Change request log

1. Concept Location

Step #	Description	Rationale
1	We booted up the Tomcat server, ran the Mango software, and logged in as an admin user.	
2	We explored the watch list page and the point details page.	To identify the modules, we will be working on for this change request
3	We used the inspect element feature of the web browser (Firefox) web browser and checked the network tab for analysis.	We wanted to check what methods were invoked when the watchlist loads
4	From the results, we decided to inspect “watchListDwr” and “MiscDwr” classes	We found out that our data is being populated from methods of this class
5	We inspected both the classes	Upon inspecting and Eclipse IDE’s call hierarchy and other features, we found out the data points were polled in the doLongPoll method of the MiscDwr class
6	We followed similar steps on the “point details” page	Since the change was supposed to be reflected on the point details page as well
7	From the results, it was obvious “DataPointDetailsDwr” is our class	Again, form the classname, we figured this class was responsible for populating the history table with point values
8	We inspected the DataPointDetailsDwr class	From step 6, we decided to inspect the “getHistoryTableData” method since it populated the point data in the history table
9	We temporarily updated the code to set the value of rendered point value time to “1” instead of the value it fetched from the database	We wanted to check if the value for our frontend was being set in this method
10	We marked the DataPointDetailsDwr and MiscDwr classes as located	From the previous steps, we confirmed this was these were the classes we needed to perform changes in.

Time spent (in minutes): 40.

Classes and methods inspected:

- /Mango/src/com/serotonin/mango/web/dwr/MiscDwr.java
    - doLongPoll
- /Mango/src/com/serotonin/mango/web/dwr/WatchListDwr
    - getPointData
    - getPointDataImpl
- /Mango/src/com/serotonin/mango/web/dwr/DataPointDetailsDwr
    - getHistoryTableData

2. Impact Analysis

Step #	Description	Rationale
1	We listed all the methods called by DataPointDetailsDwr class	To track the source, we need to update in order to incorporate the change in the history table

2	We checked out the getHistoryTableData method and using Eclipse IDE features, we tracked the getPointData method	Upon tracking, we realized the data for the History table was fetched using this method
3	Upon further analysis, we figured the data for our frontend was being prepared in the getHistoryTableData	getPointData method was responsible for fetching the data. However, getHistoryTableData was the method used to prepare the data to be displayed
4	To test the impact, we updated the RenderedPointValueTime list with "1" instead of setting the data fetched from the backend	Upon building and deploying the project, we found out the history table for all the values was filled with our new value and hence we found out our class to be updated
5	We followed similar steps for WatchList page, using the IDE's drill down/up, call history features.	We found data for our WatchListDwr was prepared in the MiscDwr class
6	We performed similar steps of updating the state value to 1 in the doLongPoll method of the MiscDwr class	While drilling, we came across multiple getPointData classes, but all these methods were responsible for fetching data from the backend and not preparing our data to display on the webpage
7	While drilling down and exploring all the call histories and method usages, we landed on NumericValue class	Since the toString method was frequently called while preparing the data, we landed on the source where the method was overridden. The class NumericValue had numerous overridden methods and one Double toString method seemed to be the source for our change
8	We updated the overridden toString method	We updated the toString method to truncate the Double with at the most 2 decimal places.
9	Build and redeployed the code with this new update.	Upon redeploying, we saw all the point values (Doubles) were being displayed as intended anywhere in the system, without updating multiple methods that were displaying the point data on the webpage (doLongPoll method and getHistoryTableData method)
10	Since changing Numeric Value class methods resulted in modifying the data layer, we did not mark it to be changed and instead marked MiscDwr and DataPointDetailsDwr classes to be changed	

Time spent (in minutes): 70.

Classes and methods inspected:

- /Mango/src/com/serotonin/mango/web/dwr/MiscDwr.java
    - doLongPoll
- /Mango/src/com/serotonin/mango/web/dwr/WatchListDwr
    - getPointData
    - getPointDataImpl
- /Mango/src/com/serotonin/mango/web/dwr/DataPointDetailsDwr.java
    - getHistoryTableData
    - getPointData
- /Mango/src/com/serotonin/mango/rt/dataImage/PointValueTime
- /Mango/src/com/serotonin/mango/rt/dataImage/DataPointRT
    - getPointValues
    - getPointValue
- /Mango/src/com/serotonin/mango/rt/dataImage/types/NumericValue
    - toString

3. Actualization

Step #	Description	Rationale
1	Updated the doLongPoll method in MiscDwr	To truncate the point value, we updated the value in newState List of doLongPoll method (Lines – 327 to 334). We added a try catch block just to catch any exceptions.
2	Similarly, we updated the getHistoryTableData method in DataPointDetailsDwr	To truncate the point value in the History table, we updated the value in rpvt.setValue code in the getHistoryTableData method (lines – 97 to 110). We added a try catch block just to catch any exceptions.
3	We then redeployed the code and manually performed a regression test.	To verify if all the webpages loaded correctly with correct data.
4	We then created functional tests to test our updated code	To verify if our update did not break any functionalities of the system and if our intended changes are reflected on the webpage

Time spent (in minutes): 30.

Classes and methods updated:

- /Mango/src/com/serotonin/mango/web/dwr/DataPointDetailsDwr.java
    - getHistoryTableData
- /Mango/src/com/serotonin/mango/web/dwr/MiscDwr.java
    - doLongPoll

4. Validation

Step #	Description	Rationale
1	Test Case 01: Test if the following web pages load correctly: 1. login 2. Watchlist 3. Points history 4. Reports Inputs: None Expected output: All web pages should load and display the correct information.	All the web pages should load and display correct information post changes. Test case passed.
2	Test case 02: Check if point values on the existing watchlist display data upto 2 decimal places: 1. Greenhouse 2. Digester 3. Power Inputs: None Expected output: Point values should be displayed with upto 2 decimal points	To check if the update in the source code displays correct data on the front end. Test case passed.
3	Test case 03: Check if point values in the history table are displayed with upto 2 decimal places for the following: 1. GH-Power - rest of GH (W) – for multiple different values with decimal places	To check if the update in the source code displays correct data on the front end. Test case passed.

	2. GH-Power - Rm1 (W) – for multiple values with 0 as their decimal place 3. 3_Light_windowINT – for the same value repeated multiple times in the history table	
4	Test case 04: Check if point values in the statistics box are displayed with upto 2 decimal places for the following: 1. GH-Power - rest of GH (W) – since stats will be calculated based on numerous different values, the results should be displayed with upto 2 decimal place values 3_Light_windowINT – since stats will be calculated based on numerous different values, the results should be displayed with upto 2 decimal place values	To check if the update in the source code displays correct data on the front end. Test case passed.

Time spent (in minutes): 15.

5. Summary of the change request

Phase	Time (minutes)	No. of classes inspected	No. of classes changed	No. of methods inspected	No. of methods changes
Concept location	40	3	0	4	0
Impact Analysis	70	6	7	10	5
Actualization	30	2	2	2	2
Verification	15	0	0	0	0
Total	155	11	9	16	7

6. Conclusions

For this change request, we took some time in understanding the software. Concept location was easy since we were able to use the browser inspect element feature. This helped us analyze all the methods that were being invoked and check what data was being provided by the same.

We invested a lot of time in impact analysis. In this step, we thoroughly checked the impact of our changes as we did not want to break any existing functionality in order to incorporate the new change. To do this, we actively used Eclipse's numerous features like call history tracking, method declaration, usages, etc. Also, in this step, we found the base class that consisted of the overridden methods. We initially had marked this method to update since making a change in this method resulted in the intended change, without updating any other methods but since updating this method resulted in changing the data layer, we unmarked it and instead selected the doLongPoll and getHistoryTableData classes. We manually tested our code and verified if the system performed without any issues.