# Use Case Action Sequence in Detail for BDS40 faults 6/24/2013

Black indicates stuff already implemented or user actions  
Red indicates stuff to change/add.  
Green indicates optional changes.   
Blue indicates email quote  
Purple indicates important concepts  
~~Crossed Out~~ indicates ideas I've rejected/ tasks completed

# Temporary Instrument faults implementation

1. Start Program
2. File > Connect to Database…
3. *DataViewer Opens. Connection Window Opens and Closes.*
4. Skip to 9. Hacked for Greg database.
5. Setup the keys for the database. These keys determine the structure of the data. Emerson’s battery data must group rows into batteries, and batteries into cabinets.
6. Select the appropriate keys. Emerson’s battery data groups batteries by String Tag, String Number, Jar Number.
7. Select the Independent and dependent axis to display in the data table on the right. Choose “Reading Date” and “CellResistance”
8. Filters will display at the top. You must hit the refresh button to load the drop down menus.
9. These filters will correspond to the chosen keys. Choose a battery by selecting something for each filter.
10. The Data is shown in a table in the Data Viewer.
11. Click ‘Graph This’ to view the data in the main window.
12. Navigate to the Cleanse tab.
13. Click BDS40 Faults to find Instrumentation faults. This is hardcoded with an ugly hack for now. Plan on a generic approach. Only works with this specific database.
14. Choose cleanse options. Choose to ignore any found BDS40 faults for this step (NYI)
15. Click Cleanse.
16. Click Graph This.
17. Click Step 2.
18. Click Graph Step 2.

# Proposals

1. Add a “Type 10 error” for rows that were entered incorrectly. Example: StringTag is not a number
2. Create various perspectives of the database where short summaries of what is going on can be viewed efficiently.
   1. Show a broad perspective of every site, located on a graphical map. Dots are situated where sites are on the map. The dot is colored according to priority.
   2. Show a broad perspective of every string within a site. Show a graphical display of the strings in their general orientation within the building on a map of the building. One dot represents one string, colored according to priority.
   3. Show a perspective of a single string and all of the cells on the string and display their last recorded readings.
   4. Rotate the string perspective to view the past readings listed by ReadingDate.

# To Do

* Note: Greg database has an issue: StringTag is a VarChar but should be a float and there are rows in the database that were entered incorrectly: the model was written in the StringTag column.
* Ask Emerson: The BDS40 Faults document specified that some faults occur when CellResistance is Null. There are no Null values in the Greg database. Could it be possible that, instead of Null values, we could expect something else? Random 0-65535? 0? Unchanging?
* Ask Emerson: Need clarification on Fault 3 “Internal resistance value is over ranged”: There are no values with 65535. ~~Should we test only for R=65535?~~ Should we be concerned about if V < 13.1 and what does that mean? ~~What/when is a resistance test? Could it be possible that, instead of 65535, we could expect something else? Values > 50000? High, outlying values?~~ **In the BDS-40 Fault conditions.pdf for fault 5 there is an example that shows 8 adjacent Cells that have maxed out resistance, however it is not 65535, but 61366. All 8 batteries have the same resistance. It is speculated that maxed out resistance may be less than 65535, but is nonetheless a Type 5 error.**
* ~~Change the Type 4/5 errors to account for maxed readings instead of exactly 65535~~
* Type 6 🡪 There appears to be no consistent pattern for the number of hours/days between readings on a single battery, which means there will be a large amount of these errors.
* Type 7 🡪 In the figures for type 7 in BDS-40 Fault conditions.pdf, what do the yellow and red lines indicate and how are they determined?
* Type 8/9 🡪 Speculated that these could be done faster in TSQL.
* These queries may only be needed to be calculated once, therefore maybe simply select new data.
* Table name may be vulnerable to sql injection.
* **Type 8/9 (previously step 1 and step 2) need to use the Faults table instead of adding a column to the BatteryReadings table.**
* Set outlier removal to 5x instead of 3x.
* Type 3 🡪 Instead of 65535, check 50000 threshold.

**Phases**

Phase 1a will examine the data and search for two different categories of faults: those arising in the instrumentation used to gather the data, and those arising from problems with the underlying battery systems being monitored. Within each of these groups, individual fault types will be identified, spanning all fault types we are aware of in both categories. We will also note whether a fault merits immediate action (such as replacing a misbehaving instrumentation unit or jar), should be marked and preserved for future processing, or should be discarded entirely as being useless for additional fault monitoring tasks.

Phase 1b will operate on data which has been processed through Phase 1a. Because Phase 1a  will catch all instrumentation faults and flag them for appropriate action, Phase 1b will focus on battery faults, as well as long-term trends in battery behavior which do not immediately rise to the level of fault (e.g., the “normal” data which was unflagged by Phase 1a). This tool will use sensor information along with both experimental and domain knowledge about battery trends to predict when a battery is nearing end-of-life and should be replaced. Reports will be generated giving this information for all of the batteries being monitored.

**Deliverables**:

In both phases, the primary deliverable will be a software tool implementing the fault detection and prognosis task. Depending on the client's specifications, we will either develop two distinct tools (one for Phase 1a and another for Phase 1b), or one tool will be developed which begins with Phase 1a functionality and later incorporates Phase 1b as well. This tool will operate directly on the SQL database, reading entries from this database and writing new information back as appropriate, and may also prepare reports for direct parsing by human operators.