**Basic Guidelines for Implementing New Manual Intervention Analysis**

The entire process can be classified broadly into 3 different procedures, namely,

1. Data Collection
2. Data Cleaning
3. Data Extraction and Classification

The end product of this is a consolidated data file that is ready for various kinds of analysis that can be done on it.

**Scope of this Document:** We are looking to generally define the approach that was adopted to generate the manual intervention data through this new initiative. The implementation was done in Python. However, other implementation option exploration is also encouraged.

**Data Collection:**

The primary source of data for manual intervention is the operator action report that is generated by DCS system. This contains records of every activity done by any user along with User ID, Station and time at which the action was exerted on the system.

These reports are generated at regular intervals, usually 6 to 10 reports per day are generated based on configuration. It is vital to ensure to use Operator action report for the entire plant rather than for any specific block.

To get a good insight into the system, evaluating a month’s worth of actions is optimal. Having said that, it is also cumbersome as the number of reports generated on monthly basis is huge. Data collection step essentially combines the entire data into one single data file

**Data Cleaning:**

The generated excel file often contains junk values in large numbers owing to the large number of empty cells in the documents. For effective analysis and for data efficiency, we need to remove the junk values prior to proceeding further. It is also vital to ensure that the generated file has data is the required format that is easy for analysis.

**Data Extraction and Classification:**

Now, with a consolidated data free from junk values in our hand, we are ready to start extracting information and classify actions.

Some information we have so far decided to extract from the data includes,

1. Block
2. Change Type or Category
3. Equipment Group
4. Sequence
5. Interaction Object
6. UserID
7. Action Variable
8. Object Type
9. Action Class

However, we are constantly exploring many other information that can be extracted from the file. We can also explore combining the data from here with MES data to get more insights.

We shall now discuss in brief the extraction procedure for the said list of information from the raw data file.

**Block:** This signifies the block to which the action belongs. This is discerned from the “Path” information from the raw data file. “Path” has a constant structure to it like,

[Control Structure]Root/Control Network RB/RB\_Block/Applications/Application\_1/Control Modules/ReaTraDir/FCV84301

From here, we can directly extract the block information using regular expression or search expressions.

**Change Type/ Category:** In general, interactions associated with control are associated with “Control Structure”. There are other structures as well that are linked with other kinds of interactions like “Location Structure” which in general is used for graphic related change. In Khandala, we have decided to focus only on Control Structure related interactions. This information is extracted from “Path” using regular expressions or search expressions.

**Equipment Group:** The general structure of a DCS application is as follows,

***PROJECT(Block level) 🡪 APPLICATIONS (Classify sub area in blocks) 🡪 Single Control Module (Group Equipment under a single category) 🡪 Control Modules (Individual Sequences)***

Equipment group essentially picks up the Single Control Module information from path to decide the group of equipment under which a particular interaction belongs to. If the said structure is violated, then caution is to be practiced while analysis.

**Sequence:** It denotes to which particular sequence the interaction belongs to. The information is sourced from “Path” and can be extracted from the control module section of the path.

**Interaction Object:** This is the end object upon which the interaction was done. This is also extracted from path information. It is usually the last of the second last information under path.

**User ID:** There is a separate field called User Account in Operator Action report which contains this information. It is directly fetched from there and standardized for ease of analysis.

**Action Variable:** This is the kind of action that was taken on the object. This information is directly sourced from “Message” field from the report.

**Object Type:** This involves classifying the object upon which the action was made into broad categories like ***Sequence, Valve, Agitator*** etc. This is tricky and is based on the nomenclature followed in the respective plants. For Khandala, broadly, the nomenclature is standardized with few exceptions. The Tag gives us information about the type of object it is and a unique ID. The idea is to separate the two and use the part that identifies the type of object to classify it. For example, in **XV1\_12345** “XV1” signifies that it is a single stage **Valve**. For implementation in Khandala, we used delimiters to separate out the type from tag and dictionary data structure to look up the type and classify the same.

**Action Class:** The operator action report captures every single interaction done with the system by any user. While this is a great thing, we need to identify the actions that we are interested in and have to define them. In Khandala, for every object, we defined the kind of interaction we would track and consider them as manual interventions. Like for example, putting an equipment in auto is not MI while opening a valve is considered MI. The operator action report captures both. So we want to classify every action into MI and NA. In Khandala, this was implemented by leveraging dictionary data structure and by using a dictionary file to define the actions.

The source code used to generate and classify is still in beta testing stage and is being updated. Any change in the approach or the source code will be updated in the report.