\_\_

Rule does activate while in on position (yes/no):\_\_\_\_\_

[Passed/Failed]



## Test 54: Able to add rule.

### Verifies user story: 3

Test by adding a new rule.

### Process

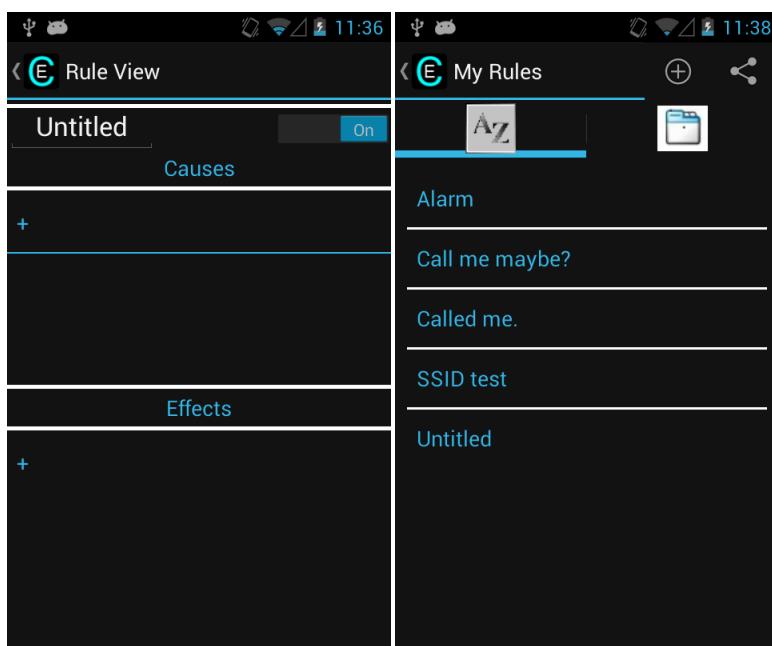
From main menu, tap “New Rule” to start making a new rule. A blank rule should show up with “Untitled” as the name. Add a cause and/or an effect. This makes the rule permanent. Use the up button to go back to the rules list. Your new rule should appear there.

### Results

Blank rule appears (yes/no):\_\_\_\_\_

New rule after instantiated appears (yes/no):\_\_\_\_\_

[Passed/Failed]



Adding new rule

New rule appears in list

### **Test 55: Able to add rule (blank).**

#### Verifies user story: 3

Test by adding a new rule.

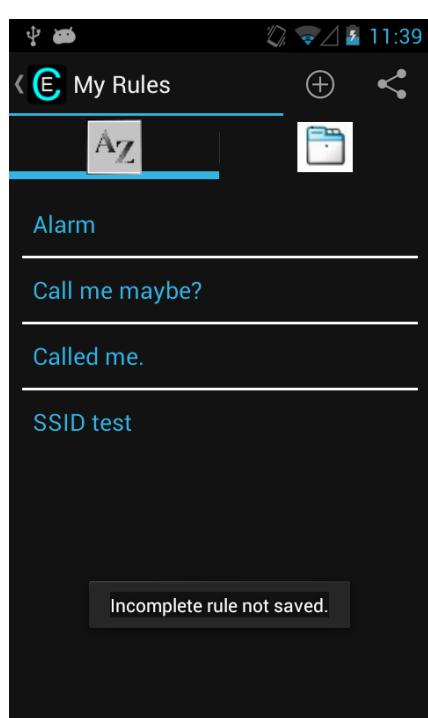
#### Process

From main menu, tap “New Rule” to start making a new rule. A blank rule should show up with “Untitled” as the name. Do not add any causes or effects. Use the up button to go back to the rules list. The new rule should not appear in the list. A Toast message should notify you that the rule was not added.

#### Results

New rule does not appear (yes/no):\_\_\_\_\_

[Passed/Failed]



**Notification warning of blank rule**

## Test 56: Test the sharing functions (NFC).

Verifies user story: 97,98

Test by trying to share a rule through NFC.

### Process

From the Main Menu, tap the share button. Choose a rule that you wish to share. When the phone says it is "Ready to send" your rule over NFC, pickup the receiving phone (which has the application installed). Press the backs of the two phones together. As they touch, the sending phone should vibrate and shrink the UI to a smaller size. When this occurs, tap and hold the screen to send the rule to the other device. As long as the application is installed on the other phone, the phone is on, and the application has been opened at least once, the other phone will receive the rule from any screen. This should occur even when a different application is open on the receiving phone. Verify that the rule was transferred by viewing the rule list on the receiving phone, and test that the rule still works properly.

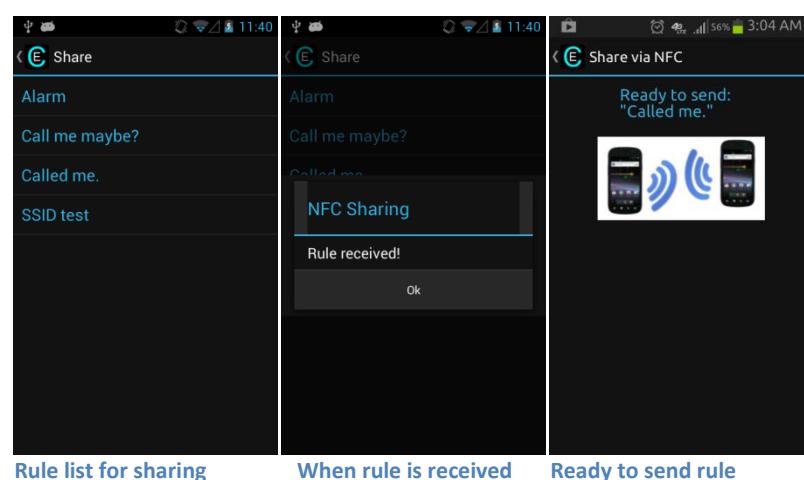
### Results

Able to share rule via NFC (yes/no): \_\_\_\_\_

Shared rule appears in rule list (yes/no): \_\_\_\_\_

Rule works as originally intended (yes/no): \_\_\_\_\_

[Passed/Failed]



## Test 57: Make sure the Boolean algebra is added correctly.

Verifies user story: 74,100

Test by adding a 2<sup>nd</sup> cause on any rule.

### Process

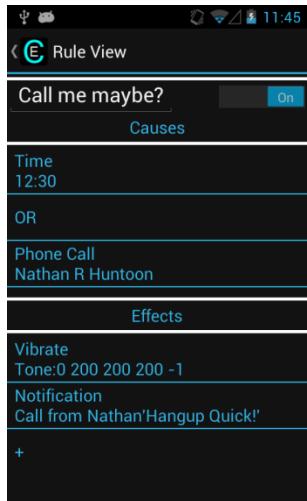
Go to any rule that has at least 1 cause, and add any cause. An OR should be added between the causes. Then, delete a cause and the OR/AND should be removed from the cause list.

### Results

OR is added between causes (yes/no):\_\_\_\_\_

OR/AND is deleted when cause is deleted (yes/no):\_\_\_\_\_

[Passed/Failed]



Rule with multiple causes

## Test 58: Make sure the Boolean algebra works correctly (AND).

Verifies user story: 74,100

Test by making a rule with ANDs in the cause list, and test that it works properly.

### Process

Set up one of the rules with multiple causes. Tap on an OR box to change it to an AND. Activate the rule, and test that the logic works properly (see below).

Note: with the AND rule, make sure both causes can be active at the same time (e.g. time and location, time and Wi-Fi are good examples).

### Results

Able to change OR to AND between causes (yes/no):\_\_\_\_\_

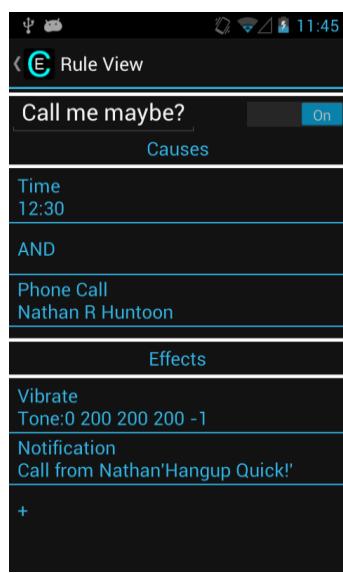
Rule is activated when both causes true(yes/no):\_\_\_\_\_

Rule is not activated when first cause true and second cause false (yes/no):\_\_\_\_\_

Rule is not activated when second cause true and first cause false (yes/no):\_\_\_\_\_

Rule is not activated when both causes are false (yes/no):\_\_\_\_\_

[Passed/Failed]



Rule with AND between causes

## Test 59: Make sure the Boolean algebra works correctly (OR).

Verifies user story: 74,100

Test by making a rule with ORs in the cause list, and test that it works properly.

### Process

Set up one of the rules with multiple causes. Tap on an AND box to change it to an OR, if necessary. Activate the rule, and test that the logic works properly (see below).

Note: with the OR rule, either cause can trigger a rule

### Results

Able to change AND to OR between causes (yes/no):\_\_\_\_\_

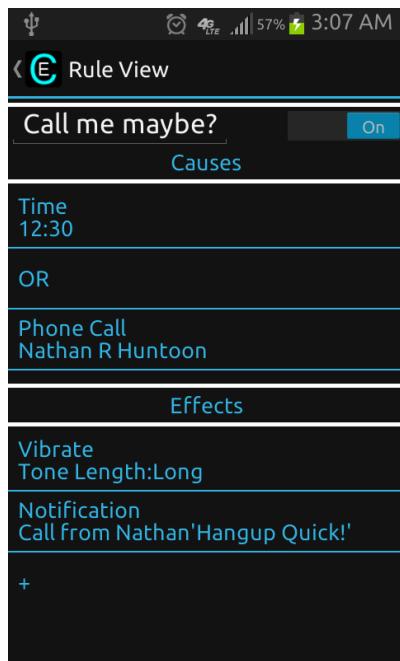
Rule is activated when both causes true(yes/no):\_\_\_\_\_

Rule is activated when first cause true and second cause false (yes/no):\_\_\_\_\_

Rule is activated when second cause true and first cause false (yes/no):\_\_\_\_\_

Rule is not activated when both causes are false (yes/no):\_\_\_\_\_

[Passed/Failed]



## Test 60: Make sure you cannot add duplicate causes.

Verifies user story: 84

Test by trying to add a duplicate cause to a rule.

### Process

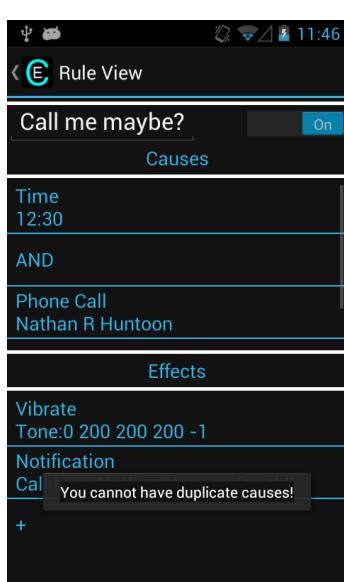
Go into the edit page of any rule and try to add a duplicate of any of the causes (same cause, same arguments). The new cause should not be added to the rule.

### Results

New duplicate cause is not added to rule (yes/no):\_\_\_\_\_

Toast is shown notifying the user that it has found a duplicate cause (yes/no):\_\_\_\_\_

[Passed/Failed]



Warning when adding duplicate causes

### **Test 61: Make sure you cannot add duplicate effects.**

Verifies user story: 84

Test by trying to add a duplicate effect to a rule.

#### Process

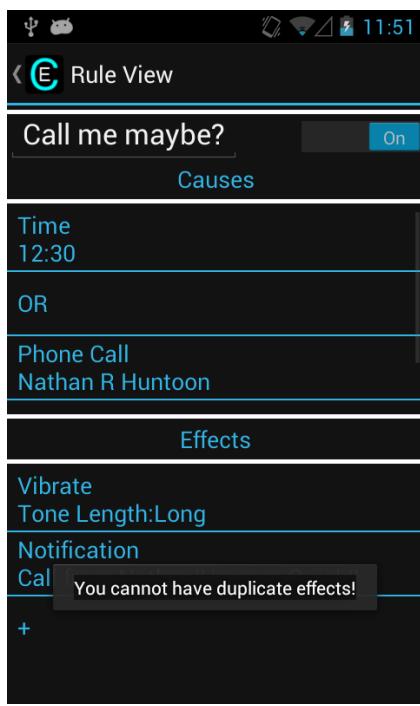
Go into the edit page of any rule and try to add a duplicate of any of the effects (same effect, same arguments). The new effect should not be added to the rule.

#### Results

New duplicate effect is not added to rule (yes/no):\_\_\_\_\_

Toast is shown notifying the user that it has found a duplicate effect (yes/no):\_\_\_\_\_

[Passed/Failed]



Warning when adding duplicate effects

## Test 62: Bring up menu in My Rules (rule list).

### Verifies user story: 25

Test by pressing the Menu key while inside of My Rules. The Menu can either be a hardware button on the bottom of the phone or 3 vertical dots on the top action bar.

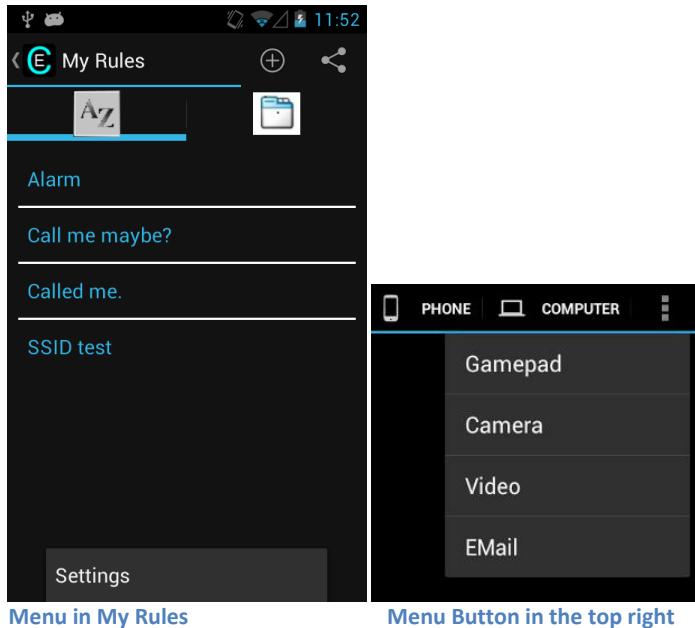
### Process

Go into the My Rules page and press Menu. The Menu should pop up with Settings/Preferences.

### Results

Menu shows up with only Settings appearing (yes/no):\_\_\_\_\_

[Passed/Failed]



### **Test 63: Bring up menu in Edit Rule.**

Verifies user story: 25

Test by pressing the Menu key while inside of Edit Rule.

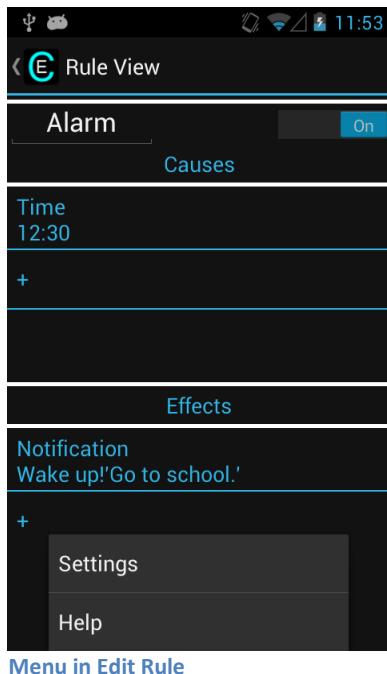
#### Process

Go into the Edit Rule page and press Menu. The Menu should pop up with Settings/Preferences and Help.

#### Results

Menu shows up with Settings and Help appearing (yes/no):\_\_\_\_\_

[Passed/Failed]



Menu in Edit Rule

#### **Test 64: Bring up menu in Help.**

Verifies user story: 23,26,76, 123

Test by pressing the Menu key while inside of Help page.

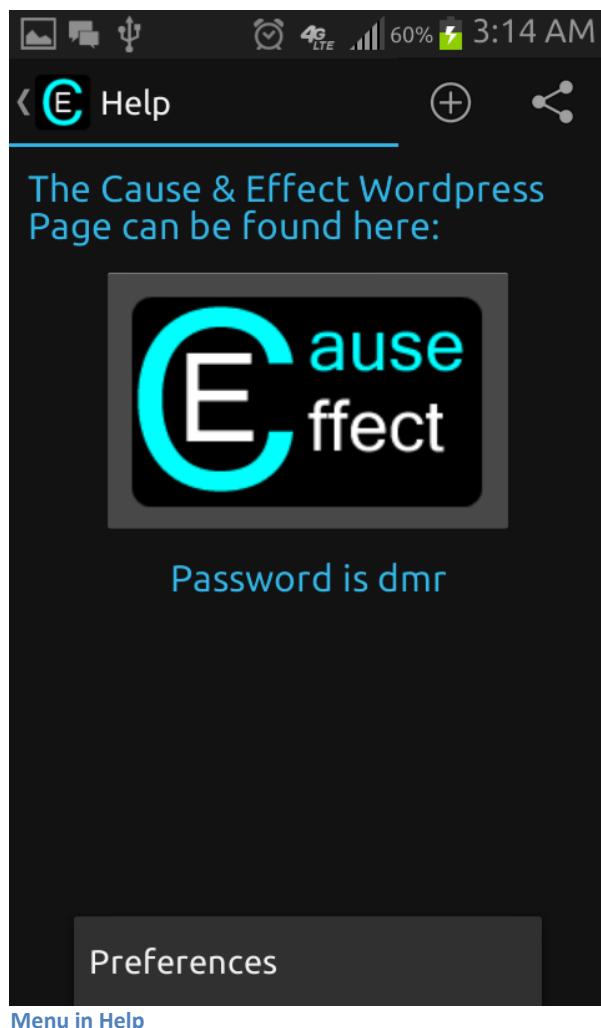
#### Process

Go into the Help page and press Menu. The Menu should pop up with Preferences.

#### Results

Menu shows up with only Settings appearing (yes/no):\_\_\_\_\_

[Passed/Failed]



## **Test 65: Bring up menu in Settings.**

Verifies user story: 27

Test by pressing the Menu key while inside of Settings page.

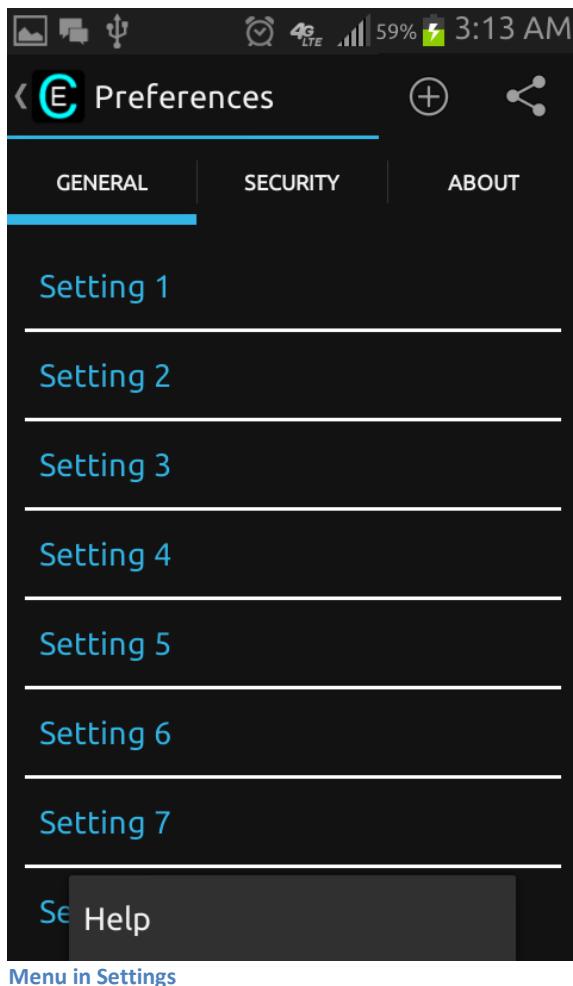
### Process

Go into the Settings page and press Menu. The Menu should pop up with Help.

### Results

Menu shows up with only Help appearing (yes/no):\_\_\_\_\_

[Passed/Failed]



## Test 66: Test Settings tabs.

Verifies user story: 9,10,11,12,13,14

Test by pressing the tabs in the Settings page to make sure all the tabs work.

### Process

Go into the Settings page and press the General tab, Accounts tab, Security tab, and About tab and make sure they work.

### Results

General tab works (yes/no):\_\_\_\_\_

Accounts tab works (yes/no):\_\_\_\_\_

Security tab works (yes/no):\_\_\_\_\_

About tab works (yes/no):\_\_\_\_\_

[Passed/Failed]



**Test 67: Test application service.**

Verifies user story: 83,90,103

Test by returning to the home screen and activating the rule.

Process

Make sure there is a relatively simple rule to activate and then exit out of the app by constantly pressing the android back button (or simply press the Home button). Then activate the rule to make sure it works.

Results

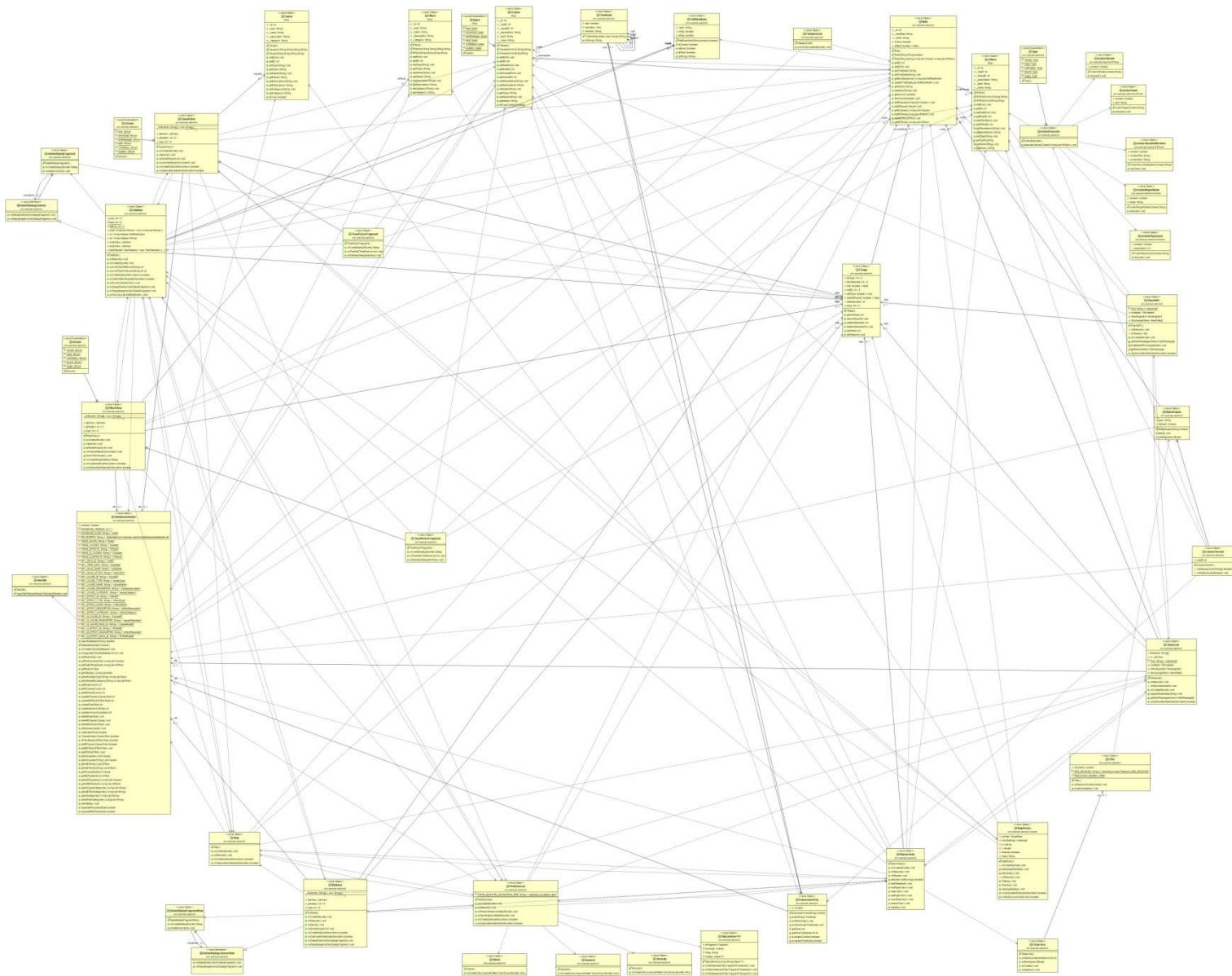
Rule activates while app is closed (yes/no): \_\_\_\_\_

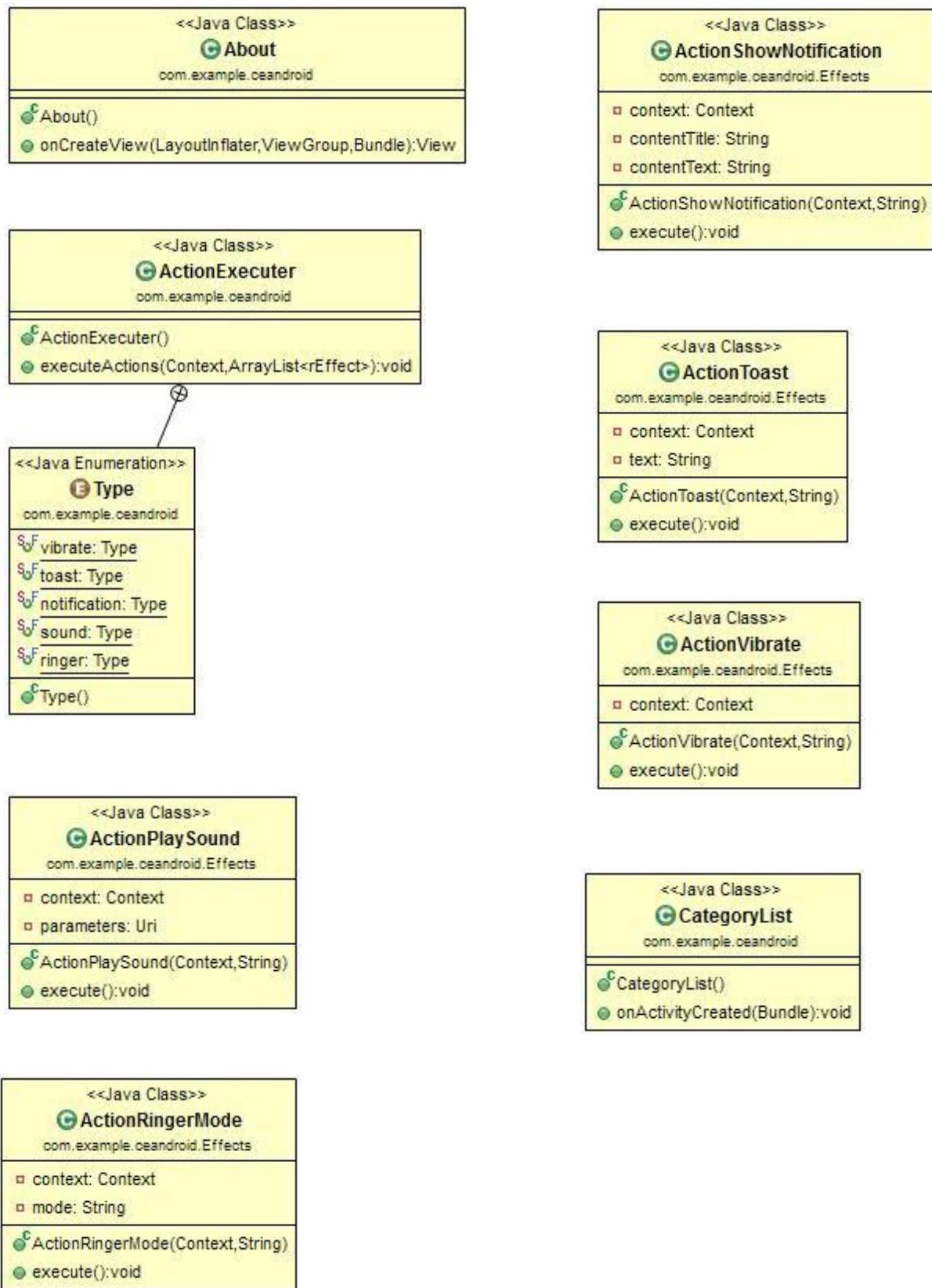
[Passed/Failed]

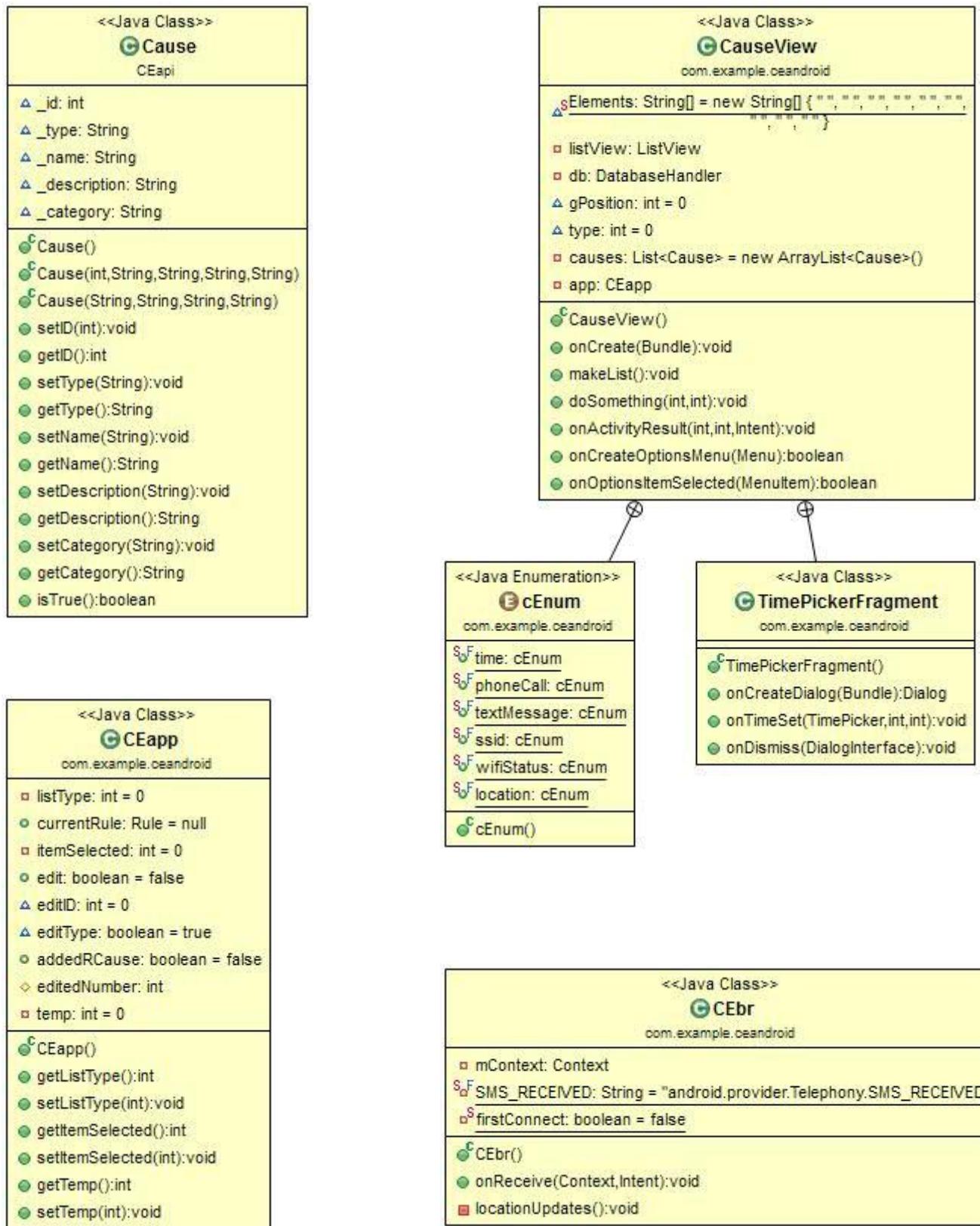
\_\_\_\_\_

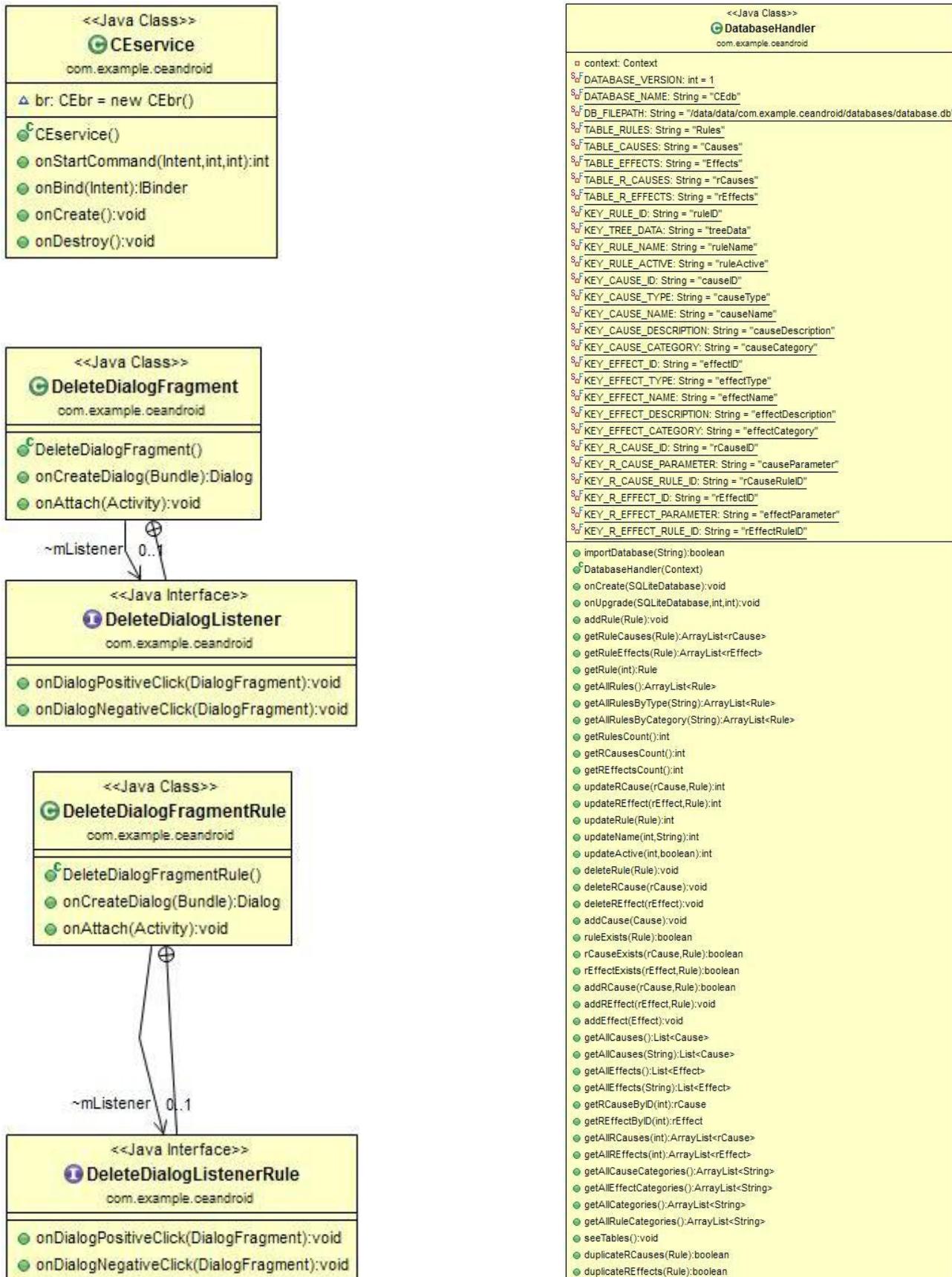
# Java Class Diagrams

This is a graph showing all of the connections for our Java Classes. After this overview, we have all of the individual class diagrams outlined.

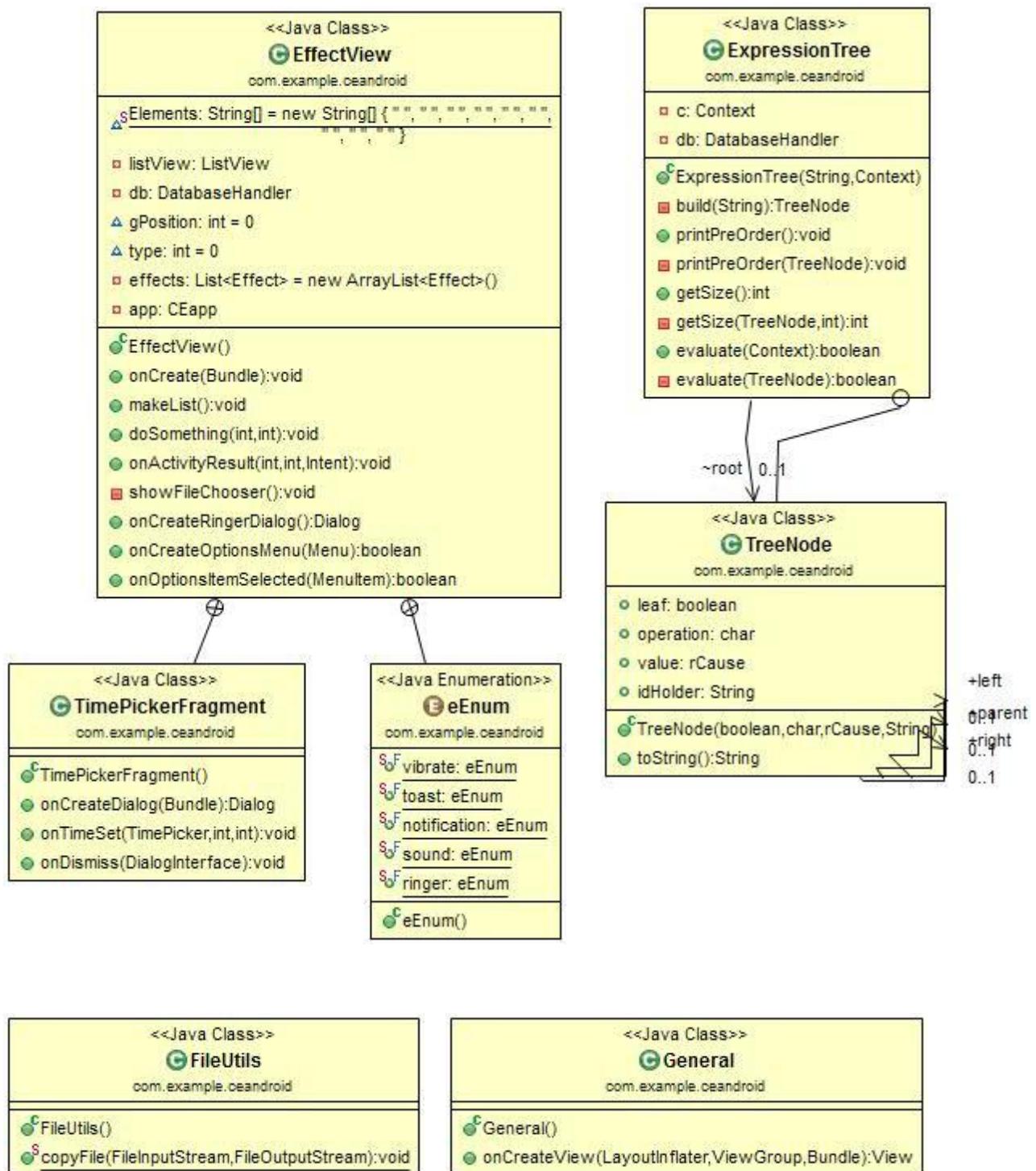








<p><b>&lt;&lt;Java Class&gt;&gt;</b></p> <p><b>C EditRule</b></p> <p>com.example.ceandroid</p> <hr/> <p>▫ app: CEapp = (CEapp) this.getApplication()</p> <p>△ type: int = 0</p> <p>△<sup>S</sup> size: int = 0</p> <p>△<sup>S</sup> delPos: int = 0</p> <p>△ cList: ArrayList&lt;EditRuleNode&gt; = new ArrayList&lt;EditRuleNode&gt;()</p> <p>△ bools: ArrayList&lt;EditRuleNode&gt; = new ArrayList&lt;EditRuleNode&gt;()</p> <p>△ eList: ArrayList&lt;String&gt; = new ArrayList&lt;String&gt;()</p> <p>△ cA: ArrayAdapter&lt;EditRuleNode&gt;</p> <p>△ eA: ArrayAdapter&lt;String&gt;</p> <p>△ cListView: ListView</p> <p>△ eListView: ListView</p> <p>△ textWatcher: TextWatcher = new TextWatcher() {...}</p> <hr/> <p>●<sup>C</sup> EditRule()</p> <p>● onResume():void</p> <p>● onCreate(Bundle):void</p> <p>● convertTypeToEEnum(String):int</p> <p>● convertTypeToCEnum(String,int):int</p> <p>● onCreateOptionsMenu(Menu):boolean</p> <p>● onOptionsItemSelected(MenuItem):boolean</p> <p>● onSwitchClicked(View):void</p> <p>● onDialogPositiveClick(DialogFragment):void</p> <p>● onDialogNegativeClick(DialogFragment):void</p> <p>● print(ArrayList&lt;EditRuleNode&gt;):void</p>	<p><b>&lt;&lt;Java Class&gt;&gt;</b></p> <p><b>C EditRuleNode</b></p> <p>com.example.ceandroid</p> <hr/> <p>△ value: String</p> <p>△ cFlag: boolean</p> <p>△ bFlag: boolean</p> <hr/> <p>●<sup>C</sup> EditRuleNode(String,boolean,boolean)</p> <p>● isCause():boolean</p> <p>● isBool():boolean</p> <p>● isPlus():boolean</p> <p>● toString():String</p>	<p><b>&lt;&lt;Java Class&gt;&gt;</b></p> <p><b>C Effect</b></p> <p>CEapi</p> <hr/> <p>△ _id: int</p> <p>△ _type: String</p> <p>△ _name: String</p> <p>△ _description: String</p> <p>△ _category: String</p> <hr/> <p>●<sup>C</sup> Effect()</p> <p>●<sup>C</sup> Effect(int,String,String,String)</p> <p>●<sup>C</sup> Effect(String,String,String,String)</p> <p>● setID(int):void</p> <p>● getID():int</p> <p>● setType(String):void</p> <p>● getType():String</p> <p>● setName(String):void</p> <p>● getName():String</p> <p>● setDescription(String):void</p> <p>● getDescription():String</p> <p>● setCategory(String):void</p> <p>● getCategory():String</p>
--	---	--



<<Java Class>>

**G Help**

com.example.ceandroid

- **Help()**
- **onCreate(Bundle):void**
- **onResume():void**
- **onCreateOptionsMenu(Menu):boolean**
- **onOptionsItemSelected(MenuItem):boolean**

<<Java Class>>

**G MainActivity**

com.example.ceandroid

- **app: CEapp**
- **MainActivity()**
- **onCreate(Bundle):void**
- **onResume():void**
- **onPause():void**
- **isMyServiceRunning():boolean**
- **testDatabase():void**
- **myRules(View):void**
- **help(View):void**
- **settings(View):void**
- **newRule(View):void**
- **share(View):void**
- **loading():void**

<<Java Class>>

**G MapPicker**

com.example.ceandroid.Causes

- **myMap: GoogleMap**
- **myUiSettings: UiSettings**
- **p: LatLng**
- **r: double**
- **finished: boolean**
- **name: String**

- **MapPicker()**
- ◊ **onCreate(Bundle):void**
- **setUpMapIfNeeded():void**
- **setUpMap():void**
- ◊ **onResume():void**
- **rDialog():void**
- **finished():void**
- **messageDialog():void**
- **onOptionsItemSelected(MenuItem):boolean**
- **onKeyDown(int,KeyEvent):boolean**

<<Java Class>>

**G MyRules**

com.example.ceandroid

- △ **Elements: String[] = new String[] { "", "", "", "", "", "" };**
- **listView: ListView**
- **db: DatabaseHandler**
- △ **gPosition: int = 0**
- △ **type: int = 0**
- **rules: List<Rule> = new ArrayList<Rule>()**
- **app: CEapp**

- **MyRules()**
- **onCreate(Bundle):void**
- **onResume():void**
- **makeList():void**
- **doSomething(int,int):void**
- **onCreateOptionsMenu(Menu):boolean**
- **onOptionsItemSelected(MenuItem):boolean**
- **onDialogPositiveClick(DialogFragment):void**
- **onDialogNegativeClick(DialogFragment):void**

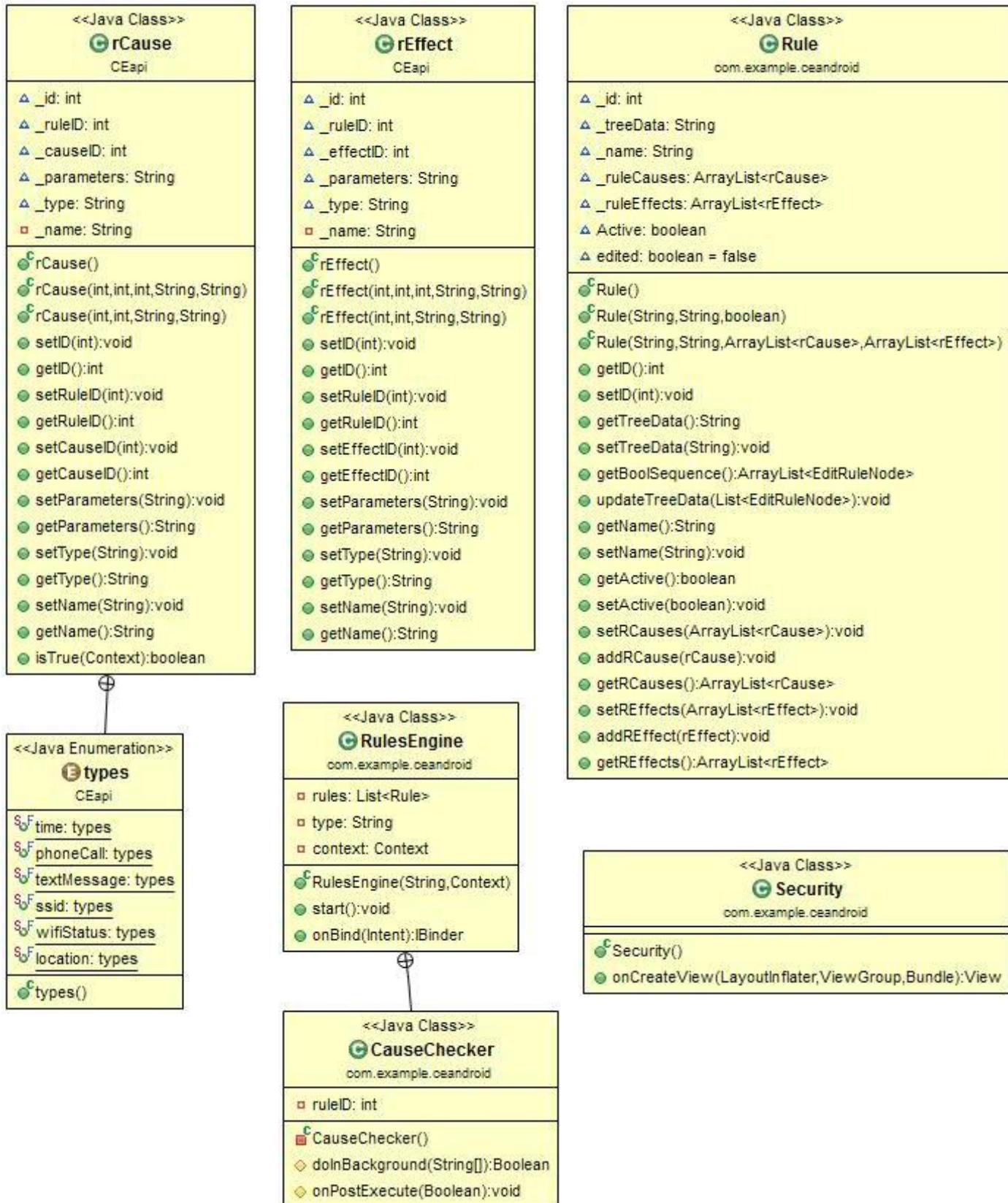
<<Java Class>>

**G Preferences**

com.example.ceandroid

S<sub>o</sub>F **STATE\_SELECTED\_NAVIGATION\_ITEM: String = "selected\_navigation\_item"**

- **Preferences()**
- **onCreate(Bundle):void**
- **onResume():void**
- **onRestoreInstanceState(Bundle):void**
- **onSaveInstanceState(Bundle):void**
- **onCreateOptionsMenu(Menu):boolean**
- **onOptionsItemSelected(MenuItem):boolean**



<p>&lt;&lt;Java Class&gt;&gt;</p> <p><b>ShareList</b></p> <p>com.example.ceandroid</p>	<p>Elements: String[]</p> <p>db: DatabaseHandler</p> <p>rules: List&lt;Rule&gt; = new ArrayList&lt;Rule&gt;()</p> <p>app: CEapp</p> <p>lv: ListView</p> <p><u>S F TAG: String = "ceandroid"</u></p> <p>    nAdapter: NfcAdapter</p> <p>    nPendingIntent: PendingIntent</p> <p>    nExchangeFilters: IntentFilter[]</p>
<p><b>ShareList()</b></p> <p>    onResume():void</p> <p>    onNewIntent(Intent):void</p> <p>    onCreate(Bundle):void</p> <p>    replaceRuleFields(String):void</p> <p>    getNdefMessages(Intent):NdefMessage[]</p> <p>    onOptionsItemSelected(MenuItem):boolean</p>	

<p>&lt;&lt;Java Class&gt;&gt;</p> <p><b>TabListener&lt;T&gt;</b></p> <p>com.example.ceandroid</p>	<p>mFragment: Fragment</p> <p><u>F mActivity: Activity</u></p> <p><u>F mTag: String</u></p> <p><u>F mClass: Class&lt;T&gt;</u></p>
<p><b>TabListener(Activity, String, Class&lt;T&gt;)</b></p> <p>    onTabSelected(Tab, FragmentTransaction):void</p> <p>    onTabUnselected(Tab, FragmentTransaction):void</p> <p>    onTabReselected(Tab, FragmentTransaction):void</p>	

# Code Appendix

## Java

### rEffect.java

```
package CEapi;

/**
 * The rEffect class is the structure that stores information for an effect
 * associated to a rule
 *
 * @author CEandroid SMU
 */
public class rEffect {
    /**
     * _id The rEffect Identifier _ruleID The Rule Identifier _effectID The
     * effect Identifier
     */
    int _id, _ruleID, _effectID;
    /**
     * _parameters Stores the information needed to execute this rEffect _type
     * The type of rEffect used for the back end
     */
    String _parameters, _type;
    /**
     * _name The name of this rEffect
     */
    private String _name;

    /**
     * Default Constructor
     */
    public rEffect() {

    }

    /**
     * Defined rEffect Constructor
     *
     * @param id
     * @param ruleID
     * @param effectID
     * @param parameters
     * @param type
     */
    public rEffect(int id, int ruleID, int effectID, String parameters,
                  String type) {
        this._id = id;
        this._ruleID = ruleID;
        this._effectID = effectID;
        this._parameters = parameters;
        this._type = type;
        this._name = "";
    }
}
```

```

}

/**
 * Defined rEffect Constructor without rEffect ID
 *
 * @param ruleID
 * @param effectID
 * @param parameters
 * @param type
 */
public rEffect(int ruleID, int effectID, String parameters, String type) {
    this._ruleID = ruleID;
    this._effectID = effectID;
    this._parameters = parameters;
    this._type = type;
    this._name = "";
}

/**
 * Set _id
 *
 * @param id
 */
public void setID(int id) {
    this._id = id;
}

/**
 * Get _id
 *
 * @return this._id
 */
public int getID() {
    return this._id;
}

/**
 * Set _ruleID
 *
 * @param ruleID
 */
public void setRuleID(int ruleID) {
    this._ruleID = ruleID;
}

/**
 * Get _ruleID
 *
 * @return this._ruleID
 */
public int getRuleID() {
    return this._ruleID;
}

/**
 * Set _effectID

```

```

*
 * @param effectID
 */
public void setEffectID(int effectID) {
    this._effectID = effectID;
}

/**
 * Get _effectID
 *
 * @return this._effectID
 */
public int getEffectID() {
    return this._effectID;
}

/**
 * Set _parameters
 *
 * @param parameters
 */
public void setParameters(String parameters) {
    this._parameters = parameters;
}

/**
 * Get _parameters
 *
 * @return this._parameters
 */
public String getParameters() {
    return this._parameters;
}

/**
 * Set _type
 *
 * @param type
 */
public void setType(String type) {
    this._type = type;
}

/**
 * Get _type
 *
 * @return this._type
 */
public String getType() {
    return this._type;
}

/**
 * Set _name
 *
 * @param name
*/

```

```
 */
public void setName(String name) {
    this._name = name;
}

/**
 * Get _name
 *
 * @return this._name
 */
public String getName() {
    return this._name;
}
}
```

## Effect.java

```
package CEapi;

/**
 * The Effect class is the structure that stores information for a specific
 * effect in the backend No rule data is stored within this class
 *
 * @author CEandroid SMU
 */
public class Effect {
    /**
     * The unique identifier for this effect
     */
    int _id;

    /**
     * The _type refers to the grouping of effects within the back end The _name
     * stores the name of this specific effect The _descripton is a more user
     * friendly representation of the _name THe _category refers to the groupin
     * g
     * of effects on the front end
     */
    String _type, _name, _description, _category;

    /**
     * Default Constructor
     */
    public Effect() {

    }

    /**
     * Defined Effect Constructor
     *
     * @param id
     * @param type
     * @param name
     * @param description
     * @param category
     */
    public Effect(int id, String type, String name, String description,
                 String category) {
        this._id = id;
        this._type = type;
        this.setName(name);
        this._description = description;
        this._category = category;
    }

    /**
     * Defined Effect Constructor without the _id
     *
     * @param type
     * @param name
     */
}
```

```

        * @param description
        * @param category
        */
    public Effect(String type, String name, String description, String category
) {
    this._type = type;
    this.setName(name);
    this._description = description;
    this._category = category;
}

/**
 * Set _id
 *
 * @param id
 */
public void setID(int id) {
    this._id = id;
}

/**
 * Get _id
 *
 * @return this._id
 */
public int getID() {
    return this._id;
}

/**
 * Set _type
 *
 * @param type
 */
public void setType(String type) {
    this._type = type;
}

/**
 * Get _type
 *
 * @return this._type
 */
public String getType() {
    return this._type;
}

/**
 * Set _name
 *
 * @param name
 */
public void setName(String name) {
    this._name = name;
}

```

```

/**
 * Get _name
 *
 * @return this._name
 */
public String getName() {
    return this._name;
}

/**
 * Set _description
 *
 * @param description
 */
public void setDescription(String description) {
    this._description = description;
}

/**
 * Get _description
 *
 * @return this._description
 */
public String getDescription() {
    return this._description;
}

/**
 * Set _category
 *
 * @param category
 */
public void setCategory(String category) {
    this._category = category;
}

/**
 * Get _category
 *
 * @return this._category
 */
public String getCategory() {
    return this._category;
}
}

```

## rCause.java

```
package CEapi;

import java.text.SimpleDateFormat;
import java.util.Date;

import com.example.ceandroid.DatabaseHandler;

import android.annotation.SuppressLint;
import android.content.Context;
import android.database.Cursor;
import android.location.Location;
import android.location.LocationManager;
import android.net.Uri;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.provider.CallLog;

/**
 * The rCause class is the structure that stores information for an effect
 * associated to a rule
 *
 * @author CEandroid SMU
 */
public class rCause {
    /**
     * id The rCause Identifier    ruleID The Rule Identifier    causeID The cause
     * Identifier
     */
    int _id, _ruleID, _causeID;
    /**
     * _parameters Stores the information needed to evaluate this rCause _type
     * The type of rCause used for the back end
     */
    String _parameters, _type;
    /**
     * _name The name of this rCause
     */
    private String _name;

    /**
     * Default Constructor
     */
    public rCause() {

    }

    /**
     * Defined rCause Constructor
     *
     * @param id
     * @param ruleID
     * @param causeID
     * @param parameters
     * @param type
     */
}
```

```

        */
    public rCause(int id, int ruleID, int causeID, String parameters,
        String type) {
        this._id = id;
        this._ruleID = ruleID;
        this._causeID = causeID;
        this._parameters = parameters;
        this._type = type;
        this._name = "";
    }

    /**
     * Defined rCause Constructor without the rCause id
     *
     * @param ruleID
     * @param causeID
     * @param parameters
     * @param type
     */
    public rCause(int ruleID, int causeID, String parameters, String type) {
        this._ruleID = ruleID;
        this._causeID = causeID;
        this._parameters = parameters;
        this._type = type;
        this._name = "";
    }

    /**
     * Set _id
     *
     * @param id
     */
    public void setID(int id) {
        this._id = id;
    }

    /**
     * Get _id
     *
     * @return this._id
     */
    public int getID() {
        return this._id;
    }

    /**
     * Set _ruleID
     *
     * @param ruleID
     */
    public void setRuleID(int ruleID) {
        this._ruleID = ruleID;
    }

    /**
     * Get _ruleID
     */

```

```

*
 * @return this._ruleID
 */
public int getRuleID() {
    return this._ruleID;
}

/**
 * Set _causeID
 *
 * @param causeID
 */
public void setCauseID(int causeID) {
    this._causeID = causeID;
}

/**
 * Get _causeID
 *
 * @return this._causeID
 */
public int getCauseID() {
    return this._causeID;
}

/**
 * Set _parameters
 *
 * @param parameters
 */
public void setParameters(String parameters) {
    this._parameters = parameters;
}

/**
 * Get _parameters
 *
 * @return this._parameters
 */
public String getParameters() {
    return this._parameters;
}

/**
 * Set _type
 *
 * @param type
 */
public void setType(String type) {
    this._type = type;
}

/**
 * Get _type
 *
 * @return this._type

```

```

        */
    public String getType() {
        return this._type;
    }

    /**
     * Set _name
     *
     * @param name
     */
    public void setName(String name) {
        this._name = name;
    }

    /**
     * Get _name
     *
     * @return this._name
     */
    public String getName() {
        return this._name;
    }

    /**
     * Evaluates the rCause as true or false
     *
     * @param c
     * @return boolean The result of the rCause evaluation
     */
    @SuppressLint("SimpleDateFormat")
    public boolean isTrue(Context c) {
        // return true;
        boolean result = false;

        if (this._type == null) {
            result = false;
        } else {
            types t = types.valueOf(this._type);
            switch (t) {
                case time: {
                    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

                    // make a string of the current time
                    Date d = new Date();
                    String curTime = sdf.format(d);

                    if (curTime.equals(this._parameters)) {
                        result = true;
                    } else {
                        result = false;
                    }
                }

                break;
            }
        }

        case phoneCall: {
    
```

```

// Code here right now will just check the parameter for the
// contact name and compare it contact name stored in this
// rCause
Uri calls = android.provider.CallLog.Calls.CONTENT_URI;// Uri.parse("content://call log/calls");
Cursor callCursor = c.getContentResolver().query(calls, null,
    null, null, null);
callCursor.moveToFirst();
@SuppressWarnings("unused")
String name = "", duration, type, number = "";
if (callCursor.getCount() > 0) {
    do {
        name = callCursor.getString(callCursor
            .getColumnIndex(CallLog.Calls.CACHED_NAME));
        number = callCursor.getString(callCursor
            .getColumnIndex(CallLog.Calls.NUMBER));
        type = callCursor
            .getString(callCursor
                .getColumnIndex(android.provider.CallLog.Calls.TYPE));
        duration = callCursor
            .getString(callCursor
                .getColumnIndex(android.provider.CallLog.Calls.DURATION))
    ;
        if ((Integer.parseInt(duration) == 0)) {
            if (Integer.parseInt(type) == android.provider.CallLog.Calls.MI
SSED_TYPE) {
                callCursor.moveToLast();
            } else {
                // System.out.println("*SKIP* Text Message: " +
                // name);
            }
        } else {
            callCursor.moveToLast();
        }
    } while (callCursor.moveToNext());
}
callCursor.close();
// int commaCounter = 0;
// char cur;
String parsedNumber = "";
// String parse1 = "";
// String parse2 = "";

for (int i = 0; i < this._parameters.length(); i++) {
    // cur = this._parameters.charAt(i);
    parsedNumber += Character.toString(this._parameters
        .charAt(i));
}

// if the passed in parameter's contact name is the same as the
// stored contact name, return true
result = parsedNumber.equals(name);
break;
}

```

```

// right now textMessage checks for the same thing as phoneCall
case textMessage: {
    // Code here right now will just check the parameter for the
    // contact name and compare it contact name stored in this
    // rCause
    Uri calls = android.provider.CallLog.Calls.CONTENT_URI;// Uri.parse("content://call_log/calls");
    Cursor callCursor = c.getContentResolver().query(calls, null,
        null, null, null);
    callCursor.moveToFirst();
    @SuppressWarnings("unused")
    String name = "", duration, type, number = "";
    if (callCursor.getCount() > 0) {
        do {
            name = callCursor.getString(callCursor
                .getColumnIndex(CallLog.Calls.CACHED_NAME));
            number = callCursor.getString(callCursor
                .getColumnIndex(CallLog.Calls.NUMBER));
            type = callCursor
                .getString(callCursor
                    .getColumnIndex(android.provider.CallLog.Calls.TYPE));
            duration = callCursor
                .getString(callCursor
                    .getColumnIndex(android.provider.CallLog.Calls.DURATION))
        ;
        if ((Integer.parseInt(duration) == 0)) {
            if (Integer.parseInt(type) == android.provider.CallLog.Calls.MISSED_TYPE) {
                // System.out.println("*SKIP* Missed Phone Call: "
                // + name);
            } else {
                callCursor.moveToLast();
                System.out.println("Text Message: " + name);
            }
        } else {
            // System.out.println("*SKIP* Phone Call: " + name);
        }
    } while (callCursor.moveToNext());
}
callCursor.close();
// int commaCounter = 0;
// char cur;
String parsedNumber = "";
// String parse1 = "";
// String parse2 = "";

for (int i = 0; i < this._parameters.length(); i++) {
    // cur = this._parameters.charAt(i);
    parsedNumber += Character.toString(this._parameters
        .charAt(i));
}

// if the passed in parameter's contact name is the same as the
// stored contact name, return true
result = parsedNumber.equals(name);

```

```

        break;
    }
    case ssid: {
        @SuppressWarnings("static-access")
        WifiManager wm = (WifiManager) c
            .getSystemService(c.WIFI_SERVICE);
        WifiInfo wi = wm.getConnectionInfo();
        String SSID = wi.getSSID();

        if (SSID != null) {
            SSID.trim();
            // System.out.println("# if characters = " + SSID.)

            if (SSID.equals(this.getParameters())) {
                result = true;
            }
        } else {
            return false;
        }

        break;
    }

    case wifiStatus: {
        @SuppressWarnings("static-access")
        WifiManager wm = (WifiManager) c
            .getSystemService(c.WIFI_SERVICE);
        boolean res = wm.isWifiEnabled();
        if (res == true && this.getParameters().equals("on")) {
            result = true;
        } else if (res == false && this.getParameters().equals("off")) {
            result = true;
        }
        break;
    }

    // Location based rules. Currently evaluates Arrivals and Departures
    // only
    case location: {
        LocationManager locManager = (LocationManager) c
            .getSystemService(Context.LOCATION_SERVICE);
        Location myLocation = locManager
            .getLastKnownLocation(LocationManager.GPS_PROVIDER);
        if (myLocation == null) {
            myLocation = locManager
                .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
        if (myLocation == null) {
            myLocation = locManager
                .getLastKnownLocation(LocationManager.PASSIVE_PROVIDER);
        }
    }

    if (myLocation != null) {
        String p = this.getParameters();
        p = p.substring(p.indexOf("Lat: "));
        p = p.replace("Lat: ", "").replace("Lng: ", "")
            .replace("Radius: ", "").replace(" miles", "")
    }
}

```

```

.replace("Fired: ", "").replace("\n", " ");
String[] params = p.split("[ ]+");
Location loc = new Location("dummy");
double r = 0;
if (params.length > 2) {
    System.out.println("Param 0 is " + params[0]);
    System.out.println("Param 1 is " + params[1]);
    System.out.println("Param 2 is " + params[2]);
    System.out.println("Param 3 is " + params[3]);
    loc.setLatitude(Double.parseDouble(params[0]));
    loc.setLongitude(Double.parseDouble(params[1]));
    r = (Double.parseDouble(params[2]));
}
System.out.println("r: " + r + " dist:"
    + myLocation.distanceTo(loc));
if (this.getCauseID() == 6) {
    // Arrivals
    if (r < myLocation.distanceTo(loc)) {
        result = false;
        params[3] = "0";
        StringBuilder myParams = new StringBuilder(
            this._parameters);
        myParams.setCharAt(myParams.length() - 2, '0');
        this.setParameters(myParams.toString());
        DatabaseHandler db = new DatabaseHandler(c);
        db.updateRCause(this, db.getRule(this._ruleID));
        db.close();
    } else if (params[3].equals("0")) {
        result = true;
        params[3] = "1";
        StringBuilder myParams = new StringBuilder(
            this._parameters);
        myParams.setCharAt(myParams.length() - 2, '1');
        this.setParameters(myParams.toString());
        DatabaseHandler db = new DatabaseHandler(c);
        db.updateRCause(this, db.getRule(this._ruleID));
        db.close();
    }
} else {
    // Departures
    if (r > myLocation.distanceTo(loc)) {
        result = false;
        params[3] = "0";
        StringBuilder myParams = new StringBuilder(
            this._parameters);
        myParams.setCharAt(myParams.length() - 2, '0');
        this.setParameters(myParams.toString());
        DatabaseHandler db = new DatabaseHandler(c);
        db.updateRCause(this, db.getRule(this._ruleID));
        db.close();
    } else if (params[3].equals("0")) {
        result = true;
        params[3] = "1";
        StringBuilder myParams = new StringBuilder(
            this._parameters);
    }
}

```

```
        myParams.setCharAt(myParams.length() - 2, '1');
        this.setParameters(myParams.toString());
        DatabaseHandler db = new DatabaseHandler(c);
        db.updateRCause(this, db.getRule(this._ruleID));
        db.close();
    }
}
} else {
    System.out.println("My Location was null");
    result = false;
}

break;
}

default:
    result = false;
    break;
}
}
return result;
}

/**
 * Types of rCauses currently stored on the app This can be removed if usin
g
 * strings for the switch in isTrue(c)
 *
 * @author CEandroid SMU
 */
public enum types {
    time, phoneCall, textMessage, ssid, wifiStatus, location
}
```

## Cause.java

```
package CEapi;

/**
 * The Cause class is the structure that stores information for a specific cause
 * in the back end No rule data is stored within this class
 *
 * @author CEandroid SMU
 */
public class Cause {
    /**
     * The unique identifier for this cause
     */
    int _id;

    /**
     * The _type refers to the grouping of causes within the back end The _name
     * stores the name of this specific cause The _description is a more user
     * friendly representation of the _name The _category refers to the grouping
     * of causes on the front end
     */
    String _type, _name, _description, _category;

    /**
     * Default Constructor
     */
    public Cause() {

    }

    /**
     * Defined Cause Constructor
     *
     * @param id
     * @param type
     * @param name
     * @param description
     * @param category
     */
    public Cause(int id, String type, String name, String description,
                String category) {
        this._id = id;
        this._type = type;
        this._name = name;
        this._description = description;
        this._category = category;
    }

    /**
     * Defined Cause Constructor without the id
     *
     * @param type
     * @param name
     */
}
```

```

* @param description
* @param category
*/
public Cause(String type, String name, String description, String category)
{
    this._type = type;
    this._name = name;
    this._description = description;
    this._category = category;
}

/**
 * Set _id
 *
 * @param id
 */
public void setID(int id) {
    this._id = id;
}

/**
 * Get _id
 *
 * @return this._id
 */
public int getID() {
    return this._id;
}

/**
 * Set _type
 *
 * @param type
 */
public void setType(String type) {
    this._type = type;
}

/**
 * Get _type
 *
 * @return this._type
 */
public String getType() {
    return this._type;
}

/**
 * Set _name
 *
 * @param name
 */
public void setName(String name) {
    this._name = name;
}

```

```

/**
 * Get _name
 *
 * @return this._name
 */
public String getName() {
    return this._name;
}

/**
 * Set _description
 *
 * @param description
 */
public void setDescription(String description) {
    this._description = description;
}

/**
 * Get _description
 *
 * @return this._description
 */
public String getDescription() {
    return this._description;
}

/**
 * Set _category
 *
 * @param category
 */
public void setCategory(String category) {
    this._category = category;
}

/**
 * Get _category
 *
 * @return this._category
 */
public String getCategory() {
    return this._category;
}

/**
 * Default Cause Evaluation returns true This is not used in normal rule
 * evaluation
 *
 * @return boolean true
 */
public boolean isTrue() {
    return true;
}
}

```

## DatabaseHandler.java

```
package com.example.ceandroid;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import CEapi.Cause;
import CEapi.Effect;
import CEapi.rCause;
import CEapi.rEffect;
import android.annotation.SuppressLint;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
import android.widget.Toast;

/**
 * Database class used for everything Database-related
 *
 * @author CEandroid SMU
 */
public class DatabaseHandler extends SQLiteOpenHelper {

    /**
     * Context variable for holding the context of the application
     */
    private Context context;

    /**
     * Database version number
     */
    private static final int DATABASE_VERSION = 1;

    /**
     * Database name
     */
    private static final String DATABASE_NAME = "CEdb";

    /**
     * The Location of the Database on the Device
     */
    @SuppressLint("SdCardPath")
    private static final String DB_FILEPATH = "/data/data/com.example.ceandroid
/databases/database.db";

    /**
     * Rules table name

```

```

*
 * @see Rule
 */
private static final String TABLE_RULES = "Rules";
/**/
* Causes table name
*
* @see Cause
*/
private static final String TABLE_CAUSES = "Causes";
/**/
* Effects table name
*
* @see Effect
*/
private static final String TABLE_EFFECTS = "Effects";
/**/
* rCauses table name
*
* @see rCause
*/
private static final String TABLE_R_CAUSES = "rCauses";
/**/
* rEffects table name
*
* @see rEffect
*/
private static final String TABLE_R_EFFECTS = "rEffects";

/**/
* Rule ID column
*
* @see Rule#_id
* @see rCause#_ruleID
* @see rEffect#_ruleID
*/
private static final String KEY_RULE_ID = "ruleID";
/**/
* Tree Data column
*
* @see Rule#_treeData
*/
private static final String KEY_TREE_DATA = "treeData";
/**/
* Rule name column
*
* @see Rule#_name
*/
private static final String KEY_RULE_NAME = "ruleName";
/**/
* Rule active column
*
* @see Rule#Active
*/
private static final String KEY_RULE_ACTIVE = "ruleActive";

```

```

/**
 * Cause ID column
 *
 * @see Cause#_id
 * @see rCause# causeID
 */
private static final String KEY_CAUSE_ID = "causeID";
/**
 * Cause type column
 *
 * @see Cause#_type
 */
private static final String KEY_CAUSE_TYPE = "causeType";
/**
 * Cause name column
 *
 * @see Cause#_name
 */
private static final String KEY_CAUSE_NAME = "causeName";
/**
 * Cause description column
 *
 * @see Cause#_description
 */
private static final String KEY_CAUSE_DESCRIPTION = "causeDescription";
/**
 * Cause category column
 *
 * @see Cause#_category
 */
private static final String KEY_CAUSE_CATEGORY = "causeCategory";

/**
 * Effect ID column
 *
 * @see Effect#_id
 * @see rEffect#_effectID
 */
private static final String KEY_EFFECT_ID = "effectID";
/**
 * Effect type column
 *
 * @see Effect#_type
 */
private static final String KEY_EFFECT_TYPE = "effectType";
/**
 * Effect name column
 *
 * @see Effect#_name
 */
private static final String KEY_EFFECT_NAME = "effectName";
/**
 * Effect description column
 *
 * @see Effect#_description
 */

```

```

private static final String KEY_EFFECT_DESCRIPTION = "effectDescription";
/**
 * Effect category column
 *
 * @see Effect# category
 */
private static final String KEY_EFFECT_CATEGORY = "effectCategory";

/**
 * rCause ID column
 *
 * @see rCause#_id
 */
private static final String KEY_R_CAUSE_ID = "rCauseID";
/**
 * rCause parameter column
 *
 * @see rCause#_parameters
 */
private static final String KEY_R_CAUSE_PARAMETER = "causeParameter";
/**
 * rCause rule ID column
 *
 * @see rCause#_ruleID
 */
private static final String KEY_R_CAUSE_RULE_ID = "rCauseRuleID";

/**
 * rEffect ID column
 *
 * @see rEffect#_id
 */
private static final String KEY_R_EFFECT_ID = "rEffectID";
/**
 * rEffect parameter column
 *
 * @see rEffect#_parameters
 */
private static final String KEY_R_EFFECT_PARAMETER = "effectParameter";
/**
 * rEffect rule ID column
 *
 * @see rEffect#_ruleID
 */
private static final String KEY_R_EFFECT_RULE_ID = "rEffectRuleID";

/**
 * Copies the database file at the specified location over the current
 * internal application database.
 */
public boolean importDatabase(String dbPath) throws IOException {

    // Close the SQLiteOpenHelper so it will commit the created empty
    // database to internal storage.
    close();
    File newDb = new File(dbPath);
}

```

```

File oldDb = new File(DB_FILEPATH);
if (newDb.exists()) {
    FileUtils.copyFile(new FileInputStream(newDb),
                      new FileOutputStream(oldDb));
    // Access the copied database so SQLiteHelper will cache it and mark
    // it as created.
    getWritableDatabase().close();
    return true;
}

return false;
}

/**
 * Constructor
 *
 * The constructor stores the application context, database name, and
 * database version.
 *
 * @param context
 *         global interface that allows access application resources
 */
public DatabaseHandler(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
    this.context = context;
}

/**
 * Creating Tables
 *
 * Runs all of the SQLite instructions to create all of the tables.
 *
 * @param db
 *         current installation of SQLiteDatabase
 */
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_RULES_TABLE = "CREATE TABLE " + TABLE_RULES + " (" +
        + KEY_RULE_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        + KEY_TREE_DATA + " TEXT, " + KEY_RULE_NAME + " TEXT, " +
        + KEY_RULE_ACTIVE + " BOOLEAN" + ")";
    String CREATE_CAUSES_TABLE = "CREATE TABLE " + TABLE_CAUSES + " (" +
        + KEY_CAUSE_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        + KEY_CAUSE_TYPE + " TEXT, " + KEY_CAUSE_NAME + " TEXT, " +
        + KEY_CAUSE_DESCRIPTION + " TEXT, " + KEY_CAUSE_CATEGORY +
        + " TEXT" + ";";
    String CREATE_EFFECTS_TABLE = "CREATE TABLE " + TABLE_EFFECTS + " (" +
        + KEY_EFFECT_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        + KEY_EFFECT_TYPE + " TEXT, " + KEY_EFFECT_NAME + " TEXT, " +
        + KEY_EFFECT_DESCRIPTION + " TEXT, " + KEY_EFFECT_CATEGORY +
        + " TEXT" + ";";
    String CREATE_R_CAUSES_TABLE = "CREATE TABLE " + TABLE_R_CAUSES + "(" +
        + KEY_R_CAUSE_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        + KEY_R_CAUSE_RULE_ID + " INT NOT NULL, " + KEY_CAUSE_ID +
        + " INT NOT NULL, " + KEY_R_CAUSE_PARAMETER + " TEXT, " +
        + "FOREIGN KEY(" + KEY_R_CAUSE_RULE_ID + ") REFERENCES "
}

```

```

        + TABLE_RULES + "(" + KEY_RULE_ID + ")" + ");";
String CREATE_R_EFFECTS_TABLE = "CREATE TABLE " + TABLE_R_EFFECTS + "("
        + KEY_R_EFFECT_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
        + KEY_R_EFFECT_RULE_ID + " INT NOT NULL, " + KEY_EFFECT_ID
        + " INT NOT NULL, " + KEY_R_EFFECT_PARAMETER + " TEXT, "
        + "FOREIGN KEY(" + KEY_R_EFFECT_RULE_ID + ") REFERENCES "
        + TABLE_RULES + "(" + KEY_RULE_ID + ")" + ");";
db.execSQL(CREATE_RULES_TABLE);
db.execSQL(CREATE_CAUSES_TABLE);
db.execSQL(CREATE_EFFECTS_TABLE);
db.execSQL(CREATE_R_CAUSES_TABLE);
db.execSQL(CREATE_R_EFFECTS_TABLE);
}

/**
 * Upgrading Database
 *
 * The method used to upgrade the database if needed.
 *
 * @param db
 *         current running SQLiteDatabase
 * @param oldVersion
 *         the old version database number
 * @param newVersion
 *         the new version database number
 */
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Drop older table if existed
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_RULES);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_R_CAUSES);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_R_EFFECTS);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CAUSES);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_EFFECTS);

    // Create tables again
    onCreate(db);
}

/**
 * Add a rule to the Database
 *
 * Adds a rule to the Database that is passed in.
 *
 * @param rule
 *         the rule to be added to the database
 * @see Rule
 */
public void addRule(Rule rule) {
    if (!ruleExists(rule)) {
        ContentValues ruleValues = new ContentValues();
        // ContentValues rCauseValues = new ContentValues();
        // ContentValues rEffectValues = new ContentValues();
        ruleValues.put(KEY_TREE_DATA, rule.getTreeData());
        ruleValues.put(KEY_RULE_NAME, rule.getName());
        ruleValues.put(KEY_RULE_ACTIVE, true);
    }
}

```

```

SQLiteDatabase db = this.getWritableDatabase();
db.insert(TABLE_RULES, null, ruleValues);
rule.setID(getAllRules().get(getRulesCount() - 1).getID());
System.out.println("Rule id is " + rule.getID());
db.close(); // Closing database connection
for (int i = 0; i < rule.getRCauses().size(); i++) {
    addRCause(rule.getRCauses().get(i), rule);
}
// rEffectValues.put(KEY_R_EFFECT_RULE_ID, getRulesCount() + 1);
for (int i = 0; i < rule.getREffects().size(); i++) {
    addREffect(rule.getREffects().get(i), rule);
}
else {
    Toast.makeText(null, "Rule already exists", Toast.LENGTH_LONG)
        .show();
}
}

new ArrayList<rCause>();
    SQLiteDatabase db = this.getReadableDatabase();
    String selectQuery = "SELECT " + TABLE_R_CAUSES + " ." + KEY_R_CAUSE_ID
        + ", " + TABLE_R_CAUSES + " ." + KEY_R_CAUSE_RULE_ID + ", "
        + TABLE_R_CAUSES + " ." + KEY_CAUSE_ID + ", " + TABLE_R_CAUSES
        + " ." + KEY_R_CAUSE_PARAMETER + ", " + TABLE_CAUSES + "."
        + KEY_CAUSE_TYPE + " FROM " + TABLE_R_CAUSES + " INNER JOIN "
        + TABLE_CAUSES + " ON " + TABLE_R_CAUSES + " ." + KEY_CAUSE_ID
        + " = " + TABLE_CAUSES + " ." + KEY_CAUSE_ID + " WHERE "
        + KEY_R_CAUSE_RULE_ID + " = " + rule.getID();
    Cursor cursor = db.rawQuery(selectQuery, null);
    if (cursor.moveToFirst()) {
        do {
            causes.add(new rCause(Integer.parseInt(cursor.getString(1)),
                Integer.parseInt(cursor.getString(2)), cursor
                    .getString(3), cursor.getString(4)));
        } while (cursor.moveToNext());
    }
    if (cursor != null) {
        cursor.close();
    }
    db.close();
}

return causes;
}
*/

```

```

* Get all of a rule's effects
*
* Gets all of a rule's effects that is passed in.
*
* @param rule
*      the rule to get the effects for
* @see Rule
* @see rEffect
* @return ArrayList<rEffect> of rEffect from given Rule
*/
public ArrayList<rEffect> getRuleEffects(Rule rule) {
    ArrayList<rEffect> effects = new ArrayList<rEffect>();
    SQLiteDatabase db = this.getReadableDatabase();
    String selectQuery = "SELECT " + TABLE_R_EFFECTS + "."
        + KEY_R_EFFECT_ID + ", " + TABLE_R_EFFECTS + "."
        + KEY_R_EFFECT_RULE_ID + ", " + TABLE_R_EFFECTS + "."
        + KEY_EFFECT_ID + ", " + TABLE_R_EFFECTS + "."
        + KEY_R_EFFECT_PARAMETER + ", " + TABLE_EFFECTS + "."
        + KEY_EFFECT_TYPE + " FROM " + TABLE_R_EFFECTS + " INNER JOIN "
        + TABLE_EFFECTS + " ON " + TABLE_R_EFFECTS + "."
        + KEY_EFFECT_ID + "=" + TABLE_EFFECTS + "."
        + KEY_EFFECT_ID
        + " WHERE " + KEY_R_EFFECT_RULE_ID + "=" + rule.getID();
    Cursor cursor = db.rawQuery(selectQuery, null);
    if (cursor.moveToFirst()) {
        do {
            effects.add(new rEffect(rule.getID(), Integer.parseInt(cursor
                .getString(2)), cursor.getString(3), cursor
                .getString(4)));
        } while (cursor.moveToNext());
    }
    if (cursor != null) {
        cursor.close();
    }
    db.close();

    return effects;
}

/**
* Get a rule
*
* Returns a rule given the rule ID.
*
* @param id
*      the ID of the rule
* @see Rule
* @return Rule with given ID or null if not found
*/
public Rule getRule(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(TABLE_RULES, new String[] { KEY_RULE_ID,
        KEY_TREE_DATA, KEY_RULE_NAME, KEY_RULE_ACTIVE }, KEY_RULE_ID
        + "=?", new String[] { String.valueOf(id) }, null, null, null,
        null);
    if (cursor != null)

```

```

        cursor.moveToFirst();

        Rule rule;
        if (cursor.getString(3).equals("0")) {
            rule = new Rule(cursor.getString(1), cursor.getString(2), false);
            rule.setID(Integer.parseInt(cursor.getString(0)));
        } else {
            rule = new Rule(cursor.getString(1), cursor.getString(2), true);
            rule.setID(Integer.parseInt(cursor.getString(0)));
        }

        db.close();
        rule.setRCauses(this.getRuleCauses(rule));
        rule.setREffects(this.getRuleEffects(rule));
        return rule;
    }

    /**
     * Get all rules
     *
     * Returns all of the Database's rules
     *
     * @see Rule
     * @return ArrayList<Rule> of all Rule in Database
     */
    public ArrayList<Rule> getAllRules() {
        ArrayList<Rule> ruleList = new ArrayList<Rule>();
        // Select All Query
        String selectQuery = "SELECT * FROM " + TABLE_RULES + " ORDER BY "
                + KEY_RULE_ID;

        SQLiteDatabase db = this.getWritableDatabase();
        Cursor cursor = db.rawQuery(selectQuery, null);

        // looping through all rows and adding to list
        if (cursor.moveToFirst()) {
            do {
                Rule rule = new Rule();
                rule.setID(Integer.parseInt(cursor.getString(0)));
                rule.setTreeData(cursor.getString(1));
                rule.setName(cursor.getString(2));
                if (cursor.getString(3).equals("0")) {
                    rule.setActive(false);
                } else {
                    rule.setActive(true);
                }
                // Adding rule to list
                ruleList.add(rule);
                rule.edited = false;
            } while (cursor.moveToNext());
        }
        close();
        // return rule list
        return ruleList;
    }
}

```

```

/**
 * Get all rules of a certain type
 *
 * Returns all of the Database's rules that have the type given.
 *
 * @param type
 *          the type of the rules you want returned
 * @see Rule
 * @see Cause#_type
 * @return ArrayList<Rule> of Rule that have the type given
 */
public ArrayList<Rule> getAllRulesByType(String type) {
    ArrayList<Rule> ruleList = new ArrayList<Rule>();

    SQLiteDatabase db = this.getWritableDatabase();
    String selectQuery = "SELECT * FROM " + TABLE_CAUSES + " WHERE "
        + KEY_CAUSE_TYPE + "=?";
    Cursor cursor = db.rawQuery(selectQuery, new String[] { type });

    // looping through all rows and adding to list of ints (cause id's)
    List<Integer> tempList = new ArrayList<Integer>();

    try {
        if (cursor.moveToFirst()) {
            do {
                tempList.add(Integer.parseInt(cursor.getString(0)));
            } while (cursor.moveToNext());
        }
    } finally {
        cursor.close();
    }

    List<Integer> tempList2 = new ArrayList<Integer>();
    for (int i = 0; i < tempList.size(); i++) {
        selectQuery = "SELECT * FROM " + TABLE_R_CAUSES + " WHERE "
            + KEY_CAUSE_ID + "=?";
        cursor = db.rawQuery(selectQuery,
            new String[] { "" + tempList.get(i) });
        try {
            if (cursor.moveToFirst()) {
                do {
                    if (!tempList2.contains(Integer.parseInt(cursor
                        .getString(1))))
                        tempList2
                            .add(Integer.parseInt(cursor.getString(1)));
                } while (cursor.moveToNext());
            }
        } finally {
            cursor.close();
        }
    }

    for (int i = 0; i < tempList2.size(); i++) {
        selectQuery = "SELECT * FROM " + TABLE_RULES + " WHERE "
            + KEY_RULE_ID + "=? AND " + KEY_RULE_ACTIVE;
        cursor = db.rawQuery(selectQuery,

```

```

        new String[] { "" + tempList2.get(i) });
    try {
        if (cursor.moveToFirst()) {
            Rule rule = new Rule();
            rule.setID(Integer.parseInt(cursor.getString(0)));
            rule.setTreeData(cursor.getString(1));
            rule.setName(cursor.getString(2));
            if (cursor.getString(3).equals("0")) {
                rule.setActive(false);
            } else {
                rule.setActive(true);
            }
            // Adding rule to list
            ruleList.add(rule);
            rule.edited = false;
        }
    } finally {
        cursor.close();
    }
}
close();
// return rule list
return ruleList;
}

/**
 * Get all rules of a certain category
 *
 * Returns all of the Database's rules that have the category given.
 *
 * @param category
 *          the type of the rules you want returned
 * @see Rule
 * @see Cause#_category
 * @return ArrayList<Rule> of Rule that have category given
 */
public ArrayList<Rule> getAllRulesByCategory(String category) {
    ArrayList<Rule> ruleList = new ArrayList<Rule>();

    SQLiteDatabase db = this.getWritableDatabase();
    String selectQuery = "SELECT * FROM " + TABLE_CAUSES + " WHERE "
        + KEY_CAUSE_CATEGORY + "=?"; // ORDER BY " + KEY_RULE_NAME;
    Cursor cursor = db.rawQuery(selectQuery, new String[] { category });

    // looping through all rows and adding to list of ints (cause id's)
    List<Integer> tempList = new ArrayList<Integer>();
    if (cursor.moveToFirst()) {
        do {
            tempList.add(Integer.parseInt(cursor.getString(0)));
        } while (cursor.moveToNext());
    }

    List<Integer> tempList2 = new ArrayList<Integer>();
    for (int i = 0; i < tempList.size(); i++) {
        selectQuery = "SELECT * FROM " + TABLE_R_CAUSES + " WHERE "
            + KEY_CAUSE_ID + "=?";
    }
}

```

```

cursor = db.rawQuery(selectQuery,
    new String[] { String.valueOf(tempList.get(i)) });
if (cursor.moveToFirst()) {
    do {
        tempList2.add(Integer.parseInt(cursor.getString(1)));
    } while (cursor.moveToNext());
}
}

for (int i = 0; i < tempList2.size(); i++) {
    selectQuery = "SELECT * FROM " + TABLE_RULES + " WHERE "
        + KEY_RULE_ID + "=?" + AND + KEY_RULE_ACTIVE;
    cursor = db.rawQuery(selectQuery,
        new String[] { String.valueOf(tempList2.get(i)) });
    if (cursor.moveToFirst()) {
        Rule rule = new Rule();
        rule.setID(Integer.parseInt(cursor.getString(0)));
        rule.setTreeData(cursor.getString(1));
        rule.setName(cursor.getString(2));
        if (cursor.getString(3).equals("0")) {
            rule.setActive(false);
        } else {
            rule.setActive(true);
        }
        // Adding rule to list
        ruleList.add(rule);
        rule.edited = false;
    }
}
close();
// return rule list
return ruleList;
}

/**
 * Get the number of Rules
 *
 * Returns the number of Rules in the Database.
 *
 * @see Rule
 * @return number of Rule objects in Database
 */
public int getRulesCount() {
    String countQuery = "SELECT * FROM " + TABLE_RULES;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(countQuery, null);
    int count = cursor.getCount();
    cursor.close();
    // return count
    return count;
}

/**
 * Get the number of rCauses
 *
 * Returns the number of rCauses in the Database.

```

```

*
* @see rCause
* @return returns number of rCause objects in Database
*/
public int getRCausesCount() {
    String countQuery = "SELECT * FROM " + TABLE_R_CAUSES;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(countQuery, null);
    int count = cursor.getCount();
    cursor.close();
    // return count
    return count;
}

/**
* Get the number of rEffects
*
* Returns the number of rEffects in the Database.
*
* @see rEffect
* @return returns number of rEffect objects in Database
*/
public int getREffectsCount() {
    String countQuery = "SELECT * FROM " + TABLE_R_EFFECTS;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(countQuery, null);
    int count = cursor.getCount();
    cursor.close();
    // return count
    return count;
}

/**
* Update an rCause
*
* Updates an rCause in the Database that correlates to the rCause and rule
* given. Returns the number of rCauses updated in the Database.
*
* @param rc
*          the rCause to be updated
* @param r
*          the rule that correlates to the rCause
* @see rCause
* @see Rule
* @return returns number of rCauses updated
*/
public int updateRCause(rCause rc, Rule r) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues rCauseValues = new ContentValues();
    rCauseValues.put(KEY_R_CAUSE_RULE_ID, r.getID());
    rCauseValues.put(KEY_CAUSE_ID, rc.getCauseID());
    rCauseValues.put(KEY_R_CAUSE_PARAMETER, rc.getParameters());
    return db.update(TABLE_R_CAUSES, rCauseValues, KEY_R_CAUSE_ID + " = ?",
        new String[] { String.valueOf(rc.getID()) });
}

```

```

/**
 * Update an rEffect
 *
 * Updates an rEffect in the Database that correlates to the rEffect and
 * rule given. Returns the number of rEffect updated in the Database.
 *
 * @param re
 *          the rEffect to be updated
 * @param r
 *          the rule that correlates to the rEffect
 * @see rEffect
 * @see Rule
 * @return returns the number of rEffect updated
 */
public int updateREffect(rEffect re, Rule r) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues rEffectValues = new ContentValues();
    rEffectValues.put(KEY_R_EFFECT_RULE_ID, r.getID());
    rEffectValues.put(KEY_EFFECT_ID, re.getEffectID());
    rEffectValues.put(KEY_R_EFFECT_PARAMETER, re.getParameters());
    return db.update(TABLE_R_EFFECTS, rEffectValues, KEY_R_EFFECT_ID
        + " = ?", new String[] { String.valueOf(re.getID()) });
}

/**
 * Update a Rule
 *
 * Updates a Rule in the Database. Returns the number of Rules updated in
 * the Database.
 *
 * @param rule
 *          the Rule to be updated
 * @see Rule
 * @return returns the number of Rules updated
 */
public int updateRule(Rule rule) {
    int valuesChanged = 0;

    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();

    values.put(KEY_TREE_DATA, rule.getTreeData());

    // updating row
    if (db.update(TABLE_RULES, values, KEY_RULE_ID + " = ?",
        new String[] { String.valueOf(rule.getID()) }) < 1) {
        addRule(rule);
        valuesChanged = 1;
    } else {
        ArrayList<rCause> causes = rule.getRCauses();
        for (int i = 0; i < causes.size(); i++) {
            if (updateRCause(causes.get(i), rule) < 1) {
                addRCause(causes.get(i), rule);
            }
        }
        ArrayList<rEffect> effects = rule.getREffects();
}

```

```

        for (int i = 0; i < effects.size(); i++) {
            if (updateREffect(effects.get(i), rule) < 1) {
                addREffect(effects.get(i), rule);
            }
        }
    }
    db.close();
    return valuesChanged;
}

/**
 * Update a Rule's name
 *
 * Updates an Rule's name in the Database. Returns the number of Rules
 * updated in the Database.
 *
 * @param ruleID
 *          the ID of the Rule to be updated
 * @param name
 *          the new name of the Rule
 * @see Rule
 * @return returns the number of Rule updated
 */
public int updateName(int ruleID, String name) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues ruleValues = new ContentValues();
    ruleValues.put(KEY_RULE_NAME, name);
    return db.update(TABLE_RULES, ruleValues, KEY_RULE_ID + "=" + ruleID,
                    null);
}

/**
 * Update a Rule's active
 *
 * Updates whether a Rule's active in the Database. Returns the number of
 * Rules updated in the Database.
 *
 * @param ruleID
 *          the ID of the Rule to be updated
 * @param isActive
 *          boolean of whether the rule is now active or not
 * @see Rule
 * @return returns the number of Rules updated
 */
public int updateActive(int ruleID, boolean isActive) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues ruleValues = new ContentValues();
    ruleValues.put(KEY_RULE_ACTIVE, isActive);
    return db.update(TABLE_RULES, ruleValues, KEY_RULE_ID + "=" + ruleID,
                    null);
}

/**
 * Delete a Rule
 *
 * Deletes a Rule in the Database.

```

```

*
* @param rule
*           the Rule to be deleted
* @see Rule
*/
public void deleteRule(Rule rule) {
    for (int i = rule.getRCauses().size() - 1; i >= 0; i--) {
        deleteRCause(rule.getRCauses().get(i));
    }
    for (int i = rule.getREffects().size() - 1; i >= 0; i--) {
        deleteREffect(rule.getREffects().get(i));
    }
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_RULES, KEY_RULE_ID + " = ?",
              new String[] { String.valueOf(rule.getID()) });
    db.close();
}

/**
 * Delete an rCause
 *
 * Deletes an rCause in the Database.
 *
 * @param rc
 *           the rCause to be deleted
 * @see rCause
*/
public void deleteRCause(rCause rc) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_R_CAUSES, KEY_R_CAUSE_ID + " = ?",
              new String[] { String.valueOf(rc.getID()) });
    db.close();
}

/**
 * Delete an rEffect
 *
 * Deletes an rEffect in the Database.
 *
 * @param re
 *           the rEffect to be deleted
 * @see rEffect
*/
public void deleteREffect(rEffect re) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_R_EFFECTS, KEY_R_EFFECT_ID + " = ?",
              new String[] { String.valueOf(re.getID()) });
    db.close();
}

/**
 * Create a Cause
 *
 * Adds a Cause to the Database.
 *
 * @param cause

```

```

        *           the Cause to be created
        * @see Cause
        */
public void addCause(Cause cause) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues causeValues = new ContentValues();
    causeValues.put(KEY_CAUSE_TYPE, cause.getType());
    causeValues.put(KEY_CAUSE_NAME, cause.getName());
    causeValues.put(KEY_CAUSE_DESCRIPTION, cause.getDescription());
    causeValues.put(KEY_CAUSE_CATEGORY, cause.getCategory());
    db.insert(TABLE_CAUSES, null, causeValues);
    db.close();
}

/**
 * Rule exists checker
 *
 * Determines whether a Rule exists or not in the Database.
 *
 * @param rule
 *           the Rule to be checked to see if it exists
 * @see Rule
 * @returns true = returns whether a rule exists or not
 */
public boolean ruleExists(Rule rule) {
    boolean exists = true;
    for (int rules = 1; (rules <= getRulesCount()) && exists; rules++) {
        if (getRule(rules).getRCauses().size() != rule.getRCauses().size()
            || getRule(rules).getREffects().size() != rule
                .getREffects().size()) {
            exists = false;
        }
        for (int rcs = 0; (rcs < getRule(rules).getRCauses().size())
            && exists; rcs++) {
            if (!rCauseExists(getRule(rules).getRCauses().get(rcs), rule)) {
                exists = false;
            }
        }
        for (int res = 0; (res < getRule(rules).getREffects().size())
            && exists; res++) {
            if (!rEffectExists(getRule(rules).getREffects().get(res), rule)) {
                exists = false;
            }
        }
    }
    if (getRulesCount() == 0) {
        exists = false;
    }
    return exists;
}

/**
 * rCause exists checker
 *
 * Determines whether a rCause exists or not in the Database. Needs the Rule
e

```

```

* since the rCause is not in the Database yet.
*
* @param rc
*      the rCause to be checked to see if it exists
* @param rule
*      the Rule that the rCause may exist inside of
* @see rCause
* @see Rule
* @return returns whether an rCause exists or not
*/
public boolean rCauseExists(rCause rc, Rule rule) {
    SQLiteDatabase db = this.getWritableDatabase();
    String selectQuery = "SELECT * FROM " + TABLE_R_CAUSES + " WHERE "
        + KEY_R_CAUSE_RULE_ID + "=? AND " + KEY_CAUSE_ID + "=? AND "
        + KEY_R_CAUSE_PARAMETER + "=?";
    Cursor cursor = db.rawQuery(
        selectQuery,
        new String[] { "" + rule.getID(), "" + rc.getCauseID(),
            rc.getParameters() });
    if (cursor.moveToFirst()) {
        db.close();
        return true;
    }
    db.close();
    return false;
}

/**
* rEffect exists checker
*
* Determines whether a rEffect exists or not in the Database. Needs the
* Rule since the rEffect is not in the Database yet.
*
* @param re
*      the rEffect to be checked to see if it exists
* @param rule
*      the Rule that the rEffect may exist inside of
* @see rEffect
* @see Rule
* @return returns whether an rEffect exists or not
*/
public boolean rEffectExists(rEffect re, Rule rule) {
    SQLiteDatabase db = this.getWritableDatabase();
    String selectQuery = "SELECT * FROM " + TABLE_R_EFFECTS + " WHERE "
        + KEY_R_EFFECT_RULE_ID + "=? AND " + KEY_EFFECT_ID + "=? AND "
        + KEY_R_EFFECT_PARAMETER + "=?";
    Cursor cursor = db.rawQuery(
        selectQuery,
        new String[] { "" + rule.getID(), "" + re.getEffectID(),
            re.getParameters() });
    if (cursor.moveToFirst()) {
        db.close();
        return true;
    }
    db.close();
    return false;
}

```

```

}

/**
 * Create an rCause
 *
 * Checks to see if a rCause already exists from
 * {@link #rCauseExists(rCause, Rule)} and if returns false will add the
 * rCause to the Database.
 *
 * @param rc
 *          the rCause to be added to the Database
 * @param rule
 *          the Rule that the rCause is inside of
 * @see rCause
 * @see Rule
 * @see #rCauseExists(rCause, Rule)
 * @return returns whether the rCause was added or not
 */
public boolean addRCause(rCause rc, Rule rule) {
    boolean b = rCauseExists(rc, rule);
    if (!b) {
        ContentValues rCauseValues = new ContentValues();
        rCauseValues.put(KEY_R_CAUSE_RULE_ID, rule.getID());
        rCauseValues.put(KEY_CAUSE_ID, rc.getCauseID());
        rCauseValues.put(KEY_R_CAUSE_PARAMETER, rc.getParameters());
        SQLiteDatabase db = this.getWritableDatabase();
        db.insert(TABLE_R_CAUSES, null, rCauseValues);
        db.close(); // Closing database connection
    } else {
        Toast.makeText(this.context, "You cannot have duplicate causes!",
                      Toast.LENGTH_SHORT).show();
    }
    return b;
}

/**
 * Create an rEffect
 *
 * Checks to see if a rEffect already exists from
 * {@link #rEffectExists(rEffect, Rule)} and if returns false will add the
 * rEffect to the Database.
 *
 * @param re
 *          the rEffect to be added to the Database
 * @param rule
 *          the Rule that the rEffect is inside of
 * @see rEffect
 * @see Rule
 * @see #rEffectExists(rEffect, Rule)
 * @return returns whether the rCause was added or not
 */
public void addREffect(rEffect re, Rule rule) {
    if (!rEffectExists(re, rule)) {
        ContentValues rEffectValues = new ContentValues();
        rEffectValues.put(KEY_R_EFFECT_RULE_ID, rule.getID());
        rEffectValues.put(KEY_EFFECT_ID, re.getEffectID());
    }
}

```

```

rEffectValues.put(KEY_R_EFFECT_PARAMETER, re.getParameters());
SQLiteDatabase db = this.getWritableDatabase();
db.insert(TABLE_R_EFFECTS, null, rEffectValues);
db.close(); // Closing database connection
} else {
    Toast.makeText(this.context, "You cannot have duplicate effects!",
        Toast.LENGTH_SHORT).show();
}
}

/**
 * Create an Effect
 *
 * Creates an Effect in the Database.
 *
 * @param effect
 *         the Effect to be added
 * @see Effect
 */
public void addEffect(Effect effect) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues effectValues = new ContentValues();
    effectValues.put(KEY_EFFECT_TYPE, effect.getType());
    effectValues.put(KEY_EFFECT_NAME, effect.getName());
    effectValues.put(KEY_EFFECT_DESCRIPTION, effect.getDescription());
    effectValues.put(KEY_EFFECT_CATEGORY, effect.getCategory());
    db.insert(TABLE_EFFECTS, null, effectValues);
    db.close();
}

/**
 * Get all Causes
 *
 * Finds and returns all Cause objects in the Database.
 *
 * @see Cause
 * @return List full of all Cause objects in Database
 */
public List<Cause> getAllCauses() {
    List<Cause> causeList = new ArrayList<Cause>();
    String selectQuery = "SELECT * FROM " + TABLE_CAUSES + " ORDER BY "
        + KEY_CAUSE_NAME;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Cause cause = new Cause();
            cause.setID(cursor.getColumnIndex("_id"));
            cause.setType(cursor.getString(1));
            cause.setName(cursor.getString(2));
            cause.setDescription(cursor.getString(3));
            cause.setCategory(cursor.getString(4));
            causeList.add(cause);
        }
    }
}

```

```

        } while (cursor.moveToNext());
    }
    close();
    return causeList;
}

/**
 * Get all Causes with given category
 *
 * Finds and returns all Cause objects in the Database with given category.
 *
 * @see Cause
 * @see Cause#_category
 * @return List full of all Cause objects in Database with given category
 */
public List<Cause> getAllCauses(String category) {
    List<Cause> causeList = new ArrayList<Cause>();

    SQLiteDatabase db = this.getWritableDatabase();
    String selectQuery = "SELECT * FROM " + TABLE_CAUSES + " WHERE "
        + KEY_CAUSE_CATEGORY + " =? ORDER BY " + KEY_CAUSE_NAME;
    Cursor cursor = db.rawQuery(selectQuery, new String[] { category });

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Cause cause = new Cause();
            cause.setID(cursor.getColumnIndex("_id"));
            cause.setType(cursor.getString(1));
            cause.setName(cursor.getString(2));
            cause.setDescription(cursor.getString(3));
            cause.setCategory(cursor.getString(4));
            causeList.add(cause);
        } while (cursor.moveToNext());
    }
    close();
    return causeList;
}

/**
 * Get all Effects
 *
 * Finds and returns all Effect objects in the Database.
 *
 * @see Effect
 * @return List full of all Effect objects in Database
 */
public List<Effect> getAllEffects() {
    List<Effect> effectList = new ArrayList<Effect>();
    String selectQuery = "SELECT * FROM " + TABLE_EFFECTS + " ORDER BY "
        + KEY_EFFECT_NAME;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list

```

```

    if (cursor.moveToFirst()) {
        do {
            Effect effect = new Effect();
            effect.setID(cursor.getColumnIndex("_id"));
            effect.setType(cursor.getString(1));
            effect.setName(cursor.getString(2));
            effect.setDescription(cursor.getString(3));
            effect.setCategory(cursor.getString(4));
            effectList.add(effect);
        } while (cursor.moveToNext());
    }
    close();
    return effectList;
}

/**
 * Get all Effects with given category
 *
 * Finds and returns all Effect objects in the Database with given category
 *
 * @see Effect
 * @see Effect#_category
 * @return List<Effect> of Effect objects in Database with given category
 */
public List<Effect> getAllEffects(String category) {
    List<Effect> effectList = new ArrayList<Effect>();

    SQLiteDatabase db = this.getWritableDatabase();
    String selectQuery = "SELECT * FROM " + TABLE_EFFECTS + " WHERE "
        + KEY_EFFECT_CATEGORY + " =? ORDER BY " + KEY_EFFECT_NAME;
    Cursor cursor = db.rawQuery(selectQuery, new String[] { category });

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Effect effect = new Effect();
            effect.setID(cursor.getColumnIndex("_id"));
            effect.setType(cursor.getString(1));
            effect.setName(cursor.getString(2));
            effect.setDescription(cursor.getString(3));
            effect.setCategory(cursor.getString(4));
            effectList.add(effect);
        } while (cursor.moveToNext());
    }
    close();
    return effectList;
}

/**
 * Get rCause with given ID
 *
 * Finds and returns the rCause with the given ID.
 *
 * @see rCause
 * @return rCause with given ID or null if not found

```

```

*/
public rCause getRCauseByID(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    String selectQuery = "SELECT " + TABLE_R_CAUSES + " ." + KEY_R_CAUSE_ID
        + ", " + TABLE_R_CAUSES + " ." + KEY_R_CAUSE_RULE_ID + ", "
        + TABLE_R_CAUSES + " ." + KEY_CAUSE_ID + ", " + TABLE_R_CAUSES
        + " ." + KEY_R_CAUSE_PARAMETER + ", " + TABLE_CAUSES + " ."
        + KEY_CAUSE_TYPE + " FROM " + TABLE_R_CAUSES + " INNER JOIN "
        + TABLE_CAUSES + " ON " + TABLE_R_CAUSES + " ." + KEY_CAUSE_ID
        + " = " + TABLE_CAUSES + " ." + KEY_CAUSE_ID + " WHERE "
        + KEY_R_CAUSE_ID + " = " + id;
    Cursor cursor = db.rawQuery(selectQuery, null);
    rCause cause = null;
    if (cursor.moveToFirst()) {
        cause = new rCause(Integer.parseInt(cursor.getString(0)),
            Integer.parseInt(cursor.getString(1)),
            Integer.parseInt(cursor.getString(2)), cursor.getString(3),
            cursor.getString(4));
    }
    close();
    // return contact
    return cause;
}

/**
 * Get rEffect with given ID
 *
 * Finds and returns the rEffect with the given ID.
 *
 * @see rEffect
 * @return rEffect with given ID or null if not found
 */
public rEffect getREffectByID(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    String selectQuery = "SELECT " + TABLE_R_EFFECTS + " ."
        + KEY_R_EFFECT_ID + ", " + TABLE_R_EFFECTS + " ."
        + KEY_R_EFFECT_RULE_ID + ", " + TABLE_R_EFFECTS + " ."
        + KEY_EFFECT_ID + ", " + TABLE_R_EFFECTS + " ."
        + KEY_R_EFFECT_PARAMETER + ", " + TABLE_EFFECTS + " ."
        + KEY_EFFECT_TYPE + " FROM " + TABLE_R_EFFECTS + " INNER JOIN "
        + TABLE_EFFECTS + " ON " + TABLE_R_EFFECTS + " ."
        + KEY_EFFECT_ID + " = " + TABLE_EFFECTS + " ." + KEY_EFFECT_ID
        + " WHERE " + KEY_R_EFFECT_ID + " = " + id;
    Cursor cursor = db.rawQuery(selectQuery, null);
    rEffect effect = null;
    if (cursor.moveToFirst()) {
        effect = new rEffect(Integer.parseInt(cursor.getString(0)),
            Integer.parseInt(cursor.getString(1)),
            Integer.parseInt(cursor.getString(2)), cursor.getString(3),
            cursor.getString(4));
    }
    close();
    // return contact
    return effect;
}

```

```

}

/**
 * Get rCauses with given Rule ID
 *
 * Finds and returns all rCauses with the given Rule ID.
 *
 * @see Rule
 * @see rCause
 * @return ArrayList<rCause> of rCause with given rule ID
 */
public ArrayList<rCause> getAllRCauses(int ruleID) {
    ArrayList<rCause> rCauseList = new ArrayList<rCause>();

    SQLiteDatabase db = this.getWritableDatabase();
    String selectQuery = "SELECT " + TABLE_R_CAUSES + " ." + KEY_R_CAUSE_ID
        + ", " + TABLE_R_CAUSES + " ." + KEY_R_CAUSE_RULE_ID + ", "
        + TABLE_R_CAUSES + " ." + KEY_CAUSE_ID + ", " + TABLE_R_CAUSES
        + " ." + KEY_R_CAUSE_PARAMETER + ", " + TABLE_CAUSES + "."
        + KEY_CAUSE_TYPE + ", " + TABLE_CAUSES + " ." + KEY_CAUSE_NAME
        + " FROM " + TABLE_R_CAUSES + " INNER JOIN " + TABLE_CAUSES
        + " ON " + TABLE_R_CAUSES + " ." + KEY_CAUSE_ID + " = "
        + TABLE_CAUSES + " ." + KEY_CAUSE_ID + " WHERE "
        + TABLE_R_CAUSES + " ." + KEY_R_CAUSE_RULE_ID + " = " + ruleID;
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            rCause cause = new rCause();
            cause.setID(Integer.parseInt(cursor.getString(0)));
            cause.setRuleID(Integer.parseInt(cursor.getString(1)));
            cause.setCauseID(Integer.parseInt(cursor.getString(2)));
            cause.setParameters(cursor.getString(3));
            cause.setType(cursor.getString(4));
            cause.setName(cursor.getString(5));
            rCauseList.add(cause);
        } while (cursor.moveToNext());
    }
    close();
    return rCauseList;
}

/**
 * Get rEffects with given Rule ID
 *
 * Finds and returns all rEffects with the given Rule ID.
 *
 * @see Rule
 * @see rEffect
 * @return ArrayList<rEffect> of rEffect with given rule ID
 */
public ArrayList<rEffect> getAllREffects(int ruleID) {
    ArrayList<rEffect> rEffectList = new ArrayList<rEffect>();

    SQLiteDatabase db = this.getWritableDatabase();

```

```

String selectQuery = "SELECT " + TABLE_R_EFFECTS + "."
+ KEY_R_EFFECT_ID + ", " + TABLE_R_EFFECTS + "."
+ KEY_R_EFFECT_RULE_ID + ", " + TABLE_R_EFFECTS + "."
+ KEY_EFFECT_ID + ", " + TABLE_R_EFFECTS + "."
+ KEY_R_EFFECT_PARAMETER + ", " + TABLE_EFFECTS + "."
+ KEY_EFFECT_TYPE + ", " + TABLE_EFFECTS + "."
+ KEY_EFFECT_NAME + " FROM " + TABLE_R_EFFECTS + " INNER JOIN "
+ TABLE_EFFECTS + " ON " + TABLE_R_EFFECTS + "."
+ KEY_EFFECT_ID + "=" + TABLE_EFFECTS + ". " + KEY_EFFECT_ID
+ " WHERE " + TABLE_R_EFFECTS + ". " + KEY_R_EFFECT_RULE_ID
+ " = " + ruleID;
Cursor cursor = db.rawQuery(selectQuery, null);

// looping through all rows and adding to list
if (cursor.moveToFirst()) {
    do {
        rEffect effect = new rEffect();
        effect.setID(Integer.parseInt(cursor.getString(0)));
        effect.setRuleID(Integer.parseInt(cursor.getString(1)));
        effect.setEffectID(Integer.parseInt(cursor.getString(2)));
        effect.setParameters(cursor.getString(3));
        effect.setType(cursor.getString(4));
        effect.setName(cursor.getString(5));
        rEffectList.add(effect);
    } while (cursor.moveToNext());
}
close();
return rEffectList;
}

/**
 * Get all Cause categories
 *
 * Finds and returns all categories found within any Cause.
 *
 * @see Cause#_category
 * @return ArrayList<String> of categories
 */
public ArrayList<String> getAllCauseCategories() {
    ArrayList<String> list = new ArrayList<String>();
    String selectQuery = "SELECT * FROM " + TABLE_CAUSES + " ORDER BY "
        + KEY_CAUSE_NAME;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            if (!list.contains(cursor.getString(4)))
                list.add(cursor.getString(4));
        } while (cursor.moveToNext());
    }
    close();
    return list;
}

```

```

/**
 * Get all Effect categories
 *
 * Finds and returns all categories found within any Effect.
 *
 * @see Effect#_category
 * @return ArrayList<String> of categories
 */
public ArrayList<String> getAllEffectCategories() {
    ArrayList<String> list = new ArrayList<String>();
    String selectQuery = "SELECT * FROM " + TABLE_EFFECTS + " ORDER BY "
        + KEY_EFFECT_NAME;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            if (!list.contains(cursor.getString(4)))
                list.add(cursor.getString(4));
        } while (cursor.moveToNext());
    }
    close();
    return list;
}

/**
 * Get all categories
 *
 * Finds and returns all categories found within any Cause or Effect.
 *
 * @see Cause#_category
 * @see Effect#_category
 * @return ArrayList<String> of categories
 */
public ArrayList<String> getAllCategories() {
    ArrayList<String> list = new ArrayList<String>();
    list.addAll(getAllCauseCategories());
    list.addAll(getAllEffectCategories());
    return list;
}

/**
 * Get all Rule categories
 *
 * Finds and returns all categories found within any Rule.
 *
 * @see Cause#_category
 * @see Effect#_category
 * @return ArrayList<String> of categories
 */
public ArrayList<String> getAllRuleCategories() {
    ArrayList<String> list = new ArrayList<String>();
    String selectQuery = "SELECT " + TABLE_CAUSES + ".";

```

```

        + KEY_COUSE_CATEGORY + " FROM " + TABLE_R_COUSES
        + " INNER JOIN " + TABLE_COUSES + " ON " + TABLE_R_COUSES + "."
        + KEY_COUSE_ID + "=" + TABLE_COUSES + "." + KEY_COUSE_ID;

SQLiteDatabase db = this.getWritableDatabase();
Cursor cursor = db.rawQuery(selectQuery, null);

// looping through all rows and adding to list
if (cursor.moveToFirst()) {
    do {
        if (!list.contains(cursor.getString(0)))
            list.add(cursor.getString(0));
    } while (cursor.moveToNext());
}

selectQuery = "SELECT " + KEY_EFFECT_CATEGORY + " FROM "
        + TABLE_R_EFFECTS + " INNER JOIN " + TABLE_EFFECTS + " ON "
        + TABLE_R_EFFECTS + "." + KEY_EFFECT_ID + "=" + TABLE_EFFECTS
        + "." + KEY_EFFECT_ID;
cursor = db.rawQuery(selectQuery, null);
if (cursor.moveToFirst()) {
    do {
        if (!list.contains(cursor.getString(0)))
            list.add(cursor.getString(0));
    } while (cursor.moveToNext());
}
close();
return list;
}

/**
 * Print all Tables
 *
 * Prints all of the Tables (all rows and columns) in the Database.
 */
public void seeTables() {
    String selectQuery = "SELECT * FROM " + TABLE_RULES;
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);
    Log.d("rules table", "");
    if (cursor.moveToFirst()) {
        do {
            for (int i = 0; i < cursor.getColumnNames().length; i++)
                Log.d("see table",
                    cursor.getColumnName(i) + " : "
                    + cursor.getString(i));

        } while (cursor.moveToNext());
    }
    Log.d("Causes table", "");
    selectQuery = "SELECT * FROM " + TABLE_COUSES;
    cursor = db.rawQuery(selectQuery, null);
    if (cursor.moveToFirst()) {
        do {
            for (int i = 0; i < cursor.getColumnNames().length; i++)
                Log.d("see table",

```

```

        cursor.getColumnName(i) + " : "
        + cursor.getString(i));

    } while (cursor.moveToNext());
}

Log.d("Effects table", "");
selectQuery = "SELECT * FROM " + TABLE_EFFECTS;
cursor = db.rawQuery(selectQuery, null);
if (cursor.moveToFirst()) {
    do {
        for (int i = 0; i < cursor.getColumnNames().length; i++)
            Log.d("see table",
                  cursor.getColumnName(i) + " : "
                  + cursor.getString(i));

    } while (cursor.moveToNext());
}
Log.d("rCauses table", "");
selectQuery = "SELECT * FROM " + TABLE_R_CAUSES;
cursor = db.rawQuery(selectQuery, null);
if (cursor.moveToFirst()) {
    do {
        for (int i = 0; i < cursor.getColumnNames().length; i++)
            Log.d("see table",
                  cursor.getColumnName(i) + " : "
                  + cursor.getString(i));

    } while (cursor.moveToNext());
}
Log.d("rEffects table", "");
selectQuery = "SELECT * FROM " + TABLE_R_EFFECTS;
cursor = db.rawQuery(selectQuery, null);
if (cursor.moveToFirst()) {
    do {
        for (int i = 0; i < cursor.getColumnNames().length; i++)
            Log.d("see table",
                  cursor.getColumnName(i) + " : "
                  + cursor.getString(i));

    } while (cursor.moveToNext());
}
close();
}

/**
 * Duplicate rCause checker
 *
 * Checks the Database for rCause duplicates.
 *
 * @see rCause
 * @return returns <code>true</code> if there are duplicates
 */
public boolean duplicateRCauses(Rule rule) {
    String selectQuery = "SELECT * FROM " + TABLE_R_CAUSES + " WHERE "
        + KEY_R_CAUSE_RULE_ID + "=? AND " + KEY_CAUSE_ID + "=? AND "
        + KEY_R_CAUSE_PARAMETER + "=?";

```

```

SQLiteDatabase db = this.getReadableDatabase();
Cursor cursor = null;
boolean duplicate = false;
for (int i = 0; (i < rule.getRCauses().size() - 1) && !duplicate; i++) {
    cursor = db.rawQuery(selectQuery, new String[] { "" + rule.getID(),
        "" + rule.getRCauses().get(i).getCauseID(),
        "" + rule.getRCauses().get(i).getParameters() });
    if (cursor.getCount() > 1)
        duplicate = true;
}
if (cursor != null) {
    cursor.close();
}
db.close();
return duplicate;
}

/**
 * Duplicate rEffect checker
 *
 * Checks the Database for rEffect duplicates.
 *
 * @see rEffect
 * @return returns <code>true</code> if there are duplicates
 */
public boolean duplicateREffects(Rule rule) {
    String selectQuery = "SELECT * FROM " + TABLE_R_EFFECTS + " WHERE "
        + KEY_R_EFFECT_RULE_ID + "=?" + AND + KEY_EFFECT_ID + "=?" + AND +
        KEY_R_EFFECT_PARAMETER + "=?" ;
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = null;
    boolean duplicate = false;
    for (int i = 0; (i < rule.getREffects().size() - 1) && !duplicate; i++) {
        cursor = db.rawQuery(selectQuery, new String[] { "" + rule.getID(),
            "" + rule.getREffects().get(i).getEffectID(),
            "" + rule.getREffects().get(i).getParameters() });
        if (cursor.getCount() > 1)
            duplicate = true;
    }
    if (cursor != null) {
        cursor.close();
    }
    db.close();
    return duplicate;
}
}

```

## Help.java

```

package com.example.ceandroid;

import android.net.Uri;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

```

```

import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.Toast;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;

/**
 * This Help Class provides an image link to our private tutorial on Wordpress
 *
 * @author CEandroid SMU
 */
public class Help extends Activity {
    /**
     * Creates the Help Page and adds the click listener
     *
     * @see android.app.Activity#onCreate(android.os.Bundle)
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help);

        // Turn on "up" navigation
        getActionBar().setDisplayHomeAsUpEnabled(true);

        ImageButton hLink = (ImageButton) findViewById(R.id.helpLink);
        hLink.setOnClickListener(new OnClickListener() {
            /**
             * Opens the Wordpress Website
             *
             * @see android.view.View.OnClickListener#onClick(android.view.View)
             */
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_VIEW, Uri
                        .parse("http://ceandroid.wordpress.com/"));
                startActivity(i);
            }
        });
    }

    /**
     * Checks the current rule, clears if not null
     *
     * @see android.app.Activity#onResume()
     */
    @Override
    public void onResume() {
        super.onResume();
        CEapp app = (CEapp) this.getApplication();
        if (app.currentRule != null) {
            if (app.currentRule.getRCauses().isEmpty()
                    && app.currentRule.getREffects().isEmpty()) {
                DatabaseHandler db = new DatabaseHandler(this);

```

```

        db.deleteRule(app.currentRule);
        Toast.makeText(this, "Incomplete rule not saved.",
                      Toast.LENGTH_LONG).show();
        app.currentRule = null;
        db.close();
    }
}

/**
 * Help Menu Creation
 *
 * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
 */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_help, menu);
    return true;
}

/**
 * Help Menu Actions
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case android.R.id.home: {
            // This is called when the Home (Up) button is pressed
            // in the Action Bar.
            Intent parentActivityIntent = new Intent(this, MainActivity.class);
            parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                | Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(parentActivityIntent);
            finish();
            return true;
        }
        case R.id.menu_new_rule: {
            // New Rule
            CEapp app = (CEapp) this.getApplication();
            app.currentRule = new Rule();
            /*
             * DatabaseHandler db = new DatabaseHandler(this);
             * app.currentRule.setID(db.getRulesCount() + 1); db.close();
             */
            Intent myIntent = new Intent(this, EditRule.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
            return true;
        }
        case R.id.menu_share: {
            PackageManager p = this.getPackageManager();
            if (!p.hasSystemFeature("android.hardware.nfc")) {

```

```
        Toast.makeText(this, "NFC is not available on this device.",  
                    Toast.LENGTH_LONG).show();  
    } else {  
        NfcAdapter na = NfcAdapter.getDefaultAdapter(this);  
        if (na.isEnabled()) {  
            Intent myIntent = new Intent(this, ShareList.class)  
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);  
            startActivity(myIntent);  
        } else {  
            Toast.makeText(this,  
                        "Please enable NFC to use this feature.",  
                        Toast.LENGTH_LONG).show();  
        }  
    }  
    return true;  
}  
case R.id.menu_settings: {  
    Intent myIntent = new Intent(this, Preferences.class)  
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);  
    startActivity(myIntent);  
    return true;  
}  
default: {  
    return super.onOptionsItemSelected(item);  
}  
}  
}
```

## FileUtils.java

```
package com.example.ceandroid;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.channels.FileChannel;

/**
 * FileUtils handles the File Utility Byte for byte copies
 *
 * @author CEandroid SMU
 */
public class FileUtils {
    /**
     * Creates the specified <code>toFile</code> as a byte for byte copy of the
     * <code>fromFile</code>. If <code>toFile</code> already exists, then it
     * will be replaced with a copy of <code>fromFile</code>. The name and path
     * of <code>toFile</code> will be that of <code>toFile</code>. <br/>
     * <br/>
     * <i> Note: <code>fromFile</code> and <code>toFile</code> will be closed b
y
     * this function.</i>
     *
     * @param fromFile
     *          - FileInputStream for the file to copy from.
     * @param toFile
     *          - FileOutputStream for the file to copy to.
     */
    public static void copyFile(FileInputStream fromFile,
        FileOutputStream toFile) throws IOException {
        FileChannel fromChannel = null;
        FileChannel toChannel = null;
        try {
            fromChannel = fromFile.getChannel();
            toChannel = toFile.getChannel();
            fromChannel.transferTo(0, fromChannel.size(), toChannel);
        } finally {
            try {
                if (fromChannel != null) {
                    fromChannel.close();
                }
            } finally {
                if (toChannel != null) {
                    toChannel.close();
                }
            }
        }
    }
}
```

## About.java

```
package com.example.ceandroid;

import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

/**
 * About page - accessed via Preferences
 *
 * @author CEAndroid SMU
 */
public class About extends Fragment {
    /**
     * About Page is a fragment within Preferences
     *
     * @see android.app.Fragment#onCreateView(android.view.LayoutInflater,
     *      android.view.ViewGroup, android.os.Bundle)
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.activity_about, container, false);
    }
}
```

## ShareNFC.java

```
package com.example.ceandroid;

import java.util.ArrayList;

import CEapi.rCause;
import CEapi.rEffect;
import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.IntentFilter.MalformedMimeTypeException;
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.os.Parcelable;
import android.util.Log;
import android.view.MenuItem;
import android.widget.TextView;

/**
 * Page to push rules to another device via NFC
 *
 * @author CEandroid SMU
 *
 */
public class ShareNFC extends Activity {
    /** Used to get the current rule selected */
    private CEapp app;

    /** Tag for NFC */
    private static final String TAG = "ceandroid";

    /** Adapter for NFC usage. */
    NfcAdapter nAdapter;

    /** PendingIntent for NFC usage. */
    PendingIntent nPendingIntent;

    /** Filters for Ndef exchange. */
    IntentFilter[] nExchangeFilters;

    /**
     * (non-Javadoc)
     *
     * @see android.app.Activity#onResume()
     */
    @Override
    protected void onResume() {
        super.onResume();
        enableNdefExchangeMode();
    }

    /*

```

```

        * (non-Javadoc)
        *
        * @see android.app.Activity#onPause()
        */
    @SuppressWarnings("deprecation")
    @Override
    protected void onPause() {
        super.onPause();
        nAdapter.disableForegroundNdefPush(this);
    }

    /**
     * (non-Javadoc)
     *
     * @see android.app.Activity#onCreate(android.os.Bundle)
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sharenfc);

        // Turn on "up" navigation
        getActionBar().setDisplayHomeAsUpEnabled(true);

        app = (CEapp) getApplication();

        // NFC adapter initialization
        nAdapter = NfcAdapter.getDefaultAdapter(this);

        // this activity handles NFC intents
        nPendingIntent = PendingIntent.getActivity(this, 0, new Intent(
            getBaseContext(), ShareList.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP), 0);

        // intent filters for p2p exchange
        @SuppressWarnings("static-access")
        IntentFilter ndefDetected = new IntentFilter(
            nAdapter.ACTION_NDEF_DISCOVERED);
        try {
            ndefDetected.addDataType("text/plain");
        } catch (MalformedMimeTypeException e) {
        }
        nExchangeFilters = new IntentFilter[] { ndefDetected };

        TextView tv = (TextView) findViewById(R.id.textView1);
        tv.setText("Ready to send: \n\"" + app.currentRule.getName() + "\"");
    }

    /**
     * Grabs Ndef messages as they are discovered.
     *
     * @param intent
     *         used to find when Ndefs are discovered
     * @return array of Ndef messages
     */
    NdefMessage[] getNdefMessages(Intent intent) {

```

```

// Parse the intent
NdefMessage[] msgs = null;
String action = intent.getAction();
if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
    Parcelable[] rawMsgs = intent
        .getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
    if (rawMsgs != null) {
        msgs = new NdefMessage[rawMsgs.length];
        for (int i = 0; i < rawMsgs.length; i++) {
            msgs[i] = (NdefMessage) rawMsgs[i];
        }
    } else {
        // Unknown tag type
        byte[] empty = new byte[] {};
        NdefRecord record = new NdefRecord(NdefRecord.TNF_UNKNOWN,
            empty, empty, empty);
        NdefMessage msg = new NdefMessage(new NdefRecord[] { record });
        msgs = new NdefMessage[] { msg };
    }
} else {
    Log.d(TAG, "Unknown intent.");
    finish();
}
return msgs;
}

/**
 * Enables exchange of NFC data.
 */
@SuppressWarnings("deprecation")
private void enableNdefExchangeMode() {
    nAdapter.enableForegroundNdefPush(ShareNFC.this, getRuleAsNdef());
    nAdapter.enableForegroundDispatch(this, nPendingIntent,
        nExchangeFilters, null);
}

/**
 * Converts a rule to an Ndef payload.
 *
 * @return NdefMessage with rule string as its payload
 */
private NdefMessage getRuleAsNdef() {
    String rString = "";
    Rule r = app.currentRule;
    ArrayList<rCause> rCauses = r.getRCauses();
    ArrayList<rEffect> rEffects = r.getREffects();

    rString += r.getName() + '\n' + r.getTreeData() + '\n';

    // add number of rCauses for use in reading string
    rString += String.valueOf(rCauses.size()) + '\n';
    for (int i = 0; i < rCauses.size(); i++) {
        rCause cur = rCauses.get(i);
        if (cur.getParameters().equals(""))
            cur.setParameters(" ");
    }
}

```

```

        rString += String.valueOf(cur.getCauseID()) + '\n'
            + cur.getParameters() + '\n' + cur.getType() + '\n';
    }

    // do the same for rEffects
    rString += String.valueOf(rEffects.size()) + '\n';
    for (int i = 0; i < rEffects.size(); i++) {
        rEffect cur = rEffects.get(i);
        if (cur.getParameters().equals(""))
            cur.setParameters(" ");
        rString += String.valueOf(cur.getEffectID()) + '\n'
            + cur.getParameters() + '\n' + cur.getType() + '\n';
    }

    byte[] rBytes = rString.getBytes();
    NdefRecord record = new NdefRecord(NdefRecord.TNF_MIME_MEDIA,
        "text/plain".getBytes(), new byte[] {}, rBytes);
    return new NdefMessage(new NdefRecord[] { record });
}

/*
 * (non-Javadoc)
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case android.R.id.home:
        // This is called when the Home (Up) button is pressed
        // in the Action Bar.
        Intent parentActivityIntent = new Intent(this, ShareList.class);
        parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
            | Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(parentActivityIntent);
        finish();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
}

```

## ShareList.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import java.util.List;

import CEapi.rCause;
import CEapi.rEffect;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.app.PendingIntent;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.IntentFilter.MalformedMimeTypeException;
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.os.Parcelable;
import android.util.Log;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

/**
 * Page for displaying available rules to share.
 *
 * @author CEAndroid SMU
 *
 */
public class ShareList extends ListActivity {
    /** Array for holding rule names. Used for displaying in the list. */
    private String[] Elements;

    /** Handler for fetching rules. */
    private DatabaseHandler db;

    /** Holds rules to be used. */
    private List<Rule> rules = new ArrayList<Rule>();

    /** Used for setting the current rule for sharing. */
    private CEapp app;

    /** ListView for displaying rule names on screen. */
    private ListView lv;

    // NFC stuff
    /** Tag for NFC */
    private static final String TAG = "ceandroid";

    /** NfcAdapter for receiving Ndefs */
```

```

NfcAdapter nAdapter;

/** PendingIntent for NFC use */
PendingIntent nPendingIntent;

/** Intent filter for picking up Ndef messages */
IntentFilter[] nExchangeFilters;

/*
 * (non-Javadoc)
 *
 * @see android.app.Activity#onResume()
 */
@SuppressLint("StaticFieldLeak")
@Override
public void onResume() {
    super.onResume();

    /** Detect Ndefs coming in and create a rule based on the payload */
    if (nAdapter.ACTION_NDEF_DISCOVERED.equals(getIntent().getAction())) {
        NdefMessage[] messages = getNdefMessages(getIntent());
        byte[] rPayload = messages[0].getRecords()[0].getPayload();
        String rString = new String(rPayload);

        replaceRuleFields(rString);

        setIntent(new Intent());
    }
}

/*
 * (non-Javadoc)
 *
 * @see android.app.Activity#onNewIntent(android.content.Intent)
 */
@SuppressWarnings("StaticFieldLeak")
@Override
protected void onNewIntent(Intent intent) {
    /** Detects Ndefs and creates a rule */
    if (nAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())) {
        NdefMessage[] messages = getNdefMessages(intent);
        String rString = new String(
            messages[0].getRecords()[0].getPayload());
        replaceRuleFields(rString);
    }
}

/*
 * (non-Javadoc)
 *
 * @see android.app.Activity#onCreate(android.os.Bundle)
 */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    app = (CEapp) getApplication();
}

```

```

db = new DatabaseHandler(this);
rules = db.getAllRules();
Elements = new String[rules.size()];

// Turn on "up" navigation
getActionBar().setDisplayHomeAsUpEnabled(true);

for (int i = 0; i < rules.size(); i++) {
    Elements[i] = rules.get(i).getName();
}

@SuppressWarnings("rawtypes")
ArrayAdapter a = new ArrayAdapter<String>(this,
    R.layout.activity_sharelist, Elements);

setListAdapter(a);
a.notifyDataSetChanged();

lv = getListView();
lv.setTextFilterEnabled(true);

lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
        app.currentRule = rules.get(position);
        app.currentRule.setRCauses(db.getAllRCauses(app.currentRule
            .getID()));
        app.currentRule.setREffects(db.getAllREffects(app.currentRule
            .getID()));

        Intent intent = new Intent(ShareList.this, ShareNFC.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(intent);
    }
});

// Nfc adapter initialization
nAdapter = NfcAdapter.getDefaultAdapter(this);

// this activity handles NFC intents
nPendingIntent = PendingIntent.getActivity(this, 0, new Intent(this,
    getClass()).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);

// intent filters for p2p exchange
@SuppressWarnings("static-access")
IntentFilter ndefDetected = new IntentFilter(
    nAdapter.ACTION_NDEF_DISCOVERED);
try {
    ndefDetected.addDataType("text/plain");
} catch (MalformedMimeTypeException e) {
}
nExchangeFilters = new IntentFilter[] { ndefDetected };
}

/**
 * Takes in a C&E Ndef payload (plain-

```

```

text string) and creates a rule in the
* database after parsing it.
*
* @param rString
*          Plain-text representation of a rule object
*/
private void replaceRuleFields(String rString) {
    int charPtr = 0;
    DatabaseHandler db = new DatabaseHandler(this);
    ArrayList<rCause> rCauses = new ArrayList<rCause>();
    ArrayList<rEffect> rEffects = new ArrayList<rEffect>();
    int ceCount;
    String name = "";
    char cur = ' ';
    while (true) {
        cur = rString.charAt(charPtr);
        if (cur == '\n')
            break;
        name += cur;
        charPtr++;
    }

    charPtr++;
    System.out.println(name);

    // tree data, FIX ME!
    String tree = "";
    while (true) {
        cur = rString.charAt(charPtr);
        if (cur == '\n')
            break;
        tree += cur;
        charPtr++;
    }
    System.out.println(tree);

    charPtr++;

    String rCount = "";
    while (true) {
        cur = rString.charAt(charPtr);
        if (cur == '\n')
            break;
        rCount += cur;
        charPtr++;
    }
    System.out.println(rCount);

    charPtr++;
    Rule r = new Rule(tree, name, true);
    db.addRule(r);
    r.setID(db.getAllRules().get(db.getRulesCount() - 1).getID());
    ceCount = db.getRCausesCount();
    for (int i = 0; i < Integer.parseInt(rCount); i++) {
        ceCount++;
        String rid = "", rParam = "", rType = "";

```

```

while (true) {
    cur = rString.charAt(charPtr);
    if (cur == '\n')
        break;
    rid += cur;
    charPtr++;
}
System.out.println(rid);
charPtr++;

while (true) {
    cur = rString.charAt(charPtr);
    if (cur == '\n')
        break;
    rParam += cur;
    charPtr++;
}
System.out.println(rParam);
charPtr++;

while (true) {
    cur = rString.charAt(charPtr);
    if (cur == '\n')
        break;
    rType += cur;
    charPtr++;
}
System.out.println(rType);
charPtr++;

rCauses.add(new rCause(ceCount, r.getID(), Integer.parseInt(rid),
                      rParam, rType));
}

rCount = "";
while (true) {
    cur = rString.charAt(charPtr);
    if (cur == '\n')
        break;
    rCount += cur;
    charPtr++;
}
System.out.println(rCount);
charPtr++;

ceCount = db.getREffectsCount();
for (int i = 0; i < Integer.parseInt(rCount); i++) {
    ceCount++;
    String rid = "", rParam = "", rType = "";
    while (true) {
        cur = rString.charAt(charPtr);
        if (cur == '\n')
            break;
        rid += cur;
        charPtr++;
    }
}

```

```

System.out.println(rid);
charPtr++;

while (true) {
    cur = rString.charAt(charPtr);
    if (cur == '\n')
        break;
    rParam += cur;
    charPtr++;
}
System.out.println(rParam);
charPtr++;

while (true) {
    cur = rString.charAt(charPtr);
    if (cur == '\n')
        break;
    rType += cur;
    charPtr++;
}
System.out.println(rType);
charPtr++;

rEffects.add(new rEffect(ceCount, r.getID(), Integer.parseInt(rid),
    rParam, rType));
}

r.setRCauses(rCauses);
r.setREffects(rEffects);
// r.updateTreeData();
r.updateTreeData(r.getBoolSequence());
db.updateRule(r);

rules = db.getAllRules();
Elements = new String[rules.size()];

for (int i = 0; i < rules.size(); i++) {
    Elements[i] = rules.get(i).getName();
}

@SuppressWarnings("rawtypes")
ArrayAdapter a = new ArrayAdapter<String>(this,
    R.layout.activity_sharelist, Elements);

setListAdapter(a);
a.notifyDataSetChanged();
AlertDialog.Builder adb = new AlertDialog.Builder(this);
adb.setTitle("NFC Sharing").setMessage("Rule received!")
    .setNeutralButton("Ok", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });
adb.show();
db.close();
}

```

```

/**
 * Grabs Ndef messages from an NFC push from another Android device
 *
 * @param intent
 *          Used to check for an Ndef discovered action
 * @return array of received Ndefs
 */
NdefMessage[] getNdefMessages(Intent intent) {
    // Parse the intent
    NdefMessage[] msgs = null;
    String action = intent.getAction();
    if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
        Parcelable[] rawMsgs = intent
            .getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        if (rawMsgs != null) {
            msgs = new NdefMessage[rawMsgs.length];
            for (int i = 0; i < rawMsgs.length; i++) {
                msgs[i] = (NdefMessage) rawMsgs[i];
            }
        } else {
            // Unknown tag type
            byte[] empty = new byte[] {};
            NdefRecord record = new NdefRecord(NdefRecord.TNF_UNKNOWN,
                empty, empty, empty);
            NdefMessage msg = new NdefMessage(new NdefRecord[] { record });
            msgs = new NdefMessage[] { msg };
        }
    } else {
        Log.d(TAG, "Unknown intent.");
        finish();
    }
    return msgs;
}

/*
 * (non-Javadoc)
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case android.R.id.home:
        // This is called when the Home (Up) button is pressed
        // in the Action Bar.
        Intent parentActivityIntent = new Intent(this, MainActivity.class);
        parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
            | Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(parentActivityIntent);
        finish();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
}

```

## DeleteDialogFragmentRule.java

```
package com.example.ceandroid;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;

/**
 * Creates the DialogFragment for deleting rules from the MyRules Activity
 *
 * @author CEandroid SMU
 */
public class DeleteDialogFragmentRule extends DialogFragment {

    /**
     * Creates the DeleteDialogFragmentRule
     *
     * @see android.app.DialogFragment#onCreateDialog(android.os.Bundle)
     */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        // Set the dialog title
        builder.setTitle(R.string.delete)
            .setMessage(R.string.delete_confirmation_rule)
        // Set the action buttons
        .setPositiveButton(R.string.delete_confirm,
            new DialogInterface.OnClickListener() {
                /**
                 * Confirms the Delete of the Rule
                 *
                 * @see android.content.DialogInterface.OnClickListener#onClick
                 (android.content.DialogInterface,
                 *      int)
                 */
                public void onClick(DialogInterface dialog, int id) {
                    // Send the positive button event back to the
                    // host activity
                    mListener
                        .onDialogPositiveClick(DeleteDialogFragmentRule.this);
                }
            })
        .setNegativeButton(R.string.cancel,
            new DialogInterface.OnClickListener() {
                /**
                 * DeleteDialogFragmentRule Cancelled
                 *
                 * @see android.content.DialogInterface.OnClickListener#onClick
                 (android.content.DialogInterface,
                 *      int)
                 */
            });
    }

    .setOnCancelListener(new DialogInterface.OnCancelListener() {
        /**
         * DeleteDialogFragmentRule Cancelled
         *
         * @see android.content.DialogInterface.OnCancelListener#onCancel
         (android.content.DialogInterface)
         */
    });
}
```

```

    public void onClick(DialogInterface dialog, int id) {
        // Send the negative button event back to the
        // host activity
        mListener
            .onDialogNegativeClick(DeleteDialogFragmentRule.this);
    }
});

return builder.create();
}

/**
 * DeleteDialogListenerRule Interface
 *
 * @author CEandroid SMU
 */
public interface DeleteDialogListenerRule {
    public void onDialogPositiveClick(DialogFragment dialog);

    public void onDialogNegativeClick(DialogFragment dialog);
}

/**
 * DeleteDialogListenerRule that implements the DeleteDialogListenerRule
 * Interface
 */
DeleteDialogListenerRule mListener;

/**
 * Attaches the DialogFragment to the MyRules Activity
 *
 * @see android.app.DialogFragment#onAttach(android.app.Activity)
 */
@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    // Verify that the host activity implements the callback interface
    try {
        // Instantiate the NoticeDialogListener so we can send events to the
        // host
        mListener = (DeleteDialogListenerRule) activity;
    } catch (ClassCastException e) {
        // The activity doesn't implement the interface, throw exception
        throw new ClassCastException(activity.toString()
            + " must implement DeleteDialogListenerRule");
    }
}
}

```

## Preferences.java

```
package com.example.ceandroid;

import android.app.ActionBar;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.support.v4.app.FragmentActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

/**
 * Preferences page - tabbed interface to get access to other pages
 *
 * @author CEandroid SMU
 */
public class Preferences extends FragmentActivity {
    /**
     * Selected item constant referring to which fragment is in the foreground
     */
    private static final String STATE_SELECTED_NAVIGATION_ITEM = "selected_navigation_item";

    /**
     * Creates the action bar and FragmentActivity Contains General Settings,
     * Security Settings, and the About Page
     *
     * @see android.support.v4.app.FragmentActivity#onCreate(android.os.Bundle)
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_preferences);

        // Set up the action bar.
        final ActionBar actionBar = getActionBar();
        if (actionBar != null) {
            // Set Navigation mode
            actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

            // Turn on "up" navigation
            actionBar.setDisplayHomeAsUpEnabled(true);

            // For each of the sections in the app, add a tab to the action bar.
            actionBar.addTab(actionBar
                    .newTab()
                    .setText("General")
                    .setTabListener(
                            new TabListener<General>(this, "general",
                                    General.class)));
            actionBar.addTab(actionBar
                    .newTab()
                    .setText("Security"))
    }
}
```

```

        .setTabListener(
            new TabListener<Security>(this, "security",
                Security.class)));
    actionBar
        .addTab(actionBar
            .newTab()
            .setText("About")
            .setTabListener(
                new TabListener<About>(this, "about",
                    About.class)));
    }
}

/**
 * Checks the current rule, clears if not null
 *
 * @see android.app.Activity#onResume()
 */
@Override
public void onResume() {
    super.onResume();
    CEapp app = (CEapp) this.getApplication();
    if (app.currentRule != null) {
        if (app.currentRule.getRCauses().isEmpty()
            && app.currentRule.getREffects().isEmpty()) {
            DatabaseHandler db = new DatabaseHandler(this);
            db.deleteRule(app.currentRule);
            Toast.makeText(this, "Incomplete rule not saved.",
                Toast.LENGTH_LONG).show();
            app.currentRule = null;
            db.close();
        }
    }
}

/**
 * When the state is returned, the current tab selection is brought to the
 * front
 *
 * @see android.app.Activity#onRestoreInstanceState(android.os.Bundle)
 */
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    if (savedInstanceState.containsKey(STATE_SELECTED_NAVIGATION_ITEM)) {
        getActionBar(). setSelectedNavigationItem(
            savedInstanceState.getInt(STATE_SELECTED_NAVIGATION_ITEM));
    }
}

/**
 * Saves the instance state when the application is sent to the background
 *
 * @see android.support.v4.app.FragmentActivity#onSaveInstanceState(android
 * .os.Bundle)
 */
@Override

```

```

public void onSaveInstanceState(Bundle outState) {
    outState.putInt(STATE_SELECTED_NAVIGATION_ITEM, getActionBar()
        .getSelectedNavigationIndex());
}

< /**
 * Preferenes Menu Creation
 *
 * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
 */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_preferences, menu);
    return true;
}

< /**
 * Preferences Menu Actions
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case android.R.id.home: {
            // This is called when the Home (Up) button is pressed
            // in the Action Bar.
            Intent parentActivityIntent = new Intent(this, MainActivity.class);
            parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                | Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(parentActivityIntent);
            finish();
            return true;
        }
        case R.id.menu_new_rule: {
            // New Rule
            CEapp app = (CEapp) this.getApplication();
            app.currentRule = new Rule();

            Intent myIntent = new Intent(this, EditRule.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
            return true;
        }
        case R.id.menu_share: {
            PackageManager p = this.getPackageManager();
            if (!p.hasSystemFeature("android.hardware.nfc")) {
                Toast.makeText(this, "NFC is not available on this device.",
                    Toast.LENGTH_LONG).show();
            } else {
                NfcAdapter na = NfcAdapter.getDefaultAdapter(this);
                if (na.isEnabled()) {
                    Intent myIntent = new Intent(this, ShareList.class)
                        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                    startActivity(myIntent);
                }
            }
        }
    }
}

```

```
        } else {
            Toast.makeText(this,
                "Please enable NFC to use this feature.",
                Toast.LENGTH_LONG).show();
        }
    }
    return true;
}
case R.id.menu_help: {
    Intent myIntent = new Intent(this, Help.class)
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
    return true;
}
default: {
    return super.onOptionsItemSelected(item);
}
}
}
}
```

## CategoryList.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

/**
 * Category List Fragment This class is used to organize Rules/Causes/Effects
 * by
 * Category Currently not being used
 *
 * @author CEandroid SMU
 */
public class CategoryList extends Fragment {

    /**
     * Category List as the parent activity creates this fragment
     *
     * @see android.app.Fragment#onActivityCreated(android.os.Bundle)
     */
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        TextView textview = new TextView(this.getActivity());
        textview.setText("This is the Category tab");

        DatabaseHandler db = new DatabaseHandler(this.getActivity());
        try {
            ArrayList<String> categories = db.getAllCauseCategories();
            ArrayList<Rule> rule = db.getAllRules();
            for (int i = 0; i < rule.size(); i++) {
                Log.d("rule " + i, "" + rule.get(i).getName());
            }
            for (int i = 0; i < categories.size(); i++) {
                ArrayList<Rule> rules = db.getAllRulesByCategory(categories
                    .get(i));
                Log.d(categories.get(i), "" + rules.size());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        db.close();

        getActivity().setContentView(textview);
    }
}
```

## Security.java

```
package com.example.ceandroid;

import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;

/**
 * Page for showing security options
 *
 * @author CEandroid SMU
 */
public class Security extends Fragment {
    /**
     * Security Settings Page is a fragment within Preferences
     *
     * @see android.app.Fragment#onCreateView(android.view.LayoutInflater,
     *      android.view.ViewGroup, android.os.Bundle)
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.activity_security, container, false);
        ListView hList = (ListView) v.findViewById(R.id.secList);
        // List Click Adapters
        hList.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                long arg3) {
                Toast.makeText(getActivity().getApplicationContext(),
                    "Security Options not implemented", Toast.LENGTH_SHORT)
                    .show();
            }
        });
        return v;
    }
}
```

## EditRuleNode.java

```
package com.example.ceandroid;

/**
 * Custom object used for AND/OR list manipulation in EditRule
 *
 * @author CEAndroid SMU
 */
public class EditRuleNode {
    /** Can be the rCause name, boolean value ("AND"/"OR"), or "+" */
    String value;

    /** Flag to represent what kind of node this is (rCause, boolean, or +) */
    boolean cFlag, bFlag;

    /**
     * Constructor for node
     *
     * @param value
     *         rCause name, "AND"/"OR", or "+"
     * @param cFlag
     *         cause flag, true if rCause node (set bFlag to false, set both
     *         to false if +)
     * @param bFlag
     *         boolean flag, true if bool node (set cFlag to false, set both
     *         to false if +)
     */
    public EditRuleNode(String value, boolean cFlag, boolean bFlag) {
        this.value = value;
        this.cFlag = cFlag;
        this.bFlag = bFlag;
    }

    /**
     * @return true if cause node, false if bool node
     */
    public boolean isCause() {
        return cFlag;
    }

    /**
     * @return true if bool node, false if cause node
     */
    public boolean isBool() {
        return bFlag;
    }

    /**
     * @return true if plus node
     */
    public boolean isPlus() {
        if (cFlag == false && bFlag == false) {
            return true;
        } else
    }
}
```

```
        return false;
    }

/**
 * Returns this.value
 *
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
    return this.value;
}
}
```

## CauseView.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import com.example.ceandroid.Causes.MapPicker;

import CEapi.Cause;
import CEapi.rCause;
import android.annotation.SuppressLint;
import android.app.ActionBar;
import android.app.Activity;
import android.app.Dialog;
import android.app.DialogFragment;
import android.app.TimePickerDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnDismissListener;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.support.v4.app.FragmentActivity;
import android.text.format.DateFormat;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Switch;
import android.widget.TimePicker;
import android.widget.Toast;

/**
 * CauseView Class displays the list of Causes
 *
 * @author CEandroid SMU
 */
public class CauseView extends FragmentActivity {
    /**
     * Elements array holds and empty array that is later used for Cause
     * information
     */
    static String[] Elements = new String[] { " ", " ", " ", " ", " ", " ",
        " ", " ", " " };
    /**
     * List View used to list out the Causes
     */
}
```

```

private ListView listView;
< /**
 * Database Handler used to get the list of Causes
 */
private DatabaseHandler db;
< /**
 * gPosition used to handle user actions on the list of Causes
 */
int gPosition = 0;
< /**
 * The type of Cause that is selected
 */
int type = 0;
< /**
 * List of Cause Objects
 */
private List<Cause> causes = new ArrayList<Cause>();
< /**
 * Used to access global application variables
 */
private CEapp app;

< /**
 * List of Causes currently being used in the application This can be
 * removed if strings are used for the switch statement
 *
 * @author CEandroid SMU
 */
public enum cEnum {
    time, phoneCall, textMessage, ssid, wifiStatus, location
}

< /**
 * OnCreate gets the list of Causes from the database and adds them to the
 * ListView
 *
 * @see android.support.v4.app.FragmentActivity#onCreate(android.os.Bundle)
 */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.item_list);
    listView = (ListView) findViewById(R.id.fruitList);
    listView.setTextFilterEnabled(true);
    db = new DatabaseHandler(this);

    // Set up the action bar
    final ActionBar actionBar = getSupportActionBar();
    if (actionBar != null) {
        // Turn on "up" navigation
        actionBar.setDisplayHomeAsUpEnabled(true);
    }

    // Load the kind of list
    app = (CEapp) getApplication();
    makeList();
}

```

```

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_activated_1, Elements);

// set the adapter of each view with the same data
listView.setAdapter(adapter);

listView.setOnItemClickListener(new OnItemClickListener() {
    /*
     * (non-Javadoc)
     *
     * @see
     * android.widget.AdapterView.OnItemClickListener#onItemClick(android
     * .widget.AdapterView, android.view.View, int, long)
     */
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
        doSomething(position, type);
    }
});

if (app.edit) {
    if (app.editType) // cause
    {
        doSomething(app.editID, 2);
    } else // effect
    {
        doSomething(app.editID, 3);
    }
}
db.close();
}

/**
 * makeList() is what is used by the onCreate to create the list of Causes
 */
public void makeList() {
    // DatabaseHandler db = new DatabaseHandler(this.this);
    causes = db.getAllCauses();
    // db.close();
    Elements = new String[causes.size()];
    for (int i = 0; i < causes.size(); i++) {
        Elements[i] = causes.get(i).getName();
    }
}

/**
 * doSomething is called when the user selects a Cause from the list
 *
 * @param position
 * @param type
 */
public void doSomething(int position, int type) {
    // Enums Switch
    switch (cEnum.valueOf(causes.get(position).getType())) {
    case time: {
        DialogFragment newFragment = new TimePickerFragment();
    }
}

```

```

        newFragment.show(getFragmentManager(), "timePicker");
    }
    break;
case phoneCall: {
    Intent intent = new Intent(Intent.ACTION_PICK,
        ContactsContract.Contacts.CONTENT_URI)
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivityForResult(intent, 1);
}
break;
case textMessage: {
    Intent intent = new Intent(Intent.ACTION_PICK,
        ContactsContract.Contacts.CONTENT_URI)
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivityForResult(intent, 2);
}
break;
case ssid: {
    // Toast.makeText(this.getApplicationContext(),
    // "SSID not implemented", Toast.LENGTH_SHORT).show();
    final Dialog d = new Dialog(this);
    d.setContentView(R.layout.ce_dialog_ssid);
    d.setTitle("Add SSID here");
    d.setOnDismissListener(new OnDismissListener() {

        /*
         * (non-Javadoc)
         *
         * @see
         * android.content.DialogInterface.OnDismissListener#onDismiss
         * (android.content.DialogInterface)
         */
        public void onDismiss(DialogInterface arg0) {
            if (app.edit) {
                Intent myIntent = new Intent(getApplicationContext(),
                    EditRule.class)
                    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivityForResult(myIntent);
            }
        }
    });
    Button button = (Button) d.findViewById(R.id.button1);
    button.setOnClickListener(new OnClickListener() {
        /*
         * (non-Javadoc)
         *
         * @see
         * android.view.View.OnClickListener#onClick(android.view.View)
         */
        public void onClick(View v) {
            EditText edit2 = (EditText) d.findViewById(R.id.ssid);
            String text = edit2.getText().toString();
            d.dismiss();

            gPosition = 5;
        }
    });
}

```

```

        rCause rc = new rCause();
        rc.setName(causes.get(gPosition).getName());
        rc.setRuleID(app.currentRule.getID());
        rc.setType(causes.get(gPosition).getType());
        rc.setCauseID(4);
        String parameter = "" + text;
        rc.setParameters(parameter);
        app.currentRule.addRCause(rc);
        app.addedRCause = true;
        Intent myIntent = new Intent(getApplicationContext(),
            EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
})
d.show();
}
break;
case wifiStatus: {
// Toast.makeText(this.getApplicationContext(),
// "Wifi On/Off not implemented", Toast.LENGTH_SHORT).show();
final Dialog d = new Dialog(this);
d.setContentView(R.layout.ce_dialog_switch);
d.setTitle("Check if Wifi is on or off");
d.setOnDismissListener(new OnDismissListener() {

/*
 * (non-Javadoc)
 *
 * @see
 * android.content.DialogInterface.OnDismissListener#onDismiss
 * (android.content.DialogInterface)
 */
public void onDismiss(DialogInterface arg0) {
    if (app.edit) {
        Intent myIntent = new Intent(getApplicationContext(),
            EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
}

});
d.setOnDismissListener(new OnDismissListener() {

/*
 * (non-Javadoc)
 *
 * @see
 * android.content.DialogInterface.OnDismissListener#onDismiss
 * (android.content.DialogInterface)
 */
public void onDismiss(DialogInterface arg0) {
    if (app.edit) {
        Intent myIntent = new Intent(getApplicationContext(),
            EditRule.class)

```

```

        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
}

});

Button button = (Button) d.findViewById(R.id.button1);
button.setOnClickListener(new OnClickListener() {
/*
 * (non-Javadoc)
 *
 * @see
 * android.view.View.OnClickListener#onClick(android.view.View)
 */
public void onClick(View v) {
    Switch s1 = (Switch) d.findViewById(R.id.dialogSwitch);
    String text = "";
    if (s1.isChecked())
        text = "on";
    else
        text = "off";
    d.dismiss();

    gPosition = 6;
    rCause rc = new rCause();
    rc.setName(causes.get(gPosition).getName());
    rc.setRuleID(app.currentRule.getID());
    rc.setType(causes.get(gPosition).getType());
    rc.setCauseID(5);
    String parameter = text;
    rc.setParameters(parameter);
    app.currentRule.addRCause(rc);
    app.addedRCause = true;
    Intent myIntent = new Intent(getApplicationContext(),
        EditRule.class)
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
}
});
d.show();
}

break;
case location: {
    Intent intent = new Intent(this.getApplicationContext(),
        MapPicker.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    app.setTemp(position);
    startActivityForResult(intent, 2);
}
break;
default:
break;
}
db.close();
}

// Time Picker Dialog

```

```

/**
 * TimePickerFragment is called when the user selects the Time Cause
 *
 * @author CEandroid SMU
 */
@SuppressWarnings("ValidFragment")
public class TimePickerFragment extends DialogFragment implements
    TimePickerDialog.OnTimeSetListener {
    /**
     * Creates the TimePickerFragment
     *
     * @see android.app.DialogFragment#onCreateDialog(android.os.Bundle)
     */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
            DateFormat.is24HourFormat(getActivity()));
    }

    /**
     * Called when the Time is confirmed by the user
     *
     * @see android.app.TimePickerDialog.OnTimeSetListener#onTimeSet(android.
     * widget.TimePicker,
     *      int, int)
     */
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        gPosition = 2;
        rCause rc = new rCause();
        rc.setName("Time");
        rc.setRuleID(app.currentRule.getID());
        rc.setType("time");
        rc.setCauseID(1);
        String parameter = "";
        if (hourOfDay < 10) {
            parameter = "0" + hourOfDay;
        } else {
            parameter = "" + hourOfDay;
        }
        parameter += ":";
        if (minute < 10) {
            rc.setParameters(parameter + "0" + minute);
        } else {
            rc.setParameters(parameter + minute);
        }
        app.currentRule.addRCause(rc);
        app.addedRCause = true;

        Intent myIntent = new Intent(getApplicationContext(),
            EditRule.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    }
}

```

```

        startActivity(myIntent);
    }

    /**
     * Called when the user backs out or cancels the TimePickerFragment
     *
     * @see android.app.DialogFragment#onDismiss(android.content.DialogInterface)
     */
    @Override
    public void onDismiss(DialogInterface dialog) {
        if (app.edit) {
            Intent myIntent = new Intent(getApplicationContext(),
                EditRule.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
        }
    }
}

/**
 * Called when the user is done setting parameters for a Cause and the
 * picker is returned to CauseView Creates rCauses to be associated with th
e
 * Rule based on the user's parameters
 *
 * @see android.support.v4.app.FragmentActivity#onActivityResult(int, int,
 *      android.content.Intent)
 */
@SuppressWarnings("deprecation")
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode) {
        case (1):
            if (resultCode == Activity.RESULT_OK) {
                Uri contactData = data.getData();
                Cursor c = this.managedQuery(contactData, null, null, null,
                    null);
                if (c.moveToFirst()) {
                    gPosition = 0;
                    String name = c
                        .getString(c
                            .getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
                    int hasNumber = Integer
                        .parseInt(c.getString(c
                            .getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER))
                    );
                    if (hasNumber != 0) {
                        rCause rc = new rCause();
                        rc.setName(causes.get(gPosition).getName());
                        rc.setRuleID(app.currentRule.getID());
                        rc.setType(causes.get(gPosition).getType());
                        rc.setCauseID(2);
                        String parameter = name;
                        rc.setParameters(parameter);
                    }
                }
            }
    }
}

```

```

        app.currentRule.addRCause(rc);
        app.addedRCause = true;

        Intent myIntent = new Intent(
            this.getApplicationContext(), EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    } else {
        Toast.makeText(this.getApplicationContext(),
            "Please choose a contact with a valid number",
            Toast.LENGTH_LONG).show();
        Intent intent = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivityForResult(intent, 1);
    }
} else {
    if (app.edit) {
        Intent myIntent = new Intent(
            this.getApplicationContext(), EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
}
break;
case (2):
if (resultCode == Activity.RESULT_OK) {
    Uri contactData = data.getData();
    Cursor c = this.managedQuery(contactData, null, null, null,
        null);
    if (c.moveToFirst()) {
        gPosition = 1;
        String name = c
            .getString(c
                .getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
        int hasNumber = Integer
            .parseInt(c.getString(c
                .getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER)));
    });
    if (hasNumber != 0) {
        rCause rc = new rCause();
        rc.setName(causes.get(gPosition).getName());
        rc.setRuleID(app.currentRule.getID());
        rc.setType(causes.get(gPosition).getType());
        rc.setCauseID(3);
        String parameter = name;
        rc.setParameters(parameter);
        app.currentRule.addRCause(rc);
        app.addedRCause = true;

        Intent myIntent = new Intent(
            this.getApplicationContext(), EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    } else {
}
}

```

```

        Toast.makeText(this.getApplicationContext(),
                "Please choose a contact with a valid number",
                Toast.LENGTH_LONG).show();
        Intent intent = new Intent(Intent.ACTION_PICK,
                ContactsContract.Contacts.CONTENT_URI)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivityForResult(intent, 1);
    }
} else {
    if (app.edit) {
        Intent myIntent = new Intent(
                this.getApplicationContext(), EditRule.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
}
break;
default:
    Intent myIntent = new Intent(this.getApplicationContext(),
            EditRule.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
}
if (app.edit) {
    Intent myIntent = new Intent(this.getApplicationContext(),
            EditRule.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
}
}

/**
 * CauseView Menu Creation
 *
 * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
 */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_edit_task, menu);
    return true;
}

/**
 * CauseView Menu Actions
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case android.R.id.home: {
            // This is called when the Home (Up) button is pressed
            // in the Action Bar.
            Intent parentActivityIntent = new Intent(this, EditRule.class);
            parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                    | Intent.FLAG_ACTIVITY_NEW_TASK);

```

```
        startActivity(parentActivityIntent);
        finish();
        return true;
    }
    case R.id.menu_help: {
        Intent myIntent = new Intent(this, Help.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
        return true;
    }
    case R.id.menu_settings: {
        Intent myIntent = new Intent(this, Preferences.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
        return true;
    }
    default: {
        return super.onOptionsItemSelected(item);
    }
}
}
}
```

## TabListener.java

```
package com.example.ceandroid;

import android.app.ActionBar;
import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentTransaction;

/**
 * TabListener taken from sample code at developer.android.com
 *
 * @author http://developer.android.com/guide/topics/ui/actionbar.html
 */
public class TabListener<T extends Fragment> implements ActionBar.TabListener {
    private Fragment mFragment;
    private final Activity mActivity;
    private final String mTag;
    private final Class<T> mClass;

    /**
     * Constructor used each time a new tab is created.
     *
     * @param activity
     *          The host Activity, used to instantiate the fragment
     * @param tag
     *          The identifier tag for the fragment
     * @param cls
     *          The fragment's Class, used to instantiate the fragment
     */
    public TabListener(Activity activity, String tag, Class<T> cls) {
        mActivity = activity;
        mTag = tag;
        mClass = cls;
    }

    /* The following are each of the ActionBar.TabListener callbacks */

    public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
        // Check if the fragment is already initialized
        if (mFragment == null) {
            // If not, instantiate and add it to the activity
            mFragment = Fragment.instantiate(mActivity, mClass.getName());
            ft.add(android.R.id.content, mFragment, mTag);
        } else {
            // If it exists, simply attach it in order to show it
            ft.attach(mFragment);
        }
    }

    public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction ft) {
        if (mFragment != null) {
            // Detach the fragment, because another one is being attached
            ft.detach(mFragment);
        }
    }
}
```

```
}

public void onTabReselected(ActionBar.Tab tab, FragmentTransaction ft) {
    // User selected the already selected tab. Usually do nothing.
}
}
```

## CEapp.java

```
package com.example.ceandroid;

import android.app.Application;

/**
 * CEapp stores all globally accessible information for the application
 *
 * @author CEandroid SMU
 */
public class CEapp extends Application {
    /**
     * The List Type is used to determine which list data should be retrieved at
     * any given time. 1:My Rules 2:Causes 3:Effects
     */
    private int listType = 0;
    /**
     * Stores information on the rule currently being created
     */
    public Rule currentRule = null;
    /**
     * Stores information of the position of an item selected from a List
     */
    private int itemSelected = 0;
    /**
     * Stores whether a cause or effect is currently being edited
     */
    public boolean edit = false;
    /**
     * Stores the Identifier for edit modes
     */
    int editID = 0;
    /**
     * Stores whether the edit type is to influence application logic
     */
    boolean editType = true;
    /**
     * Stores whether a rCause has been added, so that ANDs and ORs can be
     * updated
     */
    public boolean addedRCause = false;
    /**
     * Stores the edited number of items to update in the database
     */
    protected int editedNumber;
    /**
     * Temporary Variable
     */
    private int temp = 0;

    /**
     * Get listType
     *
     * @return this.listType
     */
```

```

        */
    public int getListType() {
        return listType;
    }

    /**
     * Set listType
     *
     * @param listType
     */
    public void setListType(int listType) {
        this.listType = listType;
    }

    /**
     * Get itemSelected
     *
     * @return this.itemSelected
     */
    public int getItemSelected() {
        return itemSelected;
    }

    /**
     * Set itemSelected
     *
     * @param itemSelected
     */
    public void setItemSelected(int itemSelected) {
        this.itemSelected = itemSelected;
    }

    /**
     * Get temp
     *
     * @return this.temp
     */
    public int getTemp() {
        return temp;
    }

    /**
     * Set temp
     *
     * @param temp
     */
    public void setTemp(int temp) {
        this.temp = temp;
    }
}

```

## ActionExecuter.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import com.example.ceandroid.Effects.*;

import CEapi.rEffect;
import android.content.Context;

/**
 * The Action Executor Class executes Effects
 *
 * @author CEandroid SMU
 */
public class ActionExecuter {

    /**
     * Blank constructor for ActionExecuter
     */
    public ActionExecuter() {
        // blank constructor
    }

    /**
     * Executes the effect list
     *
     * Executes all of the passed effects. Determines which effect to execute
     * using the rEffect enum type.
     *
     * @param context
     *         global interface that allows access application resources
     * @param effects
     *         list of all rEffect objects to be executed.
     */
    public void executeActions(Context context, ArrayList<rEffect> effects) {

        for (rEffect effect : effects) {
            switch (Type.valueOf(effect.getType())) {
                case vibrate: {
                    new ActionVibrate(context, effect.getParameters()).execute(); // vibr
ate
                    break;
                }
                case toast: {
                    new ActionToast(context, effect.getParameters()).execute(); // toast
                    break;
                }
                case notification: {
                    new ActionShowNotification(context, effect.getParameters())
                        .execute(); // notifications
                    break;
                }
                case sound: {
                    new ActionPlaySound(context, effect.getParameters())

```

```

        .split("\n") [0]).execute() ; // play sound
    break;
}
case ringer: {
    new ActionRingerMode(context, effect.getParameters()).execute() ; // s
et
                                // ringer
                                // mode
}
default: {
    break;
}
}
}

/**
 * Currently Stored Effect Types This can be removed if the switch statemen
t
 * above uses strings instead
 *
 * @author CEandroid SMU
 */
public enum Type {
    vibrate, toast, notification, sound, ringer
}
}

```

## RulesEngine.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import java.util.List;

import CEapi.rEffect;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;

/**
 * Used for rule evaluation. Cycles through returned rules and calls
 * ActionExecuter if a rule returns true.
 *
 * @author CEandroid SMU
 *
 */
public class RulesEngine extends Service {

    /** List for holding rules fetched from the database */
    private List<Rule> rules;

    /** Represents the type of causes needing to be fetched */
    private String type;

    /** Needed for database handler creation */
    private Context context;

    /**
     * Checks a rule's causes using AsyncTask
     *
     * @author CEandroid SMU
     *
     */
    private class CauseChecker extends AsyncTask<String, Void, Boolean> {
        /** Used for fetching effects for a rule after it returns true */
        private int ruleID;

        /**
         * (non-Javadoc)
         *
         * @see android.os.AsyncTask#doInBackground(Params[])
         */
        @Override
        protected Boolean doInBackground(String... params) {
            ExpressionTree tree = new ExpressionTree(params[0], context);
            this.ruleID = Integer.parseInt(params[1]);
            return tree.evaluate(context);
        }

        /**
         * (non-Javadoc)
         */

    }
}
```

```

/*
 * @see android.os.AsyncTask#onPostExecute(java.lang.Object)
 */
@Override
protected void onPostExecute(Boolean result) {
    if (result == true) {
        // pass context and effect list to Tomin's code
        DatabaseHandler db = new DatabaseHandler(context);
        ArrayList<rEffect> rEffects = db.getAllREffects(ruleID);
        if (!rEffects.isEmpty()) {
            ActionExecuter ae = new ActionExecuter();
            ae.executeActions(context, rEffects);
        }
        db.close();
    }
}

/**
 * Default constructor
 *
 * @param type
 *          type of rule to be evaluated
 * @param c
 *          context that allows access to application resources
 */
public RulesEngine(String type, Context c) {
    this.type = type;
    this.context = c;
}

/**
 * Tells the RulesEngine to grab all rules of a certain type and check if
 * their causes return true
 */
public void start() {
    DatabaseHandler db = new DatabaseHandler(context);
    rules = db.getAllRulesByType(type);
    db.close();
    String sequence = "";
    for (Rule r : rules) {
        if (r.getActive() == true) {
            sequence = r.getTreeData();
            CauseChecker c = new CauseChecker();
            c.execute(sequence, "" + r.getID());
        }
    }
}

/*
 * (non-Javadoc)
 *
 * @see android.app.Service# onBind(android.content.Intent)
 */
@Override
public IBinder onBind(Intent arg0) {

```

```
    return null;
}
}
```

## ExpressionTree.java

```
package com.example.ceandroid;

import java.util.Stack;

import CEapi.rCause;
import android.content.Context;

/**
 * Uses the tree string stored in a rule to build a tree, fills leaf nodes with
 * rCause objects, and evaluates/returns a result
 *
 * @author CEandroid SMU
 */
public class ExpressionTree {
    /**
     * Nodes for tree building.
     *
     * @author CEandroid SMU
     */
    private static class TreeNode {
        /** Is true if the node is a leaf */
        public boolean leaf;

        /** Either '+' or '&'amp; if not a leaf node, used for evaluation */
        public char operation;

        /** rCause to be evaluated (leaf nodes only), null if operator node */
        public rCause value;

        /** Used for getting the correct rCause from the database for evaluation */
        public String idHolder;

        /** Nodes for tree building */
        @SuppressWarnings("unused")
        public TreeNode left, right, parent;
    }

    /**
     * Constructor for a node, only called by ExpressionTree
     *
     * @param leaf
     *          flag that says if the node is a leaf or not
     * @param operation
     *          set to + or & if not a leaf
     * @param value
     *          rCause associated to a leaf node, null if not leaf
     * @param idHolder
     *          holds an rCause id for when its time to fetch rCauses
     */
    public TreeNode(boolean leaf, char operation, rCause value,
                   String idHolder) {
```

```

        this.leaf = leaf;
        this.operation = operation;
        this.value = value;
        this.idHolder = idHolder;
        this.left = null;
        this.right = null;
        this.parent = null;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Object#toString()
     */
    public String toString() {
        return leaf ? Integer.toString(this.value.getID()) : Character
            .toString(this.operation);
    }
}

// end of node class //

/** The top node of the tree */
TreeNode root = null;

/** Context for using the DatabaseHandler */
private Context c;

/** Handler to get rCause objects for evaluation */
private DatabaseHandler db;

/**
 * Constructor for tree creation
 *
 * @param code
 *          formatted string that the tree will build itself with
 * @param c
 *          context for usage of application resources
 */
public ExpressionTree(String code, Context c) {
    this.c = c;
    root = build(code);
}

/**
 * Builds the tree based on the stored code.
 *
 * @param code
 *          string that the tree uses to build itself
 * @return root node of the tree
 */
private TreeNode build(String code) {
    root = null;
    TreeNode lNode = null;
    TreeNode rNode = null;
    TreeNode curNode = null;
}

```

```

char cur;
Stack<TreeNode> s = new Stack<TreeNode>();
String idHolder = "";

for (int i = 0; i < code.length(); i++) {
    cur = code.charAt(i);

    if (cur == ',' || cur == '$') {
        if (idHolder != "") {
            curNode = new TreeNode(true, '\0', null, idHolder);
            s.push(curNode);
            idHolder = "";
        }

        if (cur == '$') {
            if (!s.isEmpty()) {
                root = s.pop();
            }
            break;
        }
    } else if (cur == '+' || cur == '=') {
        curNode = new TreeNode(false, cur, null, "");

        if (!s.isEmpty()) {
            rNode = s.pop();
        }

        if (!s.isEmpty()) {
            lNode = s.pop();
        }

        if (rNode != null) {
            rNode.parent = curNode;
            curNode.right = rNode;
        }
        if (lNode != null) {
            lNode.parent = curNode;
            curNode.left = lNode;
        }

        s.push(curNode);
        curNode = null;
        idHolder = "";
    } else if (cur != '+' && cur != '=' && cur != ',' && cur != '$') {
        idHolder += cur;
    }
}

return root;

```

```

}

/**
 * Debug purposes, prints the tree
 */
public void printPreOrder() {
    printPreOrder(root);
    System.out.println();
}

/**
 * Debug purposes - prints the tree using recursive methods
 *
 * @param node
 *          starts with root and passes next node to print child values
 */
private void printPreOrder(TreeNode node) {
    if (node == null)
        return;
    if (node.leaf) {
        System.out.print(node.idHolder + " ");
    } else {
        System.out.print(node.operation + " ");
    }
    printPreOrder(node.left);
    printPreOrder(node.right);
}

/**
 * @return number of nodes in the tree
 */
public int getSize() {
    int x = getSize(root, 0);
    return x;
}

/**
 * Recursively counts number of nodes in a tree
 *
 * @param node
 *          used for recursive calling
 * @param count
 *          increments as nodes are visited
 * @return number of nodes in the tree
 */
private int getSize(TreeNode node, int count) {
    if (node == null)
        return count;
    else if (node.left == null && node.right == null)
        count += getSize(node.left, count++);
    count += getSize(node.right, count++);
}

return count;
}

/**

```

```

* Evaluates the tree, calls recursive function
*
* @param context
*         allows access to application resources
* @return boolean result
*/
public boolean evaluate(Context context) {
    c = context;
    return root == null ? false : evaluate(root);
}

/**
* Evaluates the tree and returns the result. Fills in the leaf nodes with
* rCauses for values and uses their isTrue() method to get a value for
* them.
*
* @param node
*         needed for recursive call
* @return boolean result
*/
private boolean evaluate(TreeNode node) {
    boolean result = false;

    if (node.leaf) {
        db = new DatabaseHandler(c);
        node.value = db.getRCauseByID(Integer.valueOf(node.idHolder));
        db.close();
        if (node.value != null) {
            result = node.value.isTrue(c);
        } else {
            result = false;
        }
    } else {
        boolean left, right;
        char operator = node.operation;

        if (node.left != null) {
            left = evaluate(node.left);
        } else {
            if (operator == '&') {
                left = true;
            } else {
                left = false;
            }
        }

        if (node.right != null) {
            right = evaluate(node.right);
        } else {
            if (operator == '&') {
                right = true;
            } else {
                right = false;
            }
        }

        switch (operator) {
        case '&':
            result = left & right;
        }
    }
}

```

```
        break;
    case '+':
        result = left | right;
        break;
    }
}
return result;
}
```

## General.java

```
package com.example.ceandroid;

import android.os.Bundle;
//import android.support.v4.app.Fragment;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;

/**
 * "General settings" page
 *
 * @author CEandroid SMU
 */
public class General extends Fragment {
    /**
     * General Settings Page is a fragment within Preferences
     *
     * @see android.app.Fragment#onCreateView(android.view.LayoutInflater,
     *      android.view.ViewGroup, android.os.Bundle)
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.activity_general, container, false);
        ListView hList = (ListView) v.findViewById(R.id.genList);
        // List Click Adapters
        hList.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                long arg3) {
                Toast.makeText(getActivity().getApplicationContext(),
                    "General Options not implemented", Toast.LENGTH_SHORT)
                    .show();
            }
        });
        return v;
    }
}
```

## CEbr.java

```
package com.example.ceandroid;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.text.format.Time;

/**
 * CEbr is the BroadcastReceiver that is run on the CEservice to handle OS
 * triggers This class is the start of the entire Rules Engine flow
 *
 * @author CEandroid SMU
 */
public class CEbr extends BroadcastReceiver {
    /**
     * Application Context
     */
    private Context mContext;
    /**
     * SMS Received String
     */
    private static final String SMS_RECEIVED = "android.provider.Telephony.SMS_
RECEIVED";
    /**
     * Stores whether this is the first connection
     */
    private static boolean firstConnect = false;

    /**
     * The onReceive function handles all of the messages from the operating
     * system. The rules engine is fired immediatley to minimize time in this
     * section.
     *
     * @see android.content.BroadcastReceiver#onReceive(android.content.Context
     *
     *      android.content.Intent)
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        mContext = context;
        String action = intent.getAction();
        String phoneState;
        Bundle b = intent.getExtras();
        if (b != null) {
            phoneState = b.getString(TelephonyManager.EXTRA_STATE);
        } else {
            phoneState = "...";
        }
    }
}
```

```

String message = "Broadcast intent detected " + intent.getAction();
System.out.println(message);
System.out.println("PhoneState is " + phoneState);
int wifiState = intent.getIntExtra(WifiManager.EXTRA_WIFI_STATE,
        WifiManager.WIFI_STATE_UNKNOWN);
// The minute has ticked
if (Intent.ACTION_TIME_TICK.equals(action)) {
    Time now = new Time();
    now.setToNow();
    String timeText = "" + now.hour + now.minute;
    RulesEngine r1 = new RulesEngine("time", mContext);
    System.out.println("time_tick" + timeText);
    r1.start();
} else if (action.equals(SMS_RECEIVED)) {
    RulesEngine r1 = new RulesEngine("textMessage", mContext);
    r1.start();
} else if (action.equals(WifiManager.WIFI_STATE_CHANGED_ACTION)) {
    // need to check for first state change only to avoid wifi message
    // echoes
    if (firstConnect) {
        if (wifiState == WifiManager.WIFI_STATE_DISABLED
            || wifiState == WifiManager.WIFI_STATE_ENABLED) {
            RulesEngine r1 = new RulesEngine("wifiStatus", mContext);
            r1.start();
            firstConnect = false;
        }
    } else {
        firstConnect = true;
    }
} else if (action
        .equals(WifiManager.SUPPLICANT_CONNECTION_CHANGE_ACTION)) {
    if (intent.getBooleanExtra(WifiManager.EXTRA_SUPPLICANT_CONNECTED,
        false)) {
        RulesEngine r1 = new RulesEngine("ssid", mContext);
        r1.start();
    }
} else if (phoneState != null) {
    System.out.println("Ringing");
    if (phoneState.equals("RINGING")) {
        RulesEngine r1 = new RulesEngine("phoneCall", mContext);
        r1.start();
    }
}
locationUpdates();
}

/**
 * locationUpdates requests updates based on the most accurate service that
 * is currently on GPS, Network, and Passive Providers
 */
private void locationUpdates() {
    LocationManager locManager = (LocationManager) mContext
        .getSystemService(Context.LOCATION_SERVICE);

    // Define a listener that responds to location updates
    LocationListener locationListener = new LocationListener() {

```

```

    /**
     * Triggers when the phone receives an updated location
     *
     * @see android.location.LocationListener#onLocationChanged(android.location.Location)
     */
    public void onLocationChanged(Location location) {
        LocationManager listLocManager = (LocationManager) mContext
            .getSystemService(Context.LOCATION_SERVICE);
        if (isBetterLocation(
            location,
            listLocManager
                .getLastKnownLocation(LocationManager.GPS_PROVIDER))) {
            RulesEngine r1 = new RulesEngine("location", mContext);
            r1.start();
        } else {
            if (isBetterLocation(
                location,
                listLocManager
                    .getLastKnownLocation(LocationManager.NETWORK_PROVIDER))) {
                RulesEngine r1 = new RulesEngine("location", mContext);
                r1.start();
            } else {
                if (isBetterLocation(
                    location,
                    listLocManager
                        .getLastKnownLocation(LocationManager.PASSIVE_PROVIDER))) {
                    RulesEngine r1 = new RulesEngine("location",
                        mContext);
                    r1.start();
                }
            }
        }
    }

    /**
     * Triggers when the status is changed
     *
     * @see android.location.LocationListener#onStatusChanged(java.lang.String,
     *      int, android.os.Bundle)
     */
    public void onStatusChanged(String provider, int status,
        Bundle extras) {

    }

    /**
     * Triggers when a location provider is enabled
     *
     * @see android.location.LocationListener#onProviderEnabled(java.lang.String)
     */
    public void onProviderEnabled(String provider) {

```

```

    }

    /**
     * Triggers when a location provider is disabled
     *
     * @see android.location.LocationListener#onProviderDisabled(java.lang.
String)
     */
    public void onProviderDisabled(String provider) {

    }

    // Is not two minutes
    private static final int TWO_MINUTES = 1000 * 15;

    /**
     * Determines whether one Location reading is better than the
     * current Location fix
     *
     * @param location
     *          The new Location that you want to evaluate
     * @param currentBestLocation
     *          The current Location fix, to which you want to compare
     *          the new one
     */
    protected boolean isBetterLocation(Location location,
        Location currentBestLocation) {
        if (currentBestLocation == null) {
            // A new location is always better than no location
            return true;
        }

        // Check whether the new location fix is newer or older
        long timeDelta = location.getTime()
            - currentBestLocation.getTime();
        boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
        boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
        boolean isNewer = timeDelta > 0;

        // If it's been more than two minutes since the current
        // location, use the new location
        // because the user has likely moved
        if (isSignificantlyNewer) {
            return true;
        }
        // If the new location is more than two minutes older, it must
        // be worse
        else {
            if (isSignificantlyOlder) {
                return false;
            }
        }

        // Check whether the new location fix is more or less accurate
        int accuracyDelta = (int) (location.getAccuracy() -
currentBestLocation

```

```

        .getAccuracy());
    boolean isLessAccurate = accuracyDelta > 0;
    boolean isMoreAccurate = accuracyDelta < 0;
    boolean isSignificantlyLessAccurate = accuracyDelta > 200;

    // Check if the old and new location are from the same provider
    boolean isFromSameProvider = isSameProvider(
        location.getProvider(),
        currentBestLocation.getProvider());

    // Determine location quality using a combination of timeliness
    // and accuracy
    if (isMoreAccurate) {
        return true;
    } else {
        if (isNewer && !isLessAccurate) {
            return true;
        } else {
            if (isNewer && !isSignificantlyLessAccurate
                && isFromSameProvider) {
                return true;
            }
        }
    }
    return false;
}

/**
 * isSameProvider checks to see if the providers are the same
 *
 * @param provider1
 * @param provider2
 * @return True if the provider is the same
 */
private boolean isSameProvider(String provider1, String provider2) {
    if (provider1 == null) {
        return provider2 == null;
    }
    return provider1.equals(provider2);
};

// Register the listener with the Location Manager to receive location
// updates
locManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0,
    0, locationListener);
locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
    locationListener);
locManager.requestLocationUpdates(LocationManager.PASSIVE_PROVIDER, 0,
    0, locationListener);
}
}

```

## EffectView.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import CEapi.Effect;
import CEapi.rCause;
import CEapi.rEffect;
import android.annotation.SuppressLint;
import android.app.ActionBar;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.app.TimePickerDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnDismissListener;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.support.v4.app.FragmentActivity;
import android.text.format.DateFormat;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TimePicker;
import android.widget.Toast;

/**
 * EffectView Class displays the list of Effects
 *
 * @author CEandroid SMU
 */
public class EffectView extends FragmentActivity {
    /**
     * Elements array holds and empty array that is later used for cause
     * information
     */
    static String[] Elements = new String[] { " ", " ", " ", " ", " ", " ", " ",
        " ", " ", " " };
    /**
     * List View used to list out the Effects
     */
    private ListView listView;
```

```

/**
 * Database Handler used to get the list of Effects
 */
private DatabaseHandler db;
/**
 * gPosition used to handle user actions on the list of Effects
 */
int gPosition = 0;
/**
 * The type of Effect that is selected
 */
int type = 0;
/**
 * List of Effect Objects
 */
private List<Effect> effects = new ArrayList<Effect>();
/**
 * Used to access global application variables
 */
private CEapp app;

/**
 * List of effects currently being used in the application This can be
 * removed if strings are used for the switch statement
 *
 * @author CEandroid SMU
 */
public enum eEnum {
    vibrate, toast, notification, sound, ringer
}

/**
 * OnCreate gets the list of Effects from the database and adds them to the
 * ListView
 *
 * @see android.support.v4.app.FragmentActivity#onCreate(android.os.Bundle)
 */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    db = new DatabaseHandler(this);

    setContentView(R.layout.item_list);

    // instantiate listView
    listView = (ListView) findViewById(R.id.fruitList);

    listView.setTextFilterEnabled(true);

    // Set up the action bar
    final ActionBar actionBar = getActionBar();
    if (actionBar != null) {
        // Turn on "up" navigation
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}

```

```

// Load the kind of list
app = (CEapp) getApplication();
makeList();
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_activated_1, Elements);

// set the adapter of each view with the same data
listView.setAdapter(adapter);

listView.setOnItemClickListener(new OnItemClickListener() {
    /*
     * (non-Javadoc)
     *
     * @see
     * android.widget.AdapterView.OnItemClickListener#onItemClick(android
     * .widget.AdapterView, android.view.View, int, long)
     */
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
        doSomething(position, type);
    }
});

if (app.edit) {
    if (app.editType) // cause
    {
        doSomething(app.editID, 2);
    } else // effect
    {
        doSomething(app.editID, 3);
    }
}
db.close();
}

/**
 * makeList() is what is used by the onCreate to create the list of Effects
 */
public void makeList() {
    effects = db.getAllEffects();
    Elements = new String[effects.size()];
    for (int i = 0; i < effects.size(); i++) {
        if (effects.get(i).getName().equals("Toast")) {
            Elements[i] = "Popup Text";
        } else {
            Elements[i] = effects.get(i).getName();
        }
    }
}

/**
 * doSomething is called when the user selects an Effect from the list
 *
 * @param position
 * @param type
 */

```

```

public void doSomething(int position, int type) {
    // Enums Switch
    switch (eEnum.valueOf(effects.get(position).getType())) {
        case notification: {
            final Dialog d = new Dialog(this);
            d.setContentView(R.layout.ce_dialog_notification);
            d.setTitle("New Notification");
            d.setOnDismissListener(new OnDismissListener() {
                /*
                 * (non-Javadoc)
                 *
                 * @see
                 * android.content.DialogInterface.OnDismissListener#onDismiss
                 * (android.content.DialogInterface)
                 */
                public void onDismiss(DialogInterface arg0) {
                    if (app.edit) {
                        Intent myIntent = new Intent(getApplicationContext(),
                            EditRule.class)
                            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        startActivity(myIntent);
                    }
                }
            });
            Button button = (Button) d.findViewById(R.id.button1);
            button.setOnClickListener(new OnClickListener() {
                /*
                 * (non-Javadoc)
                 *
                 * @see
                 * android.view.View.OnClickListener#onClick(android.view.View)
                 */
                public void onClick(View v) {

                    EditText edit = (EditText) d.findViewById(R.id.title);
                    String title = edit.getText().toString();
                    EditText edit2 = (EditText) d.findViewById(R.id.text);
                    String text = edit2.getText().toString();
                    d.dismiss();

                    rEffect re = new rEffect();
                    re.setName("Notification");
                    re.setRuleID(app.currentRule.getID());
                    re.setType("notification");
                    re.setEffectID(3);
                    String parameter = title + " " + text + " ";
                    re.setParameters(parameter);
                    app.currentRule.addREffect(re);
                    Intent myIntent = new Intent(getApplicationContext(),
                        EditRule.class)
                        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                    startActivity(myIntent);
                }
            });
            d.show();
        }
    }
}

```

```

    }
    break;
  case toast: {
    final Dialog d = new Dialog(this);
    d.setContentView(R.layout.ce_dialog_toast);
    d.setTitle("New Popup Text");
    d.setOnDismissListener(new OnDismissListener() {
      /*
       * (non-Javadoc)
       *
       * @see
       * android.content.DialogInterface.OnDismissListener#onDismiss
       * (android.content.DialogInterface)
       */
      public void onDismiss(DialogInterface arg0) {
        if (app.edit) {
          Intent myIntent = new Intent(getApplicationContext(),
              EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
          startActivity(myIntent);
        }
      }
    });
    Button button = (Button) d.findViewById(R.id.button1);
    button.setOnClickListener(new OnClickListener() {
      /*
       * (non-Javadoc)
       *
       * @see
       * android.view.View.OnClickListener#onClick(android.view.View)
       */
      public void onClick(View v) {
        EditText edit2 = (EditText) d.findViewById(R.id.text);
        String text = edit2.getText().toString();
        d.dismiss();

        rEffect re = new rEffect();
        re.setName("Toast");
        re.setRuleID(app.currentRule.getID());
        re.setType("toast");
        re.setEffectID(1);
        String parameter = "Text:" + text;
        re.setParameters(parameter);
        app.currentRule.addREffect(re);
        Intent myIntent = new Intent(getApplicationContext(),
            EditRule.class)
          .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
      }
    });
    d.show();
  }
  break;
  case vibrate: {
    rEffect re = new rEffect();

```

```

        re.setName("Vibrate");
        re.setRuleID(app.currentRule.getID());
        re.setType("vibrate");
        re.setEffectID(2);
        String parameter = "Long";
        re.setParameters("Tone Length:" + parameter);
        app.currentRule.addREffect(re);
        Intent myIntent = new Intent(this.getApplicationContext(),
            EditRule.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
    break;
case sound: {
    showFileChooser();
}
break;
case ringer: {
    Dialog d = onCreateRingerDialog();
    d.setOnDismissListener(new OnDismissListener() {
        /*
         * (non-Javadoc)
         *
         * @see
         * android.content.DialogInterface.OnDismissListener#onDismiss
         * (android.content.DialogInterface)
         */
        public void onDismiss(DialogInterface arg0) {
            if (app.edit) {
                Intent myIntent = new Intent(getApplicationContext(),
                    EditRule.class)
                    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(myIntent);
            }
        }
    });
    d.show();
}
break;
default:
    break;
}
db.close();
}

// Time Picker Dialog
@SuppressLint("ValidFragment")
public class TimePickerFragment extends DialogFragment implements
    TimePickerDialog.OnTimeSetListener {
/*
 * (non-Javadoc)
 *
 * @see android.app.DialogFragment#onCreateDialog(android.os.Bundle)
 */
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {

```

```

// Use the current time as the default values for the picker
final Calendar c = Calendar.getInstance();
int hour = c.get(Calendar.HOUR_OF_DAY);
int minute = c.get(Calendar.MINUTE);

// Create a new instance of TimePickerDialog and return it
return new TimePickerDialog(getActivity(), this, hour, minute,
    DateFormat.is24HourFormat(getActivity()));
}

/*
 * (non-Javadoc)
 *
 * @see
 * android.app.TimePickerDialog.OnTimeSetListener#onTimeSet(android.
 * widget.TimePicker, int, int)
 */
public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
    gPosition = 2;
    rCause rc = new rCause();
    rc.setName("Time");
    rc.setRuleID(app.currentRule.getID());
    rc.setType("time");
    rc.setCauseID(1);
    String parameter = "";
    if (hourOfDay < 10) {
        parameter = "0" + hourOfDay;
    } else {
        parameter = "" + hourOfDay;
    }
    parameter += ":";
    if (minute < 10) {
        rc.setParameters(parameter + "0" + minute);
    } else {
        rc.setParameters(parameter + minute);
    }
    app.currentRule.addRCause(rc);
    app.addedRCause = true;

    Intent myIntent = new Intent(getApplicationContext(),
        EditRule.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
}

/*
 * (non-Javadoc)
 *
 * @see
 * android.app.DialogFragment#onDismiss(android.content.DialogInterface)
 */
@Override
public void onDismiss(DialogInterface dialog) {
    if (app.edit) {
        Intent myIntent = new Intent(getApplicationContext(),
            EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

```

```

        startActivity(myIntent);
    }
}
}

/**
 * Called when the user is done setting parameters for a Effect and the
 * picker is returned to EffectView Creates rEffects to be associated with
 * the Rule based on the user's parameters
 *
 * @see android.support.v4.app.FragmentActivity#onActivityResult(int, int,
 *      android.content.Intent)
 */
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 3) {
        if (resultCode == Activity.RESULT_OK) {
            Uri contactData = data.getData();
            String fileName = "";
            String scheme = contactData.getScheme();

            if (scheme.equals("file")) {
                fileName = contactData.getLastPathSegment();
            } else if (scheme.equals("content")) {
                String[] proj = { MediaStore.Images.Media.TITLE };
                Cursor cursor = this.getContentResolver().query(
                    contactData, proj, null, null, null);
                if (cursor != null && cursor.getCount() != 0) {
                    int columnIndex = cursor
                        .getColumnIndexOrThrow(MediaStore.Images.Media.TITLE);
                    cursor.moveToFirst();
                    fileName = cursor.getString(columnIndex);
                }
            }
        }

        rEffect re = new rEffect();
        re.setName("Play Sound");
        re.setRuleID(app.currentRule.getID());
        re.setType("sound");
        re.setEffectID(4);
        String parameter = contactData.toString() + "\n" + fileName;
        re.setParameters(parameter);
        app.currentRule.addREffect(re);
        Intent myIntent = new Intent(this.getApplicationContext(),
            EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    } else {
        if (app.edit) {
            Intent myIntent = new Intent(this.getApplicationContext(),
                EditRule.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
        }
    }
}

```

```

        }

    if (app.edit) {
        Intent myIntent = new Intent(this.getApplicationContext(),
            EditRule.class).addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
}

private void showFileChooser() {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("audio/*");
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    try {
        startActivityForResult(
            Intent.createChooser(intent, "Select a File to Play"), 3);
    } catch (android.content.ActivityNotFoundException ex) {
        // Potentially direct the user to the Market with a Dialog
        Toast.makeText(this.getApplicationContext(),
            "Please install a File Manager.", Toast.LENGTH_SHORT)
            .show();
    }
}

public Dialog onCreateRingerDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    // Set the dialog title
    builder.setTitle("Ring Mode")
        .setItems(R.array.ring_mode,
            new android.content.DialogInterface.OnClickListener() {
                /*
                 * (non-Javadoc)
                 *
                 * @see
                 * android.content.DialogInterface.OnClickListener
                 * #onClick(android.content.DialogInterface, int)
                 */
                public void onClick(DialogInterface dialog,
                    int which) {
                    rEffect re = new rEffect();
                    re.setName("Ring Mode");
                    re.setRuleID(app.currentRule.getID());
                    re.setType("ringer");
                    re.setEffectID(5);
                    String parameter = "";
                    switch (which) {
                        case 0:
                            parameter = "normal";
                
```

```

        break;
    case 1:
        parameter = "vibrate";
        break;
    case 2:
        parameter = "silent";
        break;
    default:
        parameter = "normal";
        break;
    }
    re.setParameters("Ring mode: " + parameter);
    app.currentRule.addREffect(re);
    Intent myIntent = new Intent(
        getApplicationContext(), EditRule.class)
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
}
})
.setNegativeButton("Cancel",
    new DialogInterface.OnClickListener() {
    /*
     * (non-Javadoc)
     *
     * @see
     * android.content.DialogInterface.OnClickListener
     * #onClick(android.content.DialogInterface, int)
     */
    public void onClick(DialogInterface dialog, int id) {
        if (app.edit) {
            Intent myIntent = new Intent(
                getApplicationContext(),
                EditRule.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
        }
    }
});
return builder.create();
}

/**
 * EffectView Menu Creation
 *
 * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
 */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_edit_task, menu);
    return true;
}

/**
 * EffectView Menu Actions
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)

```

```

        */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case android.R.id.home: {
            // This is called when the Home (Up) button is pressed
            // in the Action Bar.
            Intent parentActivityIntent = new Intent(this, EditRule.class);
            parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                | Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(parentActivityIntent);
            finish();
            return true;
        }
        case R.id.menu_help: {
            Intent myIntent = new Intent(this, Help.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
            return true;
        }
        case R.id.menu_settings: {
            Intent myIntent = new Intent(this, Preferences.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
            return true;
        }
        default: {
            return super.onOptionsItemSelected(item);
        }
    }
}

```

## MainActivity.java

```
package com.example.ceandroid;

import java.util.ArrayList;

import CEapi.Cause;
import CEapi.Effect;
import CEapi.rCause;
import CEapi.rEffect;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.ActivityManager;
import android.app.ActivityManager.RunningServiceInfo;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity {
    /**
     * CEapp contains the globally accessible variables
     */
    private CEapp app;

    /**
     * Creates the Main Menu Starts the Database and CEservice, if necessary
     *
     * @see android.app.Activity#onCreate(android.os.Bundle)
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        app = (CEapp) this.getApplication();
        if (app.getListType() == 0) {
            new Thread(new Runnable() {
                /**
                 * The database is created in a separate thread This keeps the
                 * application responsive
                 *
                 * @see java.lang.Runnable#run()
                 */
                public void run() {
                    testDatabase();
                }
            }).start();
        }
    }

    /**
     * Occurs when the app is resumed Starts the service if it is not currently
```

```

    * running
    *
    * @see android.app.Activity#onResume()
    */
@Override
public void onResume() {
    super.onResume();
    if ((!isMyServiceRunning()) && app.getListType() != 0) {
        Intent service = new Intent(this, CEservice.class);
        this.startService(service);
    }

    if (app.currentRule != null) {
        if (app.currentRule.getRCauses().isEmpty()
            && app.currentRule.getREffects().isEmpty()) {
            DatabaseHandler db = new DatabaseHandler(this);
            db.deleteRule(app.currentRule);
            Toast.makeText(this, "Incomplete rule not saved.",
                Toast.LENGTH_LONG).show();
            app.currentRule = null;
            db.close();
        }
    }
}

/*
 * (non-Javadoc)
 *
 * @see android.app.Activity#onPause()
 */
@Override
public void onPause() {
    super.onPause();
}

/**
 * Used to evaluate the current state of the CEservice
 *
 * @return boolean The service is either running or not running
 */
private boolean isMyServiceRunning() {
    ActivityManager manager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);
    for (RunningServiceInfo service : manager
        .getRunningServices(Integer.MAX_VALUE)) {
        if (CEservice.class.getName()
            .equals(service.service.getClassName())) {
            return true;
        }
    }
    return false;
}

// Generates test data for the database
/**
 * Used to create the database Creates Causes, Effects, and default Rules

```

```

    * Once this function is completed, the service is started, and user action
s
    * are allowed
    */
@SuppressLint("SdCardPath")
public void testDatabase() {
    DatabaseHandler db = new DatabaseHandler(this);

    try {
        db.importDatabase("/data/data/com.example.ceandroid/databases/database.
db");
    } catch (Exception e) {
        db.onUpgrade(db.getWritableDatabase(), 1, 1);
        db.addCause(new Cause("time", "Time", "time reached", "time"));
        db.addCause(new Cause("phoneCall", "Phone Call",
                "incoming phone call", "phone"));
        db.addCause(new Cause("textMessage", "Text Message",
                "new text message", "message"));
        db.addCause(new Cause("ssid", "Wifi SSID", "ssid match", "wifi"));
        db.addCause(new Cause("wifiStatus", "Wifi Status", "wifi status",
                "wifi"));
        db.addCause(new Cause("location", "Arriving at a Location",
                "Triggers when you arrive at a location", "Location"));
        db.addCause(new Cause("location", "Departing a Location",
                "Triggers when you depart a location", "Location"));
        db.addEffect(new Effect("toast", "Toast",
                "Displays a toast message", "display"));
        db.addEffect(new Effect("vibrate", "Vibrate",
                "Activates the phone's vibration", "system"));
        db.addEffect(new Effect("notification", "Notification",
                "Adds a notification", "display"));
        db.addEffect(new Effect("sound", "Play Sound",
                "Activates the phone's vibration", "media"));
        db.addEffect(new Effect("ringer", "Ring Mode",
                "Silent, Vibrate, and Normal ring modes", "system"));

        ArrayList<rCause> ruleCauses = new ArrayList<rCause>();
        ruleCauses.add(new rCause(1, 1, "12:30", "time"));
        ArrayList<rEffect> ruleEffects = new ArrayList<rEffect>();
        ruleEffects.add(new rEffect(1, 3, "Wake up! Go to school.",
                "notification"));
        db.addRule(new Rule("1$", "Alarm", ruleCauses, ruleEffects));

        ruleCauses.clear();
        ruleEffects.clear();
        ruleCauses.add(new rCause(2, 1, "12:30", "time"));
        ruleCauses.add(new rCause(2, 2, "Nathan R Huntoon", "phone call"));
        ruleEffects.add(new rEffect(2, 2, "Tone Length:Long", "vibrate"));
        ruleEffects.add(new rEffect(2, 3,
                "Call from Nathan! Hangup Quick!", "notification"));
        db.addRule(new Rule("2,3,$", "Call me maybe?", ruleCauses,
                ruleEffects));

        ruleCauses.clear();
        ruleEffects.clear();
        ruleCauses.add(new rCause(3, 1, "12:30", "time"));
    }
}

```

```

ruleCauses.add(new rCause(3, 2, "Matt Rispoli", "phone call"));
ruleEffects.add(new rEffect(3, 2, "Tone Length:Long", "vibrate"));
ruleEffects
    .add(new rEffect(3, 3,
        "Call from Matt!'Probably about Physics.'",
        "notification"));
db.addRule(new Rule("4,5,$", "Called me.", ruleCauses, ruleEffects));

ruleCauses.clear();
ruleEffects.clear();
ruleCauses.add(new rCause(4, 4, "PerunaNet", "ssid"));
ruleCauses.add(new rCause(4, 1, "12:00", "time"));
ruleEffects.add(new rEffect(4, 3, "I am connected to PerunaNet!",
    "notification"));
db.addRule(new Rule("6,7,$", "SSID test", ruleCauses, ruleEffects));
}

app.setListType(-1);
db.close();

Intent service = new Intent(this, CEservice.class);
this.startService(service);
}

/**
 * Called when the user tries to access the MyRules Activity Allows access
 * unless the database and service aren't ready yet
 *
 * @param v
 */
public void myRules(View v) {
    if (app.getListType() != 0) {
        Intent myIntent = new Intent(this, MyRules.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    } else {
        loading();
    }
}

/**
 * Called when the user tries to access the Help Activity Allows access
 * unless the database and service aren't ready yet
 *
 * @param v
 */
public void help(View v) {
    if (app.getListType() != 0) {
        Intent myIntent = new Intent(this, Help.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    } else {
        loading();
    }
}

/**

```

```

 * Called when the user tries to access the Preferences Activity Allows
 * access unless the database and service aren't ready yet
 *
 * @param v
 */
public void settings(View v) {
    if (app.getListType() != 0) {
        Intent myIntent = new Intent(this, Preferences.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    } else {
        loading();
    }
}

/**
 * Called when the user tries to create a new rule (EditRule Activity)
 * Allows access unless the database and service aren't ready yet
 *
 * @param v
 */
public void newRule(View v) {
    if (app.getListType() != 0) {
        // New Rule
        CEapp app = (CEapp) this.getApplication();
        app.currentRule = new Rule();

        Intent myIntent = new Intent(this, EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    } else {
        loading();
    }
}

/**
 * Called when the user tries to access the ShareList Activity Allows acces
s
 * unless the database and service aren't ready yet
 *
 * @param v
 */
public void share(View v) {
    if (app.getListType() != 0) {
        PackageManager p = this.getPackageManager();
        if (!p.hasSystemFeature("android.hardware.nfc")) {
            Toast.makeText(this, "NFC is not available on this device.",
                Toast.LENGTH_LONG).show();
        } else {
            NfcAdapter na = NfcAdapter.getDefaultAdapter(this);
            if (na.isEnabled()) {
                Intent myIntent = new Intent(this, ShareList.class)
                    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(myIntent);
            } else {
                Toast.makeText(this,

```

```
        "Please enable NFC to use this feature.",
        Toast.LENGTH_LONG).show();
    }
}
} else {
    loading();
}
}


/***
 * Called when the user tries to access another screen before the database
 * and service are ready
 *
 * @param v
 */


public void loading() {
    Toast.makeText(this.getApplicationContext(), "Loading Test Database",
        Toast.LENGTH_SHORT).show();
}
}
```

## DeleteDialogFragment.java

```
package com.example.ceandroid;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;

/**
 * Delete Dialog displayed when an item is to be deleted from a List
 *
 * @author CEandroid SMU
 */
public class DeleteDialogFragment extends DialogFragment {

    /**
     * Creates the Delete Dialog
     *
     * @see android.app.DialogFragment#onCreateDialog(android.os.Bundle)
     */
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        CEapp app = (CEapp) this.getActivity().getApplication();
        // Set the dialog title
        if (app.editType)
            builder.setMessage(R.string.delete_confirmation_cause);
        else
            builder.setMessage(R.string.delete_confirmation_effect);
        builder.setTitle(R.string.delete)
                // Set the action buttons
                .setPositiveButton(R.string.delete_confirm,
                        new DialogInterface.OnClickListener() {
                            /**
                             * Delete Confirmed
                             *
                             * @see android.content.DialogInterface.OnClickListener#onClick
                             (android.content.DialogInterface,
                             *         int)
                             */
                            public void onClick(DialogInterface dialog, int id) {
                                // Send the positive button event back to the
                                // host activity
                                mListener
                                        .onDialogPositiveClick(DeleteDialogFragment.this);
                            }
                        })
                .setNegativeButton(R.string.cancel,
                        new DialogInterface.OnClickListener() {
                            /**
                             * Deleted Cancelled
                             */
                        });
        return builder.create();
    }
}
```

```

        * @see android.content.DialogInterface.OnClickListener#onClick
(android.content.DialogInterface,
        *      int)
        */
    public void onClick(DialogInterface dialog, int id) {
        // Send the negative button event back to the
        // host activity
        mListener
            .onDialogNegativeClick(DeleteDialogFragment.this);
    }
});

return builder.create();
}

/**
 * Delete Dialog Listener Interface
 *
 * @author CEandroid SMU
 */
public interface DeleteDialogListener {
    public void onDialogPositiveClick(DialogFragment dialog);

    public void onDialogNegativeClick(DialogFragment dialog);
}

/**
 * Delete Dialog Listener implements the DeleteDialogListener Interface
 */
DeleteDialogListener mListener;

/**
 * Attaches the Dialog to the Activity
 *
 * @see android.app.DialogFragment#onAttach(android.app.Activity)
 */
@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    // Verify that the host activity implements the callback interface
    try {
        // Instantiate the NoticeDialogListener so we can send events to the
        // host
        mListener = (DeleteDialogListener) activity;
    } catch (ClassCastException e) {
        // The activity doesn't implement the interface, throw exception
        throw new ClassCastException(activity.toString()
            + " must implement DeleteDialogListener");
    }
}
}

```

## MyRules.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import java.util.List;

import android.app.ActionBar;
import android.app.DialogFragment;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.res.Resources;
import android.nfc.NfcAdapter;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemLongClickListener;

/**
 * MyRules Class displays the list of Rules
 *
 * @author CEandroid SMU
 */
public class MyRules extends FragmentActivity implements
    DeleteDialogFragmentRule.DeleteDialogListenerRule {
    /**
     * Elements array holds and empty array that is later used for Rule
     * information
     */
    static String[] Elements = new String[] { " ", " ", " ", " ", " ", " ",
        " ", " ", " " };
    /**
     * List View used to list out the Rules
     */
    private ListView listView;
    /**
     * Database Handler used to get the list of Rules
     */
    private DatabaseHandler db;
    /**
     * gPosition used to handle user actions on the list of Rules
     */
    int gPosition = 0;
    /**
     * The type of Rule that is selected
     */
    int type = 0;
    /**
     * List of Rule Objects
```

```

        */
    private List<Rule> rules = new ArrayList<Rule>();
    /**
     * Used to access global application variables
     */
    private CEapp app;

    /**
     * Creates the MyRules Activity and loads the list of Rules from the
     * database
     *
     * @see android.support.v4.app.FragmentActivity#onCreate(android.os.Bundle)
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.item_list);
        @SuppressWarnings("unused")
        Resources res = getResources();
        listView = (ListView) findViewById(R.id.fruitList);
        listView.setTextFilterEnabled(true);
        db = new DatabaseHandler(this);

        // Load the kind of list
        app = (CEapp) getApplication();
        app.edit = false;

        if (app.currentRule != null) {
            if (app.currentRule.getRCauses().isEmpty()
                && app.currentRule.getREffects().isEmpty()) {
                db.deleteRule(app.currentRule);
                Toast.makeText(this, "Incomplete rule not saved.",
                    Toast.LENGTH_LONG).show();
                app.currentRule = null;
            }
        }

        makeList();
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_activated_1, Elements);

        // Set up the action bar
        final ActionBar actionBar = getActionBar();
        if (actionBar != null) {
            // Turn on "up" navigation
            actionBar.setDisplayHomeAsUpEnabled(true);
        }

        // set the adapter of each view with the same data
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(new OnItemClickListener() {
            /*
             * (non-Javadoc)
             *
             * @see
             * android.widget.AdapterView.OnItemClickListener#onItemClick(android

```

```

        * .widget.AdapterView, android.view.View, int, long)
    */
public void onItemClick(AdapterView<?> parent, View view,
    int position, long id) {
    doSomething(position, type);
}
});
listView.setOnItemClickListener(new OnItemLongClickListener() {
/*
 * (non-Javadoc)
 *
 * @see
 * android.widget.AdapterView.OnItemLongClickListener#onItemLongClick
 * (android.widget.AdapterView, android.view.View, int, long)
 */
public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
    int arg2, long arg3) {
    app.editedNumber = rules.get(arg2).getID();
    DialogFragment newFragment = new DeleteDialogFragmentRule();
    newFragment.show(getFragmentManager(), "delete");
    return true;
}
});
db.close();
}

/*
 * (non-Javadoc)
 *
 * @see android.support.v4.app.FragmentActivity#onResume()
 */
@Override
public void onResume() {
    super.onResume();
    CEapp app = (CEapp) this.getApplication();
    if (app.currentRule != null) {
        if (app.currentRule.getRCauses().isEmpty()
            && app.currentRule.getREffects().isEmpty()) {
            DatabaseHandler db = new DatabaseHandler(this);
            db.deleteRule(app.currentRule);
            Toast.makeText(this, "Incomplete rule not saved.",
                Toast.LENGTH_LONG).show();
            app.currentRule = null;
            db.close();
        }
    }
}

/**
 * makeList() is what is used by the onCreate to create the list of Rules
 */
public void makeList() {
    rules = db.getAllRules();
    Elements = new String[rules.size()];
    for (int i = 0; i < rules.size(); i++) {
        Elements[i] = rules.get(i)._name;
}

```

```

        }

    }

    /**
     * doSomething is called when the user selects a Rule from the list
     *
     * @param position
     * @param type
     */
    public void doSomething(int position, int type) {
        app.currentRule = rules.get(position);
        app.currentRule.setRCauses(db.getAllRCauses(app.currentRule.getID()));
        app.currentRule.setREffects(db.getAllREffects(app.currentRule.getID()));

        // Calls the Edit Text
        Intent myIntent = new Intent(getApplicationContext(), EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
        db.close();
    }

    /**
     * MyRules Menu Creation
     *
     * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
     */
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_my_rules, menu);
        return true;
    }

    /**
     * MyRules Menu Actions
     *
     * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
     */
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
        switch (item.getItemId()) {
            case android.R.id.home: {
                // This is called when the Home (Up) button is pressed
                // in the Action Bar.
                Intent parentActivityIntent = new Intent(this, MainActivity.class);
                parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                    | Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(parentActivityIntent);
                finish();
                return true;
            }
            case R.id.menu_new_rule: {
                // New Rule
                CEapp app = (CEapp) this.getApplication();
                app.currentRule = new Rule();
                /*

```

```

        * DatabaseHandler db = new DatabaseHandler(this);
        * app.currentRule.setID(db.getRulesCount() + 1); db.close();
        */

Intent myIntent = new Intent(this, EditRule.class)
    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(myIntent);
return true;
}

case R.id.menu_share: {
    PackageManager p = this.getPackageManager();
    if (!p.hasSystemFeature("android.hardware.nfc")) {
        Toast.makeText(this, "NFC is not available on this device.",
            Toast.LENGTH_LONG).show();
    } else {
        NfcAdapter na = NfcAdapter.getDefaultAdapter(this);
        if (na.isEnabled()) {
            Intent myIntent = new Intent(this, ShareList.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
        } else {
            Toast.makeText(this,
                "Please enable NFC to use this feature.",
                Toast.LENGTH_LONG).show();
        }
    }
    return true;
}
case R.id.menu_help: {
    Intent myIntent = new Intent(this, Help.class)
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
    return true;
}
case R.id.menu_settings: {
    Intent myIntent = new Intent(this, Preferences.class)
        .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(myIntent);
    return true;
}
default: {
    return super.onOptionsItemSelected(item);
}
}
}

/***
 * onDialogPositiveClick is called when a Rule is confirmed to be deleted b
y
 * the user
 *
 * @see com.example.ceandroid.DeleteDialogFragmentRule.DeleteDialogListener
Rule#onDialogPositiveClick(android.app.DialogFragment)
 */
public void onDialogPositiveClick(DialogFragment dialog) {
    // clicked to delete the cause or effect

```

```
DatabaseHandler db = new DatabaseHandler(this);
CEapp app = (CEapp) this.getApplication();
db.deleteRule(db.getRule(app.editedNumber));
db.close();
Intent intent = getIntent().addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
finish();
startActivity(intent);
}

/**
 * onDialogNegativeClick is called when a Rule delete is cancelled
 *
 * @see com.example.ceandroid.DeleteDialogFragmentRule.DeleteDialogListener
Rule#onDialogNegativeClick(android.app.DialogFragment)
 */
public void onDialogNegativeClick(DialogFragment dialog) {
    // clicked cancel, do not do anything
}
}
```

## CEservice.java

```
package com.example.ceandroid;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.WifiManager;
import android.os.IBinder;

/**
 * Service to handle device state changes and evaluate rules accordingly
 *
 * @author CEandroid SMU
 */
public class CEservice extends Service {
    /** Broadcast receiver to listen for state changes */
    CEbr br = new CEbr();

    /**
     * @see android.app.Service#onStartCommand(android.content.Intent, int, int)
     */
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return Service.START_NOT_STICKY;
    }

    /**
     * @see android.app.Service#onBind(android.content.Intent)
     */
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    /**
     * Registers the BroadcastReceiver (and the entire rules engine) from the
     * service Uses IntentFilters to declare trigger types
     *
     * @see android.app.Service#onCreate()
     */
    @Override
    public void onCreate() {
        super.onCreate();

        IntentFilter filter = new IntentFilter();

        filter.addAction("android.provider.Telephony.SMS_RECEIVED");
        filter.addAction(Intent.ACTION_TIME_TICK);
        filter.addAction(Intent.ACTION_TIME_CHANGED);
        filter.addAction(Intent.ACTION_NEW_OUTGOING_CALL);
        filter.addAction("android.intent.action.PHONE_STATE");
        filter.addAction("android.net.wifi.WIFI_STATE_CHANGED");
        filter.addAction(WifiManager.SUPPLICANT_CONNECTION_CHANGE_ACTION);
```

```
    registerReceiver(br, filter);
}

/***
 * Unregisters the CEbr
 *
 * @see android.app.Service#onDestroy()
 */
@Override
public void onDestroy() {
    super.onDestroy();
    unregisterReceiver(br);
}
}
```

## Rule.java

```
package com.example.ceandroid;

import java.util.ArrayList;
import java.util.List;

import CEapi.rCause;
import CEapi.rEffect;

/**
 * The Rule classic is the underlying data structure for the entire program
 * Causes and Effects, as well as other rule information, is stored within this
 * object
 *
 * @author CEandroid SMU
 */
public class Rule {

    /**
     * The Rule's Unique Identifier
     */
    int _id;
    /**
     * _treeData Stores the boolean logic and Causes in a String _name Stores
     * the Rule's name
     */
    String _treeData, _name;
    /**
     * Contains the List of rCauses
     */
    ArrayList<rCause> _ruleCauses;
    /**
     * Contains the List of rEffects
     */
    ArrayList<rEffect> _ruleEffects;
    /**
     * Stores whether or not the rule is currently active
     */
    boolean Active;
    /**
     * Stores whether or not the rule is edited and needs to be updated in the
     * Database
     */
    boolean edited = false;

    /**
     * Default Constructor Creates a new rule
     */
    public Rule() {
        this._ruleCauses = new ArrayList<rCause>();
        this._ruleEffects = new ArrayList<rEffect>();
        Active = true;
        edited = false;
        this._treeData = "";
    }
}
```

```

        this._name = "Untitled";
    }

    /**
     * Defined Rule Constructor without the Rule's ID, rCauses, and rEffects
     *
     * @param treeData
     * @param name
     * @param Active
     */
    public Rule(String treeData, String name, boolean Active) {
        this.treeData = treeData;
        if (name == null)
            this._name = "Untitled";
        else
            this._name = name;
        this.Active = Active;
        this._ruleCauses = null;
        this._ruleEffects = null;
        this._ruleCauses = new ArrayList<rCause>();
        this._ruleEffects = new ArrayList<rEffect>();
        edited = false;
    }

    /**
     * Defined Rules Constructor without the Rule's ID
     *
     * @param treeData
     * @param name
     * @param ruleCauses
     * @param ruleEffects
     */
    public Rule(String treeData, String name, ArrayList<rCause> ruleCauses,
                 ArrayList<rEffect> ruleEffects) {
        this.treeData = treeData;
        if (name == null)
            this._name = "Untitled";
        else
            this._name = name;
        this.Active = true;
        this._ruleCauses = ruleCauses;
        this._ruleEffects = ruleEffects;
        edited = false;
    }

    /**
     * Get _id
     *
     * @return this._id
     */
    public int getID() {
        return this._id;
    }

    /**
     * Set _id
     */

```

```

*
 * @param id
 */
public void setID(int id) {
    this._id = id;
}

/**
 * Get _treeData
 *
 * @return this._treeData
 */
public String getTreeData() {
    return this._treeData;
}

/**
 * Set _treeData
 *
 * @param treeData
 */
public void setTreeData(String treeData) {
    this._treeData = treeData;
}

/**
 * Get's the BooleanSequence used for rCauses
 *
 * @return ArrayList<EditRuleNode> The list of Booleans
 */
public ArrayList<EditRuleNode> getBoolSequence() {
    ArrayList<EditRuleNode> bools = new ArrayList<EditRuleNode>();
    int cCounter = 0;
    boolean aoFlag = false;
    ArrayList<Integer> andLocs = new ArrayList<Integer>();
    char cur = ' ';

    for (int i = 0; i < this._treeData.length(); i++) {
        cur = this._treeData.charAt(i);

        if (cur == '+' || cur == '&') {
            bools.add(new EditRuleNode("", false, true));
            if (cur == '&') {
                andLocs.add(cCounter - 2);
            }
            aoFlag = true;
        } else if (cur == ',') {
            if (aoFlag) {
                aoFlag = false;
            } else {
                cCounter++;
            }
        }
    }

    for (int i = 0; i < andLocs.size(); i++) {

```

```

        int loc = andLocs.get(i);
        bools.get(loc).value = "AND";
    }

    for (int i = 0; i < bools.size(); i++) {
        if (!bools.get(i).value.equals("AND")) {
            bools.get(i).value = "OR";
        }
    }

    return bools;
}

/**
 * Updates the the treeData using a new list of booleans
 *
 * @param bools
 */
public void updateTreeData(List<EditRuleNode> bools) {
    String tree = "";
    rCause cur;
    boolean andFlag = false;
    if (_ruleCauses.size() > 0) {
        if (bools.isEmpty()) {
            for (int i = 0; i < _ruleCauses.size(); i++) {
                tree += Integer.toString(_ruleCauses.get(i).getID());
            }
        } else {
            for (int i = 0; i < _ruleCauses.size(); i++) {
                cur = _ruleCauses.get(i);
                if (i == bools.size()) {
                    tree += Integer.toString(cur.getID()) + ',';
                    if (andFlag == true) {
                        tree += "&";
                        andFlag = false;
                    }
                } else {
                    if (andFlag == true) {
                        tree += (cur.getID() + ",&,");
                        if (i < bools.size())
                            && bools.get(i).value.equals("OR"))
                            andFlag = false;
                    }
                }
            }
        }
    } else if (i < bools.size()
        && bools.get(i).value.equals("AND")) {
        tree += Integer.toString(cur.getID()) + ',';
        andFlag = true;
    }

    else if (i < bools.size()
        && bools.get(i).value.equals("OR")) {
        tree += Integer.toString(cur.getID()) + ',';
        andFlag = false;
    }
}

```

```

        }
    }

    for (int i = 0; i < bools.size(); i++) {
        if (bools.get(i).value.equals("OR")) {
            tree += "+,";
        }
    }
} // end of else

// dollar sign represents end of string
tree += "$";
}

System.out.println("Tree is " + tree);
setTreeData(tree);
}

return this._name;
}

true;
}

return this.Active;
}

boolean active) {
    this.Active = active;
    edited = true;
}

```

```

    /**
     * Set _ruleCauses
     *
     * @param ruleCauses
     */
    public void setRCauses(ArrayList<rCause> ruleCauses) {
        this._ruleCauses = ruleCauses;
    }

    /**
     * Add rCause to _ruleCauses
     *
     * @param cause
     */
    public void addRCause(rCause cause) {
        this._ruleCauses.add(cause);
        edited = true;
    }

    /**
     * Get _ruleCauses
     *
     * @return this._ruleCauses
     */
    public ArrayList<rCause> getRCauses() {
        return this._ruleCauses;
    }

    /**
     * Set _ruleEffects
     *
     * @param ruleEffects
     */
    public void setREffects(ArrayList<rEffect> ruleEffects) {
        this._ruleEffects = ruleEffects;
    }

    /**
     * Add rEffect to _ruleEffects
     *
     * @param effect
     */
    public void addREffect(rEffect effect) {
        this._ruleEffects.add(effect);
        edited = true;
    }

    /**
     * Get _ruleEffects
     *
     * @return this._ruleEffects
     */
    public ArrayList<rEffect> getREffects() {
        return this._ruleEffects;
    }
}

```

## EditRule.java

```
package com.example.ceandroid;

import java.util.ArrayList;

import CEapi.rCause;
import CEapi.rEffect;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.app.DialogFragment;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemLongClickListener;
import android.widget.ArrayAdapter;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.RelativeLayout;
import android.widget.Switch;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;

/**
 * The EditRule class is the main Activity for rule creation. This Activity is
 * used for creating, editing, and viewing rules. The Rule creation constantly
 * comes back to this screen to show the user the progress on the rule creation.
 *
 * The rCauses, rCause boolean logic, rEffects, Rule Name, and Rule Active
 * Status are all displayed on this Activity
 *
 * @author CEAndroid SMU
 */
public class EditRule extends Activity implements
    DeleteDialogFragment.DeleteDialogListener {
    /**
     * CEapp contains the globally accessible variables
     */
    private CEapp app = (CEapp) this.getApplication();
    /**
     * The type of rCause or rEffect used for Enums
     */
    int type = 0;
    /**
     * Generic size variable
     */
    static int size = 0;
    /**
     * The position of the rCause or rEffect that is to be deleted
     */
    static int delPos = 0;
```

```

/**
 * The list of rCauses
 */
ArrayList<EditRuleNode> cList = new ArrayList<EditRuleNode>();
/**
 * The list of ANDs and ORs for rCauses
 */
ArrayList<EditRuleNode> bools = new ArrayList<EditRuleNode>();
/**
 * The list of rEffects
 */
ArrayList<String> eList = new ArrayList<String>();
/**
 * The current rCause being updated
 */
ArrayAdapter<EditRuleNode> cA;
/**
 * The current rEffect being updated
 */
ArrayAdapter<String> eA;
/**
 * The ListView containing all of the rCauses
 */
ListView cListView;
/**
 * The ListView containing all of the rEffects
 */
ListView eListView;
/**
 * TextWatcher updates the Rules named as it is changed
 */
TextWatcher textWatcher = new TextWatcher() {

    /**
     * As the name is changing
     *
     * @see android.text.TextWatcher#onTextChanged(java.lang.CharSequence,
     *      int, int, int)
     */
    public void onTextChanged(CharSequence s, int start, int before,
        int count) {

    }

    /**
     * Just before the name is changed
     *
     * @see android.text.TextWatcher#beforeTextChanged(java.lang.CharSequence
     *
     *      int, int, int)
     */
    public void beforeTextChanged(CharSequence s, int start, int count,
        int after) {

    }
}

```

```

    /**
     * After the name has been changed The rule name is updated at this time
     *
     * @see android.text.TextWatcher#afterTextChanged(android.text.Editable)
     */
    public void afterTextChanged(Editable s) {
        app.currentRule.setName(s.toString());
        DatabaseHandler db = new DatabaseHandler(getApplicationContext());
        db.updateName(app.currentRule.getID(), app.currentRule.getName());
        db.close();
    }
};

/**
 * When the Activity is resumed, the textWatcher is added again
 *
 * @see android.app.Activity#onResume()
 */
@Override
public void onResume() {
    super.onResume();
    ((EditText) this.findViewById(R.id.textView1))
        .addTextChangedListener(textWatcher);
}

/**
 * Loads all rule information for a new rule or a specific rule in the
 * database This include rCauses, rCause boolean logic, rEffects, Rule Name
 *
 * and Rule Active Status
 *
 * @see android.app.Activity#onCreate(android.os.Bundle)
 */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_edit_rule);

    // Turn on "up" navigation
    getActionBar().setDisplayHomeAsUpEnabled(true);

    // Reset Focus
    RelativeLayout myLayout = (RelativeLayout) this
        .findViewById(R.id.relativeEditRule);
    myLayout.requestFocus();

    // Create Lists
    cListView = (ListView) findViewById(R.id.causes);
    eListView = (ListView) findViewById(R.id.effects);
    cA = new ArrayAdapter<EditRuleNode>(this,
        android.R.layout.simple_list_item_activated_1,
        android.R.id.text1, cList);
    eA = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_activated_1,
        android.R.id.text1, eList);
    cListView.setAdapter(cA);
}

```

```

eListView.setAdapter(eA);

// List Click Adapters
cListView.setOnItemClickListener(new OnItemClickListener() {
    /**
     * Cause List Item Clicked Calls the CauseView page for this Cause
     *
     * @see android.widget.AdapterView.OnItemClickListener#onItemClick(android.widget.AdapterView,
     *      android.view.View, int, long)
     */
    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
            long arg3) {
        EditRuleNode comp = (EditRuleNode) arg0.getAdapter().getItem(
                arg2);
        size = app.currentRule.getRCauses().size();
        // int size = arg0.getAdapter().getCount();
        app.editType = true;
        // if(size == (arg2+1))
        if (comp.isPlus()) {
            app.edit = false;
            Intent myIntent = new Intent(getApplicationContext(),
                    CauseView.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
        } else if (comp.isBool()) {
            DatabaseHandler db = new DatabaseHandler(
                    getApplicationContext());
            if (comp.value.equals("OR")) {
                comp.value = "AND";
                bools.get(arg2 / 2).value = "AND";
                app.currentRule.updateTreeData(bools);
                cA.notifyDataSetChanged();
            } else {
                comp.value = "OR";
                bools.get(arg2 / 2).value = "OR";
                app.currentRule.updateTreeData(bools);
                cA.notifyDataSetChanged();
            }
            db.updateRule(app.currentRule);
            db.close();
        } else {
            delPos = arg2 / 2;
            app.edit = true;
            app.editID = convertTypeToCEnum(app.currentRule
                    .getRCauses().get(arg2 / 2).getType(),
                    app.currentRule.getRCauses().get(arg2 / 2)
                    .getCauseID());
            app.editedNumber = app.currentRule.getRCauses()
                    .get(arg2 / 2).getID();
            Intent myIntent = new Intent(getApplicationContext(),
                    CauseView.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
        }
    }
}

```

```

});  

cListView.setOnItemLongClickListener(new OnItemLongClickListener() {  

    /**  

     * Called when a Cause has a long tap Delete dialog is called where  

     * appropriate  

     *  

     * @see android.widget.AdapterView.OnItemLongClickListener#onItemLongCl  

ick(android.widget.AdapterView,  

     *      android.view.View, int, long)  

    */  

    public boolean onItemLongClick(AdapterView<?> arg0, View arg1,  

        int arg2, long arg3) {  

        // int size = arg0.getAdapter().getCount();  

        EditRuleNode comp = (EditRuleNode) arg0.getAdapter().getItem(  

            arg2);  

        // if(size == (arg2+1))  

        if (comp.isCause()) {  

            app.editType = true;  

            delPos = arg2 / 2;  

            app.editedNumber = app.currentRule.getRCauses()  

                .get(arg2 / 2).getID();  

            DialogFragment newFragment = new DeleteDialogFragment();  

            newFragment.show(getFragmentManager(), "delete");  

        }  

        return true;  

    }  

});  

eListView.setOnItemClickListener(new OnItemClickListener() {  

    /**  

     * Effect List Item Clicked Calls the EffectView page for this  

     * Effect  

     *  

     * @see android.widget.AdapterView.OnItemClickListener#onItemClick(android.widget.AdapterView,  

     *      android.view.View, int, long)  

    */  

    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,  

        long arg3) {  

        int size = arg0.getAdapter().getCount();  

        app.editType = false;  

        if (size == (arg2 + 1)) {  

            app.edit = false;  

            Intent myIntent = new Intent(getApplicationContext(),  

                EffectView.class)  

                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);  

            startActivity(myIntent);  

        } else {  

            app.edit = true;  

            app.editID = convertTypeToEnum(app.currentRule  

                .getREffects().get(arg2).getType());  

            app.editedNumber = app.currentRule.getREffects().get(arg2)  

                .getID();  

            if (app.currentRule.getREffects().get(arg2).getType()  

                .equals("vibrate")) {  

                Toast.makeText(getApplicationContext(),  

                    "Cannot edit vibrate effects",

```

```

        Toast.LENGTH_LONG).show();
    } else {
        Intent myIntent = new Intent(getApplicationContext(),
            EffectView.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
}
}
});
eListView.setOnItemLongClickListener(new OnItemLongClickListener() {
    /**
     * Called when a Effect has a long tap Delete dialog is called where
     * appropriate
     *
     * @see android.widget.AdapterView.OnItemLongClickListener#onItemLongCl
     ick(android.widget.AdapterView,
         *      android.view.View, int, long)
     */
    public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
        int arg2, long arg3) {
        int size = arg0.getAdapter().getCount();
        if (size == (arg2 + 1)) {
            // do nothing, '+' was selected
        } else {
            app.editType = false;
            app.editedNumber = app.currentRule.getREffects().get(arg2)
                .getID();
            DialogFragment newFragment = new DeleteDialogFragment();
            newFragment.show(getFragmentManager(), "delete");
        }
        return true;
    }
});
// Load the kind of list
app = (CEapp) this.getApplication();

bools = app.currentRule.getBoolSequence();

if (app.edit) // editing a cause or effect
{
    DatabaseHandler db = new DatabaseHandler(this);
    if (app.editType) // Cause
    {
        size = app.currentRule.getRCauses().size();
        bools = app.currentRule.getBoolSequence();

        // added check - if > 0 then can't get negative index
        if (size > 0) {
            app.currentRule.getRCauses()
                .get(app.currentRule.getRCauses().size() - 1)
                .setID(app.editedNumber);
            db.updateRCause(
                app.currentRule.getRCauses().get(
                    app.currentRule.getRCauses().size() - 1),

```

```

        app.currentRule);
    if (db.duplicateRCauses(app.currentRule)) {
        size = app.currentRule.getRCauses().size();
        db.deleteRCause(db.getRCauseByID(app.editedNumber));
    }
}
} else // Effect
{
    int eSize = app.currentRule.getREffects().size();

    // added check - if > 0 then can't get negative index
    if (eSize > 0) {
        app.currentRule.getREffects()
            .get(app.currentRule.getREffects().size() - 1)
            .setID(app.editedNumber);
        db.updateREffect(
            app.currentRule.getREffects().get(
                app.currentRule.getREffects().size() - 1),
            app.currentRule);
        if (db.duplicateREffects(app.currentRule)) {
            db.deleteREffect(db.getREffectByID(app.editedNumber));
        }
    }
}
app.currentRule
    .setRCauses(db.getAllRCauses(app.currentRule.getID()));
app.currentRule.setREffects(db.getAllREffects(app.currentRule
    .getID()));

// remove bools until right amount in rule
while (app.currentRule.getRCauses().size() - bools.size() != 1
    && !bools.isEmpty()) {
    bools.remove(bools.size() - 1);
}
app.currentRule.updateTreeData(bools);
db.updateRule(app.currentRule);
bools = app.currentRule.getBoolSequence();
db.close();
}

// Fill Out Form
EditText title = (EditText) this.findViewById(R.id.textView1);
title.setText(app.currentRule.getName());
Switch active = (Switch) this.findViewById(R.id.switch1);
active.setChecked(app.currentRule.Active);

// toggle on/off with both clicks and swipes
active.setOnCheckedChangeListener(new Switch.OnCheckedChangeListener() {
    /**
     * Called when the Rule Active Status is changed The onSwitchClicked
     * function is then called to update the value for the rule
     *
     * @see android.widget.CompoundButton.OnCheckedChangeListener#onCheckedChanged
     */
    android.widget.CompoundButton,
    boolean)
    */
})

```

```

    public void onCheckedChanged(CompoundButton arg0, boolean arg1) {
        onSwitchClicked(arg0);
    }
});

if (app.currentRule.edited) // general adding of causes or effects
{
    DatabaseHandler db = new DatabaseHandler(this);
    bools = app.currentRule.getBoolSequence();
    if (db.updateRule(app.currentRule) > 0) {
        app.currentRule.setID(db.getAllRules()
            .get(db.getRulesCount() - 1).getID());
    }
    app.currentRule
        .setRCauses(db.getAllRCauses(app.currentRule.getID()));
    app.currentRule.setREffects(db.getAllREffects(app.currentRule
        .getID()));

    // add bools if rCause added
    if (app.addedRCause && !app.edit
        && (size != app.currentRule.getRCauses().size())) {
        bools.add(new EditRuleNode("OR", false, true));
        size = app.currentRule.getRCauses().size();
        app.addedRCause = false;
    }

    app.currentRule.updateTreeData(bools);
    db.updateRule(app.currentRule);
    bools = app.currentRule.getBoolSequence();
    app.currentRule.edited = false;
    app.edit = false;
    db.close();
}

// clear bools if only 1 cause
if (app.currentRule.getRCauses().size() <= 1) {
    bools.clear();
}

DatabaseHandler db = new DatabaseHandler(this);
if (app.currentRule != null) {
    app.currentRule.updateTreeData(bools);
    db.updateRule(app.currentRule);
    bools = app.currentRule.getBoolSequence();
}
db.close();

// add everything to lists
for (int i = 0; i < app.currentRule.getRCauses().size(); i++) {
    rCause r = app.currentRule.getRCauses().get(i);

    if (!r.getType().equals("location")) {
        cList.add(new EditRuleNode(r.getName() + "\n"
            + r.getParameters(), true, false));
    } else {
        String param = r.getParameters();

```

```

        String finalParam = "";
        char cur = ' ';
        int start = 0;
        while (cur != '\n') {
            cur = param.charAt(start);
            finalParam += cur;
            start++;
        }
        cList.add(new EditRuleNode(r.getName() + "\n" + finalParam,
            true, false));
    }

    if (i < bools.size() && !bools.isEmpty()) {
        cList.add(bools.get(i));
    }
}

for (int i = 0; i < app.currentRule.getREffects().size(); i++) {
    rEffect r = app.currentRule.getREffects().get(i);
    String finalParam = r.getParameters();
    if (r.getType().equals("sound")) {
        finalParam = finalParam.split("\n") [1];
    }

    if (r.getType().equals("toast")) {
        eList.add("Popup Text" + "\n" + finalParam);
    } else {
        eList.add(r.getName() + "\n" + finalParam);
    }
}

// Add Adapters
cList.add(new EditRuleNode("+", false, false));
eList.add("+");
}

/**
 * Enum conversions for rEffects in the list
 *
 * @param type
 * @return int The type of rEffect
 */
public int convertTypeToEEEnum(String type) {
    if (type.equals("notification"))
        return 0;
    if (type.equals("sound"))
        return 1;
    if (type.equals("ringer"))
        return 2;
    if (type.equals("toast"))
        return 3;
    if (type.equals("vibrate"))
        return 4;
    return -1;
}

```

```

/**
 * Enum conversions for rCauses in the list
 *
 * @param type
 * @return int The type of rCause
 */
public int convertTypeToCEnum(String type, int cid) {
    if (type.equals("location")) {
        if (cid == 6)
            return 0;
        else
            return 1;
    }
    if (type.equals("phoneCall"))
        return 2;
    if (type.equals("textMessage"))
        return 3;
    if (type.equals("time"))
        return 4;
    if (type.equals("ssid"))
        return 5;
    if (type.equals("wifiStatus"))
        return 6;
    return -1;
}

/**
 * Creates the Menu for the EditRule Activity
 *
 * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
 */
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_edit_task, menu);
    return true;
}

/**
 * Actions for the EditRule Menu
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case android.R.id.home: {
            // This is called when the Home (Up) button is pressed
            // in the Action Bar.
            Intent parentActivityIntent = new Intent(this, MyRules.class);
            parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                | Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(parentActivityIntent);
            finish();
            return true;
        }
    }
}

```

```

        case R.id.menu_help: {
            Intent myIntent = new Intent(this, Help.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
            return true;
        }
        case R.id.menu_settings: {
            Intent myIntent = new Intent(this, Preferences.class)
                .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(myIntent);
            return true;
        }
        default: {
            return super.onOptionsItemSelected(item);
        }
    }

    /**
     * Called when the Rule Activity status is changed This value is updated in
     * the current rule and database
     *
     * @param view
     */
    public void onSwitchClicked(View view) {
        // Is the switch on?
        boolean on = ((Switch) view).isChecked();
        app.currentRule.Active = on;
        DatabaseHandler db = new DatabaseHandler(this);
        db.updateActive(app.currentRule.getID(), on);
        db.close();
    }

    /**
     * This is called when a delete of a rCause or rEffect is confirmed
     *
     * @see com.example.ceandroid.DeleteDialogFragment.DeleteDialogListener#onDialogPositiveClick(android.app.DialogFragment)
     */
    public void onDialogPositiveClick(DialogFragment dialog) {
        // clicked to delete the cause or effect
        DatabaseHandler db = new DatabaseHandler(this);
        if (app.editType) // cause
        {
            db.deleteRCause(db.getRCauseByID(app.editedNumber));
            if (!bools.isEmpty() && bools.size() == delPos) {
                bools.remove(bools.size() - 1);
            } else if (!bools.isEmpty()) {
                bools.remove(delPos);
            }
        } else // effect
        {
            db.deleteREffect(db.getREffectByID(app.editedNumber));
        }
        app.currentRule.setRCauses(db.getAllRCauses(app.currentRule.getID()));
        app.currentRule.setREffects(db.getAllREffects(app.currentRule.getID()));
    }
}

```

```

        app.currentRule.updateTreeData(bools);
        bools = app.currentRule.getBoolSequence();
        db.close();
        finish();
        startActivity(getIntent());
    }

    /**
     * This is called when the delete of a rCause or rEffect is cancelled
     *
     * @see com.example.ceandroid.DeleteDialogFragment.DeleteDialogListener#onDialogNegativeClick(android.app.DialogFragment)
     */
    public void onDialogNegativeClick(DialogFragment dialog) {
        // clicked cancel, do not do anything
    }

    /**
     * Prints the list of booleans used for the Cause Tree within the current
     * rule Primarily used for debugging
     *
     * @param bools
     */
    public void print(ArrayList<EditRuleNode> bools) {
        for (int i = 0; i < bools.size(); i++) {
            System.out.println(bools.get(i).value);
        }
    }
}

```

## MapPicker.java

```
package com.example.ceandroid.Causes;

import CEapi.rCause;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Color;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.text.InputType;
import android.view.KeyEvent;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Toast;

import com.example.ceandroid.CEapp;
import com.example.ceandroid.CauseView;
import com.example.ceandroid.EditRule;
import com.example.ceandroid.R;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.GoogleMap.OnMapClickListener;
import com.google.android.gms.maps.GoogleMap.OnMapLongClickListener;
import com.google.android.gms.maps.UiSettings;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.PolygonOptions;
import com.google.android.gms.maps.SupportMapFragment;

/**
 * The MapPicker allows the user to choose a Location for all Location based
 * rules This class uses the Google Maps 2.0 API
 *
 * @author CEandroid SMU
 */
public class MapPicker extends android.support.v4.app.FragmentActivity {
    /**
     * The Google Maps 2.0 API Object
     */
    private GoogleMap myMap;
    /**
     * Settings for the Map
     */
    private UiSettings myUiSettings;
    /**
     * General use point variable
     */
    private LatLng p;
    /**
     * Radius from the point
     */
    private double r;
```

```

/**
 * Finished selecting a location, confirmed by the user
 */
private boolean finished;
/**
 * The name of the location, set by the user
 */
private String name;

/**
 * Creates and loads the Map on the screen
 *
 * @see android.support.v4.app.FragmentActivity#onCreate(android.os.Bundle)
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.ce_activity_map);

    Toast.makeText(MapPicker.this, "Tap the map to pick a location",
        Toast.LENGTH_LONG).show();

    p = new LatLng(0, 0);
    r = 0;
    setUpMapIfNeeded();
    finished = false;
    name = "no name";

    // Turn on "up" navigation
    getActionBar().setDisplayHomeAsUpEnabled(true);
}

/**
 * Sets up the map only if it is not already loaded
 */
private void setUpMapIfNeeded() {
    // Do a null check to confirm that we have not already instantiated the
    // map.
    if (myMap == null) {
        // Try to obtain the map from the SupportMapFragment.
        myMap = ((SupportMapFragment) this.getSupportFragmentManager()
            .findFragmentById(R.id.map)).getMap();
        // Check if we were successful in obtaining the map.
        if (myMap != null) {
            setUpMap();
        }
    }
}

/**
 * Sets up and loads the Map
 */
private void setUpMap() {
    // The Map is verified. It is now safe to manipulate the map.
    myMap.setMyLocationEnabled(true);
    myUiSettings = myMap.getUiSettings();
}

```

```

myUiSettings.setMyLocationButtonEnabled(true);
myUiSettings.setAllGesturesEnabled(true);
myUiSettings.setZoomControlsEnabled(true);

// Tap Listener
OnMapClickListener tap = new OnMapClickListener() {
    /**
     * Adds a pin to a user selected location, then asks the user to
     * name the location
     *
     * @see com.google.android.gms.maps.GoogleMap.OnMapClickListener#onMapC
     lick(com.google.android.gms.maps.model.LatLng)
     */
    public void onMapClick(LatLng point) {
        p = point;
        myMap.addMarker(new MarkerOptions().position(new LatLng(
            point.latitude, point.longitude)));
        myMap.animateCamera(
            CameraUpdateFactory.newLatLngZoom(point, 16), 1000,
            null);
        android.os.SystemClock.sleep(1000);
        messageDialog();
    }
};

OnMapLongClickListener tapLong = new OnMapLongClickListener() {
    /**
     * The user confirms a location
     *
     * @see com.google.android.gms.maps.GoogleMap.OnMapLongClickListener#on
     MapLongClick(com.google.android.gms.maps.model.LatLng)
     */
    public void onMapLongClick(LatLng point) {
        if (finished) {
            finished = false;
            finished();
        } else {
            Toast.makeText(MapPicker.this, "No location chosen",
                Toast.LENGTH_LONG).show();
        }
    }
};

// Add Listeners
myMap.setOnMapClickListener(tap);
myMap.setOnMapLongClickListener(tapLong);

// Goto my location on load
LocationManager lm = (LocationManager) MapPicker.this
    .getSystemService(Context.LOCATION_SERVICE);
Location myLocation = lm
    .getLastKnownLocation(LocationManager.GPS_PROVIDER);
if (myLocation == null) {
    myLocation = lm
        .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
    if (myLocation == null) {

```

```

        myLocation = lm
            .getLastKnownLocation(LocationManager.PASSIVE_PROVIDER);
    }
}
if (myLocation != null) {
    LatLng myLoc = new LatLng(myLocation.getLatitude(),
        myLocation.getLongitude());
    myMap.animateCamera(CameraUpdateFactory.newLatLngZoom(myLoc, 16),
        1000, null);
} else {
    LatLng myLoc = new LatLng(32.986204, -96.702003);
    myMap.animateCamera(CameraUpdateFactory.newLatLngZoom(myLoc, 16),
        1000, null);
}
android.os.SystemClock.sleep(1000);
}

/**
 * Reloads the map if needed when the application is resumed
 *
 * @see android.support.v4.app.FragmentActivity#onResume()
 */
@Override
protected void onResume() {
    super.onResume();
    setUpMapIfNeeded();
}

/**
 * Creates the region around the pin location
 */
public void rDialog() {
    r = .002;
    PolygonOptions rectOptions = new PolygonOptions().add(new LatLng(
        p.latitude - r, p.longitude - r), new LatLng(p.latitude - r,
        p.longitude + r), new LatLng(p.latitude + r, p.longitude + r),
        new LatLng(p.latitude + r, p.longitude - r));

    // Set the rectangle's stroke color to red
    rectOptions.strokeColor(Color.argb(125, 51, 181, 229));
    // Set the rectangle's fill to blue
    rectOptions.fillColor(Color.argb(50, 51, 181, 229));

    myMap.addPolygon(rectOptions);

    Toast.makeText(MapPicker.this, "Tap and hold to confirm this area",
        Toast.LENGTH_LONG).show();
    finished = true;
}

/**
 * Called when the user has confirmed a location
 */
public void finished() {
    // Call Edit Rule Again
    CEapp app = (CEapp) getApplication();
}

```

```

rCause rc = new rCause();

rc.setRuleID(app.currentRule.getID());
rc.setType("location");
if (app.getTemp() == 0) {
    rc.setCauseID(6);
} else {
    rc.setCauseID(7);
}
rc.setName(name);

String loc = "Location: " + name + "\n";
String plat = "Lat: " + p.latitude + "\n";
String plon = "Lng: " + p.longitude + "\n";
String rad = "Radius: " + (r * 50) + " miles\n";
String fired = "Fired: 0\n";
String parameter = loc + plat + plon + rad + fired;
rc.setParameters(parameter);
app.currentRule.addRCause(rc);
app.addedRCause = true;

Intent myIntent = new Intent(MapPicker.this, EditRule.class)
    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(myIntent);
}

/**
 * Asks the user to name a picked location
 */
private void messageDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Name the location");

    // Set up the input
    final EditText input = new EditText(MapPicker.this);
    // Specify the type of input expected; this, for example, sets the input
    // as a password, and will mask the text
    input.setInputType(InputType.TYPE_CLASS_TEXT);
    builder.setView(input);

    // Set up the buttons
    builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
        /**
         * Names the location and then draws the region around the pin
         * location
         *
         * @see android.content.DialogInterface.OnClickListener#onClick(android
         .content.DialogInterface,
         *      int)
         */
        public void onClick(DialogInterface dialog, int which) {
            name = input.getText().toString();
            rDialog();
        }
    });
    builder.setNegativeButton("Cancel",

```

```

        new DialogInterface.OnClickListener() {
    /**
     * Clears pins if the user doesn't like the selected
     * location
     *
     * @see android.content.DialogInterface.OnClickListener#onClick(android.content.DialogInterface,
     *      int)
     */
    public void onClick(DialogInterface dialog, int which) {
        myMap.clear();
        p = new LatLng(0, 0);
        dialog.cancel();
    }
});

builder.show();
}

/**
 * MapPicker Menu Actions
 *
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            // This is called when the Home (Up) button is pressed
            // in the Action Bar.
            CEapp app = (CEapp) getApplication();
            if (app.edit) {
                Intent myIntent = new Intent(getApplicationContext(),
                    EditRule.class)
                    .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(myIntent);
            } else {
                Intent parentActivityIntent = new Intent(this, CauseView.class);
                parentActivityIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
                    | Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(parentActivityIntent);
                finish();
            }
            return true;
    }
    return super.onOptionsItemSelected(item);
}

/**
 * Overrides the back button
 *
 * @see android.support.v4.app.FragmentActivity#onKeyDown(int,
 *      android.view.KeyEvent)
 */
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {

```

```
if (keyCode == KeyEvent.KEYCODE_BACK) {
    CEapp app = (CEapp) getApplication();
    if (app.edit) {
        Intent myIntent = new Intent(getApplicationContext(),
            EditRule.class)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(myIntent);
    }
    return super.onKeyDown(keyCode, event);
}

return super.onKeyDown(keyCode, event);
}
```

## ActionShowNotification.java

```
package com.example.ceandroid.Effects;

import com.example.ceandroid.R;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;

/**
 * Shows a Notification
 *
 * @author CEandroid SMU
 */
public class ActionShowNotification {

    /**
     * global interface that allows access application resources
     */
    private Context context;
    /**
     * the intended title of the notification
     */
    private String contentTitle;
    /**
     * the intended message inside the notification
     */
    private String contentText;

    /**
     * Constructor
     *
     * The constructor stores the application context, notification title, and
     * notification content. The parameters are split into the title and
     * content, respectively, before being stored.
     *
     * @param context
     *         global interface that allows access application resources
     * @param parameters
     *         notification title and content in the format
     *         "Content Title'Content Text"
     */
    public ActionShowNotification(Context context, String parameters) {
        this.context = context;
        String[] params = parameters.split("\\"'");
        if (params.length > 0) {
            this.contentTitle = params[0];
            if (params.length > 1) {
                this.contentText = params[1];
            }
        }
    }
}
```

```

    }

    /**
     * Executes the effect
     *
     * Shows a new notification based on the intended title and content. The
     * notification manager identifies each notification using the notification
     * ID, which is stored as the time in milliseconds at which the notificatio
n
     * was created.
     */
    public void execute() {
        PendingIntent pIntent = PendingIntent.getActivity(context, 0,
            new Intent(), 0);

        Notification notification = new NotificationCompat.Builder(context)
            .setSmallIcon(R.drawable.ic_launcher)
            .setContentTitle(contentTitle).setContentText(contentText)
            .setContentIntent(pIntent).build();

        NotificationManager notificationManager = (NotificationManager) context
            .getSystemService(Context.NOTIFICATION_SERVICE);
        int mId = (int) System.currentTimeMillis(); // notification ID is the
            // current time in
            // milliseconds

        notification.flags |= Notification.FLAG_AUTO_CANCEL;

        notificationManager.notify(mId, notification);
    }
}

```

## ActionRingerMode.java

```
package com.example.ceandroid.Effects;

import android.content.Context;
import android.media.AudioManager;
import android.widget.Toast;

/**
 * Changes the ring mode type
 *
 * @author CEandroid SMU
 */
public class ActionRingerMode {

    /**
     * global interface that allows access application resources
     */
    private Context context;
    /**
     * the intended ring mode to be changed to
     */
    private String mode;

    /**
     * Constructor
     *
     * The constructor stores the application context and the intended ringer
     * mode.
     *
     * @param context
     *          global interface that allows access application resources
     * @param parameters
     *          the ring mode to be changed to
     */
    public ActionRingerMode(Context context, String parameters) {
        this.context = context;
        this.mode = parameters.replace("Ring mode: ", "");
    }

    /**
     * Execute the effect
     *
     * Changes the ringer mode based on the intended parameter. Outputs an erro
     * r
     * if and incorrect parameter is sent.
     */
    public void execute() {
        AudioManager audioManager = (AudioManager) context
            .getSystemService(Context.AUDIO_SERVICE);
        if (mode.equals("normal"))
            audioManager.setRingerMode(AudioManager.RINGER_MODE_NORMAL);
        else if (mode.equals("vibrate"))
            audioManager.setRingerMode(AudioManager.RINGER_MODE_VIBRATE);
        else if (mode.equals("silent"))
    }
}
```

```
        audioManager.setRingerMode(AudioManager.RINGER_MODE_SILENT);  
    else  
        Toast.makeText(context.getApplicationContext(),  
                      "error: ringer mode parameters", Toast.LENGTH_SHORT).show();  
    }  
}
```

## ActionVibrate.java

```
package com.example.ceandroid.Effects;

import android.content.Context;
import android.os.Vibrator;

/**
 * Vibrates the phone
 *
 * @author CEandroid SMU
 *
 */
public class ActionVibrate {

    /**
     * global interface that allows access application resources
     */
    private Context context;

    /**
     * Constructor
     *
     * The constructor stores the application context.
     *
     * @param context
     *          global interface that allows access application resources
     * @param parameters
     *          [unused] the length of the notification
     */
    public ActionVibrate(Context context, String parameters) {
        this.context = context;
    }

    /**
     * Executes the effect
     *
     * Vibrates the device for a period of 1200 milliseconds.
     */
    public void execute() {
        Vibrator vibrator = (Vibrator) context
            .getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate(1200);
    }
}
```

## ActionToast.java

```
package com.example.ceandroid.Effects;

import android.content.Context;
import android.widget.Toast;

/**
 * Displays a Toast message
 *
 * @author CEandroid SMU
 */
public class ActionToast {

    /**
     * global interface that allows access application resources
     */
    private Context context;
    /**
     * the content text of the toast
     */
    private String text;

    /**
     * Constructor
     *
     * The constructor stores the application context and the content of the
     * toast.
     *
     * @param context
     *          global interface that allows access application resources
     * @param parameters
     */
    public ActionToast(Context context, String parameters) {
        this.context = context;
        if (parameters.length() > 0) {
            this.text = " " + parameters.substring(5);
        }
        this.text += " ";
    }

    /**
     * Executes the effect
     *
     * Shows the toast with the intended text content.
     */
    public void execute() {
        Toast.makeText(context.getApplicationContext(), text,
                      Toast.LENGTH_SHORT).show();
    }
}
```

## ActionPlaySound.java

```
package com.example.ceandroid.Effects;

import android.content.Context;
import android.content.Intent;
import android.net.Uri;

/**
 * Plays a sound Effect
 *
 * @author CEandroid SMU
 */
public class ActionPlaySound {

    /**
     * global interface that allows access application resources
     */
    private Context context;
    /**
     * URI to the media file to be played
     */
    private Uri parameters;

    /**
     * Constructor
     *
     * The constructor stores the application context and location to the audio
     * file. It also converts the location of the audio file from a String into
     * a URI.
     *
     * @param context
     *          global interface that allows access application resources
     * @param parameters
     *          the location of the audio file
     */
    public ActionPlaySound(Context context, String parameters) {
        this.context = context;
        this.parameters = Uri.parse(parameters);
    }

    /**
     * Execute the effect
     *
     * Plays the intended audio file in the default media player for audio.
     */
    public void execute() {
        Intent intent = new Intent();
        intent.setAction(android.content.Intent.ACTION_VIEW);
        intent.setDataAndType(this.parameters, "audio/*");
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                | Intent.FLAG_ACTIVITY_SINGLE_TOP);
        context.startActivity(intent);
    }
}
```

# XML

## AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ceandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="15" />
        <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission
        android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
        <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
        <uses-permission android:name="android.permission.INTERNET"/>
        <uses-permission android:name="android.permission.RECEIVE_SMS"/>
        <uses-permission android:name="android.permission.READ_SMS"/>
        <uses-permission android:name="android.permission.WRITE_SMS"/>
        <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
        <uses-permission
            android:name="android.permission.ACCESS_COARSE_LOCATION"/>
            <uses-permission
                android:name="android.permission.ACCESS_FINE_LOCATION"/>
                <permission android:name="com.example.ceandroid.permission.MAPS_RECEIVE"
                android:protectionLevel="signature"/>
                    <uses-permission
                        android:name="com.example.ceandroid.permission.MAPS_RECEIVE"/>
                        <uses-permission
                            android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
                            <uses-permission
                                android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
                                <uses-permission android:name="android.permission.NFC"/>

                                <uses-feature
                                    android:glEsVersion="0x00020000"
                                    android:required="true"/>
                                <uses-permission android:name="android.permission.VIBRATE"/>

    <application
        android:name="CEapp"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:screenOrientation="portrait"
        android:launchMode="singleTask" >

        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyARML854c14f7h-C42taDvd_j_CGTlbzgU"/>
        <meta-data
            android:name="android.nfc.disable_beam_default"
            android:value="true" />
```

```

<service
    android:name=".CEservice"
    android:icon="@drawable/ic_launcher"
    android:label="@string/service_name" >
</service>

<activity
    android:name=".MainActivity"
    android:label="@string/title_activity_main"
    android:theme="@android:style/Theme.NoTitleBar"
    android:launchMode="singleTop" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".Help"
    android:label="@string/help" >
</activity>
<activity
    android:name=".Preferences"
    android:label="@string/title_activity_preferences" >
</activity>
<activity
    android:name=".ShareList"
    android:label="@string/title_activity_sharelist" >

        <intent-filter>
            <action android:name="android.nfc.action.NDEF_DISCOVERED" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="text/plain" />
        </intent-filter>
</activity>
<activity
    android:name=".ShareNFC"
    android:label="@string/title_activity_sharenfc" >
</activity>
<activity
    android:name=".General"
    android:label="@string/title_activity_general" >
</activity>
<activity
    android:name=".About"
    android:label="@string/title_activity_about" >
</activity>
<activity
    android:name=".Security"
    android:label="@string/title_activity_security" >
</activity>
<activity
    android:name=".CauseView"
    android:label="@string/cause" >
</activity>
<activity

```

```
        android:name=".EffectView"
        android:label="@string/effect" >
    </activity>
    <activity
        android:name=".AlphabeticalList"
        android:label="@string/az" >
    </activity>
    <activity
        android:name=".CategoryList"
        android:label="@string/category" >
    </activity>
    <activity
        android:name=".MyRules"
        android:label="@string/my_rules" >
    </activity>
    <activity
        android:name=".EditRule"
        android:label="@string/title_activity_edit_task2" >
    </activity>
    <activity
        android:name="com.example.ceandroid.Causes.MapPicker"
        android:label="@string/map" >
    </activity>
    <receiver android:name="CEbr"
              android:enabled="true"
              android:exported="true">
    </receiver>
</application>
</manifest>
```

## activity\_about.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <ImageView
        android:id="@+id/logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:contentDescription="@string/logo"
        android:src="@drawable/logo" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/logo"
        android:layout_centerHorizontal="true"
        android:text="@string/about"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:text="@string/aboutDesc" />

    <ImageView
        android:id="@+id/samsung_logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/samsung"
        android:contentDescription="@string/samsung"
        android:layout_below="@+id/textView2"
        android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

## activity\_cause.xml

```
<LinearLayout android:layout_gravity="center"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    style="@style/AppTheme">

<FrameLayout android:id="@+id/cause_fragment_container"
    android:layout_height="match_parent"
    android:layout_width="match_parent"/>
</LinearLayout>
```

## activity\_edit\_rule.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/relativeEditRule"
    style="@style/AppTheme"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:focusable="true"
    android:focusableInTouchMode="true">

    <EditText
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:gravity="center"
        android:hint="@string/default_name"
        android:inputType="text"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:selectAllOnFocus="true" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:text="@string/cause"
        android:gravity="center"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView1"
        android:layout_centerHorizontal="true"
        android:text="@string/effect"
        android:gravity="center"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <ListView
        android:id="@+id/causes"
        android:layout_width="wrap_content"
        android:layout_height="155dp"
        android:dividers="#33B5E5"
        android:dividerHeight="1dp"
        android:layout_below="@+id/imageView2"
        android:layout_alignParentLeft="true" />

    <ListView
        android:id="@+id/effects"
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:divider="#33B5E5"
        android:dividerHeight="1dp"
        android:layout_below="@+id/imageView3"
        android:layout_alignParentRight="true"/>

<ImageView
    android:id="@+id/imageView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView2"
    android:layout_alignParentLeft="true"
    android:src="@drawable/dividerh"
    android:contentDescription="@string/divider"/>

<ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/causes"
    android:layout_alignParentLeft="true"
    android:src="@drawable/dividerh"
    android:contentDescription="@string/divider"/>

<ImageView
    android:id="@+id/imageView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView3"
    android:layout_alignParentLeft="true"
    android:src="@drawable/dividerh"
    android:contentDescription="@string/divider"/>

<ImageView
    android:id="@+id/imageView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView4"
    android:layout_alignParentLeft="true"
    android:src="@drawable/dividerh"
    android:contentDescription="@string/divider"/>

<Switch
    android:id="@+id/switch1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView1"
    android:layout_alignBottom="@+id/textView1"
    android:layout_alignParentRight="true"
    android:textOff="Off"
    android:textOn="On" />

</RelativeLayout>

```

## activity\_effect.xml

```
<LinearLayout android:layout_gravity="center"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    style="@style/AppTheme">

<FrameLayout android:id="@+id/effect_fragment_container"
    android:layout_height="match_parent"
    android:layout_width="match_parent"/>
</LinearLayout>
```

## activity\_general.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/genLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/genList"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:divider="#FFFFFF"
        android:dividerHeight="2dp"
        android:padding="10dp"
        android:textSize="20sp"
        android:entries="@array/general_items">
    </ListView>
</LinearLayout>
```

## activity\_help.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/helpLayout"
    style="@style/AppTheme"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/helpText"
        android:textIsSelectable="true"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/helpText"
        android:padding="10dp"
        android:textSize="20sp">
    </TextView>

    <ImageButton
        android:id="@+id/helpLink"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/helpText"
        android:layout_centerHorizontal="true"
        android:autoLink="web"
        android:padding="10dp"
        android:src="@drawable/logo" />

    <TextView
        android:id="@+id/helpTextPass"
        android:textIsSelectable="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/helpLink"
        android:layout_centerHorizontal="true"
        android:text="@string/helpTextPass"
        android:padding="10dp"
        android:textSize="20sp">
    </TextView>
</RelativeLayout>
```

## activity\_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    style="@style/AppTheme">
    <ImageView
        android:id="@+id/logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/logo"
        android:contentDescription="@string/logo"
        android:layout_centerHorizontal="true"/>
    <Button
        android:id="@+id/button1"
        android:layout_width="260dp"
        android:layout_height="60dp"
        android:layout_marginTop="160dp"
        android:layout_centerHorizontal="true"
        android:text="@string/my_rules"
        style="@style/ButtonTheme"
        android:onClick="myRules"/>
    <Button
        android:id="@+id/button2"
        android:layout_width="260dp"
        android:layout_height="60dp"
        android:layout_below="@+id/button1"
        android:layout_alignLeft="@+id/button1"
        android:text="@string/new_rule"
        style="@style/ButtonTheme"
        android:onClick="newRule"/>
    <Button
        android:id="@+id/button3"
        android:layout_width="260dp"
        android:layout_height="60dp"
        android:layout_below="@+id/button2"
        android:layout_alignLeft="@+id/button1"
        android:text="@string/help"
        style="@style/ButtonTheme"
        android:onClick="help"/>
    <Button
        android:id="@+id/button4"
        android:layout_width="260dp"
        android:layout_height="60dp"
        android:layout_below="@+id/button3"
        android:layout_alignLeft="@+id/button1"
        android:text="@string/settings"
        style="@style/ButtonTheme"
        android:onClick="settings"/>
    <Button
        android:id="@+id/button5"
        android:layout_width="260dp"
        android:layout_height="60dp"
        android:layout_below="@+id/button4"
```

```
        android:layout_alignLeft="@+id/button1"
        android:text="@string/share"
        style="@style/ButtonTheme"
        android:onClick="share"/>
<ImageView
    android:id="@+id/samsung_logo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/samsung"
    android:contentDescription="@string/samsung"
    android:layout_below="@+id/button5"
    android:layout_centerHorizontal="true"/>
</RelativeLayout>
```

## activity\_my\_rules.xml

```
<LinearLayout android:layout_gravity="center"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    style="@style/AppTheme">

<FrameLayout android:id="@+id/rules_fragment_container"
    android:layout_height="match_parent"
    android:layout_width="match_parent"/>
</LinearLayout>
```

## activity\_preferences.xml

```
<LinearLayout android:layout_gravity="center"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    style="@style/AppTheme">

<LinearLayout android:id="@+id/pref_fragment_container"
    android:layout_height="match_parent"
    android:layout_width="match_parent">

</LinearLayout>
</LinearLayout>
```

## activity\_security.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/secLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/secList"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:divider="#FFFFFF"
        android:dividerHeight="2dp"
        android:padding="10dp"
        android:textSize="20sp"
        android:entries="@array/secure_items">
    </ListView>
</RelativeLayout>
```

## activity\_sharelist.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="20sp" >
</TextView>
```

## activity\_sharenfc.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:textIsSelectable="true"
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:padding="10dp"
        android:textSize="20sp"/>

    <ImageView
        android:id="@+id/nfcImg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:layout_below="@+id/textView1"
        android:padding="10dp"
        android:src="@drawable/nfc_example"/>

</RelativeLayout>
```

## ce\_activity\_map.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    style="@style/AppTheme"
    android:id="@+id/container">

    <fragment android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.google.android.gms.maps.SupportMapFragment"/>
</RelativeLayout>
```

## **ce\_dialog\_notification.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <EditText
        android:id="@+id/title"
        android:inputType="text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="Title" />
    <EditText
        android:id="@+id/text"
        android:inputType="text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:hint="Subtext"/>
    <Button
        android:id="@+id/button1"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Submit"
        style="@style/ButtonTheme"/>
</LinearLayout>
```

## ce\_dialog\_ssid.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <EditText
        android:id="@+id/ssid"
        android:inputType="text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="4dp"
        android:hint="SSID" />
    <Button
        android:id="@+id/button1"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Submit"
        style="@style/ButtonTheme"/>
</LinearLayout>
```

## **ce\_dialog\_switch.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <Switch
        android:id="@+id/dialogSwitch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOff="Off"
        android:textOn="On" />
    <Button
        android:id="@+id/button1"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Submit"
        style="@style/ButtonTheme"/>
</LinearLayout>
```

## **ce\_dialog\_toast.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <EditText
        android:id="@+id/text"
        android:inputType="text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="4dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginBottom="16dp"
        android:hint="Message"/>
    <Button
        android:id="@+id/button1"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="Submit"
        style="@style/ButtonTheme"/>
</LinearLayout>
```

## ce\_dialog\_vibrate.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="154dp"
        android:layout_weight="326.86"
        android:scrollbars="vertical"
        android:text="Pass in an array of ints that are the durations for
which to turn on or off the vibrator in milliseconds. The first value
indicates the number of milliseconds to wait before turning the vibrator on.
The next value indicates the number of milliseconds for which to keep the
vibrator on before turning it off. Subsequent values alternate between
durations in milliseconds to turn the vibrator off or to turn the vibrator on.
To cause the pattern to repeat, pass the index into the pattern array at
which to start the repeat, or -1 to disable repeating." />

    <EditText
        android:id="@+id/text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_marginTop="4dp"
        android:hint="0, 200, 100, 200, -1"
        android:inputType="text" />

    <Button
        android:id="@+id/button1"
        style="@style/ButtonTheme"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_weight="6.83"
        android:text="Submit" />

</LinearLayout>
```

## **item\_list.xml**

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/listLayoutAlpha"
    android:orientation="vertical"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent" >

    <ListView
        android:id="@+id/fruitList"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="0.5"
        android:divider="#FFFFFF"
        android:dividerHeight="2dp"
        android:padding="10dp"
        android:textSize="20sp" >
    </ListView>

</LinearLayout>
```

## **menu\_edit\_task.xml**

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_settings"
        android:title="@string/settings"
        android:icon="@android:drawable/ic_menu_preferences"/>
    <item android:id="@+id/menu_help"
        android:title="@string/help"
        android:icon="@android:drawable/ic_menu_help"/>
</menu>
```

## **menu\_help.xml**

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_new_rule"
        android:title="@string/new_rule"
        android:icon="@android:drawable/ic_menu_add"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_share"
        android:title="@string/share"
        android:icon="@android:drawable/ic_menu_share"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_settings"
        android:title="@string/settings"
        android:icon="@android:drawable/ic_menu_preferences"/>
</menu>
```

## menu\_my\_rules.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_new_rule"
        android:title="@string/new_rule"
        android:icon="@android:drawable/ic_menu_add"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_share"
        android:title="@string/share"
        android:icon="@android:drawable/ic_menu_share"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_settings"
        android:title="@string/settings"
        android:icon="@android:drawable/ic_menu_preferences"/>
</menu>
```

## menu\_preferences.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_new_rule"
        android:title="@string/new_rule"
        android:icon="@android:drawable/ic_menu_add"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_share"
        android:title="@string/share"
        android:icon="@android:drawable/ic_menu_share"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_help"
        android:title="@string/help"
        android:icon="@android:drawable/ic_menu_help"/>
</menu>
```

## menu\_sync.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_new_rule"
        android:title="@string/new_rule"
        android:icon="@android:drawable/ic_menu_add"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_share"
        android:title="@string/share"
        android:icon="@android:drawable/ic_menu_share"
        android:showAsAction="ifRoom|withText" />
    <item android:id="@+id/menu_help"
        android:title="@string/help"
        android:icon="@android:drawable/ic_menu_help"/>
</menu>
```

## values/strings.xml

```
<resources>

    <string name="app_name">C&Elt;/string>
    <string name="menu_settings">Preferences</string>
    <string name="service_name">C&Elt;/string> Service</string>
    <string name="title_activity_main">C&Elt;/string>
    <string name="my_rules">My Rules</string>
    <string name="help">Help</string>
    <string name="settings">Preferences</string>
    <string name="new_rule">New Rule</string>
        <string name="samsung">The Samsung Logo</string>
        <string name="logo">The C&Elt;/string> Logo</string>
        <string name="main_menu">Main Menu</string>
    <string-array name="help_items">
        <item >Tutorial</item>
        <item >Creating Rules</item>
        <item >Viewing Rules</item>
        <item >Editing Rules</item>
        <item >Toggle Rules</item>
        <item >Import Rules</item>
        <item >Sharing Rules</item>
        <item >Syncing Accounts</item>
        <item >Advanced Rules</item>
        <item >Rule Lists</item>
    </string-array>
    <string name="helpText">The Cause & Effect Wordpress Page can be found here:</string>
    <string name="helpTextPass">Password is dmr</string>
    <string name="helpLink">http://ceandroid.wordpress.com/</string>
    <string name="default_name">Edit Name</string>
    <string name="cause">Causes</string>
    <string name="effect">Effects</string>
    <string name="at_home">At Home</string>
    <string name="screen_on">Screen On</string>
    <string name="new_task">+</string>
    <string name="title_activity_preferences">Preferences</string>
    <string name="title_activity_sync">Sync</string>
    <string name="title_activity_accounts">Accounts</string>
    <string name="title_activity_about">About</string>
    <string name="title_activity_security">Security</string>
    <string name="title_activity_general">General</string>
    <string name="title_activity_sharelist">Share</string>
        <string name="title_activity_sharenfc">Share via NFC</string>
        <string name="share">Share</string>
    <string name="az">A-Z View</string>
    <string name="category">Category View</string>
    <string name="list_view">List View</string>
    <string name="grid_view">Grid View</string>
    <string name="title_activity_edit_task2">Rule View</string>
    <string name="update_rule">Update Rule</string>
    <string name="active">Active</string>
    <string-array name="general_items">
        <item >Setting 1</item>
        <item >Setting 2</item>
    </string-array>
</resources>
```

```

<item >Setting 3</item>
<item >Setting 4</item>
<item >Setting 5</item>
<item >Setting 6</item>
<item >Setting 7</item>
<item >Setting 8</item>
<item >Setting 9</item>
<item >Setting 10</item>
</string-array>
<string-array name="secure_items">
    <item >Security Item 1</item>
    <item >Security Item 2</item>
    <item >Security Item 3</item>
    <item >Security Item 4</item>
    <item >Security Item 5</item>
    <item >Security Item 6</item>
    <item >Security Item 7</item>
    <item >Security Item 8</item>
    <item >Security Item 9</item>
    <item >Security Item 10</item>
</string-array>
<string name="about">About C&E:</string>
<string name="aboutDesc">For people wanting to make their smart devices truly intelligent. Today, smart phones are overly complex and jammed pack with functionality. Despite having the power to do so many spectacular things, smart devices lack the ability for users to take control and put the phone to work for them. Cause and Effect is a software solution that let users easily program their smart devices to respond to a set of rules with a chosen effect. With Cause and Effect, smart devices will become more intelligent and make user's lives easier by intuitively automating tasks, gathering information, displaying information, etc. Unlike other rule based software, Cause and Effect will provide more functionality and a better user interface to allow any type of user to take control of their phone.</string>
<string name="aboutCredits">About C&E:</string>
<string name="divider">Divider</string>
<string name="csni">Cause String Not Implemented</string>
<string name="ni">Not Implemented</string>
<string name="map">Choose a Location</string>
<string name="delete_confirmation_cause">Are you sure you want to delete this cause?</string>
<string name="delete_confirmation_effect">Are you sure you want to delete this effect?</string>
<string name="delete_confirmation_rule">Are you sure you want to delete this rule?</string>
<string name="delete_confirm">Delete</string>
<string name="cancel">Cancel</string>
<string name="delete">Delete</string>
<string-array name="ring_mode">
    <item >Normal</item>
    <item >Vibrate</item>
    <item >Silent</item>
</string-array>
</resources>

```

## values/styles.xml

```
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <style name="AppTheme" parent="android:Theme.Holo">
        <item name="android:background">#121212</item>
        <item name="android:textColor">#33B5E5</item>
        <item name="android:screenOrientation">portrait</item>
    </style>

    <style name="ButtonTheme" parent="android:style/Widget.Button">
        <item name="android:textColor">#33B5E5</item>
        <item name="android:background">@drawable/button_custom</item>
        <item
            name="android:textAppearance">?android:textAppearanceLargeInverse</item>
    </style>
</resources>
```

## values-v11/styles.xml

```
<resources>
    <style name="AppTheme" parent="android:Theme.Holo">
        <item name="android:background">#121212</item>
        <item name="android:textColor">#33B5E5</item>
        <item name="android:screenOrientation">portrait</item>
    </style>

    <style name="ButtonTheme" parent="android:style/Widget.Button">
        <item name="android:textColor">#33B5E5</item>
        <item name="android:background">@drawable/button_custom</item>
        <item
name="android:textAppearance">?android:attr/textAppearanceLargeInverse</item>
    </style>
</resources>
```

## values-v14/styles.xml

```
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <style name="AppTheme" parent="android:Theme.Holo">
        <item name="android:background">#121212</item>
        <item name="android:textColor">#33B5E5</item>
        <item name="android:screenOrientation">portrait</item>
    </style>

    <style name="ButtonTheme" parent="android:style/Widget.Button">
        <item name="android:textColor">#33B5E5</item>
        <item name="android:background">@drawable/button_custom</item>
        <item
name="android:textAppearance">?android:attr/textAppearanceLargeInverse</item>
    </style>
</resources>
```