# Rule Handling Workflow

Here’s our first idea for flow for rule creation. We will go into more depth of some options.

# Broadcast Receiver Ideas:

* One broadcast receiver for all rules
  + Pro: One manageable object in memory
  + Con: Harder to modify live object. If one rule crashes, they all crash. One object would also be very memory intensive.
* One broadcast receiver for each rule
  + Pro: Custom broadcast receivers for each rule type. Easier to turn on and off rules (create/delete a new object)
  + Con: Lots of objects in memory
* One broadcast receiver per type of rule (one for location, one for Wi-Fi, etc)
  + Pro: Less objects than the 2nd option, but specialized per rule type.
  + Con: Adding rules to each active category receiver on live.

# The “Event Handler” ideas

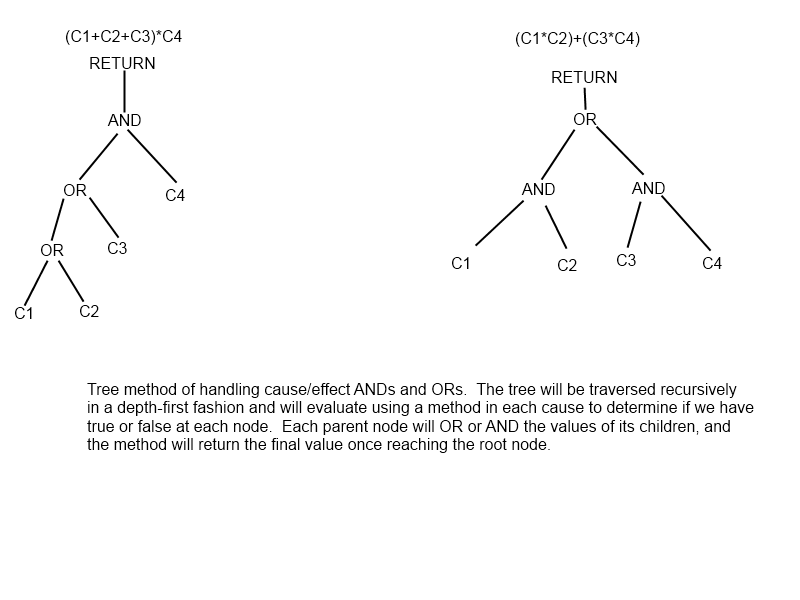
* Parallel Event Handler and Broadcast Receiver that feeds into the Rules Engine
  + Event Handler would be an object for rules not involving the OS (External Accounts, etc). Broadcast Receiver would be an object for rules handling all OS triggered events.
  + Each Object would have a unique function to call the Rules Engine for rule checking.
  + After thinking through other ideas… This one is by far the best option.

# “Rules Database” Ideas

* SQLite database on the phone (stores rules)
  + Pro: Tons of functionality, Powerful, well stored data, fast data access, in Android API
  + Con: Heavy data storage, lots of metadata
* B+ Tree (not a database)
  + Pro: Not a full database. Lighter weight storage of data
  + Con: Less powerful searches, less organized
* Hash Table
  + Pro: Simple Implementation, light weight, not a database
  + Con: Less powerful searches, could degrade to a linked list, less organized
* XML/JSON Storage
  + Pro: Simple, light weight, easy to edit (advanced users)
  + Con: Less powerful searches, less organized

# Rules Engine

## Recursive Logic Tree Function

Here is our idea for Boolean logic with rules:

* The ANDs and ORs could be evaluated recursively within the function mentioned above. The more ORs, the more efficient the tree search could be (if one side is true, the other side does not need to be evaluated at all).
* If this logic tree function returns true, then the rules engine calls the “Action Executer.” This will trigger the effects associated with the given rule.

## Check Cause Function

* A function with a switch statement could be used to make a case by case action to evaluate each cause type. This switch statement could use the event handler/broadcast receiver event as the argument. This is more efficient than a bunch of if/else statements. This function would verify the cause as true or false and then return to the logic tree above.
  + Pro: Simple, Versatile, Extensible, Efficient, not IF/ELSE spam.
  + Con: Lots of case types would be needed.

Example switch statement pseudo code:

//Function for checking a cause. Passes in the event type and any rule requirements

Bool checkCause(eventType, ruleArg)

{  
 switch(eventType) //The event type argument is from the event handler/BR

{

Case Wi-Fi: //returns true or false based on conditions of this case

Break;  
Case Location:

Break;

Case Time:

Break;

Case Phone State:

Break;

Default://returns false in case of a missing case.  
 return false;

}

}