CMS Developer Manual

Version 1.0

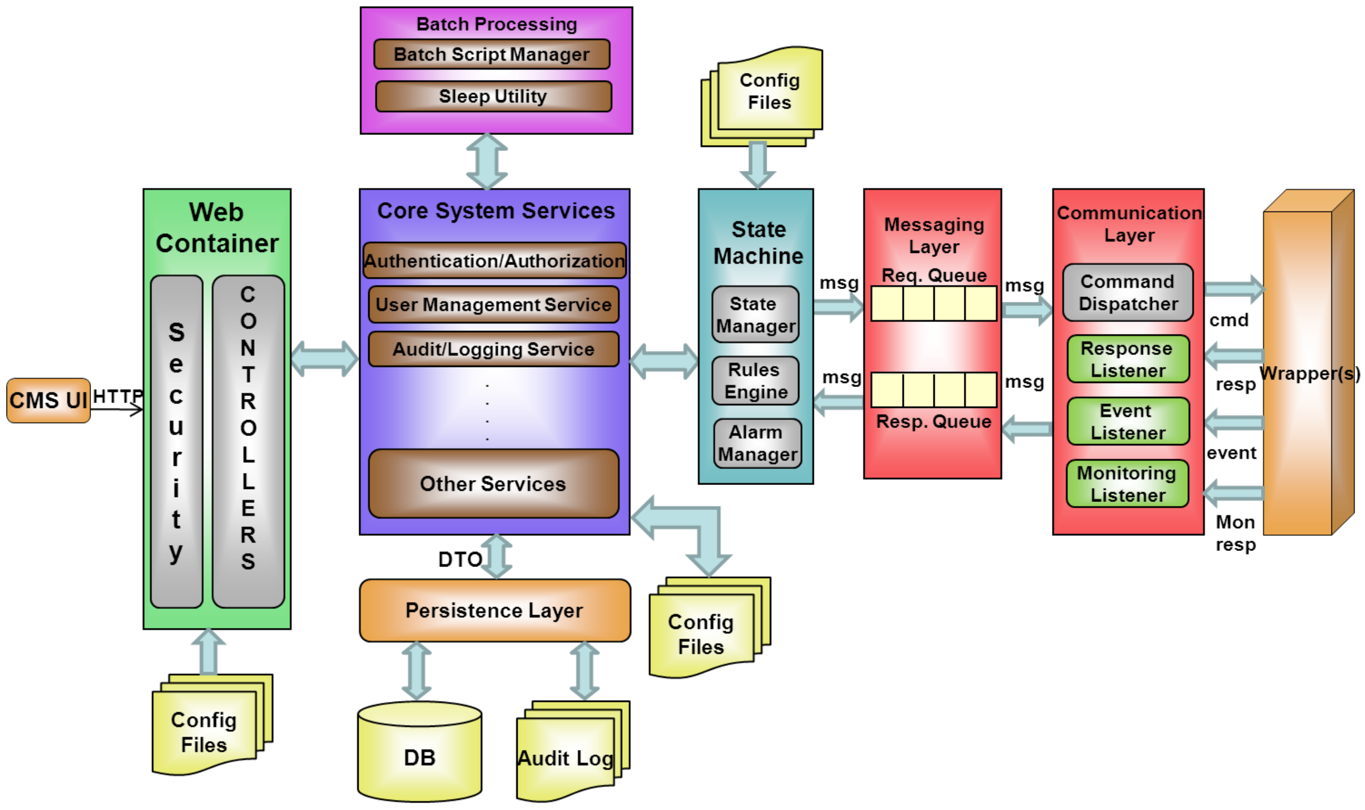
PSL TEAM

5/4/2012

Contents

1 CMS Architecture

Following diagram depicts high level architecture of CMS.



CMS Architecture

1.1 CMS UI

It is web UI displayed the as web page in browser responsible to manage the HTTP requests and responses. UI consists of HTML components and flex components.

1.2 Web Container & Configuration Files

1.2.1 Web Container

Web container is responsible for providing a platform to spring container. Spring container is responsible for managing life cycle of various application classes.

A web container along with spring container provide runtime environment for web components that includes security, concurrency, lifecycle management, transaction, deployment, and other services. The controllers routes requests from the CMS UI to core system services.

1.2.2 Configuration Files

The CMS internal configuration files like applicationcontext.xml etc

1.3 Core System Service & Configuration file

1.3.1 Core System Service

Core system service took the request from controller, analyze the request and interact with various system components (Batch processing, Sate Machine, Data layer etc...), process the request and send the response to controller. This layer can be called as business layer of CMS.

1.3.2 Configuration file

All the configuration files used for the CMS, like all the <susb_system>_command.xml etc. These files help increase the level of configurability of CMS.

1.4 Batch Processing

It processes the batch processing request coming from user, executes the batch and sends the result to caller.

1.5 Persistent Layer and DB

This layer handles database related operations; it also hides the database complexity from the programmer giving way for easy maintenance and migration to different database if needed in future.

1.6 State Machine & Config Files

1.6.1 State Machine

State machine shows status of the sub systems, manages the state of the sub systems, take the corrective actions like raise the alarm and execute the commands depending upon the condition. It is a central processing unit of CMS, responsible for overall functional behavior of CMS.

1.6.2 Configuration file

The inter configuration file used for the state machine like state configuration file, rule files etc.

1.7 Messaging Layer

It provides the asynchronous behavior to CMS by routing the messages coming from the CMS to Communication layer. It also routes responses from external system to CMS asynchronously. This asynchronous behavior is vital to increase the overall scalability of CMS.

1.8 Communication Layer

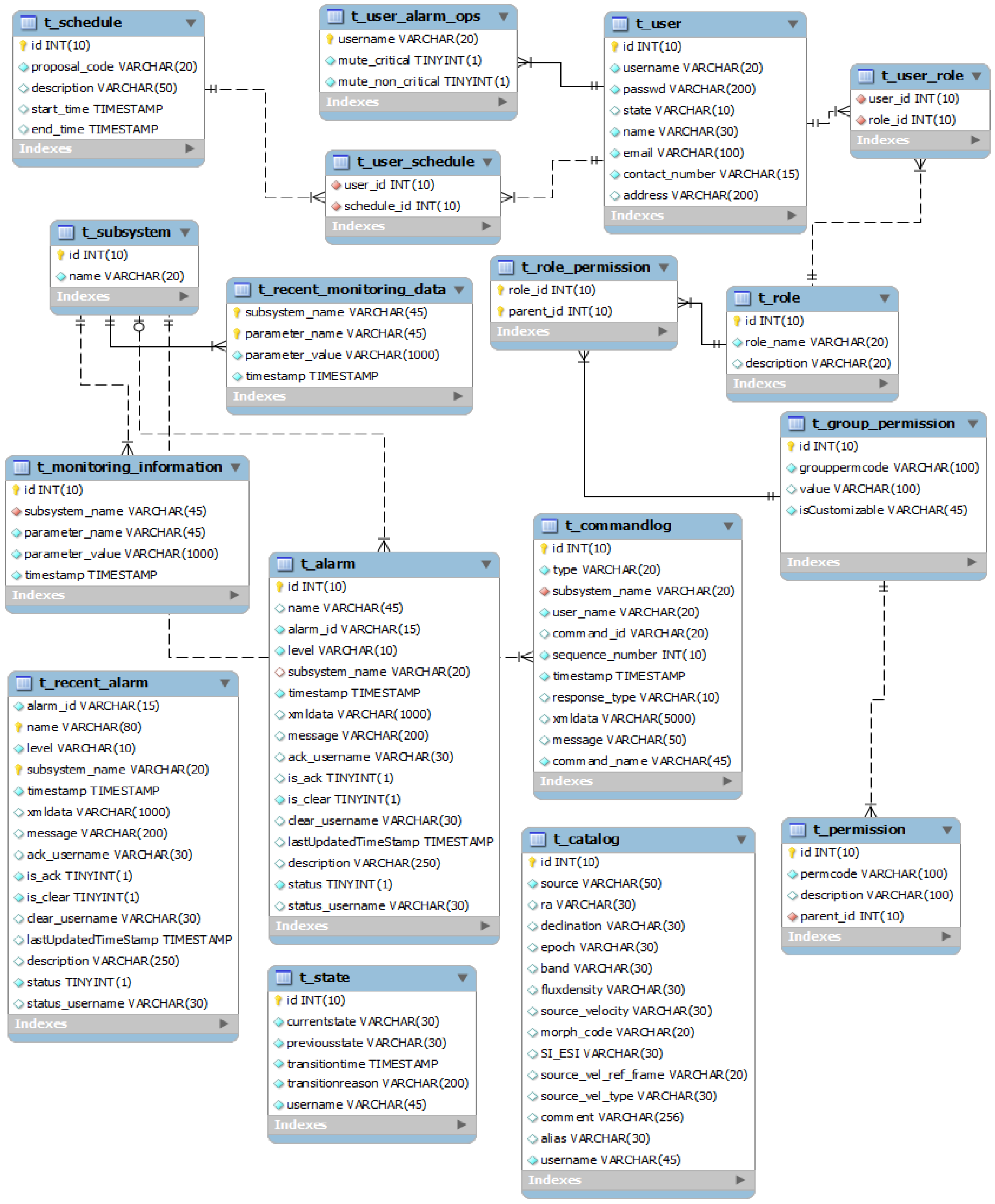
It is responsible to send the data from CMS to wrapper and wrapper to CMS. Communication layer uses sockets for communication.

1.9 Wrapper

It is responsible to handle the CMS request, process it and take the appropriate action on actual hardware and send the original response back to CMS. It is an external component for CMS.

2 Database Design

Following diagram shows the Entity relationship diagram for various tables used by CMS.



3 Wrapper Communication

3.1 Introduction

3.1.1 Purpose

This section describes the schematics communication between the CMS and wrapper.

3.1.2 Scope

The CMS and wrapper communication purpose is as mentioned below:

1. Sending Command from CMS to Wrapper
2. Receiving acknowledgement from Wrapper
3. Receiving response from Wrapper
4. Receiving asynchronous event from wrapper
5. Receiving Monitoring information from wrapper

Server socket requirement would be as below:

1. Command Server Socket:
   Server Socket for accepting commands (residing at wrapper)
   The number of Command Server socket would depend on the number of wrappers i.e. **each wrapper would run a Server Socket to accept commands**.

2. Event Server Socket:
   Server Socket listening for incoming asynchronous events and responses
   The CMS will have only one Event Server Socket.

3.1.3 Glossary & Acronyms

IUCAA	– Inter-University Centre for Astronomy and Astrophysics
NCRA	–National Centre for Radio Astrophysics
IGO	– IUCAA Girawali Observatory
GMRT	- Giant Meter Wavelength Radio Telescope
URL	– Uniform Resource Locator
I/O	– Input/output
IDE	– Integrated Development Environment
JDE	– Java Development Environment
HTTP	– Hypertext Transfer Protocol
XML	- Extensible Markup Language
GUI	- Graphical User Interface
CMS	- Control and Monitoring System
RDBMS	– Relational Database Management System
Java EE	– Java Enterprise Edition

3.1.4 References

Please refer the following documents available:

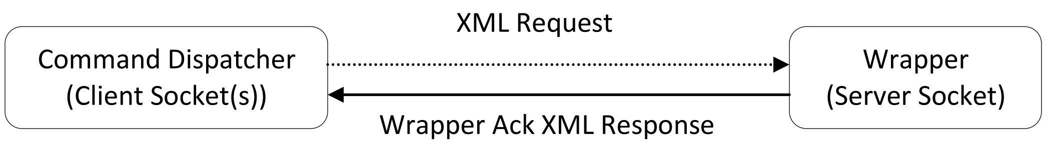
- Telescope_CMS_Requirement_Specifications_Document_V2[1].0.pdf
- Command_structure_ver1.3.pdf
- Servo_Telemetry_Commn_Protocol.pdf
- Technical Discussion meeting with IUCAA team
- Technical Discussion meeting with NCRA team

3.2 Overview

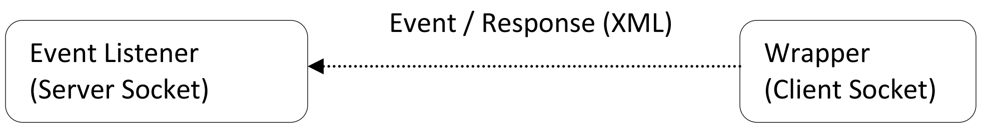
3.2.1 Communication Layer

This layer actually transports the commands to the wrapper as well as receives the response/event from the wrapper. All the communication between CMS and wrapper is in terms of structured well defined xml documents. This layer is made up of following components –

- **Command Dispatcher** – This component is responsible for delivering commands to appropriate wrapper for subsystem. The Command Dispatcher communicates with the wrapper layer using socket connection.



- **Event Listener** – This component receives various acknowledgements, responses and events from wrapper and hand it over to messaging layer so that business layer can process the response.



All wrappers will send the Events/Response to the same Server Socket.

3.2.2 Socket Communication

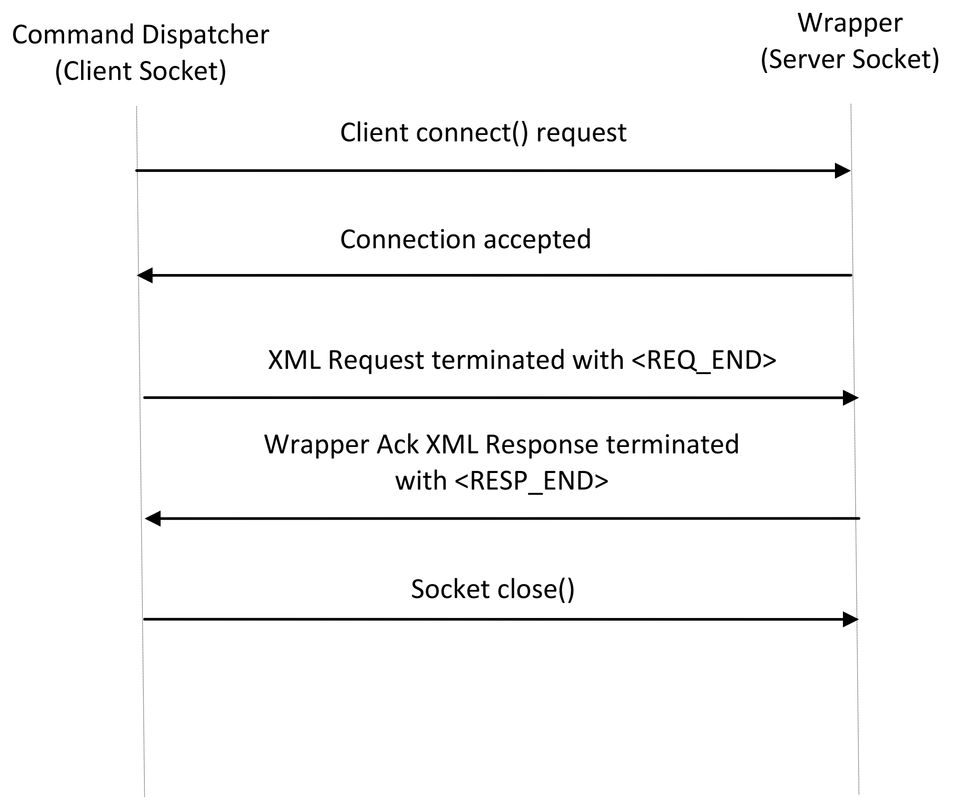
The Socket server needs to support multiple clients so needs to follow the Multithreading approach i.e. separate thread for each client connection.

Pros of Multithreaded Socket Server:

1. Multiple clients can get connected to server.
2. If one of the clients goes in IO Blocking mode, still the server is reachable through other clients.
3. This approach can support CMS scalability.

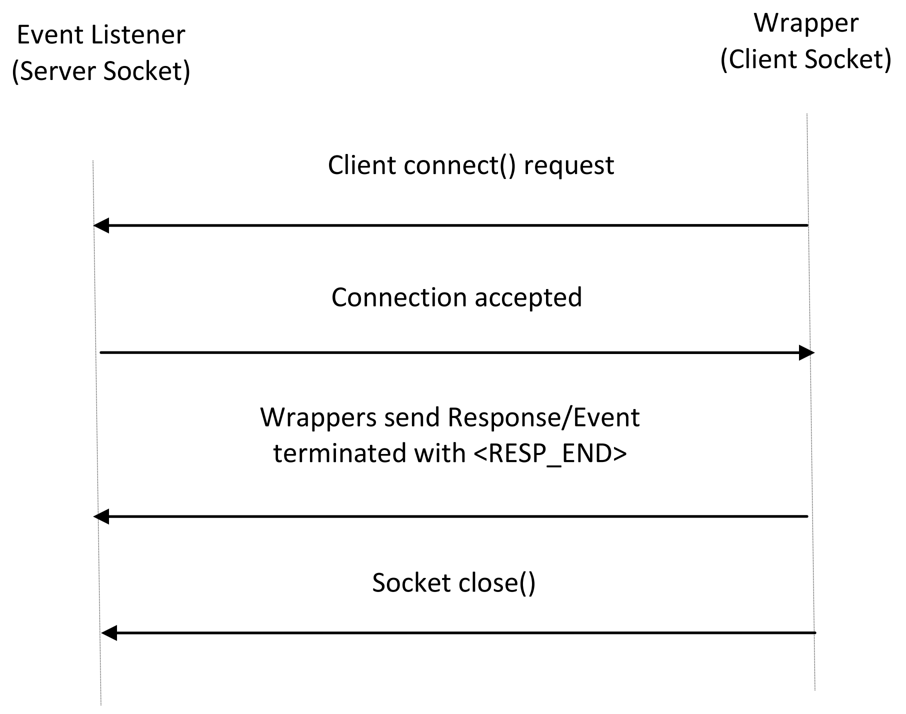
3.2.3 CMS Command Dispatcher

The command Dispatcher is responsible for sending the commands over the socket. For this it will implement a client socket connection on CMS end and a Server Socket at Wrapper end. The handshaking and data flow would be as specified in below figure:



3.2.4 CMS Event Listener

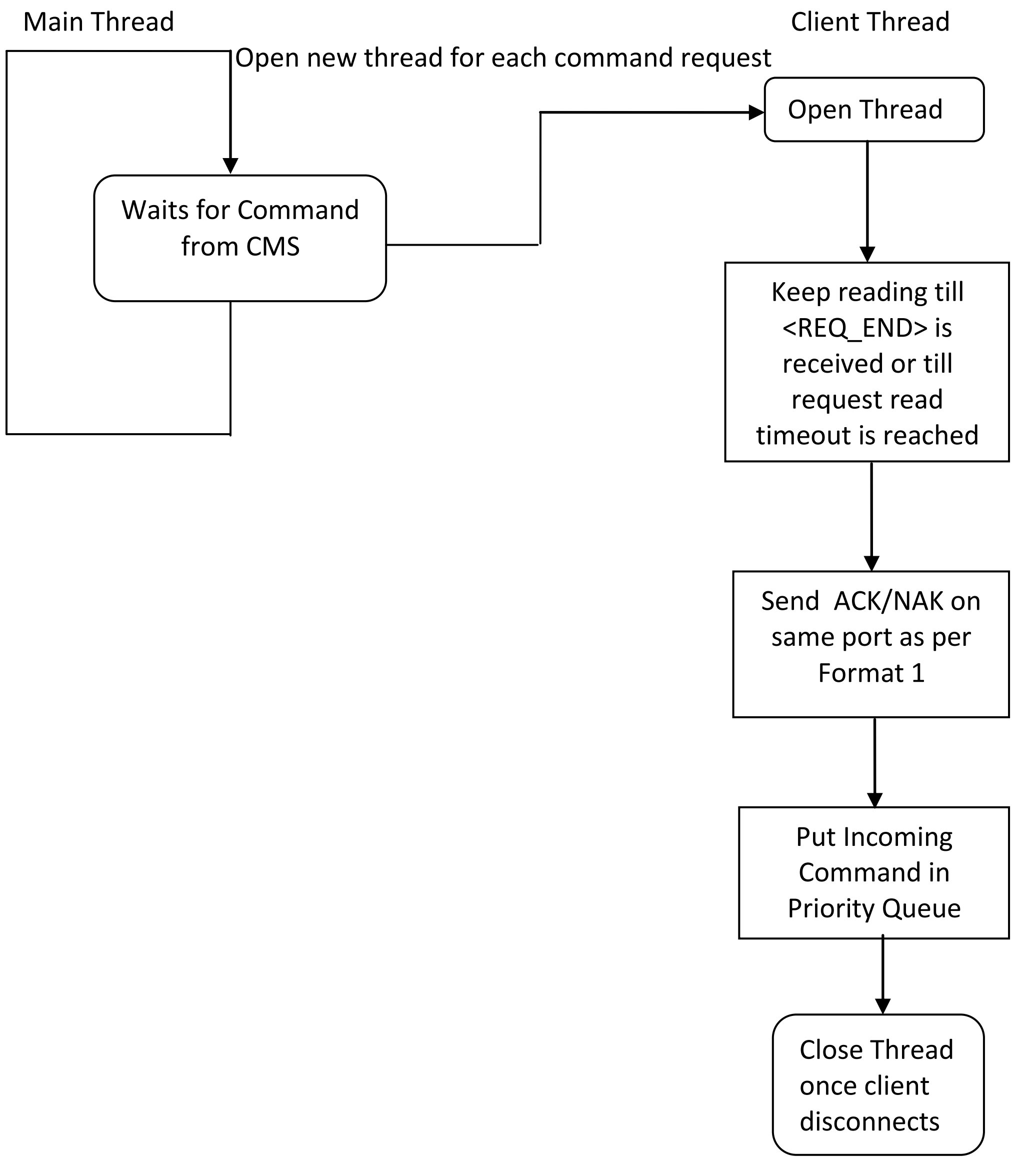
The Event Listener is responsible for various acknowledgements, responses and events from wrapper. For this the CMS will implement a Server Socket and the wrapper will open a client socket. The handshaking and data flow would be as specified in below figure:



3.2.5 Wrapper Command Acceptor

The wrapper Command Acceptor accepts the command from the CMS and sends wrapper acknowledgement on the same connection.
Each Wrapper would have to implement a Server socket to accepts commands

Command Acceptor Module (Server socket for accepting commands)


Main Thread
Client Thread
Open new thread for each command request
Waits for Command from CMS
Open Thread
Keep reading till <REQ_END> is received or till request read timeout is reached
Send ACK/NAK on same port as per Format 1
Put Incoming Command in Priority Queue
Close Thread once client disconnects

Format 1:

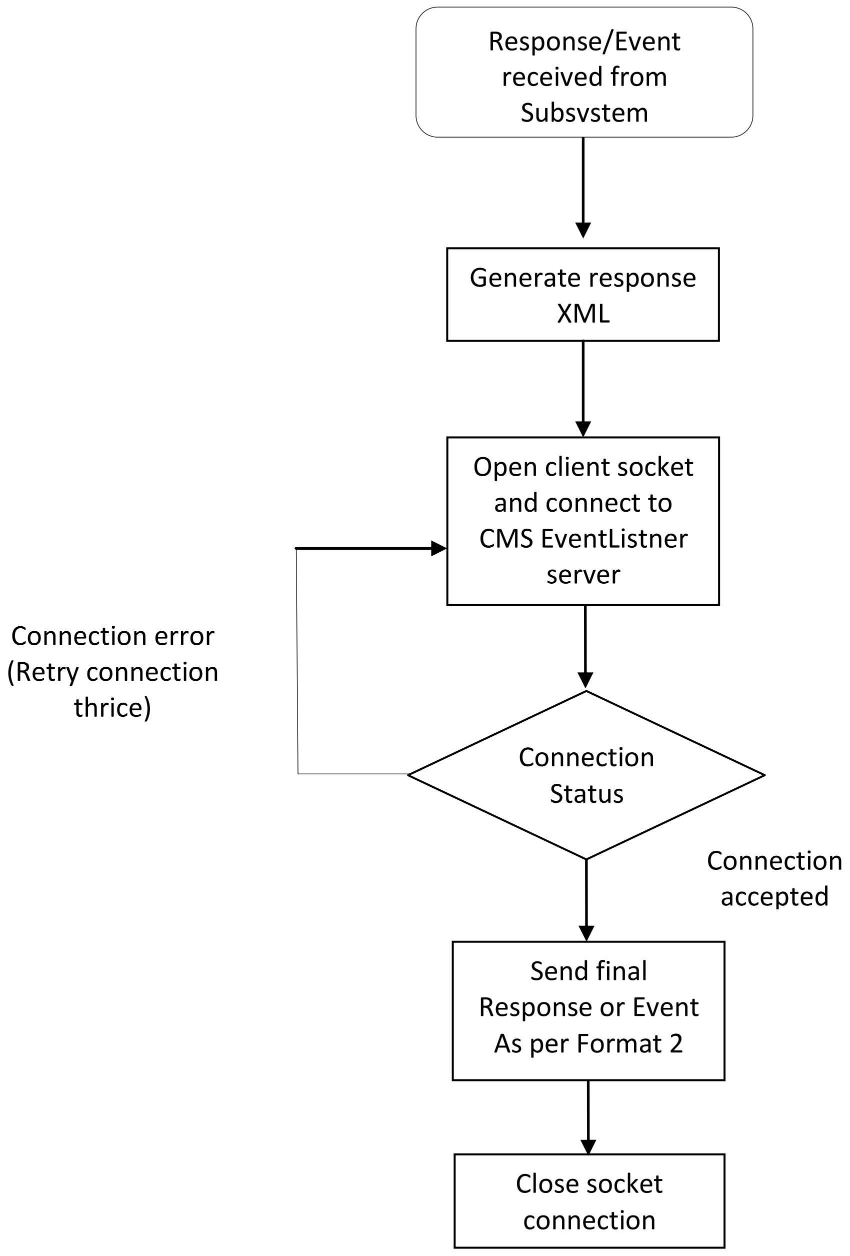
Wrapper-ACK – to the same port on which request arrived

```
<responses>
        <response>
                **<seq>1</seq>**
                **<id>44</id>**
                **<name> MOVE</name>**
                **<systemid>servo</systemid>**
                **<version>1.0</version>**
                **<timestamp></timestamp>**
                <code>10</code>
                <event>10</event>
                <message>wrapper ack</message>
                <data></data>
        </response>
</responses>
```

*Note: Data indicated in the **purple color** is command dependent and will be available from the request sent by CMS. For more details on the response format refer to Request and Response format section of this document.

3.2.6 Wrapper Event/Response sender

This component sends the event or asynchronous response (intermediate and final) to CMS. For this the wrapper needs to open a client socket connection and connects to the CMS EventListner server and writes the response. Once response is written the client can disconnect from the server.


Response/Event received from Subsvstem
Generate response XML
Open client socket and connect to CMS EventListner server
Connection error (Retry connection thrice)
Connection Status
Connection accepted
Send final Response or Event As per Format 2
Close socket connection

Format 2:

Final Response-

```
<responses>
        <response>
                <seq>1</seq>
                <id>44</id>
                <name>MOVE</name>
                <systemid>servo</systemid>
                <version>1.0</version>
                <timestamp></timestamp>
                <code>10</code>
                <event>12</event>
                <message>move successful</message>
                <data></data>
        </response>
</responses>
```

*note: Data indicated in the **purple color** is command dependent and will be available from the request sent by CMS. For more details on the response format refer to Request and Response format section of this document.

3.3 Request and Response Format

Following are request and response format for communication between CMS and Wrapper.

3.3.1 Request Format

Following is the request format of the xml that would be sent to a wrapper by CMS.

```
<commands>
    <command>
            <seq>1</seq>
            <id>41</id>
            <name>position</name>
            <systemid>subsystem1</systemid>
            <version>1.0</version>
            <timestamp>2011-01-11T19:27:23.110+05:30</timestamp>
            <priority>1</priority>
            <syntax>ax,ang1</syntax>
            <data>
                    <param>
                            <name>subsystemid</name>
                            <value>1</value>
                    </param>
                    <param>
                            <name>ax</name>
                            <value>A</value>
                    </param>
                    <param>
                            <name>ang1</name>
                            <value>10</value>
                    </param>
            </data>
    </command>
</commands >
```

After the command be sent **<REQ_END>** is sent to indicate command end.

Following is brief description about the various fields in this format –

seq – This will contain a unique number indicating the count of commands fired. CMS will populate this value and send it as part of Command request.

id – This represents the command id for CMS system, for various subsystems it would be actual opcode corresponding to the command fired. CMS will send this as part of Command request xml. This can contain alphanumeric characters, with maximum length of 20 characters.

name – This is display name of the actual command fired by the user. CMS will send this as part of Command request xml.

systemid – This is used to distinguish between various subsystems attached to CMS via wrapper. CMS will populate this as part of Command request.

version – subsystem version

timestamp – The timestamp at which the request was created.

priority – This is used to set the priority. 0 - being low priority and 1 being high priority.
Note: Since priority is added as a part of request, a separate Server Socket would not be required on wrapper end to cater to High Priority commands.

syntax – It describes the command parameter sequence to be used by wrapper to sort the parameters

data – Some of the commands may need additional input apart from opcode and systemid, all such additional information will be sent as name-value pair under DATA field. Wrapper can parse DATA fields to retrieve corresponding information.

3.3.2 Request Component Size

All the components below will go as String in the request xml, the data types should be referred while extracting the component from the xml.

Component Name	Data type	Size in Characters (UTF)	Size in bytes
request xml	String	5000	10,000
seq	Long (unsigned)		4
id	String	20	40
name	String	20	40
systemid	String	20	40
version	String	10	20
timestamp	TimeStamp		4
priority	Short		1
param-name	String	45	90
Param-value	String	45	90

3.3.3 Response Format

Following is the sample response format of the xml that would be sent by a wrapper to CMS.

```
<responses>
        <response>
        <seq>1</seq>
        <id>41</id>
        <name>position</name>
        <systemid>subsystem1</systemid>
        <version>1.0</version>
        <timestamp>2011-01-11T19:27:23.110+05:30</timestamp>
        <priority>1</priority>
                <code>10</code>
                <event>12</event>
                <message>Positioned successfully</message>
                <data>
                        <param>
                                <name>subsystemid</name>
                                <value>1</value>
                        </param>
                        <param>
                                <name>ax</name>
                                <value>A</value>
                        </param>
                        <param>
                                <name>ang1</name>
                                <value>10</value>
                        </param>
                </data>

                <alarms>
                        <alarm>
                                <id></id>
                        <name></name>
                        <message></message>
                        <level></level>
                        </alarm>
                </alarms>
        </response>
</responses>
```

Following is the descriptive information about response format that CMS expects from wrapper.

seq – This will contain a unique number indicating the count of commands fired. CMS will populate this value and send it as part of Command request, CMS expects wrapper to send back same value as part of each and every ack, intermediate or final response for corresponding command. This field is optional for events as they are generated asynchronously.

id – This represents the command id for CMS system, for various subsystems it would be actual opcode corresponding to the command fired. CMS will send this as part of Command request xml and expects wrapper to send it back for each and every ack, intermediate or final response for corresponding command. This can contain alphanumeric characters, with maximum length of 20 characters.

name – This is display name of the actual command fired by the user. CMS will send this as part of Command request xml and expects wrapper to send it back for each and every ack, intermediate or final response for corresponding command.

systemid – This is used to distinguish between various subsystems attached to CMS via wrapper. CMS will populate this as part of Command request, CMS expects wrapper to send same value back as part of each and every ack, intermediate or final response for corresponding command.

version – subsystem version

timestamp – The timestamp at which the request was created. Optional for events

priority – This provides the priority of response. High priority responses will be consumed first.

code – This field is populated by wrapper while sending response back to CMS for a given command. Following are the various values and their interpretation by CMS. CMS expects wrapper to adhere to this convention while sending back ack, intermediate, final response.

VALUE	**Meaning**
10	SUCCESS
>10	FAILURE

event – This field is populated by wrapper while sending response back to CMS for a given command or any asynchronous event. Following are the various values and their interpretation by CMS. CMS expects wrapper to adhere to this convention while sending back ack, intermediate, final response.

VALUE	**Meaning**
<10	CMS internal purpose should not be used
10	Wrapper Ack
11	Subsystem Ack
12	Final response
>12	Any asynchronous event or intermediate response, Wrapper can send multiple responses with value of **event > 12**. This would be helpful in avoiding timeouts especially in long running commands. When CMS receives intermediate response with event > 12, it increases the timeout period for corresponding command. The increased timeout period is governed by following algorithm –

	1. If timeout configured for corresponding command in *_commands.xml (let's call it **cmdTimeout**) then New timeout = old timeout + **cmdTimeout.** 2. Otherwise New timeout = old timeout + **defaultTimeout** (from cms.properties).

message – This field can be used to communication information such as command success, failure or any other message to the end user. In case of general failure like invalid command syntax wrapper will need to indicate the error message. This field can be used for the same.

data – This field will be populated by wrapper while sending intermediate, final response or asynchronous response to CMS. This field can contain any number of name-value pairs. Optional if no data is to be sent.

alarm – In some cases like hardware failure, subsystem will generate alarm and send it to CMS via wrapper. In such cases wrapper can make use of ALARM tag, which contains following fields-

id – alarm identifier
name – alarm name
message – Contains the brief description about the ALARM, which will be shown to the user in ALARM window.
level – Wrapper can assign the priority for such ALARM and share the information with CMS using this field.

Level can be 1 - Information, 2,3 - Warning, 4,5 – Critical

Note that the response structure is fairly generic to accommodate various types of responses from subsystem. It may be noted that in some cases some of the fields cannot be populated by wrapper for e.g.

a) Fields like id, name, seq are not relevant when wrapper sends hardware failure alarms
b) If a command was executed successfully it cannot have ALARM information on it
c) Response for some of commands may not have any data associated with it so <data> field cannot be populated

Only in such cases wrapper can exclude these fields while sending information back to CMS.

3.3.4 Response Component Size

All the components below will go as String in the response xml, the data types should be referred while extracting the component from the xml

Component Name	Data type	Size in Characters (UTF)	Size in bytes
response xml	String	5000	10,000
seq	Long (unsigned)		4
id	String	20	40
name	String	20	40
systemid	String	20	40
version	String	10	20
timestamp	TimeStamp		4
priority	Short		1
code	Integer		4
event	Integer		4
message	String	50	100
param-name	String	45	90
Param-value	String	45	90
id (alarm)	String	20	40
name (alarm)	String	20	40
message (alarm)	String	50	100
level	Short		1

3.4 Monitoring Configuration, Request and Response Format

Following are configuration, request and response format for communication between CMS and Wrapper for Monitoring information. This structure is still evolving and minor changes may be required to be done considering the implementation perspective.

3.4.1 Monitoring Command Configuration

Following is the configuration for doMon command in the subsystem commands xml.

```
<command>
        <name>doMon</name>
                <id>101</id>
        <syntax>frequency</syntax>
                <sample>30</sample>
 </command>
```

The Monitoring frequency will set from a **monitoring_frequency** parameter defined in cms.properties file

3.4.2 Monitoring Request Format

Following is the monitoring request format of the xml that would be sent to a wrapper by CMS.

```
<command>
        <seq>1</seq>
        <name>doMon</name>
        <id>101</id>
        <systemid>subsystem1</systemid>
        <version>1.0</version>
        <timestamp>2011-01-11T19:27:23.110+05:30</timestamp>
        <priority>0</priority>
        <syntax>ang2</syntax>
        <data>
                <param>
                        <name>frequency</name>
                        <value>30</value>
                </param>
        </data>
</command>
```

Following is the descriptive information about doMon request format that CMS sends to wrapper on cms initialization:

seq – This will contain a unique number indicating the count of command fired. CMS will populate this value and send it as part of Command request, CMS expects wrapper to send back same value as part of each and every ack, intermediate or final response for corresponding command. This field is optional for events as they are generated asynchronously.

id – This represents the command id for CMS system. CMS will send this as part of Command request xml and expects wrapper to send it back for each and every monitoring response. This can contain alphanumeric characters, with maximum length of 20 characters.

name – This is display name of the actual command fired by the user. CMS will send this as part of doMon request xml and expects wrapper to send it back for each and every ack, every monitoring response.

systemid – This is used to distinguish between various subsystems attached to CMS via wrapper. CMS will populate this as part of Command request, CMS expects wrapper to send same value back as part of each and every monitoring response.

version – subsystem version

timestamp – The timestamp at which the request was created. Optional for events

syntax – It describes the command parameter sequence to be used by wrapper to sort the parameters

priority – This provides the priority of request. High priority requests will be consumed first.

data – The data will contain the monitoring frequency. This would be same for all subsystems.

3.4.3 Monitoring Request Component Size

The monitoring request component size is same as the Request format component size. Please refer to section 3.1.2 Request Component size

3.4.4 Monitoring Response Format

Following is the monitoring response format of the xml that would be sent by wrapper to CMS.

```
<responses>
        <response>
                <id>30</id>
                <name>doMon</name>
                 <seq>1</seq>
                <systemid>servo</systemid>
                <code>10</code>
                 <event>15</event>
                <data>
                        <param>
                                <name>status</name>
```

```
                    <value>OK</value>
                </param>
                <param>
                    <name>wind_vel1</name>
                    <value>10</value>
                </param>
            </data>
            <alarms>
                <alarm>
                    <id>1</id>
                    <name>alarm1</name>
                    <message></message>
                    <level>3</level>
                </alarm>
            </alarms>
            <message></message>
        </response>
    </responses>
```

Following is the descriptive information about doMon response format that wrapper sends to to cms at the monitoring frequency:

seq – This will contain a unique number indicating the count of command fired. CMS will populate this value and send it as part of Command request, CMS expects wrapper to send back same value as part of each and every ack, intermediate or final response for corresponding command. This field is optional for events as they are generated asynchronously.

id – This represents the command id for CMS system. CMS will send this as part of Command request xml and expects wrapper to send it back for each and every monitoring response. This can contain alphanumeric characters, with maximum length of 20 characters.

name – This is display name of the actual command fired by the user. CMS will send this as part of doMon request xml and expects wrapper to send it back for each and every ack, every monitoring response.

systemid – This is used to distinguish between various subsystems attached to CMS via wrapper. CMS will populate this as part of Command request, CMS expects wrapper to send same value back as part of each and every monitoring response.

version – subsystem version

timestamp – The timestamp at which the request was created. Optional for events

data – The data will contain the monitoring frequency. This would be same for all subsystems.

code – This field is populated by wrapper while sending response back to CMS for a given command. For monitoring response the code can set to 10.

event – This field is populated by wrapper while sending response back to CMS for a given command or any asynchronous event. This field value can be set to any value greater than 12 indication that this an intermediate response

message – This field can be used to communicate some information to the user.

data – This field can contain any number of monitoring parameters.

alarm – In some cases like hardware failure, subsystem will generate alarm and send it to CMS via wrapper. In such cases wrapper can make use of ALARM tag, which contains following fields-

> **id** – alarm identifier
> **name** – alarm name
> **message** – Contains the brief description about the ALARM, which will be shown to the user in ALARM window.
> **level** – Wrapper can assign the priority for such ALARM and share the information with CMS using this field.
> Level can be 1 - Information, 2,3 - Warning, 4,5 – Critical

3.4.5 Monitoring Request Component Size

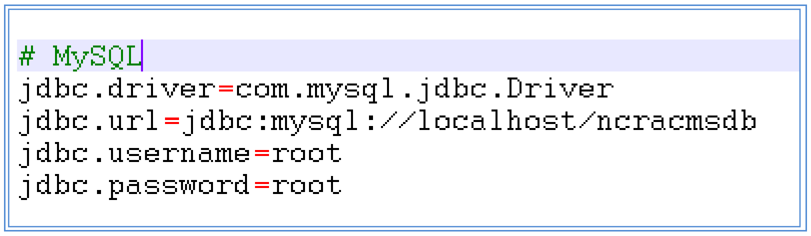
The monitoring response component size is same as the Response format component size. Please refer to section 3.1.4 Response Component size.

4 CMS Configuration

4.1 Configuration in cms.properties

Open cms.properties using vi editor and modify the following as per your environment settings

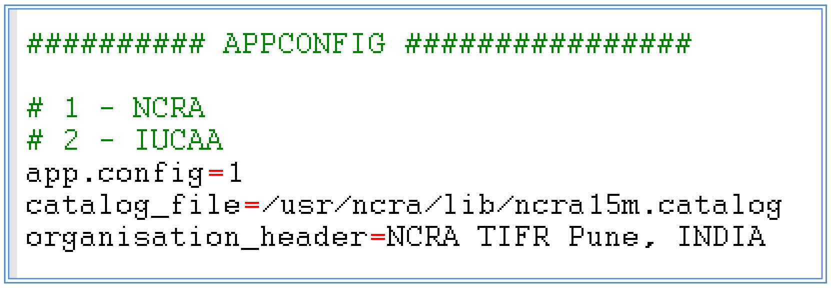
4.1.1 Database configurations



```
# MySQL
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost/ncracmsdb
jdbc.username=root
jdbc.password=root
```

1. If ncracmsdb is installed on different machine other than the tomcat installed machine then change jdbc.url to point to the new machine. Replace localhost with the machine ip address.
2. Change the jdbc.username and jdbc.password as per the username and password specified in mysql

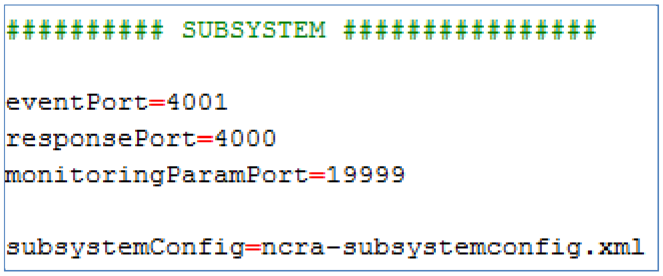
4.1.2 Catalog file settings



```
########## APPCONFIG ################

# 1 - NCRA
# 2 - IUCAA
app.config=1
catalog_file=/usr/ncra/lib/ncra15m.catalog
organisation_header=NCRA TIFR Pune, INDIA
```

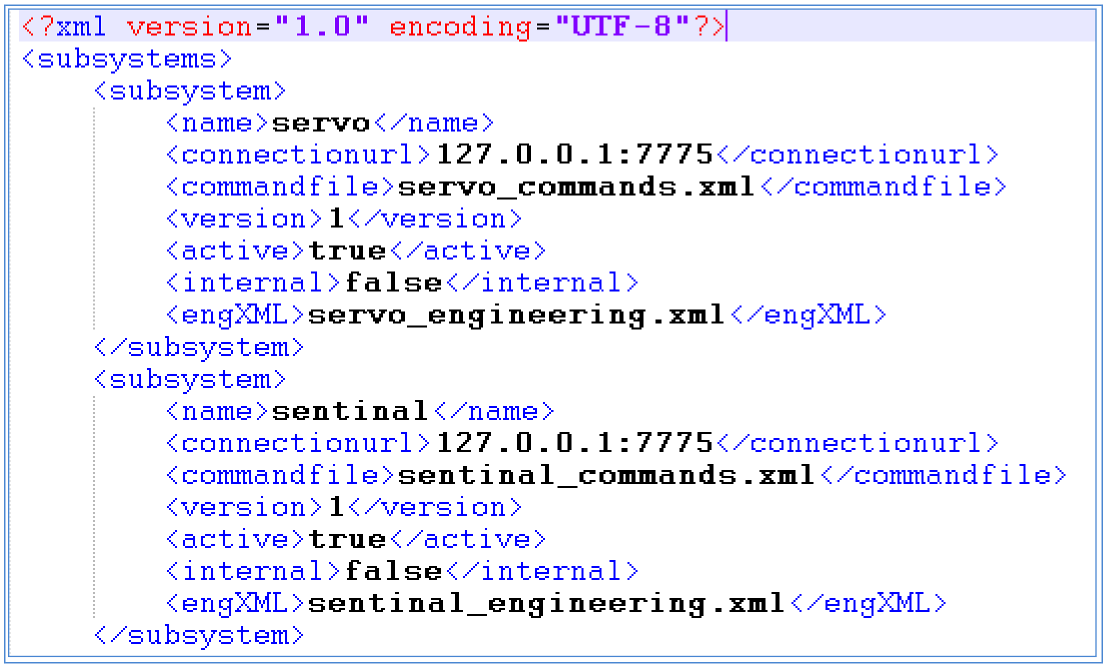
1. **app.config** specifies the app config. 1 is for NCRA and 2 for IUCAA.
2. **catalog_file** specifies the absolute path of default catalog file. Currently this is not being used in CMS, this information is for reference only.
3. **organization_header** specifies the title that will appear in header of CMS home page

4.1.3 Subsystem configurations



```
########## SUBSYSTEM ################

eventPort=4001
responsePort=4000
monitoringParamPort=19999

subsystemConfig=ncra-subsystemconfig.xml
```

1. **eventPort** specifies the port on which events will be received by CMS.
2. **responsePort** specifies the port on which response will be received by CMS. Incase both ports have the same port address; single port will receive the events and responses.
3. **monitoringParamPort** System Port number to receive monitoring information
4. Subsystem configuration has been shifted to a separate configuration file named **ncra-subsystemconfig.xml**. Following are details about this file



```
<?xml version="1.0" encoding="UTF-8"?>
<subsystems>
    <subsystem>
        <name>servo</name>
        <connectionurl>127.0.0.1:7775</connectionurl>
        <commandfile>servo_commands.xml</commandfile>
        <version>1</version>
        <active>true</active>
        <internal>false</internal>
        <engXML>servo_engineering.xml</engXML>
    </subsystem>
    <subsystem>
        <name>sentinal</name>
        <connectionurl>127.0.0.1:7775</connectionurl>
        <commandfile>sentinal_commands.xml</commandfile>
        <version>1</version>
        <active>true</active>
        <internal>false</internal>
        <engXML>sentinal_engineering.xml</engXML>
    </subsystem>
```

a. **name** – Specifies name of subsystem
b. **connectionurl** - Specifies the ip and port of servo wrapper.(sub-system specific wrapper)
c. **commandfile** - Specifies the command file name for subsystem. The command file should be placed in **$librarypath**.
d. **Version** – System version can be specified here
e. **active** – specifies whether subsystem is active. When set to false subsystem is considered as deactive.

f. **internal** – This is used by cms for internal use. For external subsystem this property should be always set to false.
g. **engXML**
h. Similarly frontend,backend,sigcon, cms can be configured.

4.1.4 Default Command Timeout

```
#default command timeout in millisec
defaultCommandTimeout=60000
```

1. **defaultCommandTimeout** specifies the default command time out in seconds. In case for a command, if any timeout is not specified, then the default command timeout value will be used as command timeout.

4.1.5 Response Timeout

```
#default response timeout in mili seconds
responseTimeout=60000
```

1. **responseTimeout** specifies the default timeout for response in milliseconds.

4.1.6 Sequence no Generator Settings

```
#sequence file
seq.file=/usr/ncra/seq.dat
```

1. It will create the seq.dat file at specified location, this file maintains current unique sequence no, which is used to track and differentiate commands from each other.

4.1.7 Menu settings

```
########### MENU ###################
refresh.delay=1000
file.name=cmsinfo.properties
```

1. **refresh.delay** specifies the delay for refreshing the menu in milliseconds
2. **file.name** specifies the menu file this contains the menu name and its url. This is visible under Information Links->Help Menu page

4.1.8 Longitude, latitude and height setting

```
#######Longitude, Latitude and Height of the Antenna#####
longitude=73.49
latitude=19:05:26.35
height=560
```

1. **longitude** (in Degrees) specifies the longitude of telescope location. It is used for various astronomical calculations.
2. **latitude** (in D:M:S Format – 19:05:26.35) Used for various astronomical calculations. **height** (in float) specifies the height of antenna. It is used for various astronomical calculations.

4.1.9 Tzone and el_lim settings

```
#####timezone#####
tzone=-5:30

el_lim=17.0
```

1. **tzone** (Supported formats are +hh:mm,-hh:mm or decimal format e.g. 5.5)Used for various astronomical calculations.
2. **el_lim** (supported formats is float) Used for various astronomical calculations

4.1.10 Command log filename

```
#######Default file name for command Log#######
commandlogfilename=file.xls
```

1. **Commandlogfilename** specifies the default filename of the command logs excel file which can be downloaded from command Log.

4.1.11 Broker URL

```
########### ACTIVEMQ ###################
brokerUrl=tcp://localhost:61616
```

1. This is the ActiveMQ URL. It must be set the port on which ActiveMQ is running.

4.1.12 Manual mode Subsystem and settings

```
#########Manual mode###########
manualmodeSubsystem=servo
manualmode.file=manualmode.properties
```

1. **manualmodeSubsystem** specifies the name of the sub system on which manual mode commands will be executed.
2. **manualmode.file** specifies the name of the file in which manual mode commands will be present.

4.1.13 Data Acquisition Subsystem and settings

```
dataAcqSubsystem=backend
dataAcq.file=dataAcq.properties
```

1. **dataAcqSubsystem** specifies the name of the sub system on which data acquisition commands will be executed.
2. **dataAcq.file** specifies the name file where data acquisition commands will be present.

4.1.14 Help Menu settings

```
manualmodehelpmenu=http://www.ncra.tifr.res.in/
cataloghelpmenu=http://www.ncra.tifr.res.in/
```

1. Currently not used .will be removed later.

4.1.15 Time zone

```
########Global Parameter ##########
time_zone=IST[GMT+5.30]
```

1. Currently not in use.

4.1.16 Server Host settings

```
serverHost=PS0672.persistent.co.in
```

1. **Server Host** for sending emails - Please change the name of this property to email server host in your organization.

4.1.17 Off Source Time Out

```
#timeout period (in milliseconds) for antenna goes off source after being on source
off_source_timeout = 180000
```

1. **off_source_timeout** specifies the maximum time out period after which alarm will be raised if antenna goes off the source after being on source.

4.1.18 AZ-EL allowable difference

```
####az-el allowable difference###
azDiffLimit=20
elDiffLimit=20
```

1. **azDiffLimit** specifies maximum allowable difference e between antenna AZ   position and target AZ position.
2. **elDiffLimit** specifies maximum allowable difference e between antenna EL position and target EL position

4.1.19 Monitoring Time Out

```
 ### monitoring timeout it is milli second###
monitoringTimeout  =  180000
```

1. **monitoringTimeout** specifies the time out period for monitoring parameters.

4.1.20 Monitoring frequency

```
monitoringfrequency=9000
```

1. **monitoringfrequency** specifies the time interval after which wrapper should send monitoring data.

4.1.21 Escape Characters

```
################ Escape Character setting #################
escapeCharacters=true
```

1. Used to handle the embedded commands .Currently not used for NCRA.

4.1.22 Connectivity Delay

```
connectivityDelay = 3000
```

1. **connectivityDelay** specifies the time interval between subsequent ping requests sent to the wrapper. CMS will ping each wrapper, if not getting connected it will wait for connectivityDelay time and then again ping the wrapper. The ping continues till the wrapper is connected or connectivityTimeout is reached.

4.1.23 Connectivity time out

```
connectivityTimeOut = 30000
```

1. **connectivityTimeOut** for checking all wrapper connections i.e. after this time interval cms will check the wrapper connections and if not connected will declare the particular wrapper as not connected.

4.1.24 Time Interval of Alarm

```
### Alarm Time interval is in milliseconds ####
timeIntervalOfAlarm = 300000
```

1. **timeintervalOfAlarm** specifies time interval between saving two same alarms rose one after another and its value is in millisecond.

4.1.25 Rules File

```
#######Rules config file############
rulesFile=CMSRules.drl
```

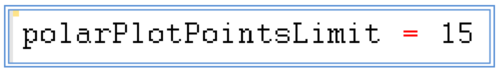
1. **rulesFile** specifies the rules defined for state machine configuration.

4.1.26 Polar Plot Refresh Interval

```
polarPlotRefreshInterval  = 900000
```

1. **PolarPlotRefreshInterval** specifies the interval value in mili-second which will be used by scheduler to periodically update the polar plot value. Similarly interval value is also specified for 2Dplot.

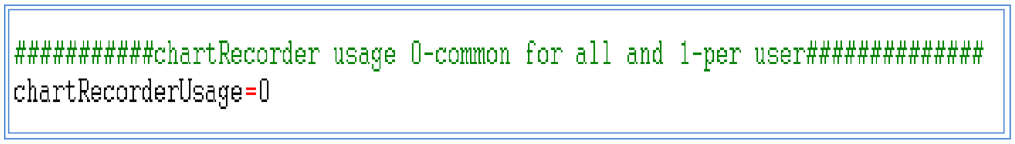
4.1.27 Polar Plot Points Limit



```
polarPlotPointsLimit = 15
```

1. **polarPlotPointsLimit** indicates how many points user wants to plot on polar plot. This in turn shows path the object has traversed.

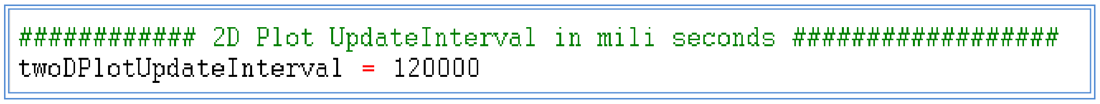
4.1.28 Chart Recorder Usage



```
##########chartRecorder usage 0-common for all and 1-per user#############
chartRecorderUsage=0
```

1. **ChartRecorderUsage** is defined for usage of Chart Recorder. If it ‘0’ the chart recorder would be common for all and if ‘1’ it would be user specific i.e per user separate chart recorder instance will run.

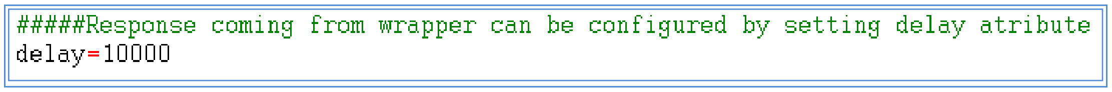
4.1.29 Two D Plot Update Interval



```
############ 2D Plot UpdateInterval in mili seconds ##################
twoDPlotUpdateInterval = 120000
```

1. **twoDPlotUpdateInterval** specifies the interval value in mili-second which will be used by scheduler to periodically update the twoDplot.

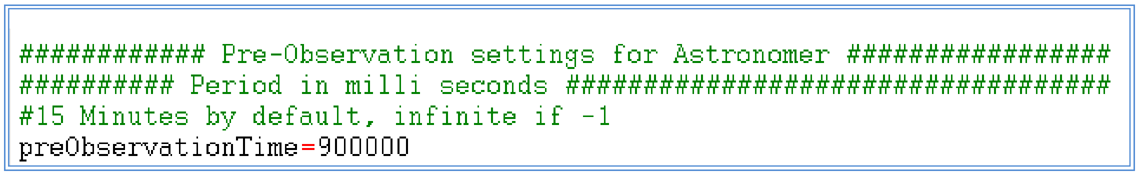
4.1.30 Delay



```
#####Response coming from wrapper can be configured by setting delay atribute
delay=10000
```

1. **Delay** specifies the time interval between two ping requests to the wrapper.

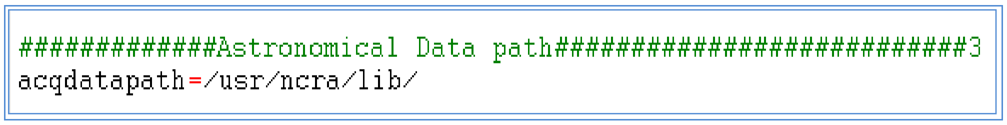
4.1.31 Pre-Observation Time



```
############ Pre-Observation settings for Astronomer #################
########## Period in milli seconds ###################################
#15 Minutes by default, infinite if -1
preObservationTime=900000
```

1. **preObservation** time before observation start time that allows astronomer to upload catalogs and validate his batch file in CMS. If “**-1**” then astronomer can login at any time to perform observation activities.

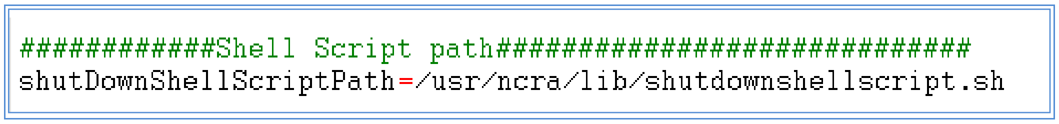
4.1.32 Acq data path



```
############Astronomical Data path###########################3
acqdatapath=/usr/ncra/lib/
```

1. **acqdatapath** specifies the directory in which astronomical data will be saved

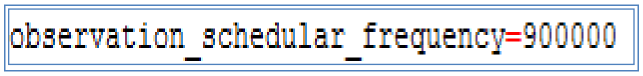
4.1.33 Shut Down Shell Script Path



```
############Shell Script path##############################
shutDownShellScriptPath=/usr/ncra/lib/shutdownshellscript.sh
```

1. **ShutDownShellScriptPath** specifies the location for complete CMS shutdown s script.

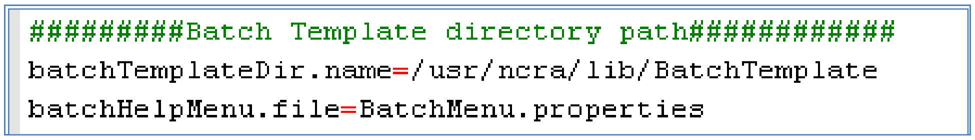
4.1.34 Observation Scheduler frequency



```
observation_schedular_frequency=900000
```

1. observation_schedular_frequency specifies the time interval after which observation scheduler will be invoked in milliseconds. Observation Scheduler keeps track of active schedule for astronomer/co-astronomer. Time interval must be greater than 15 minutes i.e. 900000 ms .If it is less, then default value of 900000 ms will be assigned.

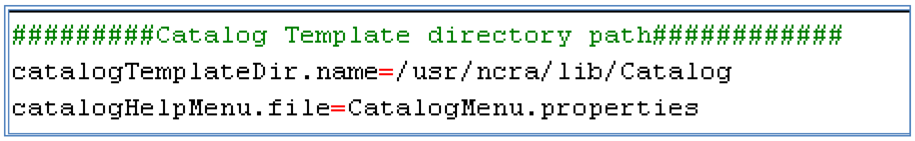
4.1.35 Batch Template Settings



```
#########Batch Template directory path############
batchTemplateDir.name=/usr/ncra/lib/BatchTemplate
batchHelpMenu.file=BatchMenu.properties
```

1. **batchTemplateDir.name** specifies the directory path in which batch templates will be stored.
2. **batchHelpMenu.file** specifies the menu configuration for batch templates to be displayed to user.

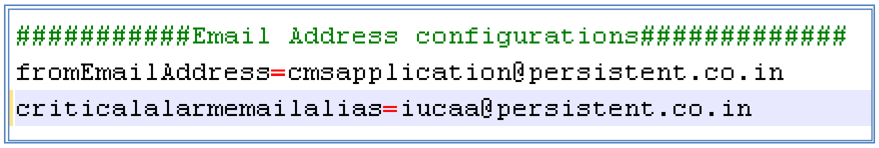
4.1.36 Catalog Template Settings



```
#########Catalog Template directory path############
catalogTemplateDir.name=/usr/ncra/lib/Catalog
catalogHelpMenu.file=CatalogMenu.properties
```

1. **catalogTemplateDir.name** specifies the directory path in which catalog templates will be stored.
2. **catalogHelpMenu.file** specifies the menu configuration for catalog templates to be displayed to user.

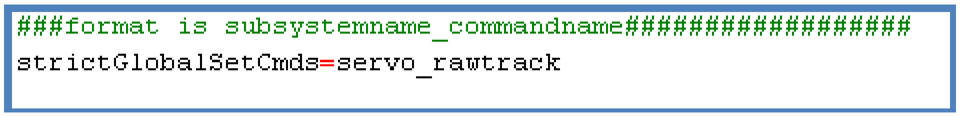
4.1.37 Email Address configurations



```
###########Email Address configurations############
fromEmailAddress=cmsapplication@persistent.co.in
criticalalarmemailalias=iucaa@persistent.co.in
```

1. **fromEmailAddress** specifies the email address which will appear in “**From:**” field in emails sent out from CMS.
2. **criticalalarmemailalias** specifies the email address to which mails will be sent out if a critical alarm is raised in or received by CMS.

4.1.38 Commands supporting strict setting of global parameters:

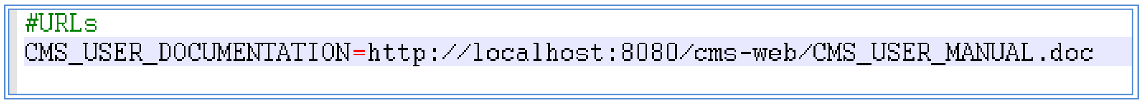


```
###format is subsystemname_commandname##################
strictGlobalSetCmds=servo_rawtrack
```

These commands will set the global parameter values at set per user strictly; no internal calculation will alter the values as set by user. The command is specified in the format: <subsystem_name>_<command_name>
To add more commands separate them using comma sepearator.

4.2 Configuration in cmsinfo.properties

Open cmsinfo.properties and modify the line with key “CMS_USER_DOCUMENTATION”. The entry against this key should be http://<serverIPAddress>/cms-web/CMS_USER_MANUAL.docx



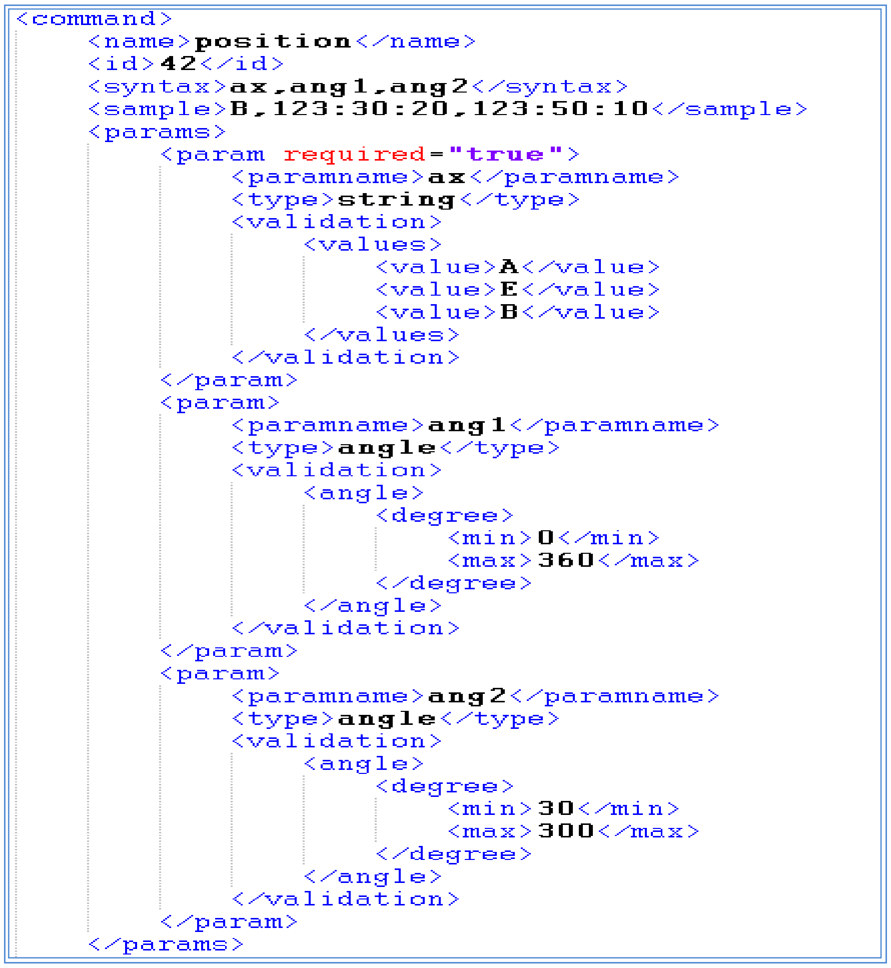
```
#URLs
CMS_USER_DOCUMENTATION=http://localhost:8080/cms-web/CMS_USER_MANUAL.doc
```

4.3 Command Configurations

To configure a command of a subsystem the corresponding subsystem’s xml file needs to be modified. The xml can be modified using any standard xml editor or any text editor. Note that the text editor will not point to any error in the xml syntax. One can open this file in browser like IE, which detects few errors in xml syntax like whether document is well-formed or not.

4.3.1 Configuring a command for servo

1. Find the command in the servo_commands.xml file. For e.g. position



```
<command>
    <name>position</name>
    <id>42</id>
    <syntax>ax,ang1,ang2</syntax>
    <sample>B,123:30:20,123:50:10</sample>
    <params>
        <param required="true">
            <paramname>ax</paramname>
            <type>string</type>
            <validation>
                <values>
                    <value>A</value>
                    <value>E</value>
                    <value>B</value>
                </values>
            </validation>
        </param>
        <param>
            <paramname>ang1</paramname>
            <type>angle</type>
            <validation>
                <angle>
                    <degree>
                        <min>0</min>
                        <max>360</max>
                    </degree>
                </angle>
            </validation>
        </param>
        <param>
            <paramname>ang2</paramname>
            <type>angle</type>
            <validation>
                <angle>
                    <degree>
                        <min>30</min>
                        <max>300</max>
                    </degree>
                </angle>
            </validation>
        </param>
    </params>
```

2.

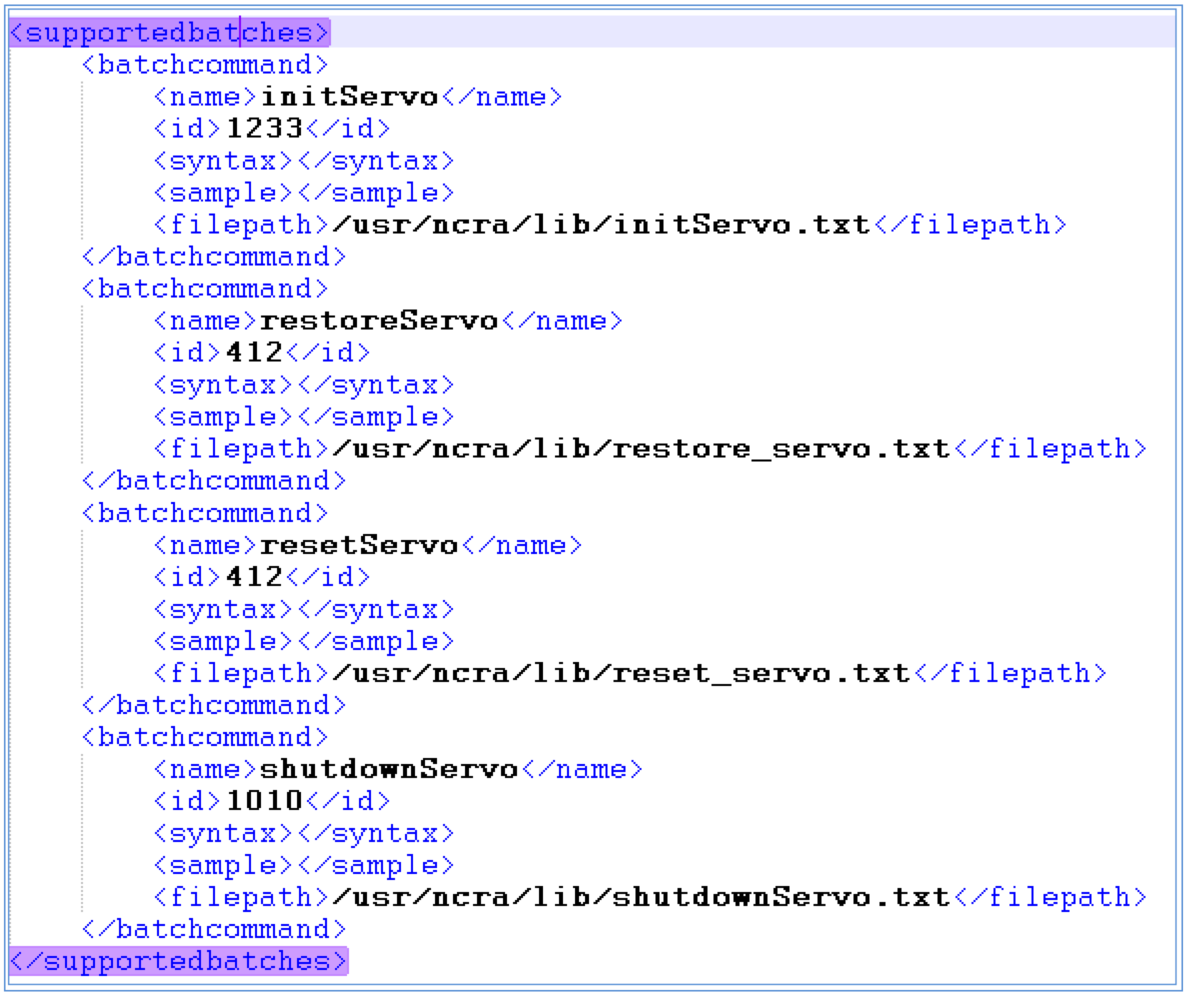
- <name> - specifies the command name
- <id> - specifies the command id.
- <syntax> - specifies the command syntax. This appears in the Expert Tab ->Syntax field. More than one parameter should be comma separated.
- <sample> -specifies the command sample. This appears in the Expert Tab-> Command Field. More than one parameter should be comma separated.
- <timeout> - This is the optional tag. It specifies timeout period for a command. If not specified defaultCommandTimeout value is considered as time out period for command.
- <params> - this tag specifies the parameter validation.
- <param> - this tag specifies validation for a single parameter
- required-“true” – specifies that this parameter is mandatory
- <paramname> - this specifies the parameter name to which validation is to be applied. This name should match to the one mentioned in syntax.
- <type> - specifies the type of parameter. Currently the following types are supported:
  integer – for integer values
  string – for a value that lies within provided string values
  long – for long values
  float – for float values
  regex – for values that can be evaluated with a regular expression
  double – for double values
- <dependency_validations> -If a particular parameters value is dependent on other parameters value <dependency_validation> tag is used. This tag is optional.

Example: When parameter “ax” of position command has value “A” then another parameter “ang1” must lie in the range 0-360.

This dependency validation must be defined as follows in <dependency_validations> tag.

```
<dependency_validations>
    <params>
        <param required="true">
            <paramname>ax</paramname>
            <type>string</type>
            <validation>
                <values>
                    <value>A</value>
                </values>
            </validation>
        </param>
        <param required="true">
            <paramname>ang1</paramname>
            <type>angle</type>
            <validation>
                <angle>
                    <degree>
                        <min>0</min>
                        <max>360</max>
                    </degree>
                </angle>
            </validation>
        </param>
    </params>
```

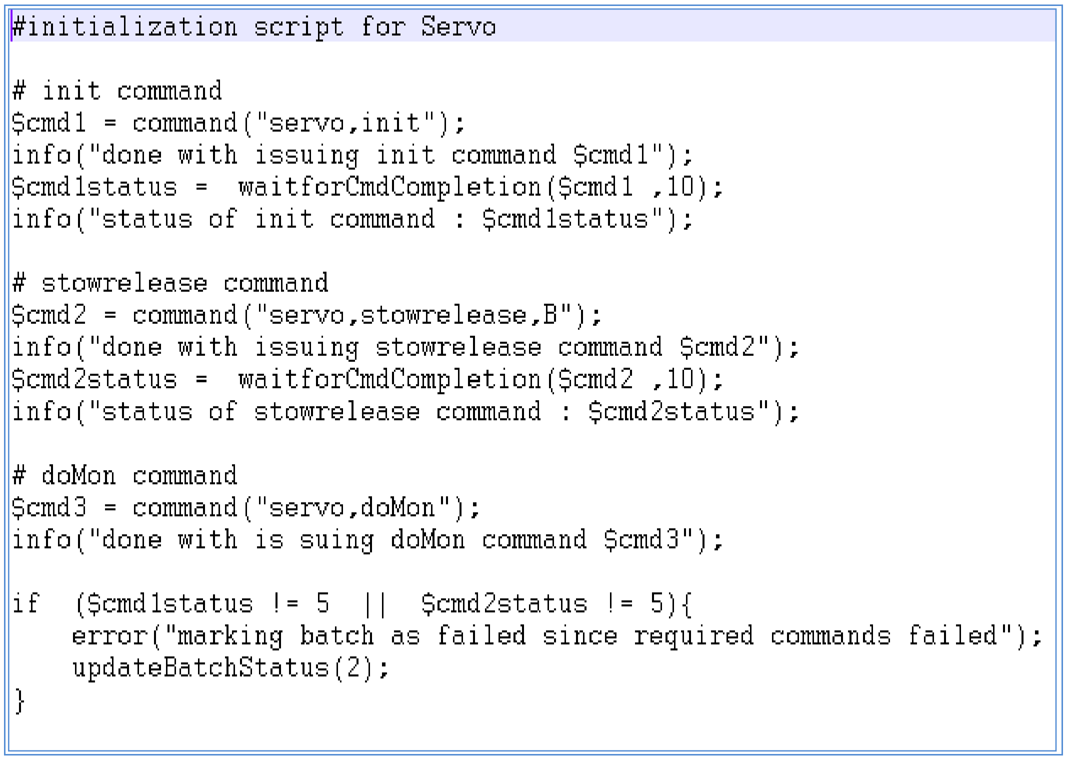
In order to make system more flexible and loosely coupled, few features have been implemented using batch scripts, .for e.g. Init and shutdown routines etc. This gives flexibility to change the logic in these scripts in future and system would be agnostic to these changes.



```
<supportedbatches>
    <batchcommand>
        <name>initServo</name>
        <id>1233</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>/usr/ncra/lib/initServo.txt</filepath>
    </batchcommand>
    <batchcommand>
        <name>restoreServo</name>
        <id>412</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>/usr/ncra/lib/restore_servo.txt</filepath>
    </batchcommand>
    <batchcommand>
        <name>resetServo</name>
        <id>412</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>/usr/ncra/lib/reset_servo.txt</filepath>
    </batchcommand>
    <batchcommand>
        <name>shutdownServo</name>
        <id>1010</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>/usr/ncra/lib/shutdownServo.txt</filepath>
    </batchcommand>
</supportedbatches>
```

- <name> - specifies the batch name
- <id> - specifies the batch id.
- <syntax> - specifies the parameter list to be sent as input parameters to batch file. This appears in the Expert Tab ->Syntax field. More than one parameter should be comma separated.
- <sample> -specifies the values for parameters mentioned above. This appears in the Expert Tab-> Command Field. More than one parameter should be comma separated.
- <filepath> - Specifies location of batch script file, please ensure this path is set correctly in your environment

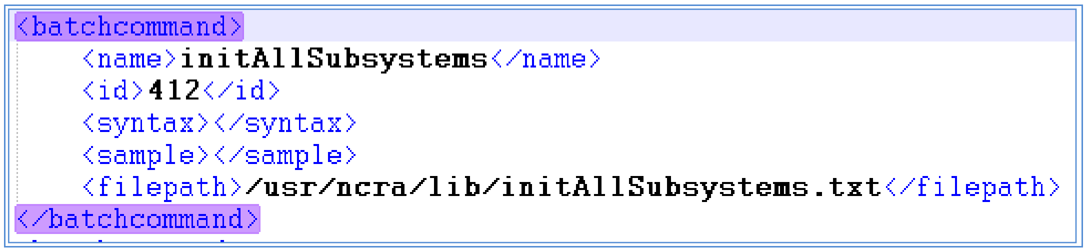
3. Please see below an excerpt from batch script: initServo.txt



```
#initialization script for Servo

# init command
$cmd1 = command("servo,init");
info("done with issuing init command $cmd1");
$cmd1status =  waitforCmdCompletion($cmd1 ,10);
info("status of init command : $cmd1status");

# stowrelease command
$cmd2 = command("servo,stowrelease,B");
info("done with issuing stowrelease command $cmd2");
$cmd2status =  waitforCmdCompletion($cmd2 ,10);
info("status of stowrelease command : $cmd2status");

# doMon command
$cmd3 = command("servo,doMon");
info("done with is suing doMon command $cmd3");

if  ($cmd1status != 5  ||  $cmd2status != 5){
    error("marking batch as failed since required commands failed");
    updateBatchStatus(2);
}
```

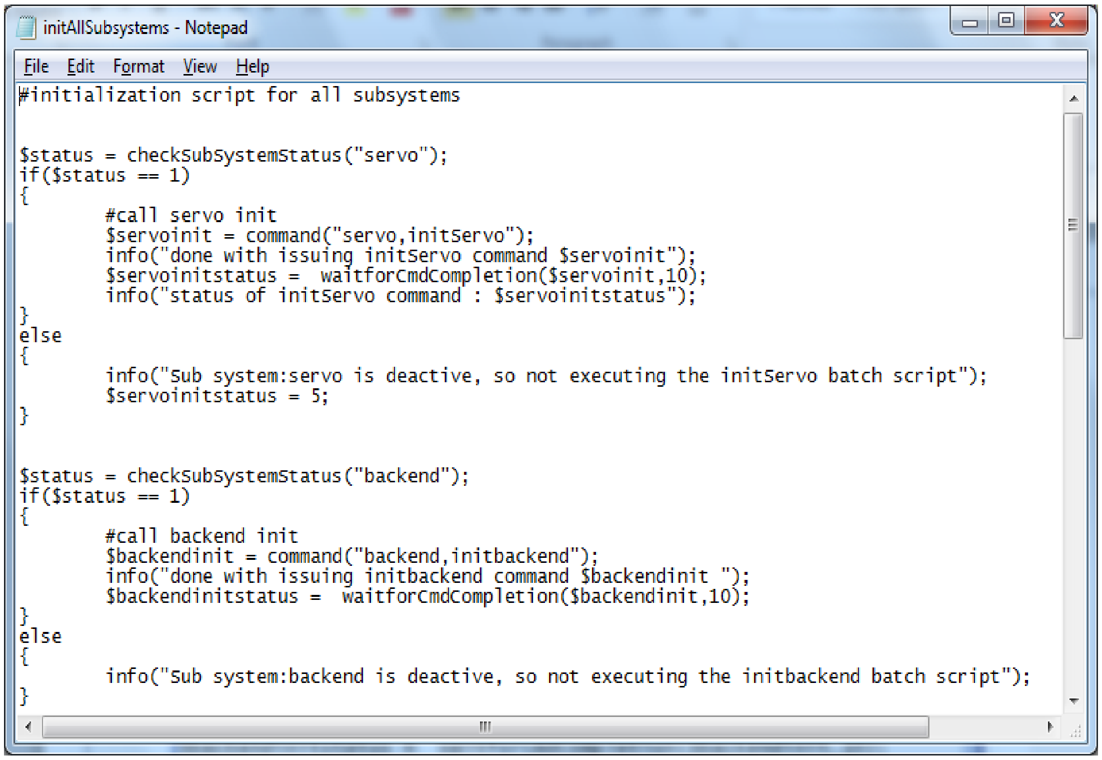
4.4 Initialization Configuration for CMS

1. To initialize all subsystems in CMS, configure the batch command “initAllSubsystems” in cms_command.xml as displayed below. The initAllSubsystems is called when CMS is started for the first time and when CMS was shut down properly.



```
<batchcommand>
        <name>initAllSubsystems</name>
        <id>412</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>/usr/ncra/lib/initAllSubsystems.txt</filepath>
</batchcommand>
```

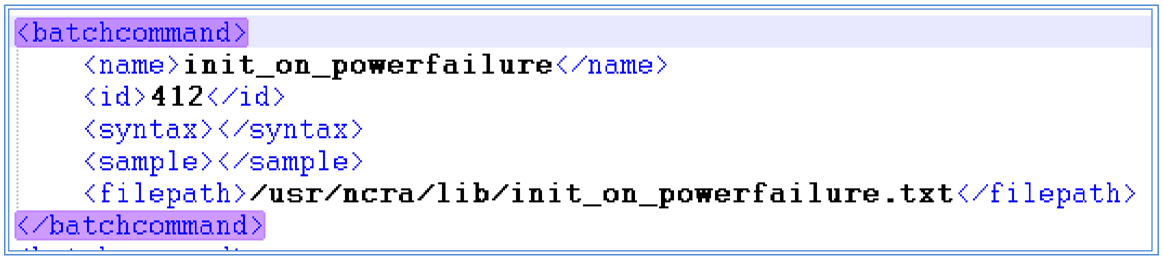
2. The **initAllSubsystems** is a batch script containing initialization commands to be sent to each subsystem. The screenshot below displays the sample initialization script.



```
#initialization script for all subsystems


$status = checkSubSystemStatus("servo");
if($status == 1)
{
        #call servo init
        $servoinit = command("servo,initServo");
        info("done with issuing initServo command $servoinit");
        $servoinitstatus =  waitforCmdCompletion($servoinit,10);
        info("status of initServo command : $servoinitstatus");
}
else
{
        info("Sub system:servo is deactive, so not executing the initServo batch script");
        $servoinitstatus = 5;
}


$status = checkSubSystemStatus("backend");
if($status == 1)
{
        #call backend init
        $backendinit = command("backend,initbackend");
        info("done with issuing initbackend command $backendinit ");
        $backendinitstatus =  waitforCmdCompletion($backendinit,10);
}
else
{
        info("Sub system:backend is deactive, so not executing the initbackend batch script");
}
```

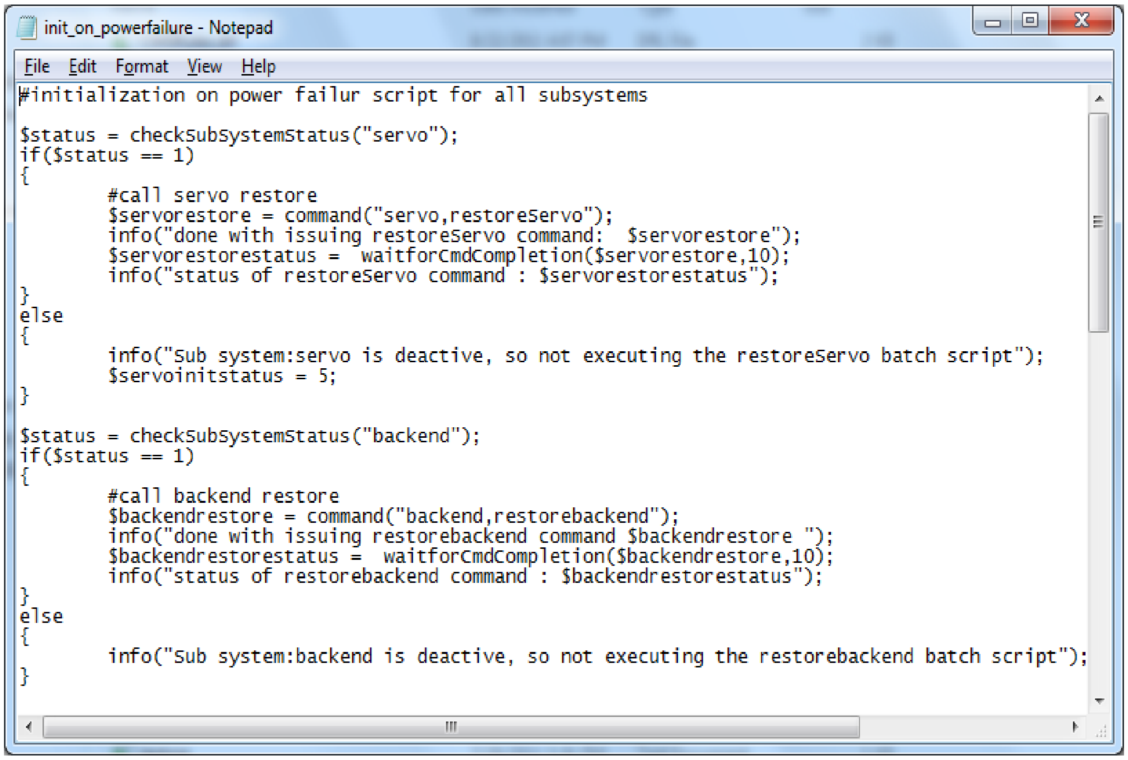
By default each batch script will be marked as successful. To mark a batch as failure use the command **updateBatchStatus (2);** the failure can be marked based on some command failure refer to initAllSubsystems.txt for the sample batch failure scenario implementation.

3. Similarly also Configure init on power failure in cms_commands.xml. The **init_on_powerfailure** script is called during CMS initialization when CMS was not properly shutdown the last time. This script contains the restoration commands in order to restore the subsystem to previous known state.



```
<batchcommand>
    <name>init_on_powerfailure</name>
    <id>412</id>
    <syntax></syntax>
    <sample></sample>
    <filepath>/usr/ncra/lib/init_on_powerfailure.txt</filepath>
</batchcommand>
```

Sample **init_on_powerfailue.txt** batch script:



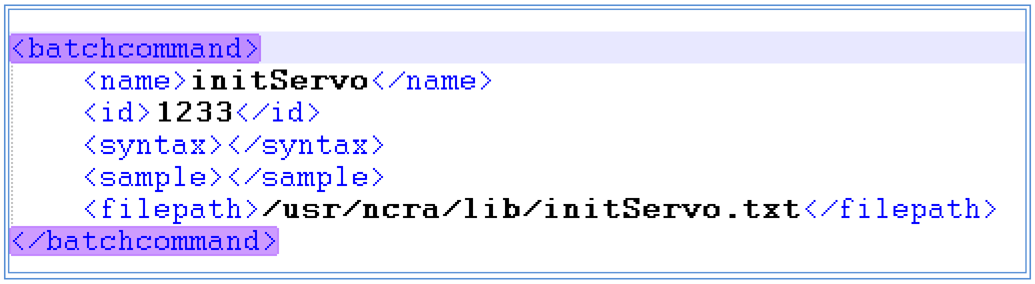
```
#initialization on power failur script for all subsystems

$status = checkSubSystemStatus("servo");
if($status == 1)
{
        #call servo restore
        $servorestore = command("servo,restoreServo");
        info("done with issuing restoreServo command:  $servorestore");
        $servorestorestatus =  waitforCmdCompletion($servorestore,10);
        info("status of restoreServo command : $servorestorestatus");
}
else
{
        info("Sub system:servo is deactive, so not executing the restoreServo batch script");
        $servoinitstatus = 5;
}

$status = checkSubSystemStatus("backend");
if($status == 1)
{
        #call backend restore
        $backendrestore = command("backend,restorebackend");
        info("done with issuing restorebackend command $backendrestore ");
        $backendrestorestatus =  waitforCmdCompletion($backendrestore,10);
        info("status of restorebackend command : $backendrestorestatus");
}
else
{
        info("Sub system:backend is deactive, so not executing the restorebackend batch script");
}
```

For the restore command to succeed, the parameter which is being restored should be present in **t_recent_monitoring_data** table. And the parameter if being used in a command the parameter should be within validation range configured in the command.

4. Similarly for each subsystem initialization script exists which are called from the initAllSubsystems batch script.

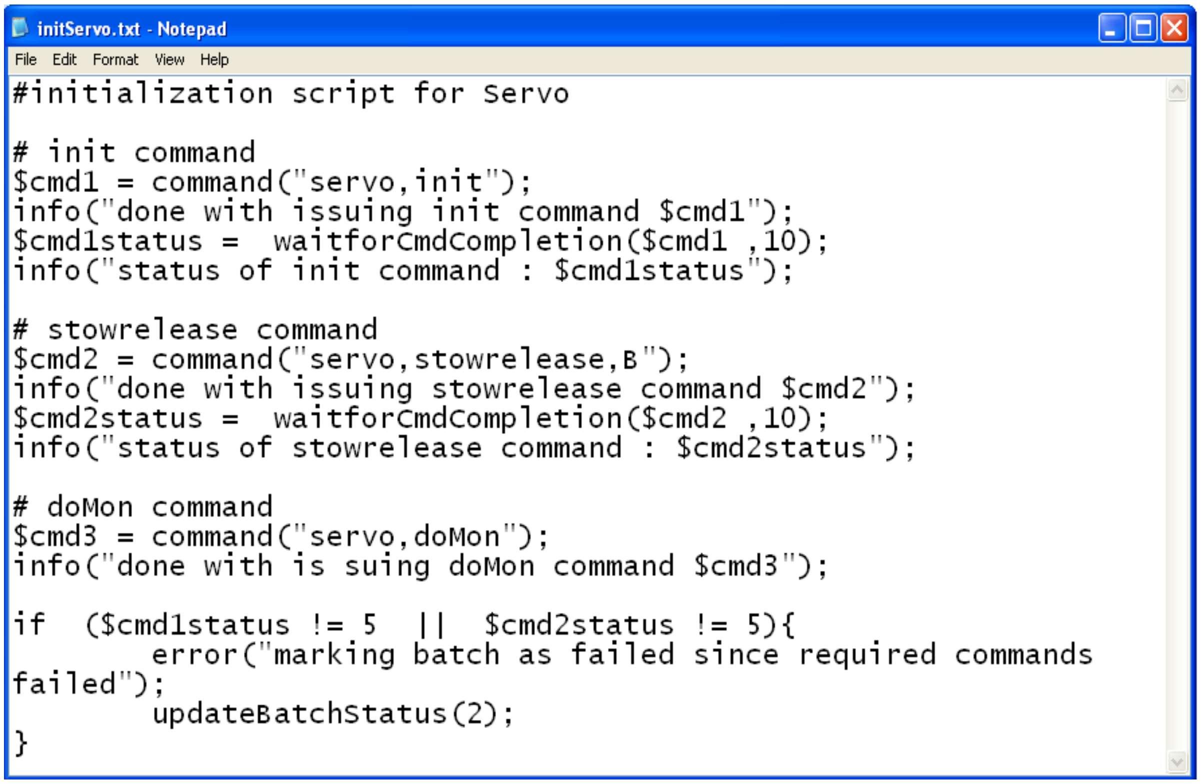
   Consider configuring initialization of servo subsystem.

   In servo_commands.xml configure" **initServo**" command.



```
<batchcommand>
    <name>initServo</name>
    <id>1233</id>
    <syntax></syntax>
    <sample></sample>
    <filepath>/usr/ncra/lib/initServo.txt</filepath>
</batchcommand>
```

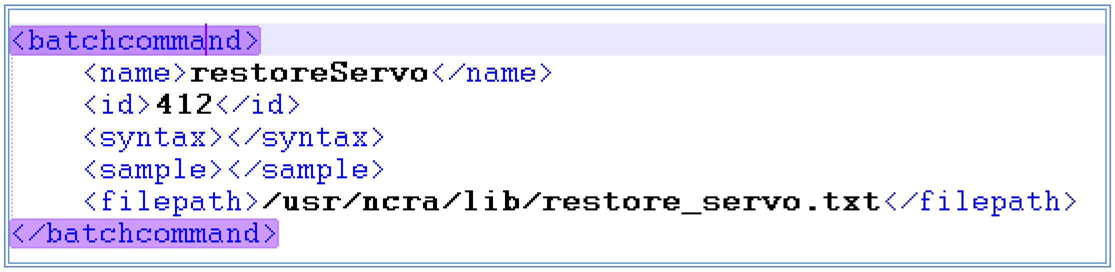
   Sample **"initServo.txt"**:



```
#initialization script for Servo

# init command
$cmd1 = command("servo,init");
info("done with issuing init command $cmd1");
$cmd1status =  waitforCmdCompletion($cmd1 ,10);
info("status of init command : $cmd1status");

# stowrelease command
$cmd2 = command("servo,stowrelease,B");
info("done with issuing stowrelease command $cmd2");
$cmd2status =  waitforCmdCompletion($cmd2 ,10);
info("status of stowrelease command : $cmd2status");

# doMon command
$cmd3 = command("servo,doMon");
info("done with is suing doMon command $cmd3");

if  ($cmd1status != 5  ||  $cmd2status != 5){
        error("marking batch as failed since required commands
failed");
        updateBatchStatus(2);
}
```

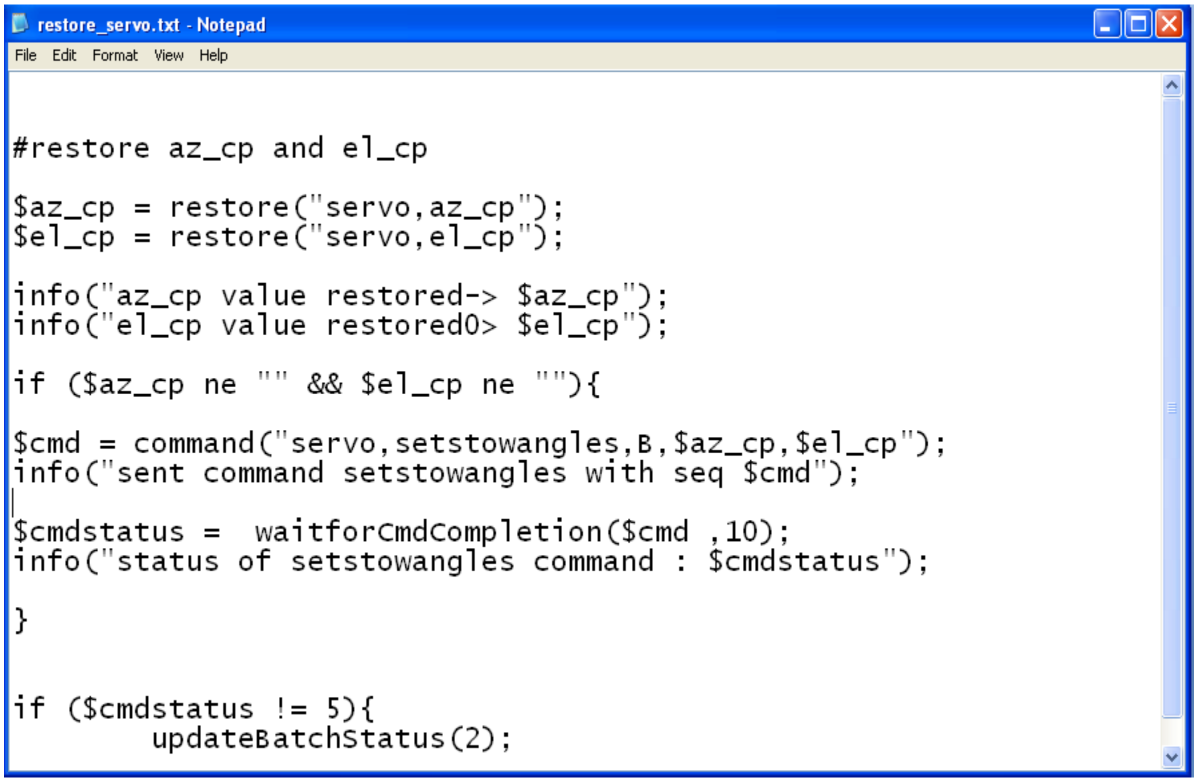
5. Similarly also configure restoration script for each subsystem. These scripts are called from the init_on_powerfailure batch script.

   Consider restoration of servo subsystem.
   In servo_commands.xml configure "**restore_servo**" command.



```
<batchcommand>
     <name>restoreServo</name>
     <id>412</id>
     <syntax></syntax>
     <sample></sample>
     <filepath>/usr/ncra/lib/restore_servo.txt</filepath>
</batchcommand>
```

Sample **"restore_servo.txt"** batch script:



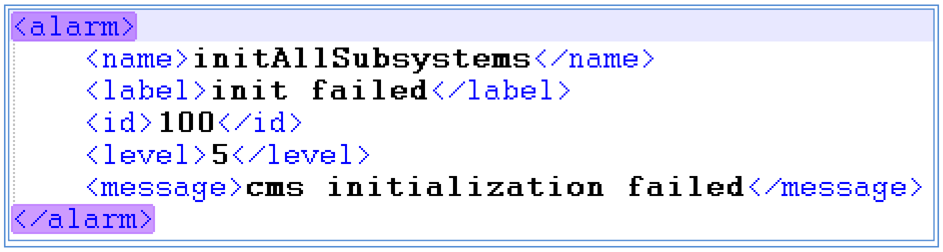
```
#restore az_cp and el_cp

$az_cp = restore("servo,az_cp");
$el_cp = restore("servo,el_cp");

info("az_cp value restored-> $az_cp");
info("el_cp value restored0> $el_cp");

if ($az_cp ne "" && $el_cp ne ""){

$cmd = command("servo,setstowangles,B,$az_cp,$el_cp");
info("sent command setstowangles with seq $cmd");

$cmdstatus =  waitforCmdCompletion($cmd ,10);
info("status of setstowangles command : $cmdstatus");

}


if ($cmdstatus != 5){
        updateBatchStatus(2);
```

4.5 Alarm Configuration

Alarm for a particular sub system can be configured in the following way.

Add the alarm tag in respective sub-system xml.

For e.g. - Following alarm will be raised when **initAllSubystems** fails.

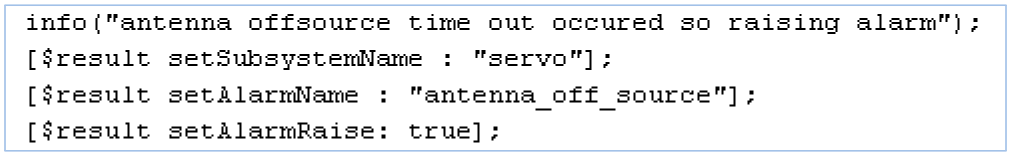


```
<alarm>
    <name>initAllSubsystems</name>
    <label>init failed</label>
    <id>100</id>
    <level>5</level>
    <message>cms initialization failed</message>
</alarm>
```

- <name> - specifies the alarm name, should be unique for a subsystem
- <label> - specifies the alarm label displayed on UI
- <id> - specifies the alarm id
- <level> - specifies the alarm level, should be from 1 to 5
  <message> - message displayed to user on UI when alarm is raised

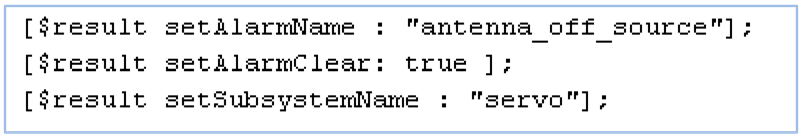
4.6 Logical validation on the received response

1. When a response is received logical validation can be done to check whether the parameters received in response are valid or not.
2. An alarm can also be raised if the logic fails. Only one alarm can be raised or clear through a single logical validator script.
3. Alarm can be raised as mentioned below, where an antenna_off_source alarm is raised:



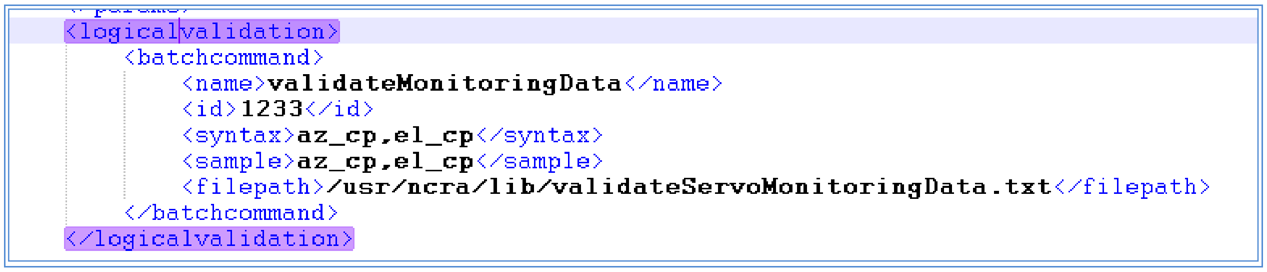
```
info("antenna offsource time out occured so raising alarm");
[$result setSubsystemName : "servo"];
[$result setAlarmName : "antenna_off_source"];
[$result setAlarmRaise: true];
```

4. Alarm can be cleared as following:



```
[$result setAlarmName : "antenna_off_source"];
[$result setAlarmClear: true ];
[$result setSubsystemName : "servo"];
```

5. The logical validation is done using a batch script, the batch script is configured in response in the following way:



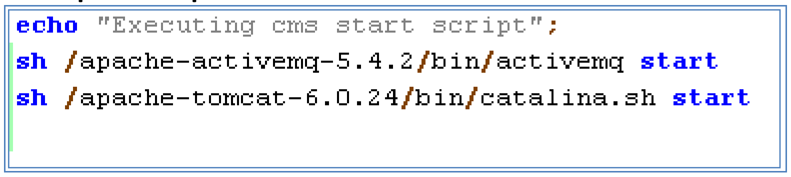
```
  </params>
  <logicalvalidation>
      <batchcommand>
          <name>validateMonitoringData</name>
          <id>1233</id>
          <syntax>az_cp,el_cp</syntax>
          <sample>az_cp,el_cp</sample>
          <filepath>/usr/ncra/lib/validateServoMonitoringData.txt</filepath>
      </batchcommand>
  </logicalvalidation>
```

Refer to servo_commands.xml for addition of logicalvalidation

4.7 Start up and Shutdown shell script configuration

1. The Startup script needs to be configured with the Tomcat and ActiveMQ server paths. A sample Start up script is provided in config folder. The startup script is not being used internally by CMS; the user will have to use it externally for starting Tomcat and ActiveMQ servers.

   startupshellscript.sh

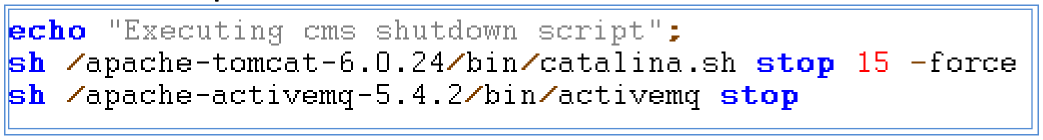
   

   ```
   echo "Executing cms start script";
   sh /apache-activemq-5.4.2/bin/activemq start
   sh /apache-tomcat-6.0.24/bin/catalina.sh start
   ```

   As displayed the Tomcat and ActiveMQ path must be correctly set for the script to work.

2. The Shutdown script needs to be configured with the Tomcat and ActiveMQ server paths. A sample Shutdown script is provided in config folder. The shutdown script is being used internally by CMS to shutdown the Tomcat and ActiveMQ server.

   shutdownhellscript.sh

   ```
   echo "Executing cms shutdown script";
   sh /apache-tomcat-6.0.24/bin/catalina.sh stop 15 -force
   sh /apache-activemq-5.4.2/bin/activemq stop
   ```

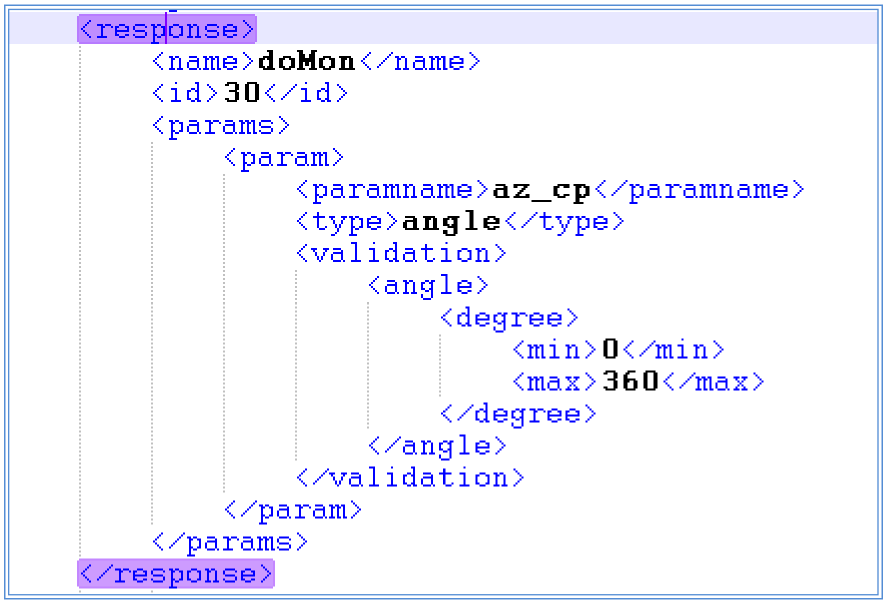
   As displayed the Tomcat and ActiveMQ path must be correctly set for the script to work.

4.8 Configure Monitoring parameters

To configure and test monitoring parameters for a particular sub-system following steps are to be followed:

Consider example for configuring "az_cp" for "**servo**" subsystem

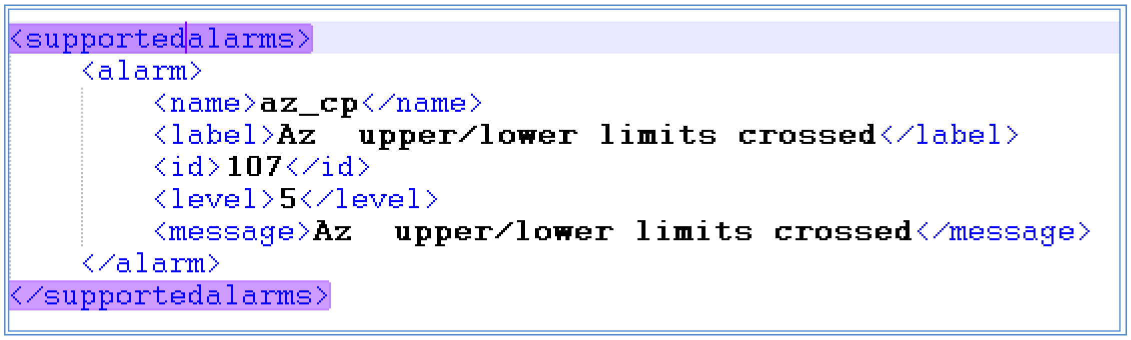
1. Make entry for parameter "**az_cp**" in the "domon" response in servo_commands.xml. Also add validation for corresponding parameter.



```
<response>
    <name>doMon</name>
    <id>30</id>
    <params>
        <param>
            <paramname>az_cp</paramname>
            <type>angle</type>
            <validation>
                <angle>
                    <degree>
                        <min>0</min>
                        <max>360</max>
                    </degree>
                </angle>
            </validation>
        </param>
    </params>
</response>
```

2. Similarly configure "**az_cp**" in servo_engineering.xml so that the parameter status can be observed on the Servo Engineering GUI.

3. If alarm is to be generated when a corresponding parameter crosses its max limit, add alarm entry for that monitoring parameter in that particular subsystem commands .xml.
   For example if alarm is generated when “**az_cp**” reaches maximum limit.
   Configure alarm for it in servo_commands.xml.

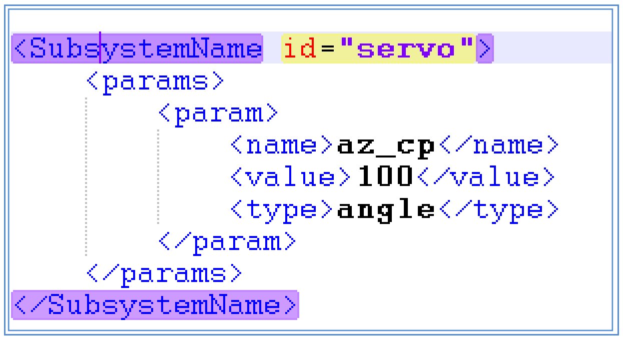


```
<supportedalarms>
    <alarm>
        <name>az_cp</name>
        <label>Az  upper/lower limits crossed</label>
        <id>107</id>
        <level>5</level>
        <message>Az  upper/lower limits crossed</message>
    </alarm>
</supportedalarms>
```

4. For testing monitoring parameters, externally xml is provided (**SubsystemResponse.xml**) to configure monitoring parameters and test them.

   Consider testing of monitoring parameters for servo subsystem.
   Configure sub system name as “**servo**” and add monitoring parameter” **az_cp**” in the param tag.

   Similarly Configurations for other subsystem’s monitoring parameter can be done.



```
<SubsystemName id="servo">
    <params>
        <param>
            <name>az_cp</name>
            <value>100</value>
            <type>angle</type>
        </param>
    </params>
</SubsystemName>
```

5 Global Parameters Configuration

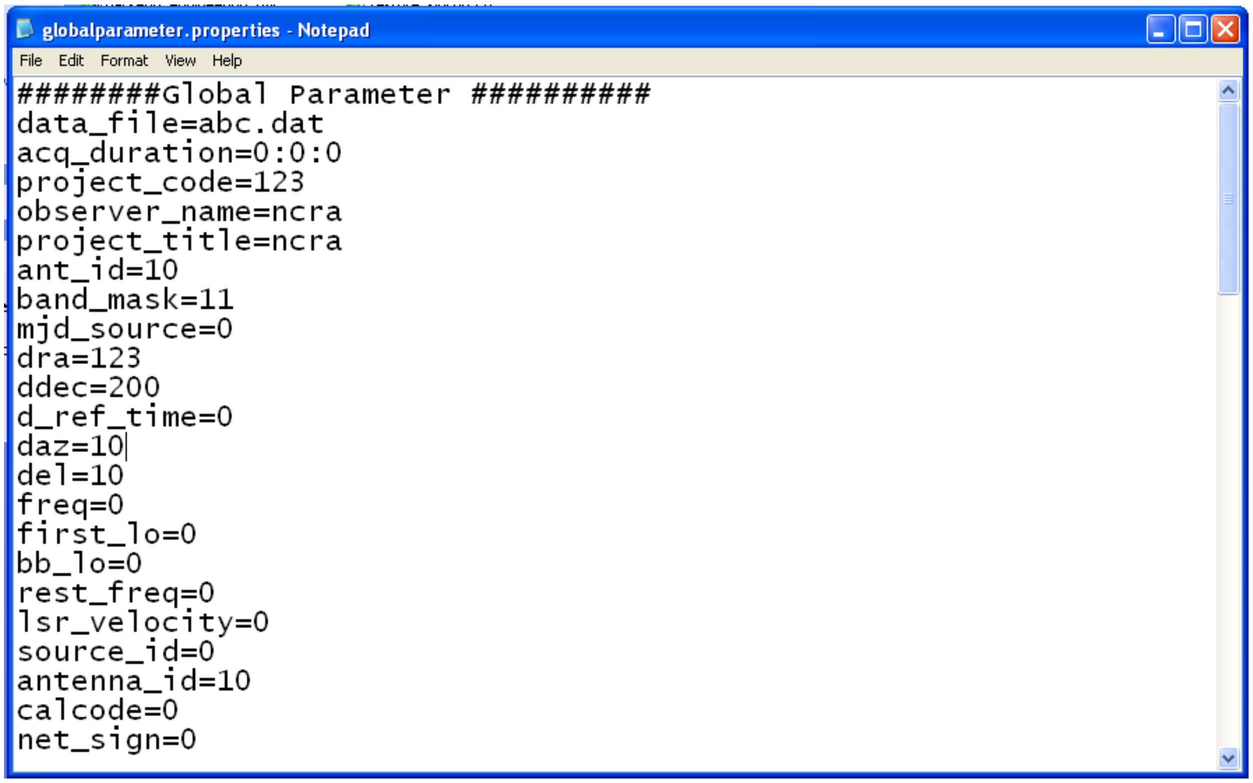
Global parameter feature allows user to configure various configuration parameters. Global parameters are set of property name and value pairs.

These property name and value pairs can be sent to various subsystems via commands, wherein these parameters act as command arguments.

For example trackobject command of servo subsystem, STRTDAS command from digital backend system uses various global parameters set though Expert tab, tune receiver respectively.

5.1 Pre-Requisites

Global parameter variables are configured in “globalparameter.properties” file


globalparameter.properties - Notepad
File Edit Format View Help

```
#######Global Parameter ##########
data_file=abc.dat
acq_duration=0:0:0
project_code=123
observer_name=ncra
project_title=ncra
ant_id=10
band_mask=11
mjd_source=0
dra=123
ddec=200
d_ref_time=0
daz=10
del=10
freq=0
first_lo=0
bb_lo=0
rest_freq=0
lsr_velocity=0
source_id=0
antenna_id=10
calcode=0
net_sign=0
```

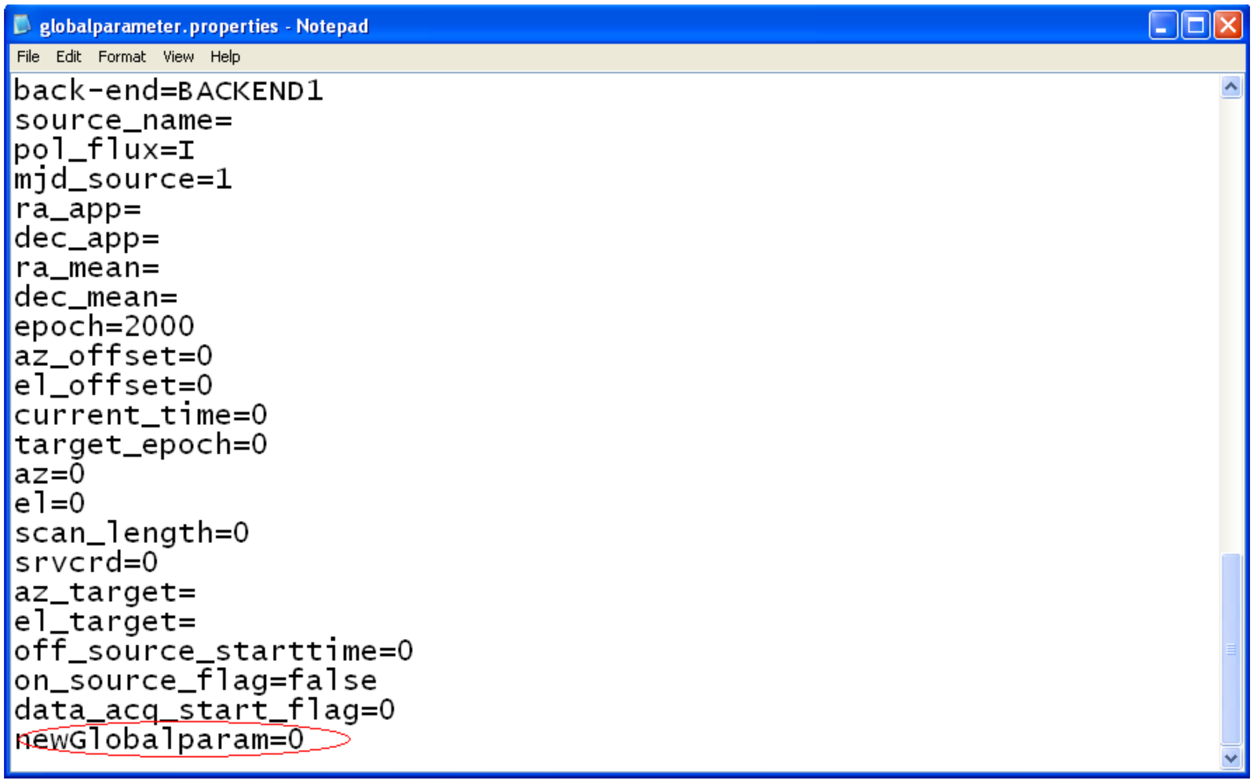
5.2 Addition of parameter in "globalparameter.properties"

1. To add a new parameter as "globalparameter", one needs to make an entry of the same in "globalparameter.properties" file.

   Example:

   User needs to add a new parameter say "**newGlobalparam**" as "globalparameter" data having data type as integer with the range of 0-1.

   Make entry of "**newGlobalparam**" in "**globalparameter.properties**" with some default valid value say "0".


globalparameter.properties - Notepad
File Edit Format View Help

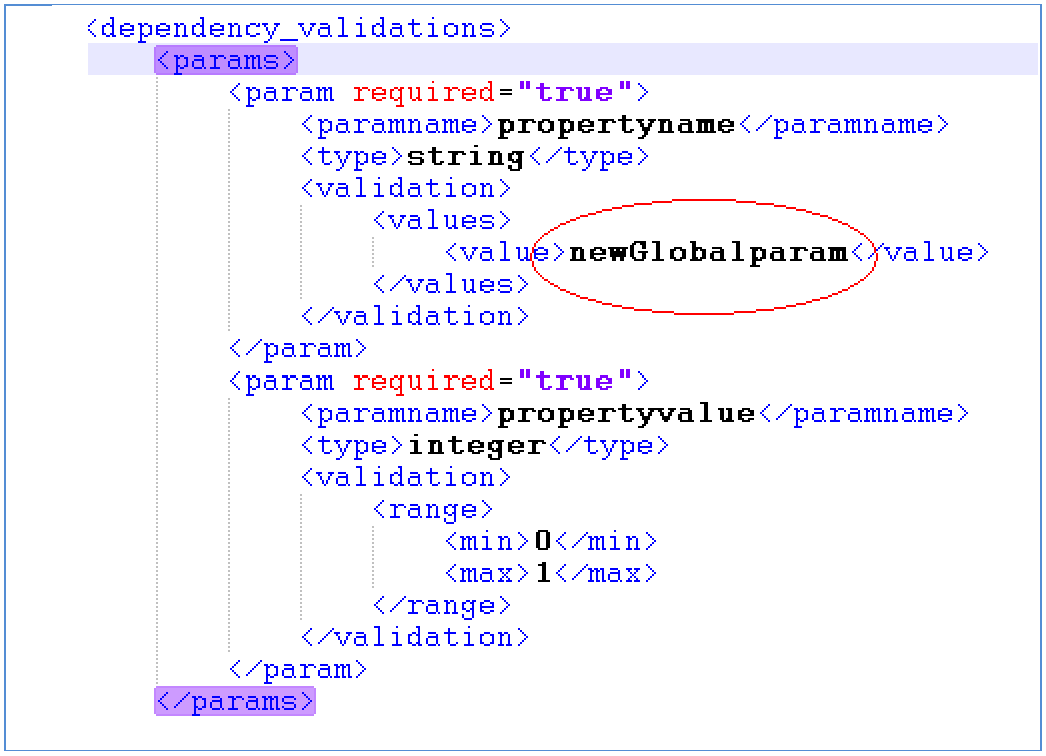
```
back-end=BACKEND1
source_name=
pol_flux=I
mjd_source=1
ra_app=
dec_app=
ra_mean=
dec_mean=
epoch=2000
az_offset=0
el_offset=0
current_time=0
target_epoch=0
az=0
el=0
scan_length=0
srvcrd=0
az_target=
el_target=
off_source_starttime=0
on_source_flag=false
data_acq_start_flag=0
newGlobalparam=0
```

5.3 Managing global parameters

1. A command named "**loadProperty**" of cms-commands.xml can be used to update global parameter values.

   Configure the property "**newGlobalparam**" in "**loadProperty**" command of **cms-commands.xml** in **<dependency_validations>** tag as follows.

   User also needs to specify validations for the added global property.



```
<dependency_validations>
    <params>
        <param required="true">
            <paramname>propertyname</paramname>
            <type>string</type>
            <validation>
                <values>
                    <value>newGlobalparam</value>
                </values>
            </validation>
        </param>
        <param required="true">
            <paramname>propertyvalue</paramname>
            <type>integer</type>
            <validation>
                <range>
                    <min>0</min>
                    <max>1</max>
                </range>
            </validation>
        </param>
    </params>
```

User can now change the value of the added global property through "**Expert"** tab.

2. Specifying the added global property in command as getter or setter argument.

   If user needs to send added global property as input to command/s of particular sub-system then user should specify the global parameter as input argument to particular command/s.

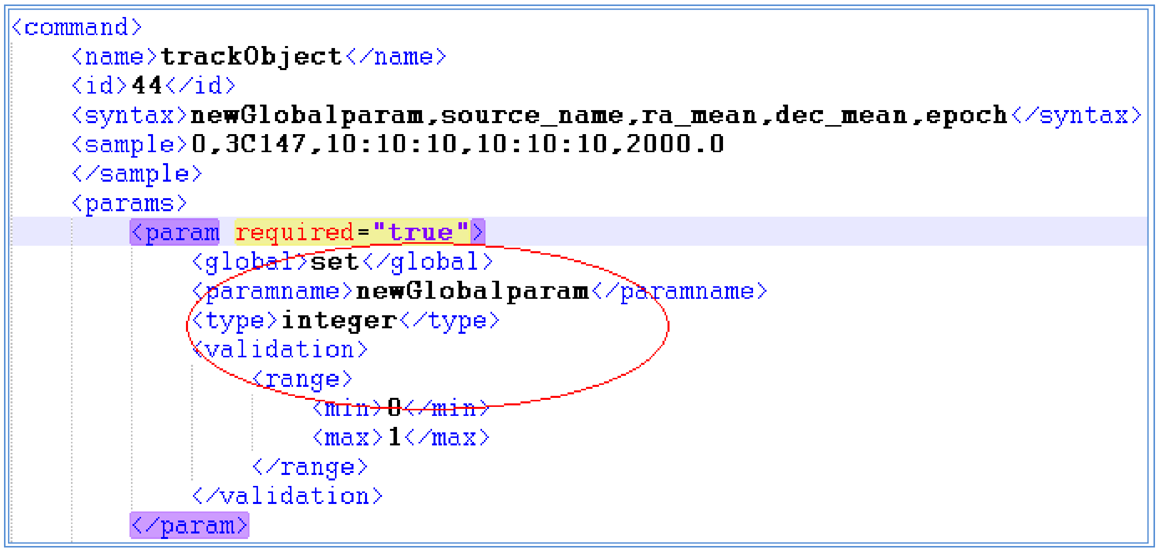
   <global> tag is used to indicate whether a particular parameter is global or not.<global > tag only accepts "**set**" and "**get**" as valid values.

   Example:

   User needs to send the global parameter "**newGlobalparam**" as one of the argument to "**trackobject**" command of servo subsystem. Then user needs to add that parameter as follows in "servo_commands.xml" under two scenarios.

   1. User can set "<**global**>" tag value as "**set**"

      It indicates that CMS will update the global parameter value during the execution of the command.

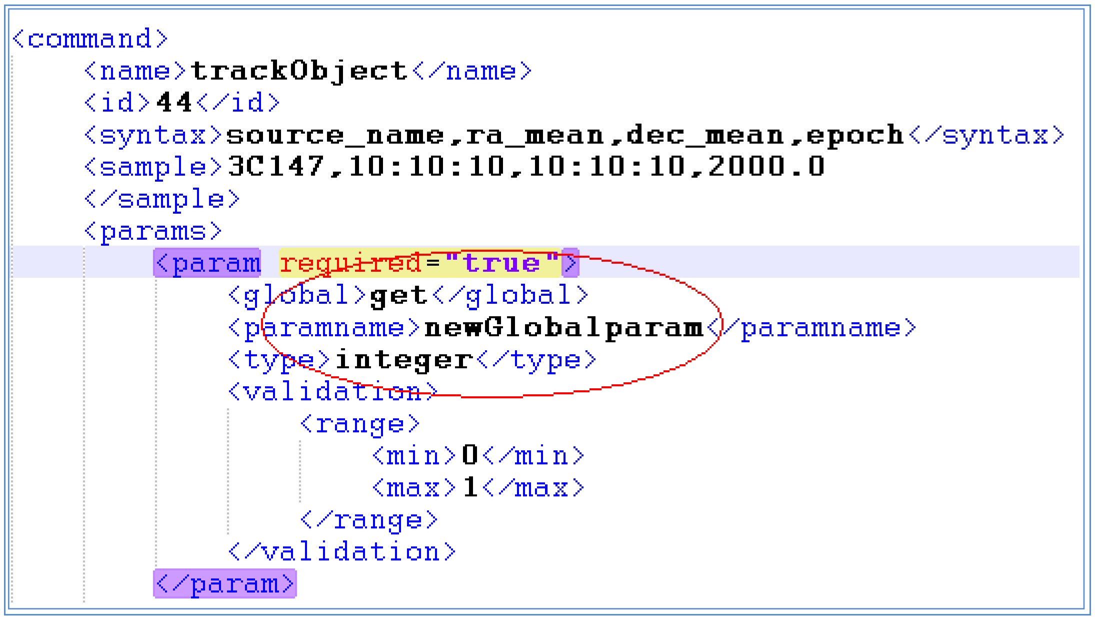


```
<command>
    <name>trackObject</name>
    <id>44</id>
    <syntax>newGlobalparam,source_name,ra_mean,dec_mean,epoch</syntax>
    <sample>0,3C147,10:10:10,10:10:10,2000.0
    </sample>
    <params>
        <param required="true">
            <global>set</global>
            <paramname>newGlobalparam</paramname>
            <type>integer</type>
            <validation>
                <range>
                    <min>0</min>
                    <max>1</max>
                </range>
            </validation>
        </param>
```

In this example, user will have to specify the “**newGlobalparam**” parameter value whenever user tries to execute the “trackobject” command as the <global> tag value is “set”. The value specified by user will be stored in”**Global parameters"** and can be used subsequently by other commands.

2. User can set “<**global**>” tag value as “**get**”.

   It indicates that the CMS will retrieve the pre-existing value from global parameter and set it as command argument.



```
<command>
    <name>trackObject</name>
    <id>44</id>
    <syntax>source_name,ra_mean,dec_mean,epoch</syntax>
    <sample>3C147,10:10:10,10:10:10,2000.0
    </sample>
    <params>
        <param required="true">
            <global>get</global>
            <paramname>newGlobalparam</paramname>
            <type>integer</type>
            <validation>
                <range>
                    <min>0</min>
                    <max>1</max>
                </range>
            </validation>
        </param>
```

6 Engineering UI Configuration

Engineering UI Configuration feature allows user to configure various configuration parameters.

6.1 Configuration files related to Engineering UI

The below mentioned configuration files are present in the lib directory.

1. **ncra-subsystemconfig.xml** :

   It contains subsystem related information along with subsystemname_engineering.XML file used to generate UI from it.

   Syntax: Servo subsystem: Under subsystem tag user can mention configuration about subsystem. Tag <engXML> specifies name of xml file used to generate enginerring UI.

```
 <subsystems>

        <subsystem>
                <name>servo</name>
                <connectionurl>127.0.0.1:7775</connectionurl>
                <commandfile>servo_commands.xml</commandfile>
                <version>1</version>
                <engXML>servo_engineering.xml</engXML>
        </subsystem>

        <subsystem>
                <name>sentinal</name>
                <connectionurl>127.0.0.1:7775</connectionurl>
                <commandfile>sentinal_commands.xml</commandfile>
                <version>1</version>
                <engXML>sentinal_engineering.xml</engXML>
        </subsystem>

</subsystems>
```

2. **subsystemname_commands.XML** (e.g. servo_commands.xml) :

   It contains all the supported commands and supported responses with validation. It also contains Monitoring parameter with validation. Refer to servo_commands.xml for a sample subsystem command, response and monitoring parameter configuration.

   The command issued from UI is validated against command mentioned in this xml. And the responses received from monitoring simulator are validating against responses mentioned in this xml.

3. **subsystemname_engineering.XML** (e.g. servo_engineering.xml) :

   It contains four section **Status param**, **Monitoring param**, **Basic commands** and **Detailed commands** for particular sub system. Engineering UI is loaded from this xml during CMS initialization. This XML defines contents of the Engineering UI.

   User can add a status parameter or Monitoring parameter to UI through this engineering xml. The newly added parameter should be pre-configured as monitoring parameter in **subsystemname_commands.XML**

   User can add a command to Basic Commands or Detailed commands section of **subsystemname_engineering.XML**. The command added should have been pre-configured in the subsystemname_commands.XML.

   If command is not specified in subsystemname_commands.xml then UI shows "Command not found" error message.

6.2 Add group of parameters in status parameter section

To add group of parameter in status or monitoring section in engineering UI, User should add tag of **<paramsgroup>** under status or monitoring section of subsystem_engineering.xml. If the parameter is added to both the sections it will be visible in both sections.

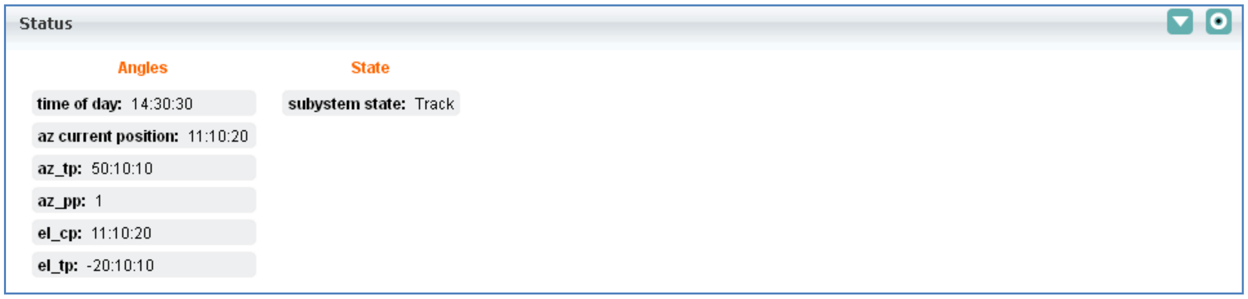
For e.g. if user wants to add XYZ group with parameter in status params section the xml syntax is:

```
<statusparams>

        <paramsgroup>
                <groupname>XYZ</groupname>
                <param>
                        <label>param1 </label>
                        <paramname>timeofday</paramname>
                        <type>datetime</type>
                        <paramvalue>10:12:20</paramvalue>
                </param>

                <param>
                        <label>param2</label>
                        <paramname>az_cp</paramname>
                        <type>angle</type>
                        <paramvalue>123:50:10</paramvalue>
                </param>
        </paramsgroup>

</statusparams>
```

The fields specified above indicate the following:

<label> - label to be displayed on screen
<paramname> - the actual name of the parameter as received in monitoring response
<type> - the data type of the parameter. To display the parameter as bit where the background color changes as per bit value the <type> should have value as **bit**
<paramvalue> - the default parameter value to be displayed on UI. If no default value is present this field can be left blank.

The added section and parameters will be visible in UI as displayed below:


Status

Angles	State
time of day: 14:30:30	subystem state: Track
az current position: 11:10:20	
az_tp: 50:10:10	
az_pp: 1	
el_cp: 11:10:20	
el_tp: -20:10:10	

6.3 Add group of parameters in monitoring parameter section

To add group of parameter in monitoring section in engineering UI, User should add tag of **<detailedparams>** under monitoring section of subsystem_engineering.xml.

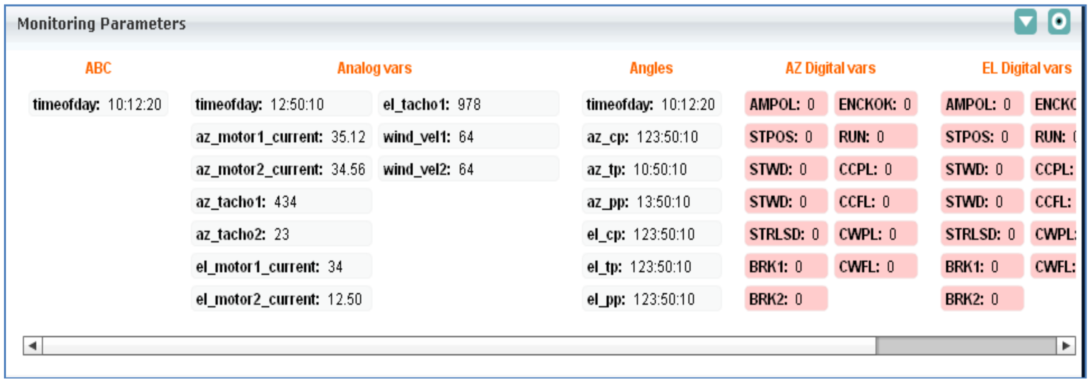
For e.g. if user wants to add XYZ group with parameter in Monitoring params section the xml syntax is:

```
<detailedparams>
        <paramsgroup>
                <groupname>ABC </groupname>
                <param>
                        <label>time of day</label>
                        <paramname>timeofday</paramname>
                        <type>datetime</type>
                        <paramvalue>10:12:20</paramvalue>
                </param>
        </paramsgroup>
</detailedparams>
```

The fields specified above indicate the following:

<label> - label to be displayed on screen
<paramname> - the actual name of the parameter as received in monitoring response
<type> - the data type of the parameter. To display the parameter as byte the <type> should have value as **bit**
<paramvalue> - the default parameter value to be displayed on UI. If no default value is present this field can be left blank.

The added section and parameters will be visible in UI as displayed below:



Monitoring Parameters

ABC	Analog vars		Angles	AZ Digital vars		EL Digital vars	
timeofday: 10:12:20	timeofday: 12:50:10	el_tacho1: 978	timeofday: 10:12:20	AMPOL: 0	ENCKOK: 0	AMPOL: 0	ENCK
	az_motor1_current: 35.12	wind_vel1: 64	az_cp: 123:50:10	STPOS: 0	RUN: 0	STPOS: 0	RUN:
	az_motor2_current: 34.56	wind_vel2: 64	az_tp: 10:50:10	STWD: 0	CCPL: 0	STWD: 0	CCPL:
	az_tacho1: 434		az_pp: 13:50:10	STWD: 0	CCFL: 0	STWD: 0	CCFL:
	az_tacho2: 23		el_cp: 123:50:10	STRLSD: 0	CWPL: 0	STRLSD: 0	CWPL:
	el_motor1_current: 34		el_tp: 123:50:10	BRK1: 0	CWFL: 0	BRK1: 0	CWFL:
	el_motor2_current: 12.50		el_pp: 123:50:10	BRK2: 0		BRK2: 0	

All status and monitoring parameter values are updated when CMS receive response from wrapper.

6.4 Add raw parameters in raw parameter section

To add a raw parameter to engineering engineering UI, User should add tag of <paramsgroup> under <rawparams> section of subsystem_engineering.xml.

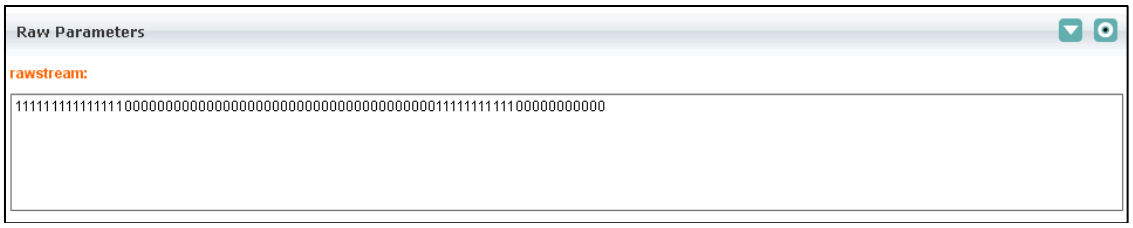
This raw param value is sent through monitoring parameter with name **rawstream**. The following is the syntax for adding a raw parameter to Engineering UI:

```
<rawparams>
        <paramsgroup>
                <groupname>Raw Stream</groupname>
                <param>
                        <label>raw stream</label>
                        <paramname>rawstream</paramname>
                        <type>string</type>
                        <paramvalue>
11111111111111100000000000000000000000000000000000000001111111111100000000000
                        </paramvalue>
                        <bgcolor>0x0000FF</bgcolor>
                </param>
        </paramsgroup>
</rawparams>
```

The fields specified above indicate the following:

<label> - label to be displayed on screen
<paramname> - the actual name of the parameter as received in monitoring response
<type> - the data type of the parameter. To display the parameter as byte the <type> should have value as **bit**
<paramvalue> - the default parameter value to be displayed on UI. If no default value is present this field can be left blank.

The raw parameter section will be visible in UI as displayed below:



rawstream:

11111111111111100000000000000000000000000000000000000001111111111100000000000

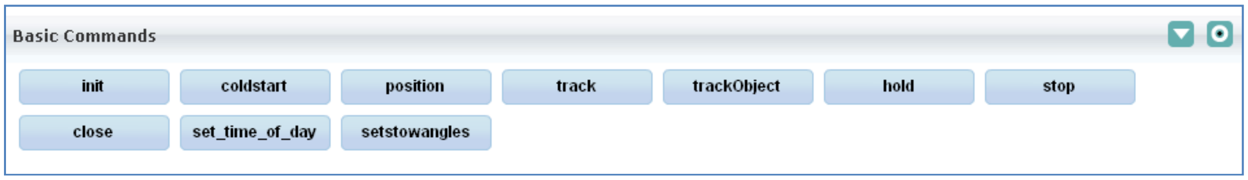
6.5 Add commands to basic command section

To add a command in basic command section user must add **<command>** tag under basic command tag of subsystemname_engineering.xml. The command syntax and validation should have been configured under supported commands tag of subsystemname_commands.xml file.

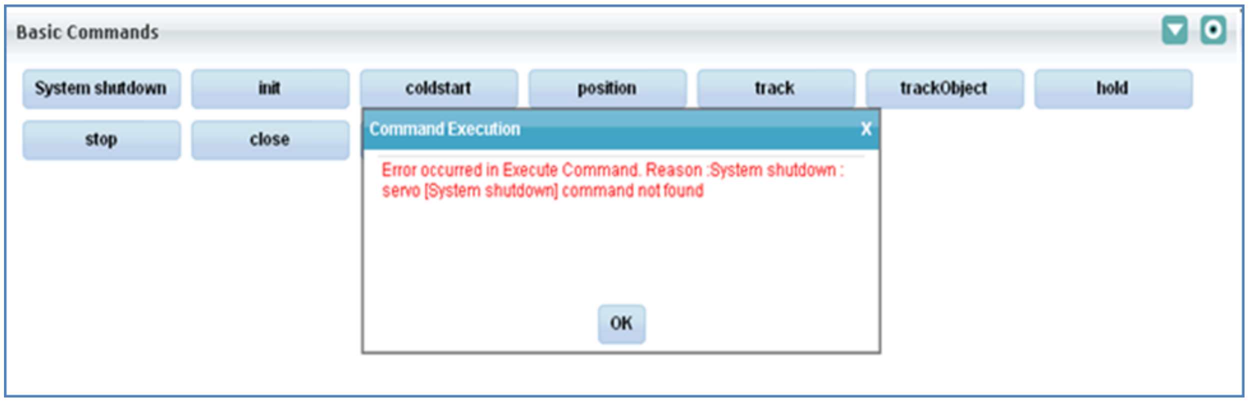
To add command System shutdown in basic section the xml syntax is:

```
<basiccommands>
        <command>
                <name>System shutdown</name>
                <id>1</id>
                <syntax></syntax>
                <sample></sample>
        </command>
</basiccommands>
```

The added command will be visible in UI as displayed below:



If command is not configured in subsystemname_commands.xml and then on executing the command then the user will get an error message "command not found".



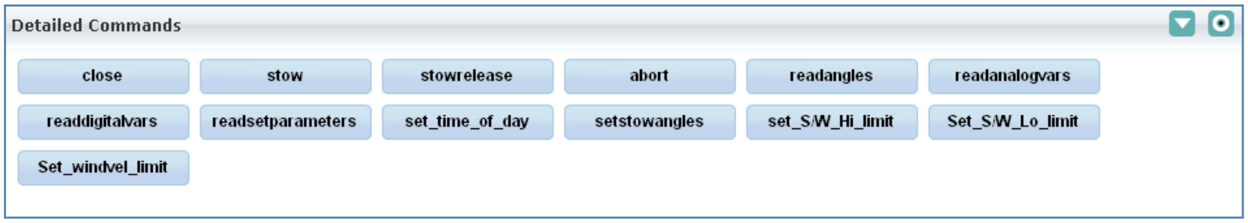
Error occurred in Execute Command. Reason :System shutdown : servo [System shutdown] command not found

6.6 Add commands to details command section

To add a command in detailed command section user must add **<command>** tag under detailed command tag of subsystemname_engineering.xml.

The command syntax and validation should have been configured under supported commands tag of subsystemname_commands.xml file.

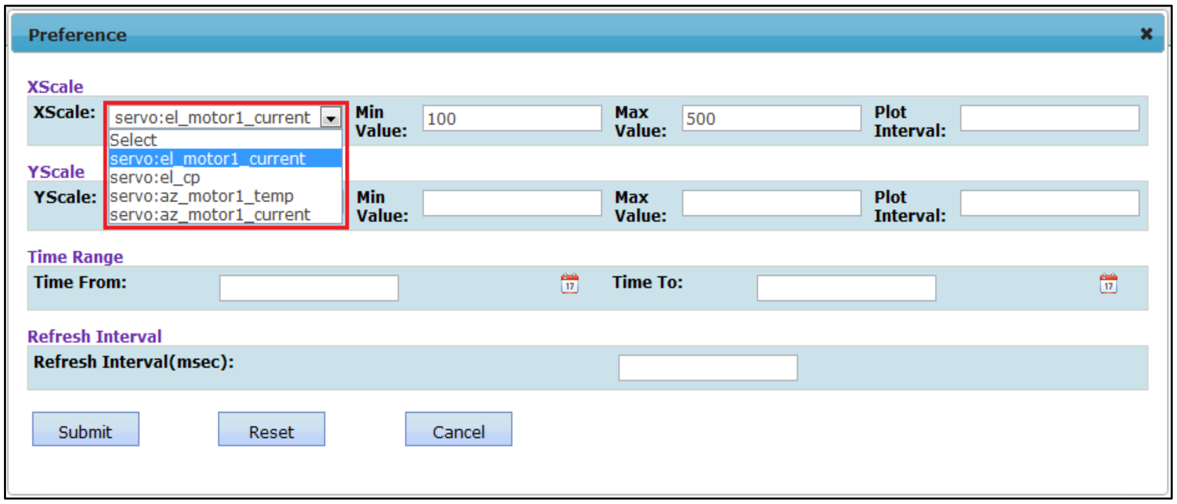
The look and feel of the added command will be similar to the basic command.



6.7 Add monitoring parameter values in trend plot section

Trend plot preferences UI is a dynamically generated UI, using servoTrendPlot.xml and ChartRecorder.xsl
User can dynamically add or delete parameters on UI by modifying servoTrendPlot.xml

It allows user to set the chart recorder's x scale and y scale values.


Preference

XScale
XScale: servo:el_motor1_current
Select
servo:el_motor1_current
servo:el_cp
servo:az_motor1_temp
servo:az_motor1_current
Min Value: 100
Max Value: 500
Plot Interval:

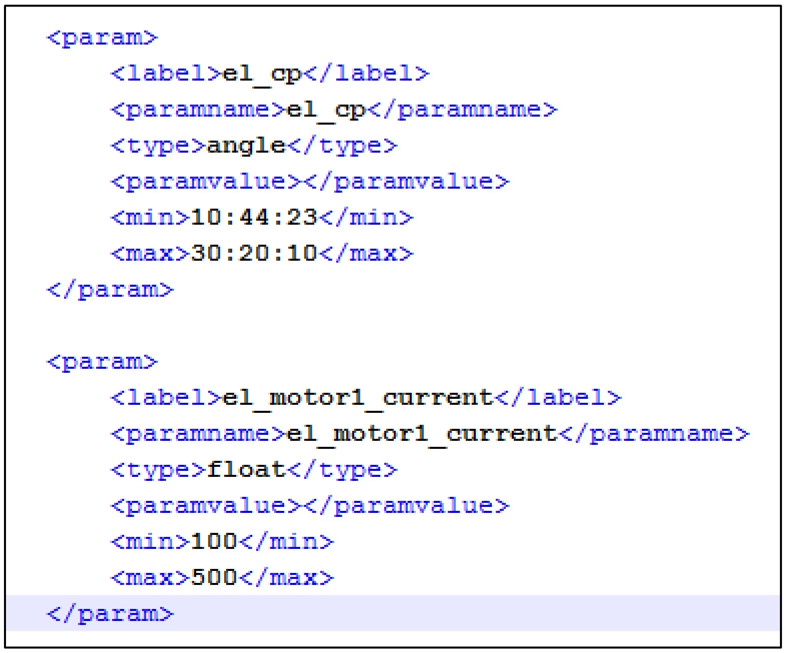
YScale
YScale:
Min Value:
Max Value:
Plot Interval:

Time Range
Time From:
Time To:

Refresh Interval
Refresh Interval(msec):

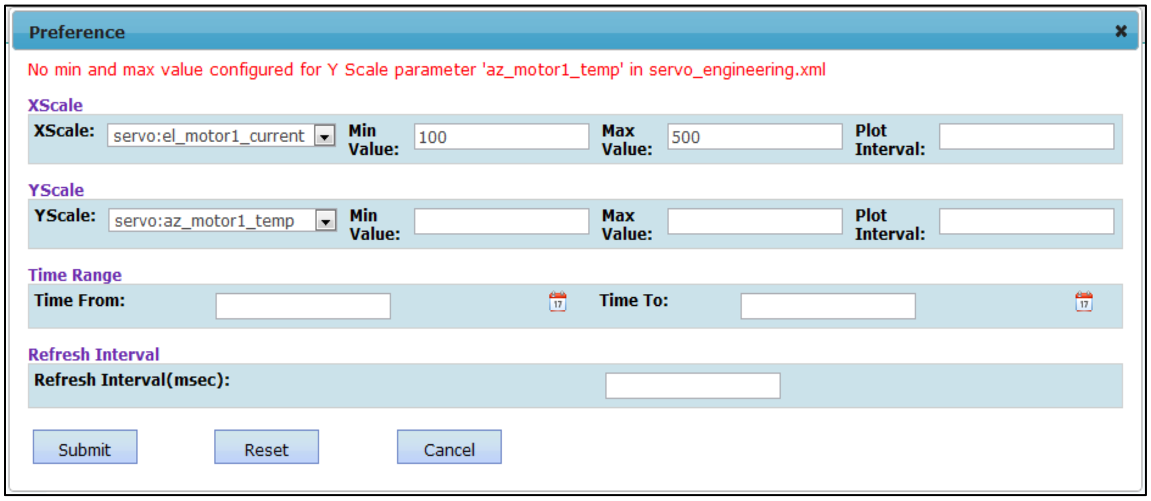
- X scale is monitoring parameter, when user select any monitoring parameter its min and max value automatically populated.

- For auto min and max value population, user needs to configure the monitoring parameter min and max value in <subsystem>_engineering.xml, here showing the servo_engineering.xml entry as below.



```
<param>
    <label>el_cp</label>
    <paramname>el_cp</paramname>
    <type>angle</type>
    <paramvalue></paramvalue>
    <min>10:44:23</min>
    <max>30:20:10</max>
</param>

<param>
    <label>el_motor1_current</label>
    <paramname>el_motor1_current</paramname>
    <type>float</type>
    <paramvalue></paramvalue>
    <min>100</min>
    <max>500</max>
</param>
```

- If any of the parameter min and max value not configured in <subsystem>_engineering.xml than user needs to manually enter these value and will get below informative message on preference page.



Preference

No min and max value configured for Y Scale parameter 'az_motor1_temp' in servo_engineering.xml

XScale

XScale: servo:el_motor1_current | Min Value: 100 | Max Value: 500 | Plot Interval:

YScale

YScale: servo:az_motor1_temp | Min Value: | Max Value: | Plot Interval:

Time Range

Time From: | Time To:

Refresh Interval

Refresh Interval(msec):

6.8 Configure Engineering UI in CMS

6.8.1 Configure in applicationcontext.xml

User must enter entry of URL in applicationcontext.xml (file path : tomcat/webapps/cms-web/web-inf/applicationcontext.xml) under <security:http> tag.

Entry should be in following format.

e.g.

```
 <security:intercept-url
          pattern="/engineeringUIController\.htm\?systemID=servo"
          access="BF_VIEW_ENGINEERING_servo" />
```

In above example user added entry for servo subsystem.

Similar entry can added for sentinal subsystem but user must change system id to sentinal which is same name as of subsystem name mentioned in ncra-subsystemconfig.xml.

6.8.2 Configure in database

Add above mentioned permission in database

e.g.

BF_VIEW_ENGINEERING_servo in database using the query specified below.

Database Query to add permission:

INSERT INTO ncracmsdb.t_group_permission values (NULL,'BF_VIEW_ENGINEERING_xxx','xxx ENGINEERING','YES');

xxx:- it is the subsystem name for which Engineering UI is required

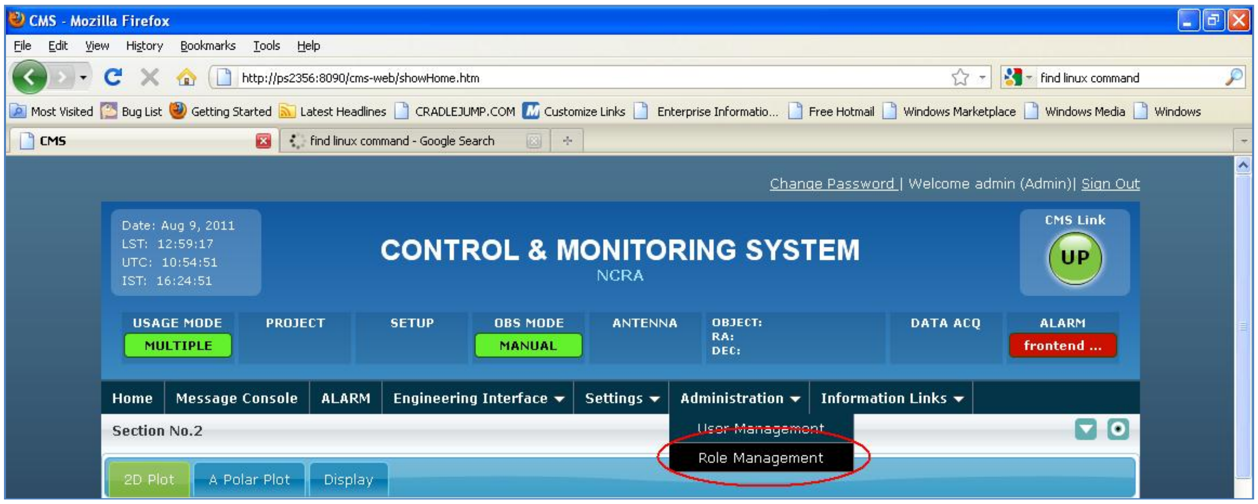
If all above configurations are done then restart tomcat.

Note: Always insert new row at end of the table "t_group_permission" and corresponding id value should be greater than maximum id value already present in database, do not insert any rows in between or with id value less than current maximum id value, this will lead to unexpected CMS behavior.

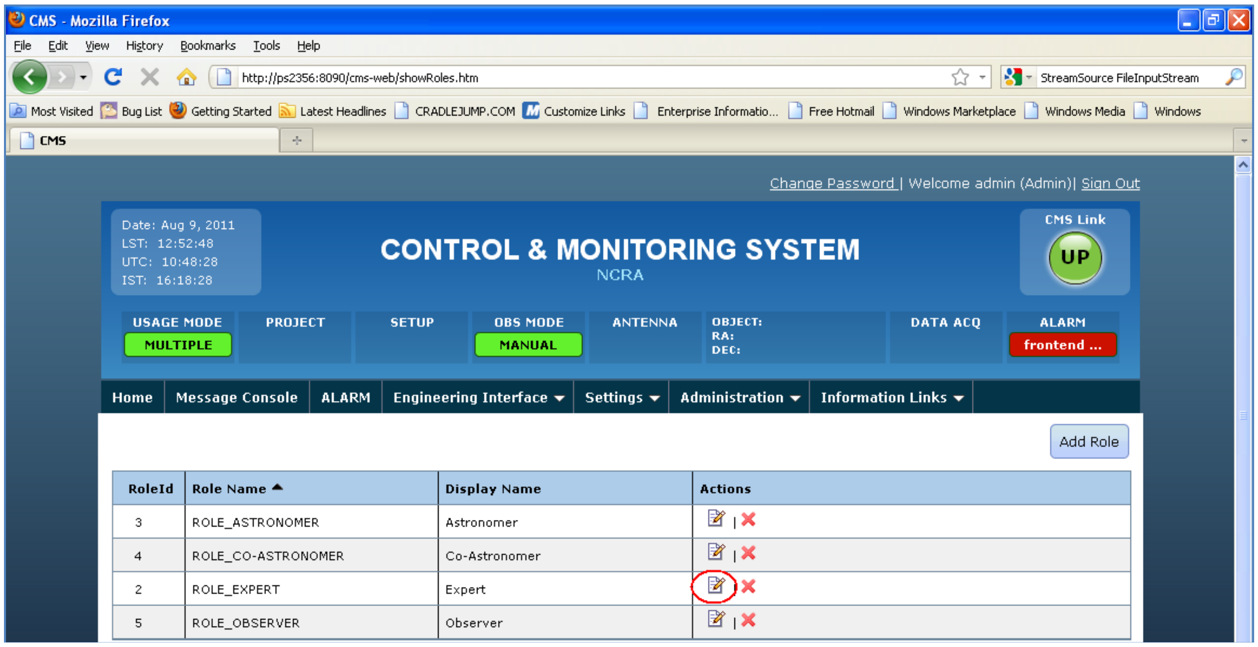
6.8.3 Configure engineering permissions in CMS

6.8.3.1 Assign the newly added permission to appropriate role

1. Login as admin
2. Select Role Management under Admin tab as shown below:

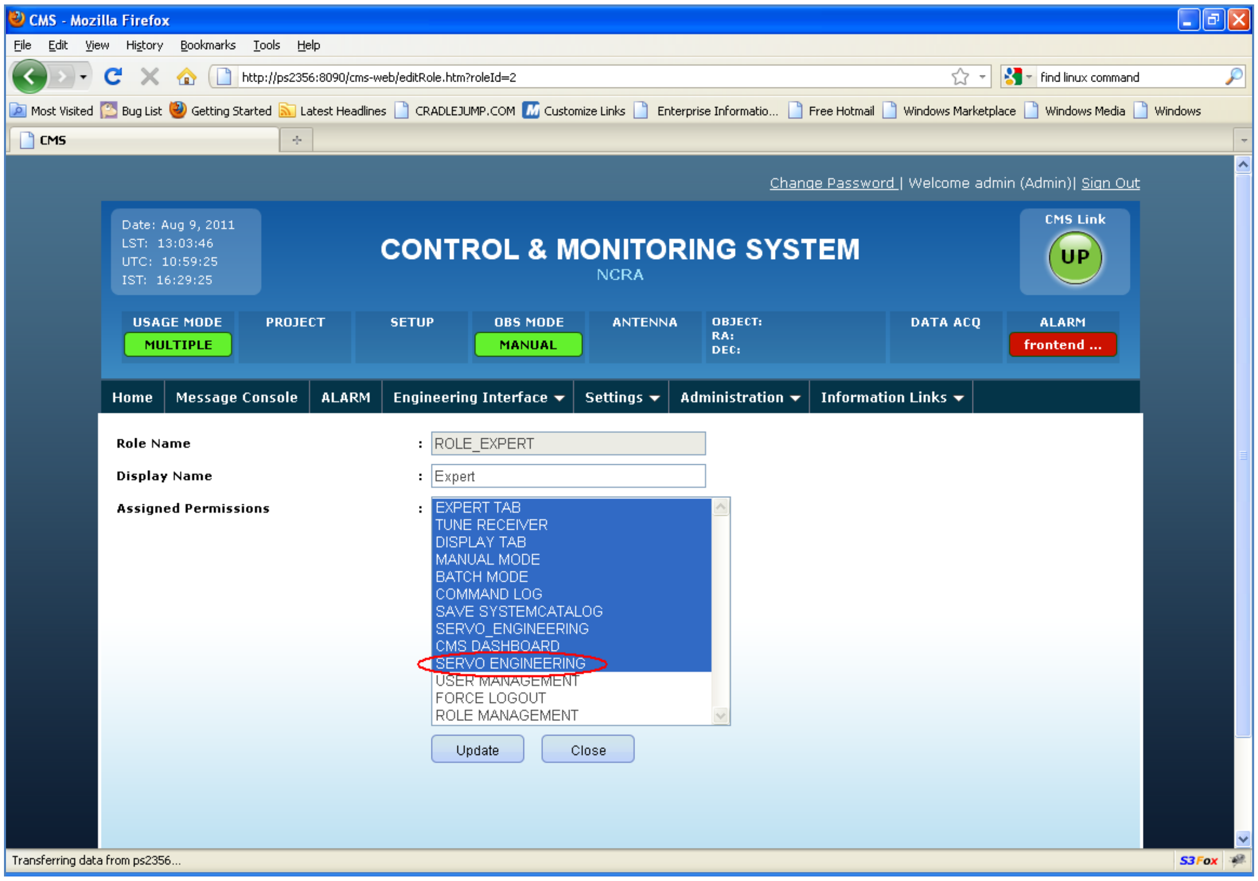


3. Select the role for which the permission needs to be assigned. For e.g. the below screen shot display the edit role operation for Expert Role.



RoleId	Role Name	Display Name	Actions
3	ROLE_ASTRONOMER	Astronomer	✎ | ✗
4	ROLE_CO-ASTRONOMER	Co-Astronomer	✎ | ✗
2	ROLE_EXPERT	Expert	✎ | ✗
5	ROLE_OBSERVER	Observer	✎ | ✗

4. Select the added permission along with other required permissions as shown below and click on update to apply the permissions to the Role.



Role Name	:	ROLE_EXPERT
Display Name	:	Expert
Assigned Permissions	:	EXPERT TAB TUNE RECEIVER DISPLAY TAB MANUAL MODE BATCH MODE COMMAND LOG SAVE SYSTEMCATALOG SERVO_ENGINEERING CMS DASHBOARD SERVO ENGINEERING USER MANAGEMENT FORCE LOGOUT ROLE MANAGEMENT

7 Dynamic UI Generation Configuration

This section contains the details about how to generate UI dynamically just by modifying few xml files. It illustrates an example each wherever dynamic UI has been generated in CMS application.

7.1 Introduction

For generating a UI dynamically we basically need 2 files

1. The template xml file which contains the UI elements. In most cases only this file needs to be changed to add new control on UI.
2.

Following is the list of templates present in CMS for various screens

   a) Receiver setup files
      - ✓ continuum.xsl
      - ✓ spectral-line.xml
      - ✓ pulsar.xml
      - ✓ planetary.xml
      - ✓ sun-moon.xml

   b) ChartRecorder.xml – Template xml to generate the UI for chart recorder.
   c) servoTrendPlot.xml – Template xml to generate the UI for the trend plot
   d) SpectralLineDisplay.xml – Template xml to generate the UI for the spectral plot
   e) PulsarDisplay.xml – Template xml to generate the UI for the pulsar plot
   f) MetaData.xml – Template xml to generate the UI for the Meta Data tab

3. The xsl file is used to transform template xml into HTML format. This file will not need any change most of the time. Server restart is needed for change in this file to reflect.

   Following is the list of xsl files already used in CMS to generate UI

   a) htmlgenerator.xsl – xsl used to generate Tune Receiver UI
   b) ChartRecorder.xml – xsl used to process ChartRecorder.xml file and generate UI for chart recorder.
   c) servoTrendPlot.xml – xsl used to process ChartRecorder.xml file and generate UI for trend Plot preferences
   d) SpectralLineDisplay.xsl – xsl used to process SpectralLineDisplay.xml file and generate UI for Spectral Line Display preferences
   e) PulsarDisplay.xsl – – xsl used to process PulsarDisplay.xml file and generate UI for Pulsar Display preferences
   f) MetaData.xsl – – xsl used to process MetaData.xml file and generate UI for Meta Data tab.

7.2 Details

Let us start explaining the generation of dynamic UI with the help of an example.

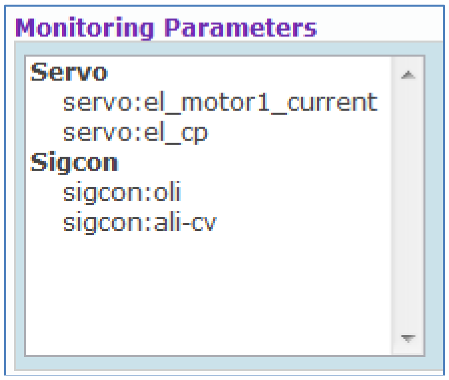
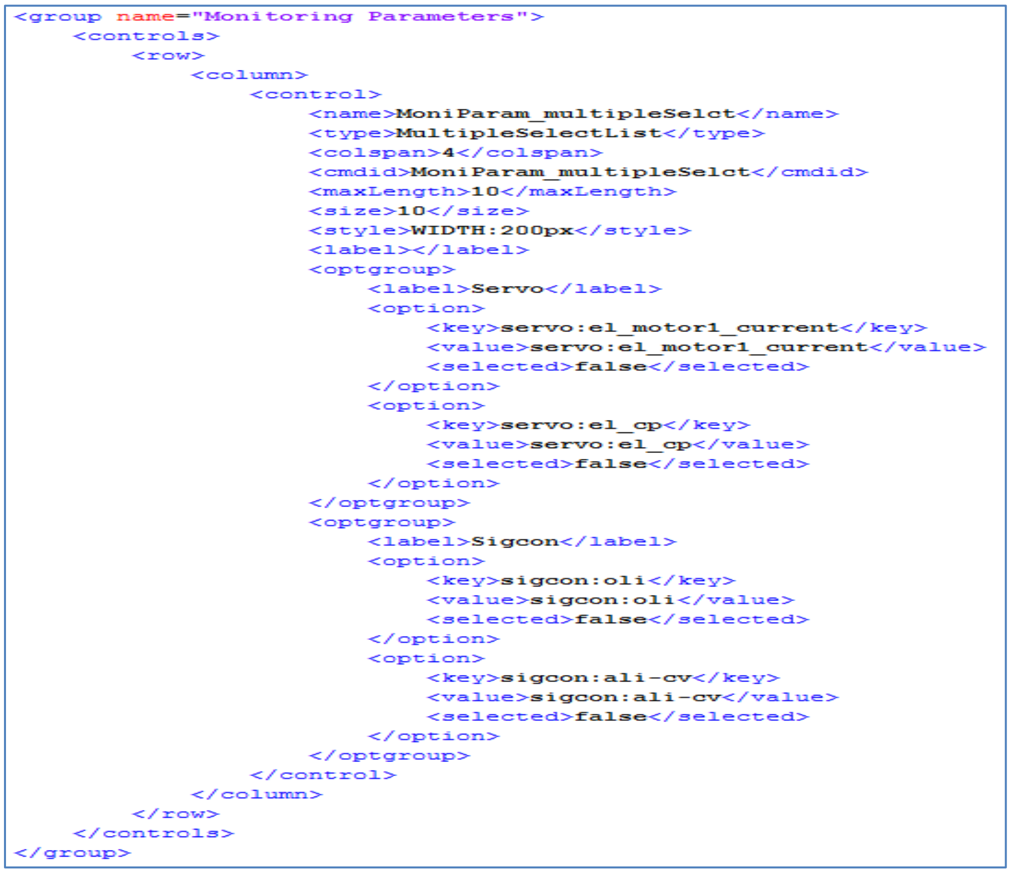


Figure 1

The figure shown above contains a select box. For generating this UI dynamically we have to make a template xml file of the UI elements and an xsl file which will transform the template xml file into the HTML format.

The template xml for the above UI will look like this:



```
<group name="Monitoring Parameters">
    <controls>
        <row>
            <column>
                <control>
                    <name>MoniParam_multipleSelct</name>
                    <type>MultipleSelectList</type>
                    <colspan>4</colspan>
                    <cmdid>MoniParam_multipleSelct</cmdid>
                    <maxLength>10</maxLength>
                    <size>10</size>
                    <style>WIDTH:200px</style>
                    <label></label>
                    <optgroup>
                        <label>Servo</label>
                        <option>
                            <key>servo:el_motor1_current</key>
                            <value>servo:el_motor1_current</value>
                            <selected>false</selected>
                        </option>
                        <option>
                            <key>servo:el_cp</key>
                            <value>servo:el_cp</value>
                            <selected>false</selected>
                        </option>
                    </optgroup>
                    <optgroup>
                        <label>Sigcon</label>
                        <option>
                            <key>sigcon:oli</key>
                            <value>sigcon:oli</value>
                            <selected>false</selected>
                        </option>
                        <option>
                            <key>sigcon:ali-cv</key>
                            <value>sigcon:ali-cv</value>
                            <selected>false</selected>
                        </option>
                    </optgroup>
                </control>
            </column>
        </row>
    </controls>
</group>
```

Figure 2

Let us start explaining each node one by one.

1. Groups – It contains the list of groups present under a section. The attribute available with node is ‘templateName’ where user can mention the name of the template. It’s useful in some cases however; it’s not used as of now.
2. Group – It groups various HTML elements to form a section, so to add a new group related to new subsystem, one need to add a new group section to template xml. The supported attribute is on this node is “name”.
3. Controls – One section can contain various HTML controls. The list of these controls is provided in this. It contains 4 attributes i.e. type, cssClass, sectionname, rowspan.
4. Row - Represents a line on GUI, if user need to add a new control on new line then this tag should be added. This is synonymous <TR> element in HTML.
5. Column - Represents a cell on UI, this acts as a container of actual UI elements (e.g. Textbox, radio button etc.). This is synonymous <TD> element in HTML.
6. Control - Represents actual UI element on the screen, user can configure various properties for control depending upon the type of control chosen, following are various configurable properties of this element.
    a. Disabled - Used to disable a control on UI, so that by default user can’t change default values for a preconfigured template.
    b. type - Used to configure type of element to be added on UI, following are various values
        i. radio – adds a radio button control on UI
        ii. checkbox – adds a checkbox element on UI
        iii. Text – adds a textbox element on UI
        iv. Dropdown – adds a dropdown element on UI
        v. MultipleSelectList –adds a multiple select dropdown element on UI
    c. name – Actual name of the control corresponding to element on UI.
    d. cmdid – This is same as “name” field mentioned and is kept their for future use (if needed), however, this might be removed in future
    e. label – Used to specify the label of the UI element (e.g. – APERTURE)
    f. value – Used to specify default value for UI element.
    g. subsystemName – Used to specify the name of the subsystem associated with element
    h. options – For controls which have range of valid values (e.g. position values) we use dropdown control. For such controls we use Dropdown on UI. This property is used to configure various values.
    i. checked – Used to specify whether checkbox element on UI should be checked by default
    j. maxLength – It is used to specify the maximum number of characters allowed in that UI element.
    k. onClick – This property is used for buttons. It contains the name of the javascript function which is to be called on click of that button.
    l. onChange – This property is used for dropdowns. It contains the name of the method which is to be called on change of the dropdown option.
    m. Tooltip – It contains the data to be shown in the tooltip.
    n. Style – It is used to specify the style, such as height, width, background color, alignment, etc.

o. Size – it is used to specify the number of the parameter shown in multiple select box without scroll, if a value is 5 than at-least 5 elements displayed.

So in the above UI all the controls fall under the section named 'Monitoring Parameters'. Here the controls are arranged in 2 rows. The first row contains the control of type dropdown and the second row contains 2 controls of type button arranged in 2 columns.

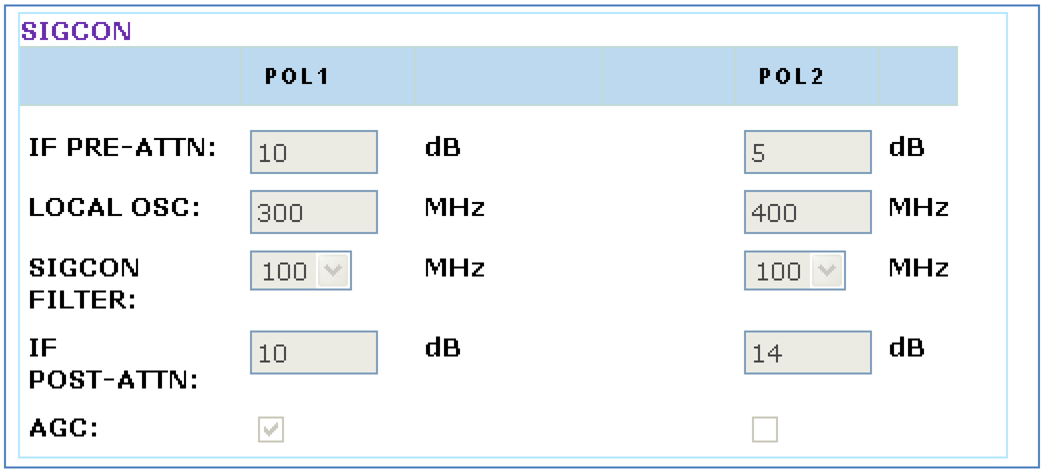
7.3 Examples

7.3.1 Tune Receiver

Following is the command configuration section for receiver setup for SIGCON system before adding a new parameter.

We will add a configuration parameter named “NEW”, which has 2 channels.

Note that NEW here is used to just indicate a new command/configuration parameter name, but it can be any command name like move, track etc.

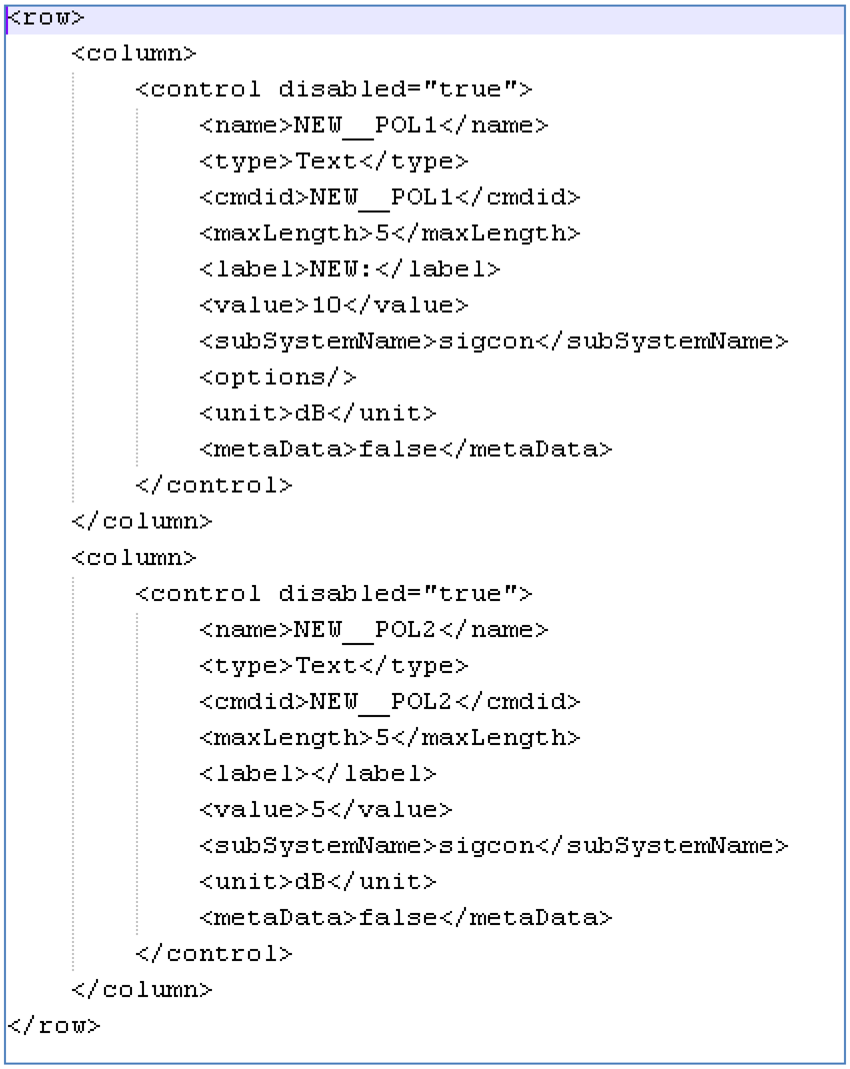


SIGCON

	POL1		POL2	
IF PRE-ATTN:	10	dB	5	dB
LOCAL OSC:	300	MHz	400	MHz
SIGCON FILTER:	100	MHz	100	MHz
IF POST-ATTN:	10	dB	14	dB
AGC:	✓		☐	

Figure 3

Let’s add following lines in continuum.xml at the end of sigcon section.



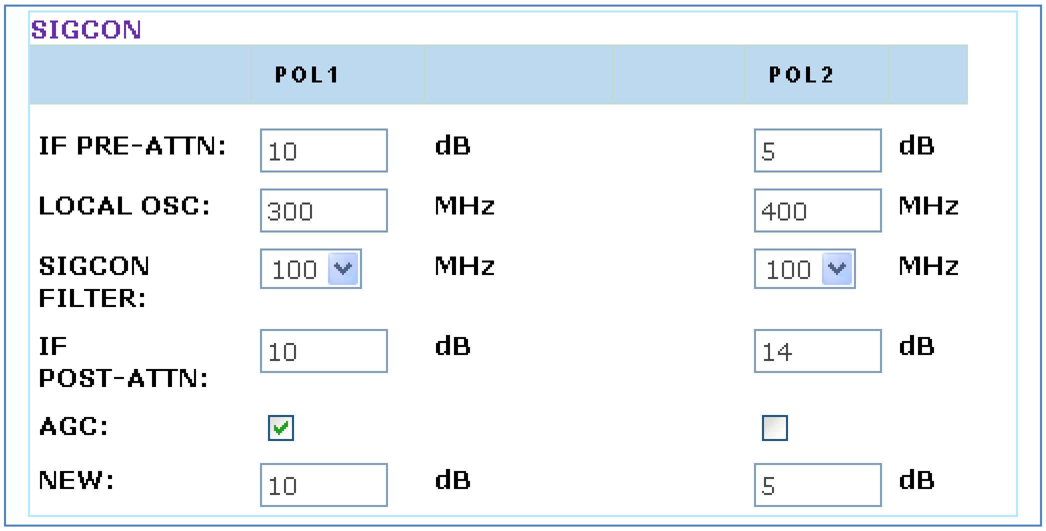
```xml
<row>
    <column>
        <control disabled="true">
            <name>NEW__POL1</name>
            <type>Text</type>
            <cmdid>NEW__POL1</cmdid>
            <maxLength>5</maxLength>
            <label>NEW:</label>
            <value>10</value>
            <subSystemName>sigcon</subSystemName>
            <options/>
            <unit>dB</unit>
            <metaData>false</metaData>
        </control>
    </column>
    <column>
        <control disabled="true">
            <name>NEW__POL2</name>
            <type>Text</type>
            <cmdid>NEW__POL2</cmdid>
            <maxLength>5</maxLength>
            <label></label>
            <value>5</value>
            <subSystemName>sigcon</subSystemName>
            <unit>dB</unit>
            <metaData>false</metaData>
        </control>
    </column>
</row>
```

Figure 4

Please notice the contents of “name” tag above, it has following format X__Y,

X here represents the command name, whereas Y indicates the parameter name. Both of these should be separated by ‘__’ (double underscore). If a command has multiple parameters then one can just repeat entries of X__Y1, X__Y2 and so on in xml file under “column/control” nodes.

Following is effect of adding these lines on tune receiver UI.

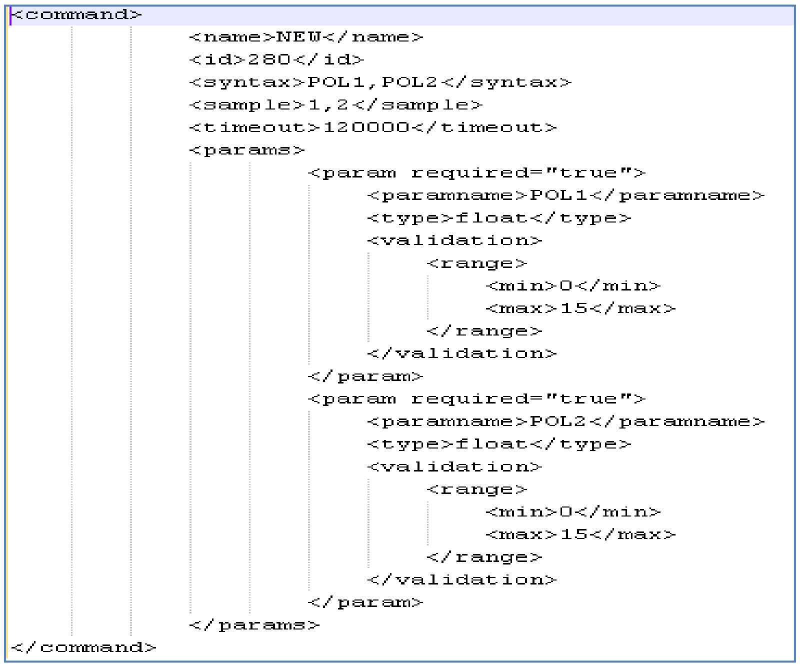


SIGCON

	POL1		POL2	
IF PRE-ATTN:	10	dB	5	dB
LOCAL OSC:	300	MHz	400	MHz
SIGCON FILTER:	100	MHz	100	MHz
IF POST-ATTN:	10	dB	14	dB
AGC:	✓		☐	
NEW:	10	dB	5	dB

Figure 5

In order to execute this newly added command we also need to add new command in sigcon_commands.xml –



```
<command>
            <name>NEW</name>
            <id>280</id>
            <syntax>POL1,POL2</syntax>
            <sample>1,2</sample>
            <timeout>120000</timeout>
            <params>
                    <param required="true">
                        <paramname>POL1</paramname>
                        <type>float</type>
                        <validation>
                            <range>
                                <min>0</min>
                                <max>15</max>
                            </range>
                        </validation>
                    </param>
                    <param required="true">
                        <paramname>POL2</paramname>
                        <type>float</type>
                        <validation>
                            <range>
                                <min>0</min>
                                <max>15</max>
                            </range>
                        </validation>
                    </param>
            </params>
</command>
```

Figure 6

7.3.2 Chart Recorder

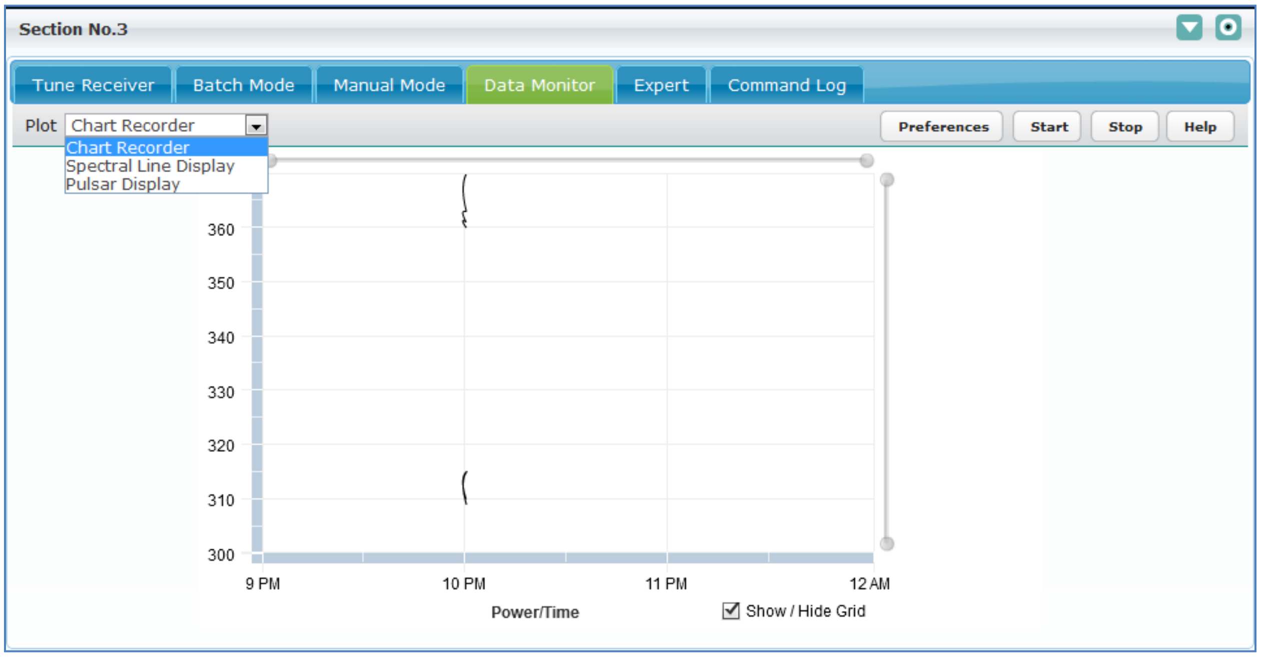


Figure 7

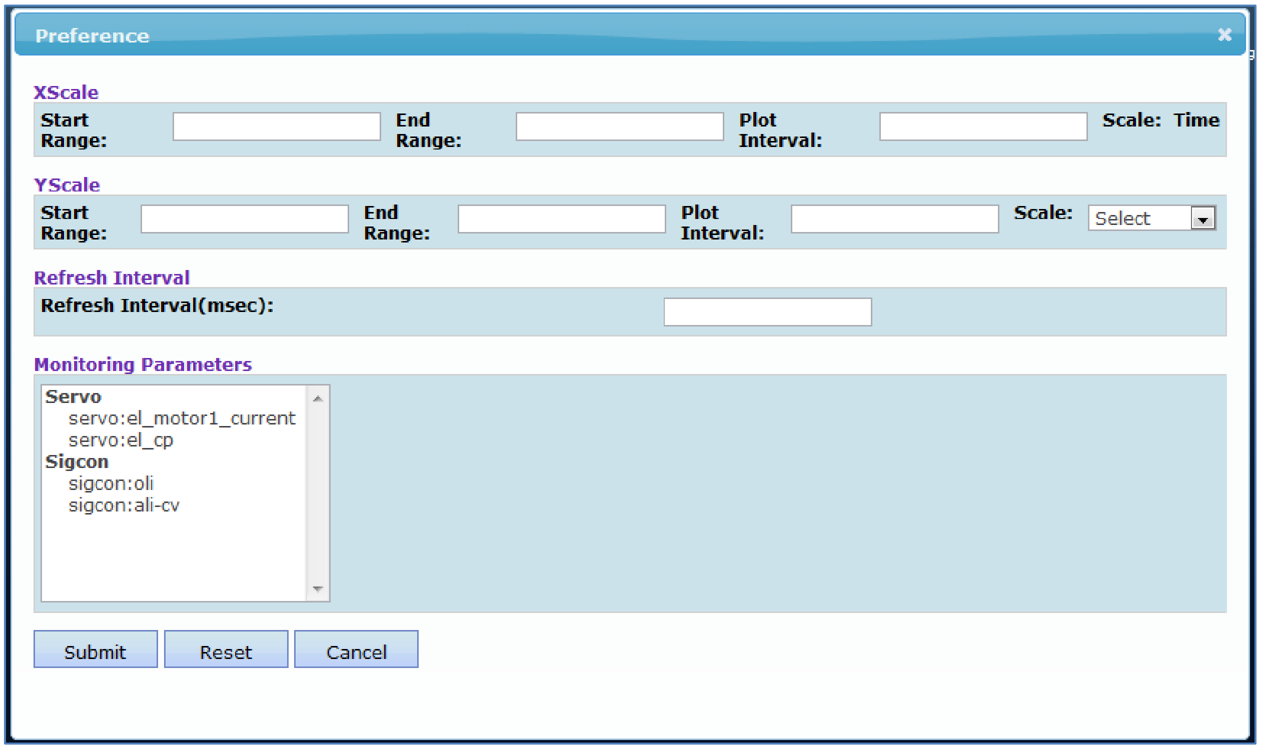
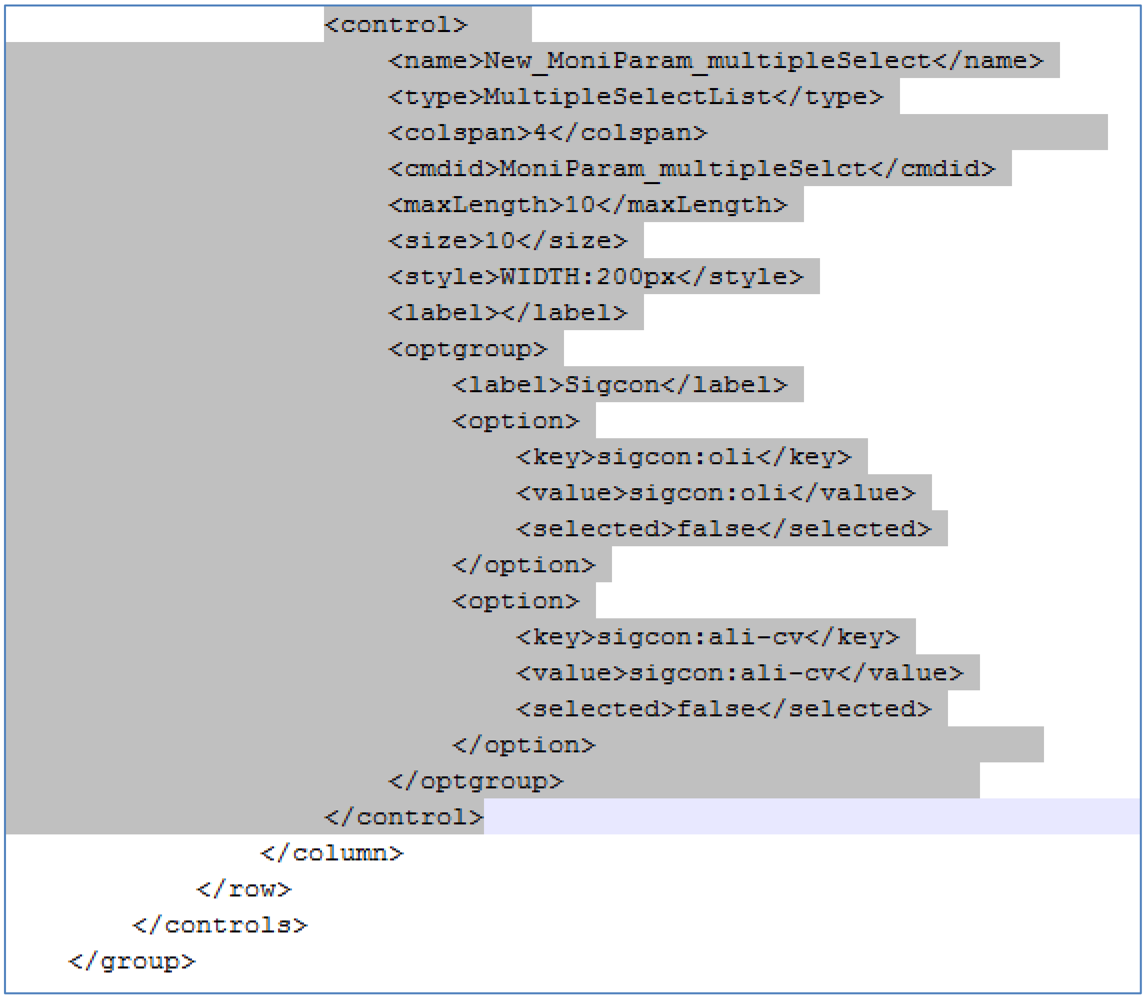


Figure 8

The chart recorder Preference screen shot is dynamically generated. The files which are used for generating chart recorder Preference dynamically is ChartRecorder.xml (template xml file) and ChartRecorder.xsl (xsl file).

User can add a new section similar to that of Monitoring Parameters or filter just by changing the template xml file. Steps for adding a new section named 'New Monitoring Parameters' are as follows.

1. Copy the complete 'controls' node of Monitoring Parameters from ChartRecoder.xml file shown in Figure 2.
2. Paste it just below the closing tag of 'controls' node of 'MoniParam_multipleSelect' as shown in Figure 9.

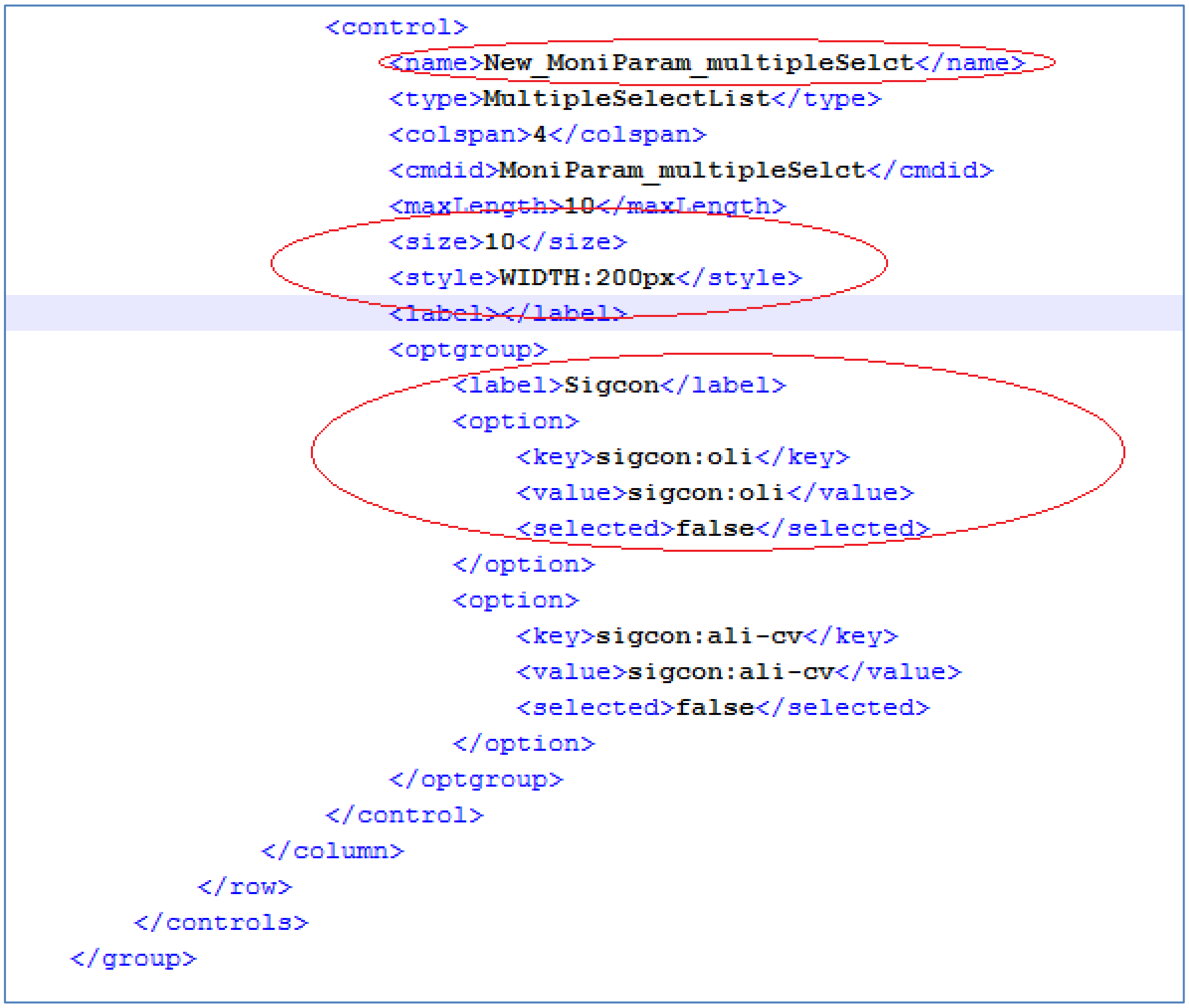


```
                    <control>
                        <name>New_MoniParam_multipleSelect</name>
                        <type>MultipleSelectList</type>
                        <colspan>4</colspan>
                        <cmdid>MoniParam_multipleSelct</cmdid>
                        <maxLength>10</maxLength>
                        <size>10</size>
                        <style>WIDTH:200px</style>
                        <label></label>
                        <optgroup>
                            <label>Sigcon</label>
                            <option>
                                <key>sigcon:oli</key>
                                <value>sigcon:oli</value>
                                <selected>false</selected>
                            </option>
                            <option>
                                <key>sigcon:ali-cv</key>
                                <value>sigcon:ali-cv</value>
                                <selected>false</selected>
                            </option>
                        </optgroup>
                    </control>
                </column>
            </row>
        </controls>
    </group>
```

Figure 9

3. Populate various values for newly added "New_MoniParam_multipleSelect", we need to add corresponding entry in ChartRecoder.xml file as well. For this just copy existing "<control>" node of 'MoniParam_multipleSelect' and change few attributes highlighted below in figure 10. Following are the nodes to be modified

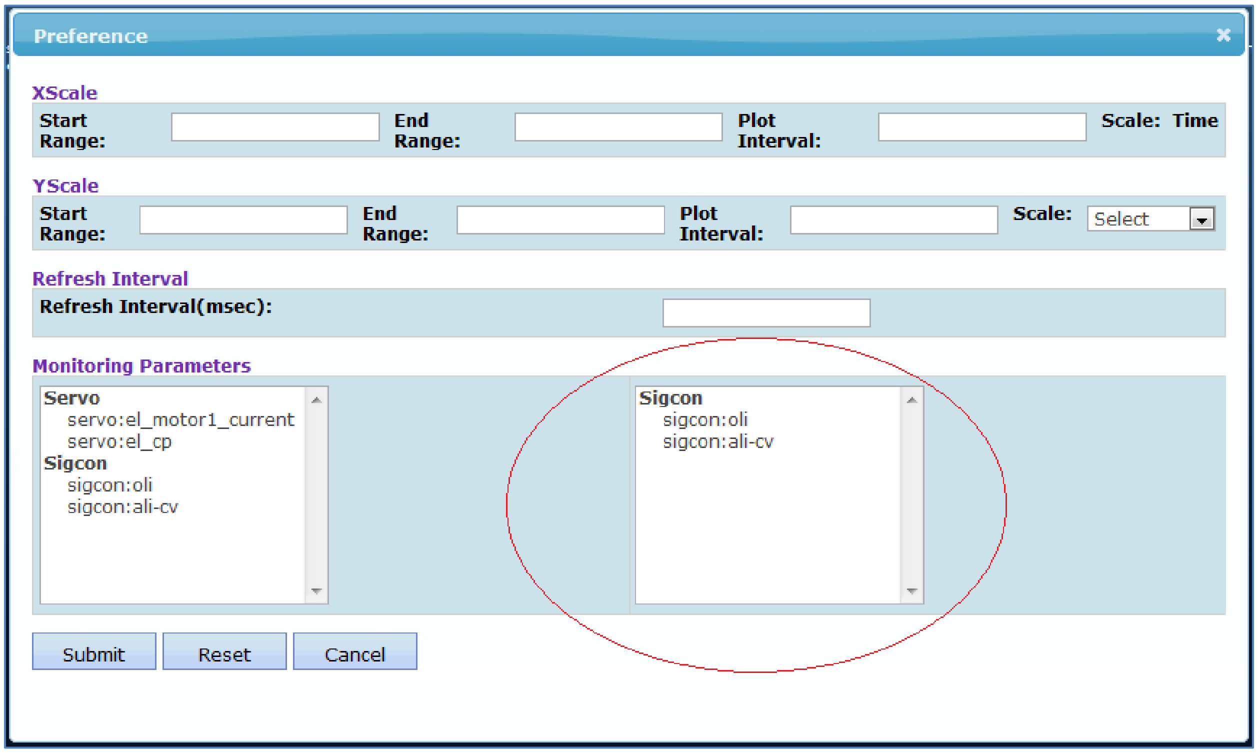
a. <Name> – Set name as any name like New_MoniParam_multipleSelect
b. <Size> - At a time how many parameters you want to display without scroll bar.
c. <Style> - Here user can set the width, height, background color etc.
d. <optgroup> - This very important section which contains multiple <options> where you can specify the monitoring parameters, here configuring two monitoring parameters, sigcon:oli, sigcon:ali-cv.



```
                    <control>
                        <name>New_MoniParam_multipleSelct</name>
                        <type>MultipleSelectList</type>
                        <colspan>4</colspan>
                        <cmdid>MoniParam_multipleSelct</cmdid>
                        <maxLength>10</maxLength>
                        <size>10</size>
                        <style>WIDTH:200px</style>
                        <label></label>
                        <optgroup>
                            <label>Sigcon</label>
                            <option>
                                <key>sigcon:oli</key>
                                <value>sigcon:oli</value>
                                <selected>false</selected>
                            </option>
                            <option>
                                <key>sigcon:ali-cv</key>
                                <value>sigcon:ali-cv</value>
                                <selected>false</selected>
                            </option>
                        </optgroup>
                    </control>
                </column>
            </row>
        </controls>
    </group>
```

Figure 10

4. With above changes ui looks like,



XScale

Start Range:		End Range:		Plot Interval:		Scale: Time

YScale

Start Range:		End Range:		Plot Interval:		Scale:	Select

Refresh Interval

Refresh Interval(msec):

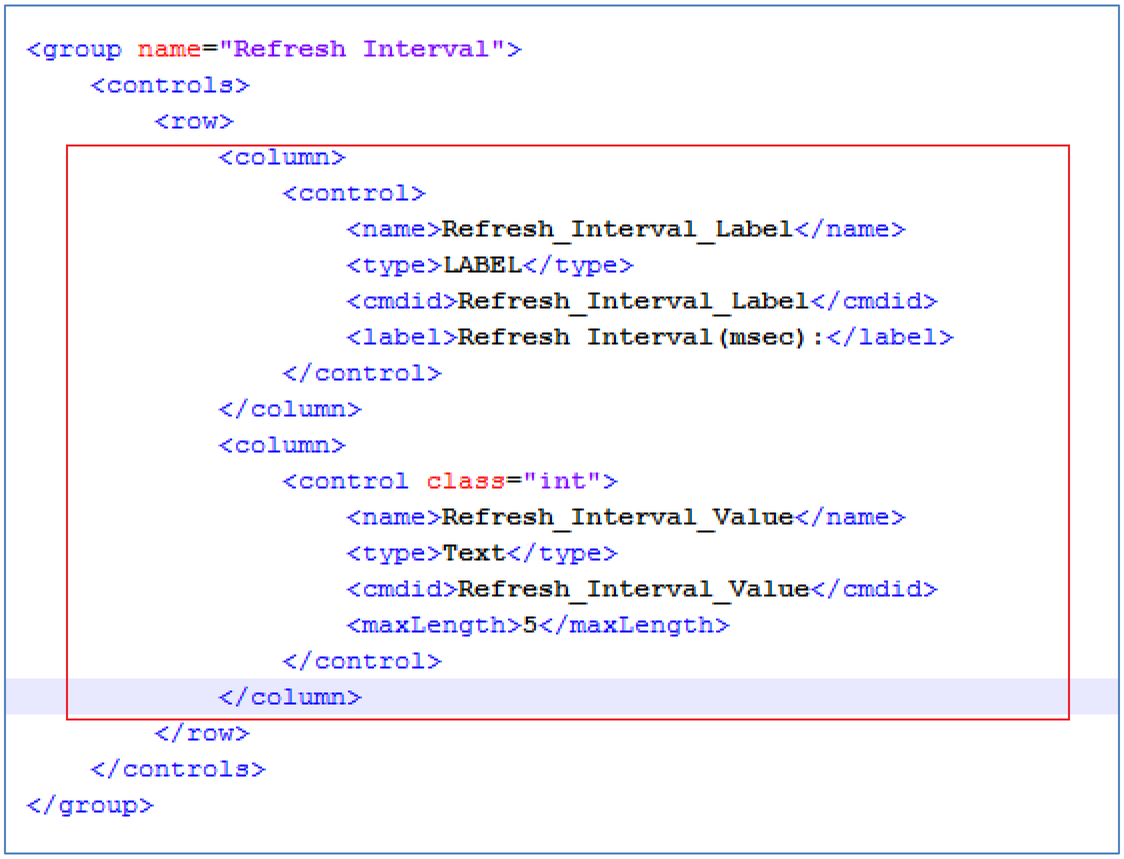
Monitoring Parameters

```
Servo
    servo:el_motor1_current
    servo:el_cp
Sigcon
    sigcon:oli
    sigcon:ali-cv
```

```
Sigcon
    sigcon:oli
    sigcon:ali-cv
```

Figure 11

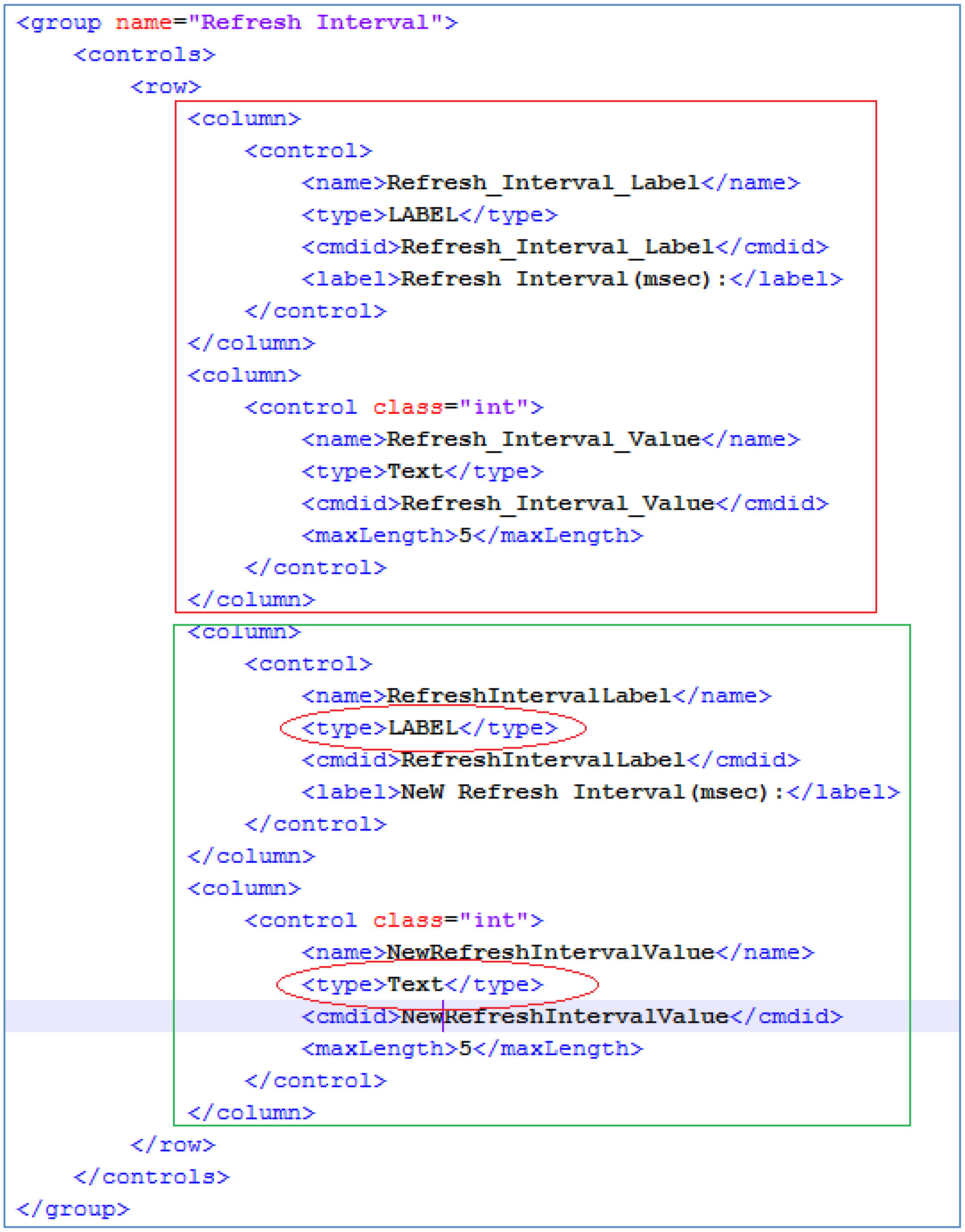
5. Now Add new label and Text box into existing UI
6. Open ChartRecoder.xml now look into refresh interval section as show in below



```
<group name="Refresh Interval">
    <controls>
        <row>
            <column>
                <control>
                    <name>Refresh_Interval_Label</name>
                    <type>LABEL</type>
                    <cmdid>Refresh_Interval_Label</cmdid>
                    <label>Refresh Interval(msec):</label>
                </control>
            </column>
            <column>
                <control class="int">
                    <name>Refresh_Interval_Value</name>
                    <type>Text</type>
                    <cmdid>Refresh_Interval_Value</cmdid>
                    <maxLength>5</maxLength>
                </control>
            </column>
        </row>
    </controls>
</group>
```

Figure 12

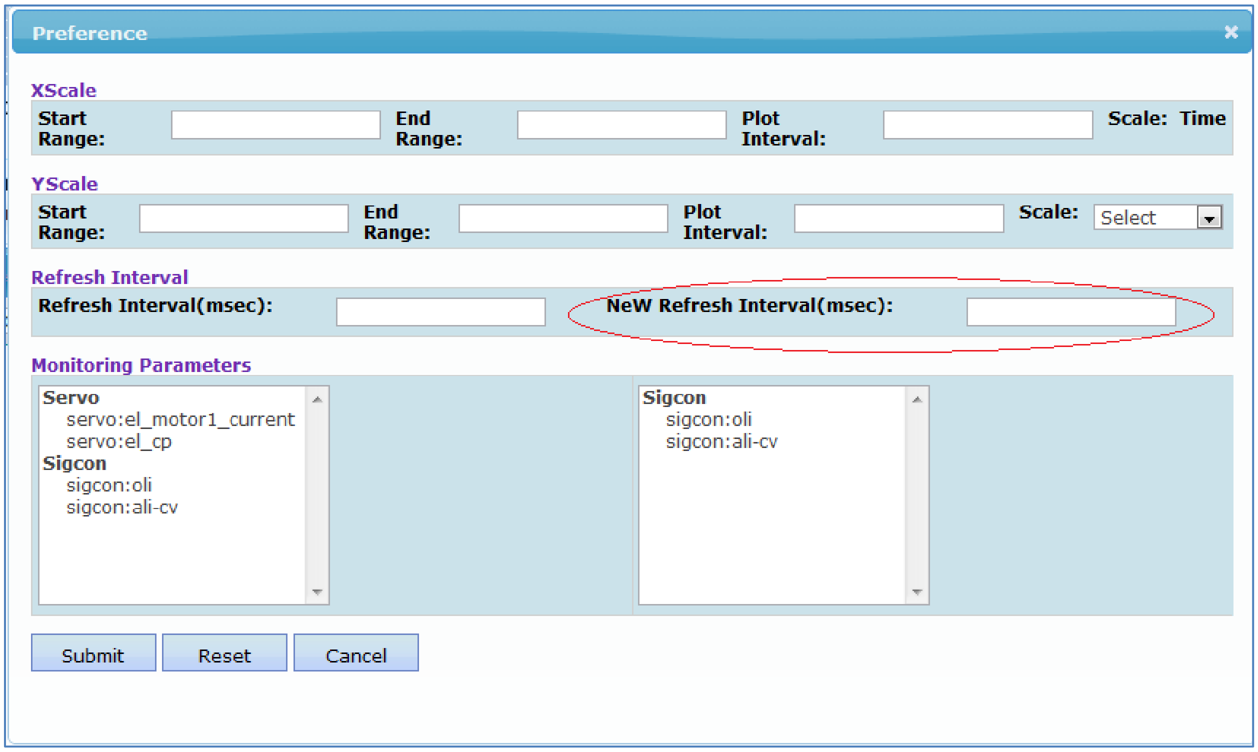
a. Copy above both <control> and pest it below it look like,



```
<group name="Refresh Interval">
    <controls>
        <row>
            <column>
                <control>
                    <name>Refresh_Interval_Label</name>
                    <type>LABEL</type>
                    <cmdid>Refresh_Interval_Label</cmdid>
                    <label>Refresh Interval(msec):</label>
                </control>
            </column>
            <column>
                <control class="int">
                    <name>Refresh_Interval_Value</name>
                    <type>Text</type>
                    <cmdid>Refresh_Interval_Value</cmdid>
                    <maxLength>5</maxLength>
                </control>
            </column>
            <column>
                <control>
                    <name>RefreshIntervalLabel</name>
                    <type>LABEL</type>
                    <cmdid>RefreshIntervalLabel</cmdid>
                    <label>NeW Refresh Interval(msec):</label>
                </control>
            </column>
            <column>
                <control class="int">
                    <name>NewRefreshIntervalValue</name>
                    <type>Text</type>
                    <cmdid>NewRefreshIntervalValue</cmdid>
                    <maxLength>5</maxLength>
                </control>
            </column>
        </row>
    </controls>
</group>
```

Figure 13

b. Now notice the first <control> type is LABEL and second control type is Text, change the attributes <name> and <cmdid> attributes.
c. Now update UI looks like as below,


Preference

XScale

Start Range:		End Range:		Plot Interval:		Scale: Time

YScale

Start Range:		End Range:		Plot Interval:		Scale:	Select

Refresh Interval

Refresh Interval(msec):		NeW Refresh Interval(msec):	

Monitoring Parameters

```
Servo
    servo:el_motor1_current
    servo:el_cp
Sigcon
    sigcon:oli
    sigcon:ali-cv
```

```
Sigcon
    sigcon:oli
    sigcon:ali-cv
```

Figure 14

<u>Note & Assumptions:</u>

- Existing ChartRecoder.xml contains 4 section as below,
  - XScale –
    - All the values Start Range, End Rand, Plot Interval, Scale name must be start with Xscale_
    - If dynamically add any component than please start <name> <cmdid> values must be start with Xscale_
    - <name> and <cmdid> nodes values are used in java script and other java implementation so don’t change these value if user is changing these node values than need to modify the code according.
  - YScale –
    - All the values Start Range, End Rand, Plot Interval, Scale name must be start with Yscale_
    - If dynamically add any component than please start <name> <cmdid> values must be start with Yscale_
    - <name> and <cmdid> nodes values are used in java script and other java implementation so don’t change these value if user is changing these node values than need to modify the code according.
  - Monitoring Parameters –
    - If dynamically add any component than please start <name> <cmdid> values must be start with MoniParam_

7.3.3 Spectral Line Display

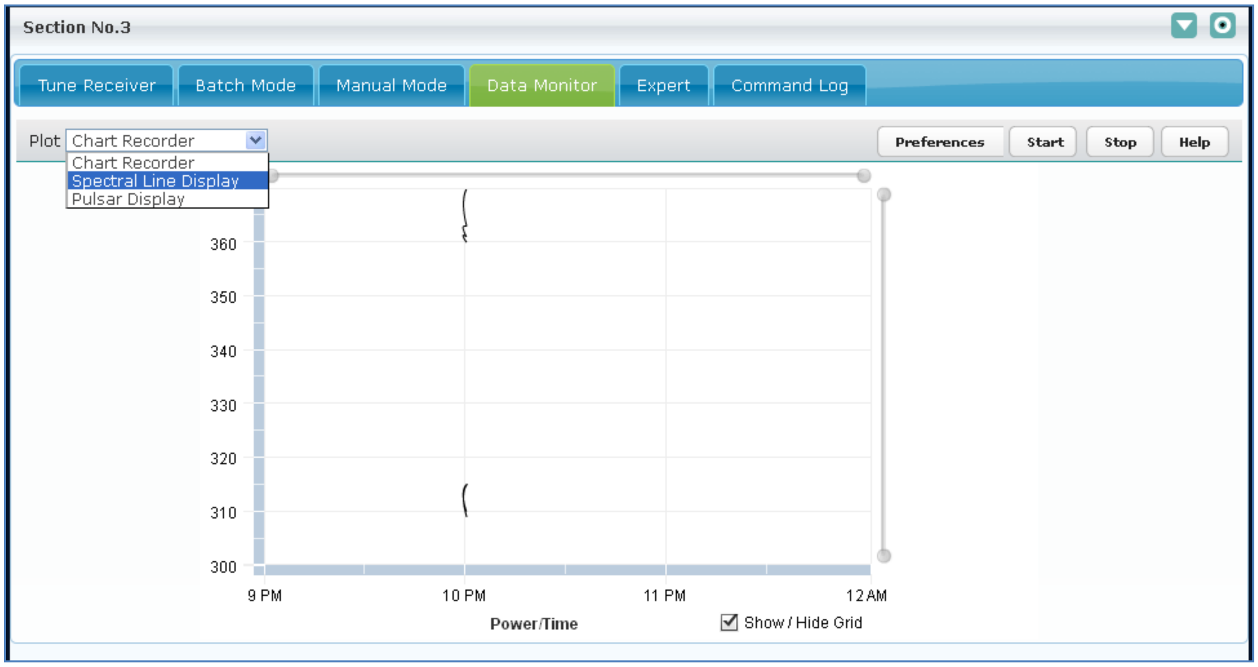
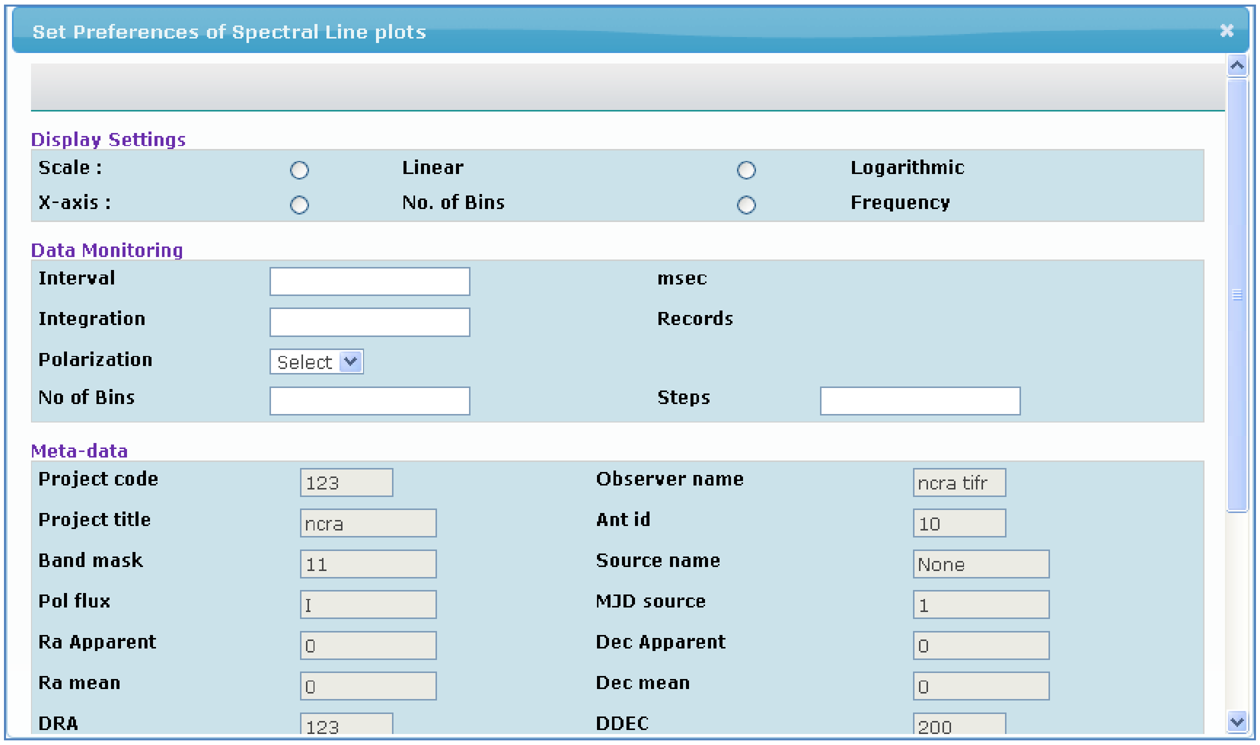


Figure 15



Display Settings

Scale :	○	Linear	○	Logarithmic
X-axis :	○	No. of Bins	○	Frequency

Data Monitoring

Interval		msec	
Integration		Records	
Polarization	Select		
No of Bins		Steps	

Meta-data

Project code	123	Observer name	ncra tifr
Project title	ncra	Ant id	10
Band mask	11	Source name	None
Pol flux	I	MJD source	1
Ra Apparent	0	Dec Apparent	0
Ra mean	0	Dec mean	0
DRA	123	DDEC	200

Figure 16

The spectral line display Preference screen shot is dynamically generated. The files which are used for generating chart recorder Preference dynamically is SpectralLineDisplay.xml (template xml file) and SpectralLineDisplay.xsl (xsl file).

User can add new parameters by adding the controls in template xml and making the entry for the same parameter in backend-commands.xml.

1. Steps to add a control say radio button name "**new_scale**" with values say "**new_linear**" and "**new_logarithmic**" in "Display Settings" section of template.xml.

   Example:
   User wants to add two radio controls.
   Following conventions must be followed while setting the attributes of radio button.

   a. **<type>:** Type will be "**radio**"

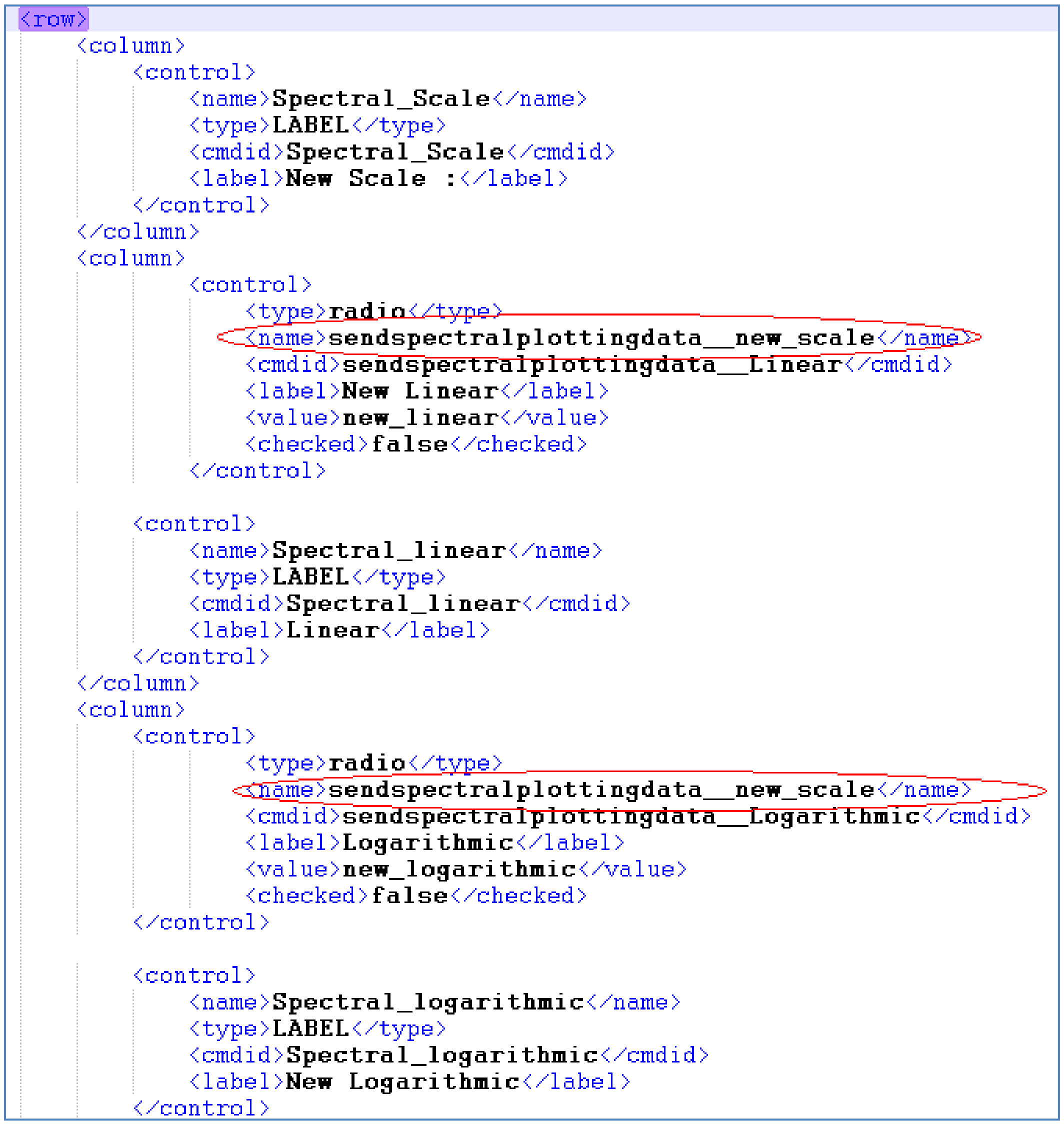
   b. **<name>** : Name of all radio buttons in a single radio group must be same. In this example we have added two new radio buttons. Name of radio button must start with "**sendspectralplottingdata__**" followed by its name.
   Example : sendspectralplottingdata__new_scale

   c. **<cmdid>:** Cmdid of the control must start with "**sendspectralplottingdata __**" followed by its name.
   Example : sendspectralplottingdata__Linear

   d. **<label> :** Label can be any valid name.
   Example : New Linear

   e. **<value>:** Value will be the one which user needs to send to backend system when that particular radio button will be selected.
   Example : new_linear

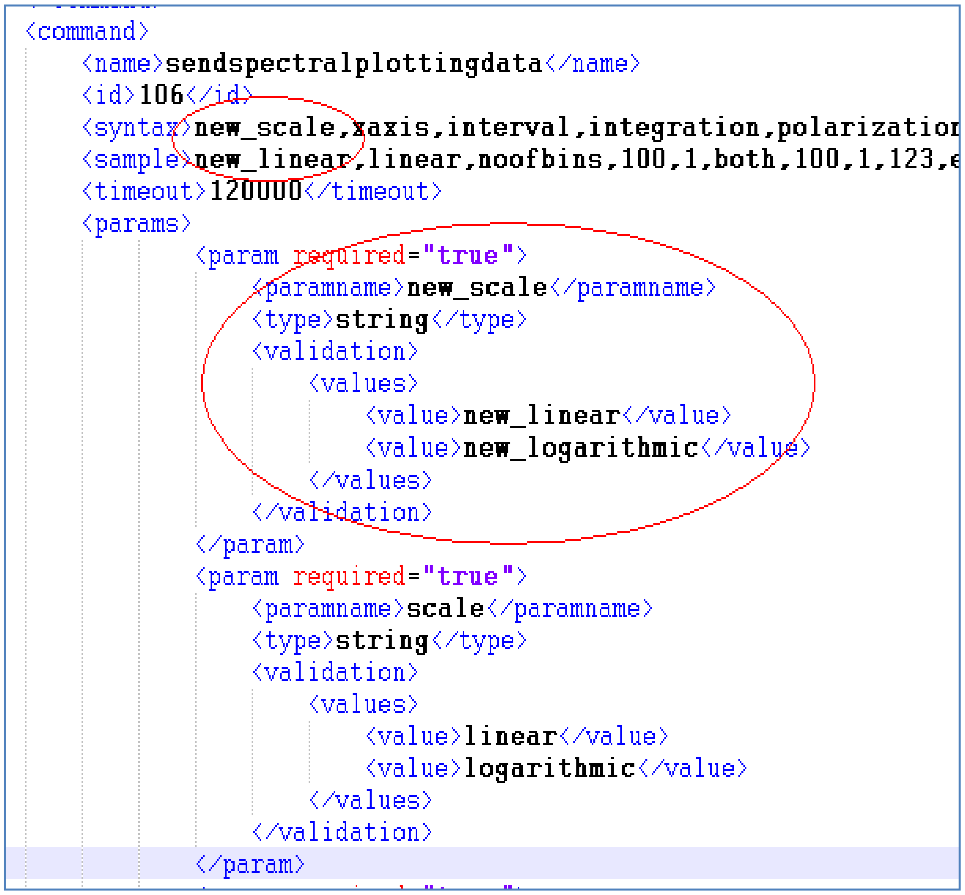
   f. **<checked>:** true/false depending upon user wants the radio button to be by default checked/unchecked.



```
<row>
    <column>
        <control>
            <name>Spectral_Scale</name>
            <type>LABEL</type>
            <cmdid>Spectral_Scale</cmdid>
            <label>New Scale :</label>
        </control>
    </column>
    <column>
            <control>
                <type>radio</type>
                <name>sendspectralplottingdata__new_scale</name>
                <cmdid>sendspectralplottingdata__Linear</cmdid>
                <label>New Linear</label>
                <value>new_linear</value>
                <checked>false</checked>
            </control>

        <control>
            <name>Spectral_linear</name>
            <type>LABEL</type>
            <cmdid>Spectral_linear</cmdid>
            <label>Linear</label>
        </control>
    </column>
    <column>
        <control>
                <type>radio</type>
                <name>sendspectralplottingdata__new_scale</name>
                <cmdid>sendspectralplottingdata__Logarithmic</cmdid>
                <label>Logarithmic</label>
                <value>new_logarithmic</value>
                <checked>false</checked>
        </control>

        <control>
            <name>Spectral_logarithmic</name>
            <type>LABEL</type>
            <cmdid>Spectral_logarithmic</cmdid>
            <label>New Logarithmic</label>
        </control>
```

Figure 17

2. User also needs to make entry of the added radio control in “**backend_commands.xml**” in “**sendspectralplottingdata**” command in following way.

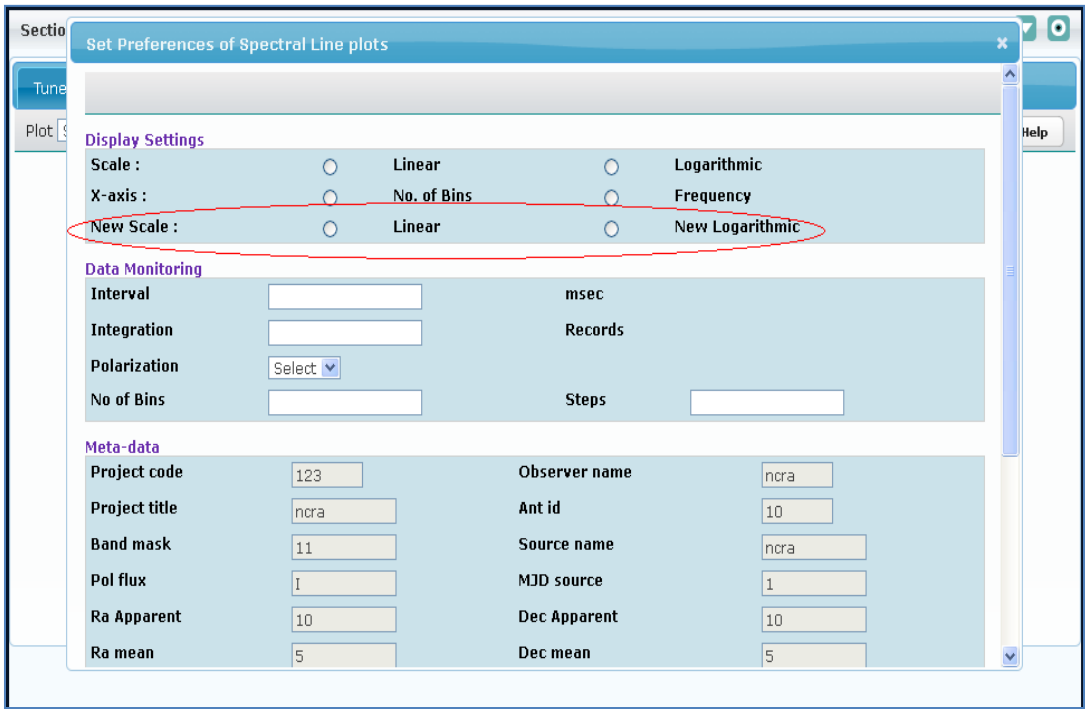

```
<command>
    <name>sendspectralplottingdata</name>
    <id>106</id>
    <syntax>new_scale,xaxis,interval,integration,polarizatio
    <sample>new_linear,linear,noofbins,100,1,both,100,1,123,e
    <timeout>120000</timeout>
    <params>
            <param required="true">
                <paramname>new_scale</paramname>
                <type>string</type>
                <validation>
                    <values>
                        <value>new_linear</value>
                        <value>new_logarithmic</value>
                    </values>
                </validation>
            </param>
            <param required="true">
                <paramname>scale</paramname>
                <type>string</type>
                <validation>
                    <values>
                        <value>linear</value>
                        <value>logarithmic</value>
                    </values>
                </validation>
            </param>
```
Figure 18

a. **<syntax>** : When the radio control was added ,its name was "**sendspectralplottingdata__new_scale**" (refer above).
Here the name of the parameter will be only "**new_scale**" i.e. "sendspectralplottingdata__" must be removed and the parameter must be added in the syntax.

b. **<param>:** Add a param tag with values as mentioned in "value" of radio buttons.
In this case they will be "new_linear" and "new_logarithmic".

c. **<sample>:** Add one of the values as sample value.
In this case it will be "new_linear".

User will be able to see the new radio control added on the preferences dialog of Spectral line display.



Set Preferences of Spectral Line plots

Display Settings

Scale :	○ Linear	○ Logarithmic
X-axis :	○ No. of Bins	○ Frequency
New Scale :	○ Linear	○ New Logarithmic

Data Monitoring

Interval		msec	
Integration		Records	
Polarization	Select		
No of Bins		Steps	

Meta-data

Project code	123	Observer name	ncra
Project title	ncra	Ant id	10
Band mask	11	Source name	ncra
Pol flux	I	MJD source	1
Ra Apparent	10	Dec Apparent	10
Ra mean	5	Dec mean	5

Figure 19

Note & Assumptions:

- Existing SpectralLineDisplay.xml contains 3 sections Display Settings, Data Monitoring and Meta data section.
- Names of all the input fields must start with "**sendspectralplottingdata__**"
- <name> and <cmdid> nodes values are used in java script and other java implementation so don't change these value .If user is changing these node values than need to modify the code according.
- Meta-data section fields are all read-only and they correspond to Global parameter" bean data. User can change this data through "Expert " or "Tune Receiver"

7.3.4 Pulsar Display

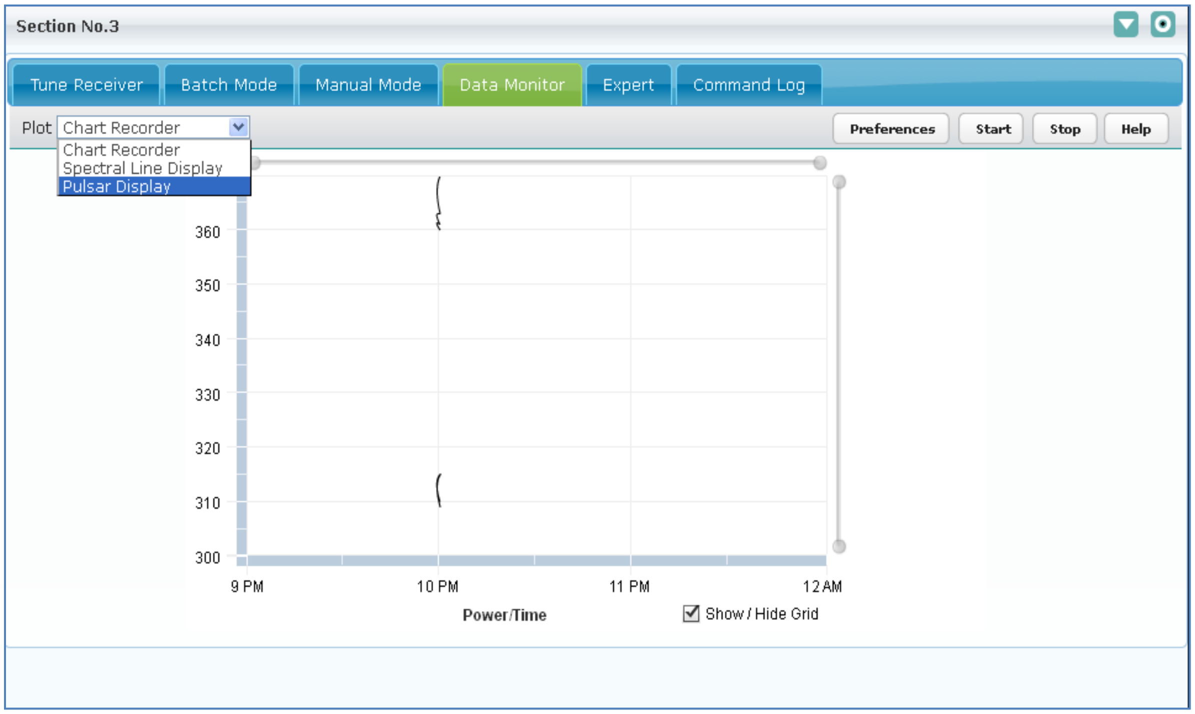
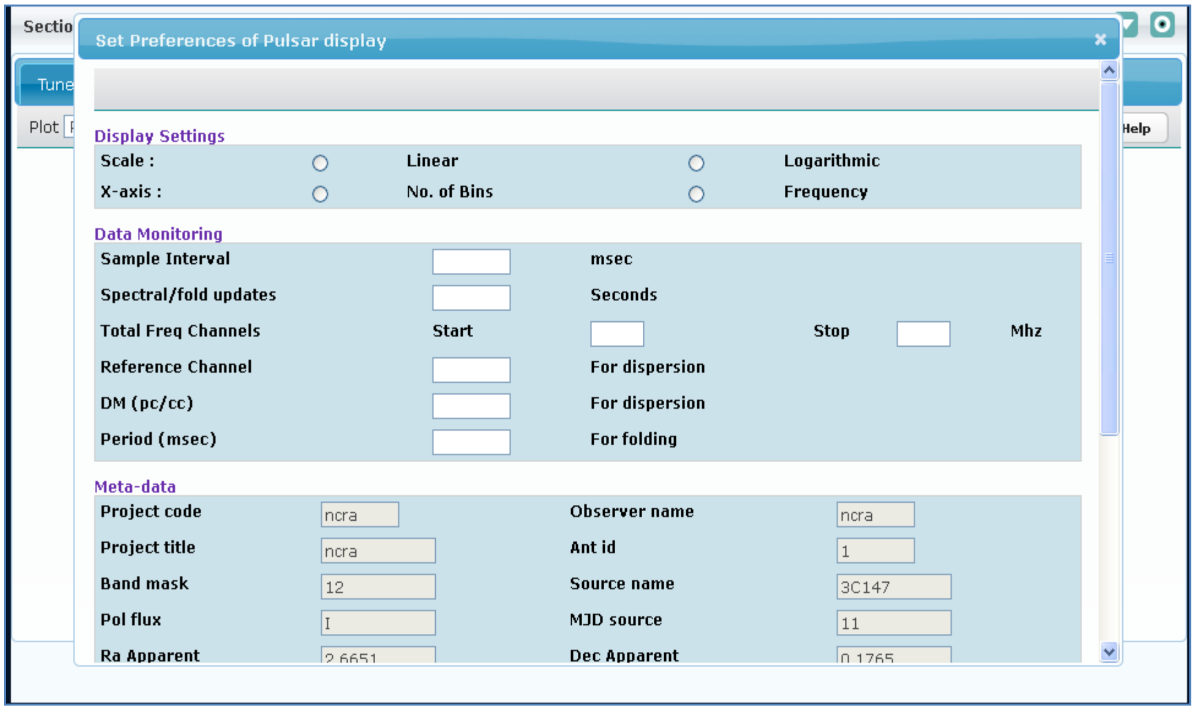


Figure 20



Display Settings

Scale :	○	Linear	○	Logarithmic
X-axis :	○	No. of Bins	○	Frequency

Data Monitoring

Sample Interval		msec			
Spectral/fold updates		Seconds			
Total Freq Channels	Start		Stop		Mhz
Reference Channel		For dispersion			
DM (pc/cc)		For dispersion			
Period (msec)		For folding			

Meta-data

Project code	ncra	Observer name	ncra
Project title	ncra	Ant id	1
Band mask	12	Source name	3C147
Pol flux	I	MJD source	11
Ra Apparent	2.6651	Dec Apparent	0.1765

Figure 21

The pulsar display Preferences dialog is dynamically generated. The files which are used for generating chart recorder Preference dynamically is PulsarDisplay.xml (template xml file) and PulsarDisplay.xsl (xsl file).

User can add new parameters by adding the controls in template xml and making the entry for the same parameter in backend-commands.xml.

1. Steps to add a control say drop down name "**sample**" with dropdown values say "**sample1**" and "**sample2** in "Display Settings" section of template.xml .

   Example:
   Following conventions must be followed while setting the attributes of dropdown.

   a. **<type> :** Type will be "**Dropdown**"
   b. **<name>** : Name of the control must start with "**sendpulsarplottingdata__"** followed by its name.

      Example : sendpulsarplottingdata__sample

   c. **<cmdid>** : Cmdid of the control must start with "**sendpulsarplottingdata__"** followed by its name.
      Example : sendpulsarplottingdata__sample

   d. **<colspan>** : If required colspan can be specified for that particular column
      Example: <colspan>2</colspan>

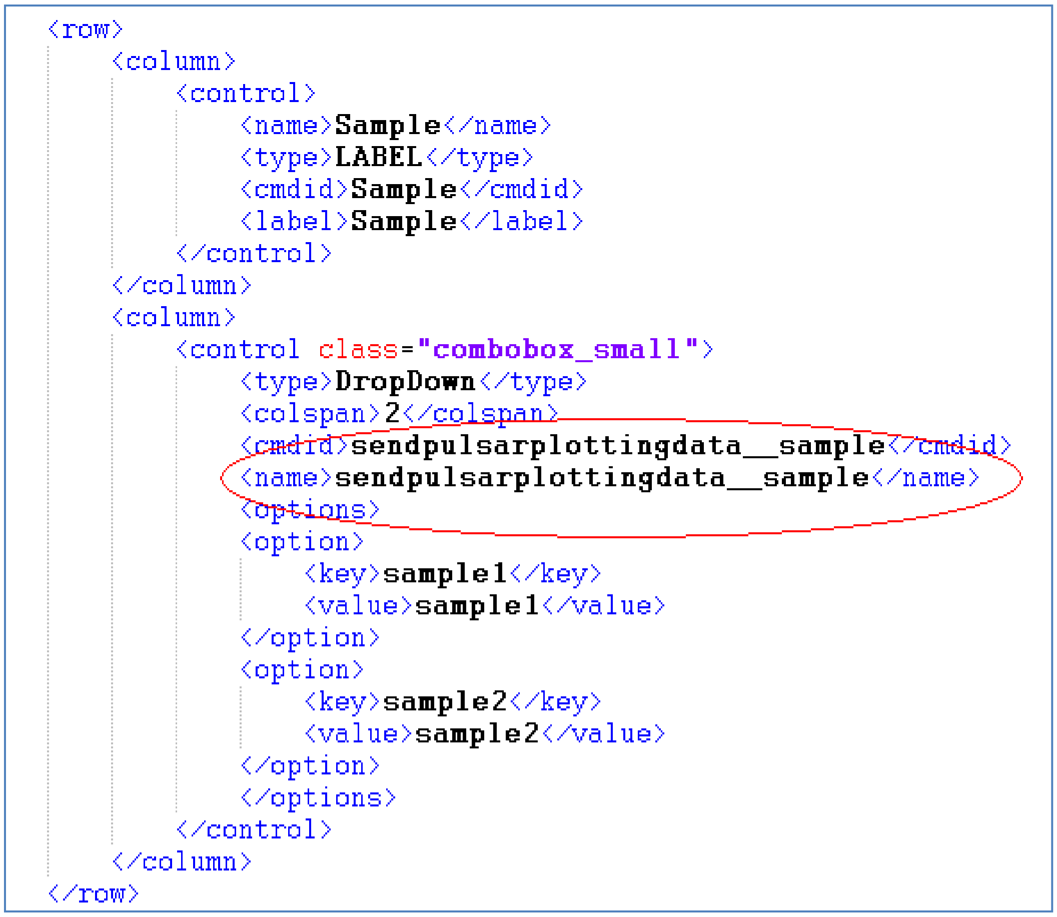
   e. **<options>** : Specifies the options that will be shown in the dropdown.

   f. **<option>** : User needs to specify the key and value for each option in option tag.

   g. **<key>** : Key is the one that is displayed to user on the dropdown.

   h. **<value>** : Value is the one that is sent to backend after a particular option in dropdown is selected.
      Example :

```
<options>
        <option>
                <key>sample1</key>
                <value>sample1</value>
        </option>
</options>
```



```
<row>
    <column>
        <control>
            <name>Sample</name>
            <type>LABEL</type>
            <cmdid>Sample</cmdid>
            <label>Sample</label>
        </control>
    </column>
    <column>
        <control class="combobox_small">
            <type>DropDown</type>
            <colspan>2</colspan>
            <cmdid>sendpulsarplottingdata__sample</cmdid>
            <name>sendpulsarplottingdata__sample</name>
            <options>
            <option>
                <key>sample1</key>
                <value>sample1</value>
            </option>
            <option>
                <key>sample2</key>
                <value>sample2</value>
            </option>
            </options>
        </control>
    </column>
</row>
```

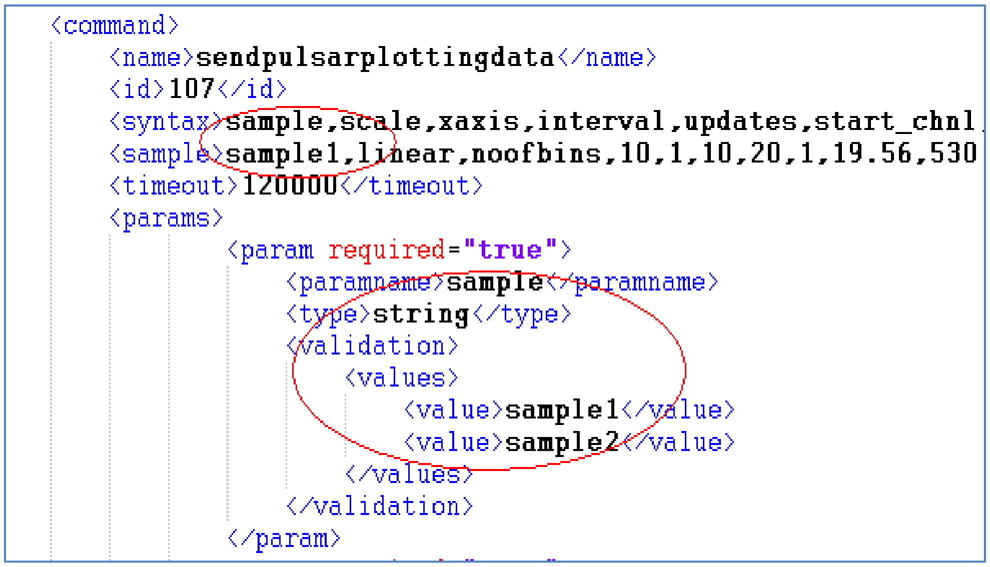
Figure 22

2. User also needs to make entry of the added dropdown control in "**backend_commands.xml**" in "**sendpulsarplottingdata**" command in following way.

   a. **<syntax>** : When the dropdown control was added ,its name was "**sendpulsarplottingdata __sample**" (refer above).
   Here the name of the parameter will be only "**sample**" i.e. "sendpulsarplottingdata__" must be removed and the parameter must be added in the syntax.

   b. **<param>:** Add a param tag with values as mentioned in "value" of dropdown options. In this case they will be "sample1" and "sample2".

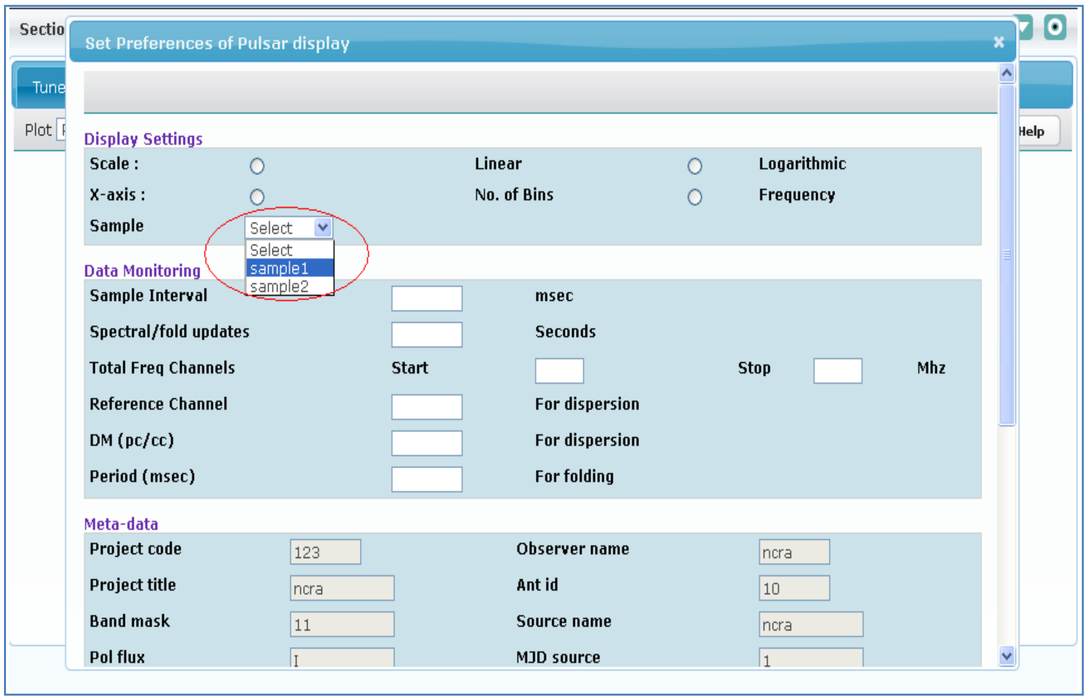
   c. **<sample>:** Add one of the values as sample value. In this case it will be "sample1".



```
<command>
    <name>sendpulsarplottingdata</name>
    <id>107</id>
    <syntax>sample,scale,xaxis,interval,updates,start_chnl
    <sample>sample1,linear,noofbins,10,1,10,20,1,19.56,530
    <timeout>120000</timeout>
    <params>
            <param required="true">
                <paramname>sample</paramname>
                <type>string</type>
                <validation>
                    <values>
                        <value>sample1</value>
                        <value>sample2</value>
                    </values>
                </validation>
            </param>
```

Figure 23

User will be able to see the new dropdown control added on the preferences dialog of Pulsar display.



Set Preferences of Pulsar display

Display Settings

Scale :	○	Linear	○	Logarithmic
X-axis :	○	No. of Bins	○	Frequency
Sample	Select (Select / sample1 / sample2)			

Data Monitoring

Sample Interval		msec		
Spectral/fold updates		Seconds		
Total Freq Channels	Start		Stop	Mhz
Reference Channel		For dispersion		
DM (pc/cc)		For dispersion		
Period (msec)		For folding		

Meta-data

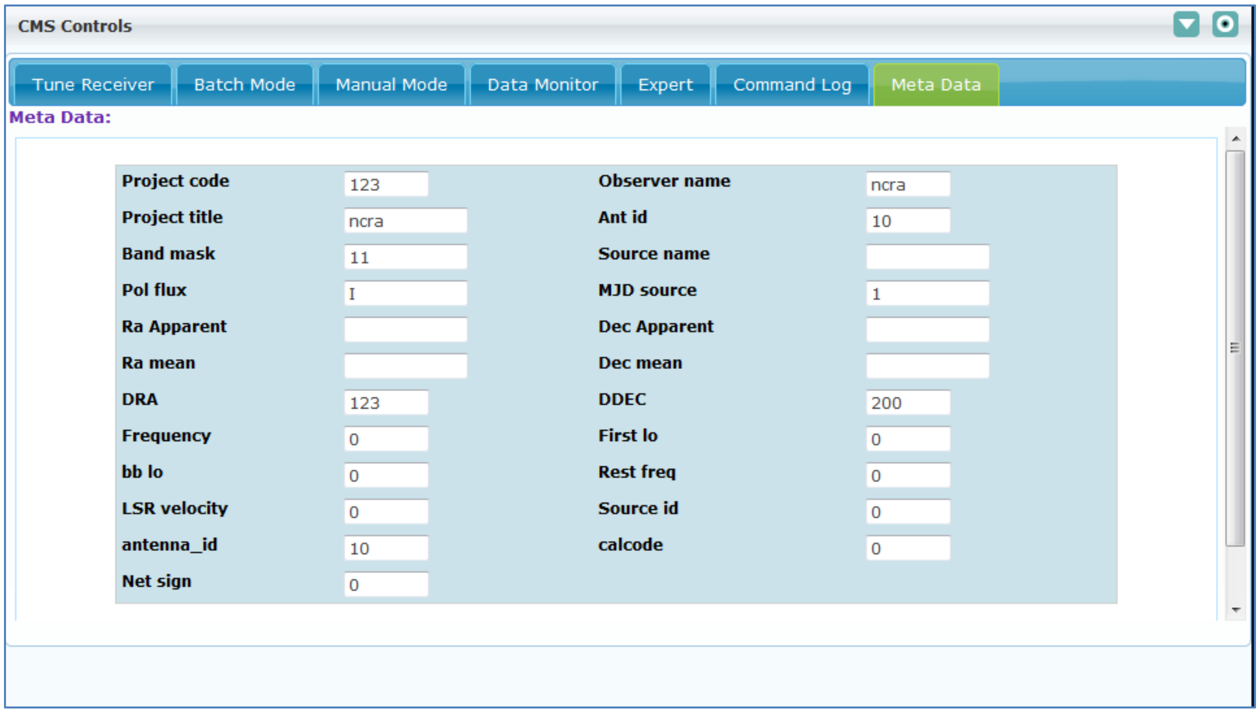
Project code	123	Observer name	ncra
Project title	ncra	Ant id	10
Band mask	11	Source name	ncra
Pol flux	I	MJD source	1

Figure 24

Note & Assumptions:

- Existing Pulsar.xml contains 3 sections Display Settings, Data Monitoring and Meta data section.
- Names of all the input fields must start with "**sendpulsarplottingdata** __".
- <name> and <cmdid> nodes values are used in java script and other java implementation so don't change these value .If user is changing these node values than need to modify the code according.
- Meta-data section fields are all read-only and they correspond to Global parameter" bean data. User can change this data through "Expert" or "Tune Receiver"

7.3.5 Meta Data Tab



Meta Data:

Project code	123	Observer name	ncra
Project title	ncra	Ant id	10
Band mask	11	Source name	
Pol flux	I	MJD source	1
Ra Apparent		Dec Apparent	
Ra mean		Dec mean	
DRA	123	DDEC	200
Frequency	0	First lo	0
bb lo	0	Rest freq	0
LSR velocity	0	Source id	0
antenna_id	10	calcode	0
Net sign	0		

Figure 25

The Meta Data tab is dynamically generated usingMetaData.xml (template xml file) and MetaData.xsl (xsl file).

If any new Meta data parameter is added in system, this file (MetaData.xml) should be modified accordingly.

1. Steps to add a control say drop down name “**sample**” with dropdown values say “**sample1**” and ”**sample2** in template.xml .

Example:

Following conventions must be followed while setting the attributes of dropdown.

a. **<type> :** Type will be “**Dropdown**”
b. **<name>** : Name of the control must start with “**sendpulsarplottingdata__”** followed by its name.
   Example : sendpulsarplottingdata__sample

c. **<cmdid>** : Cmdid of the control must start with “**sendpulsarplottingdata__”** followed by its name.
   Example : sendpulsarplottingdata__sample

d. **<colspan>** : If required colspan can be specified for that particular column
   Example: <colspan>2</colspan>

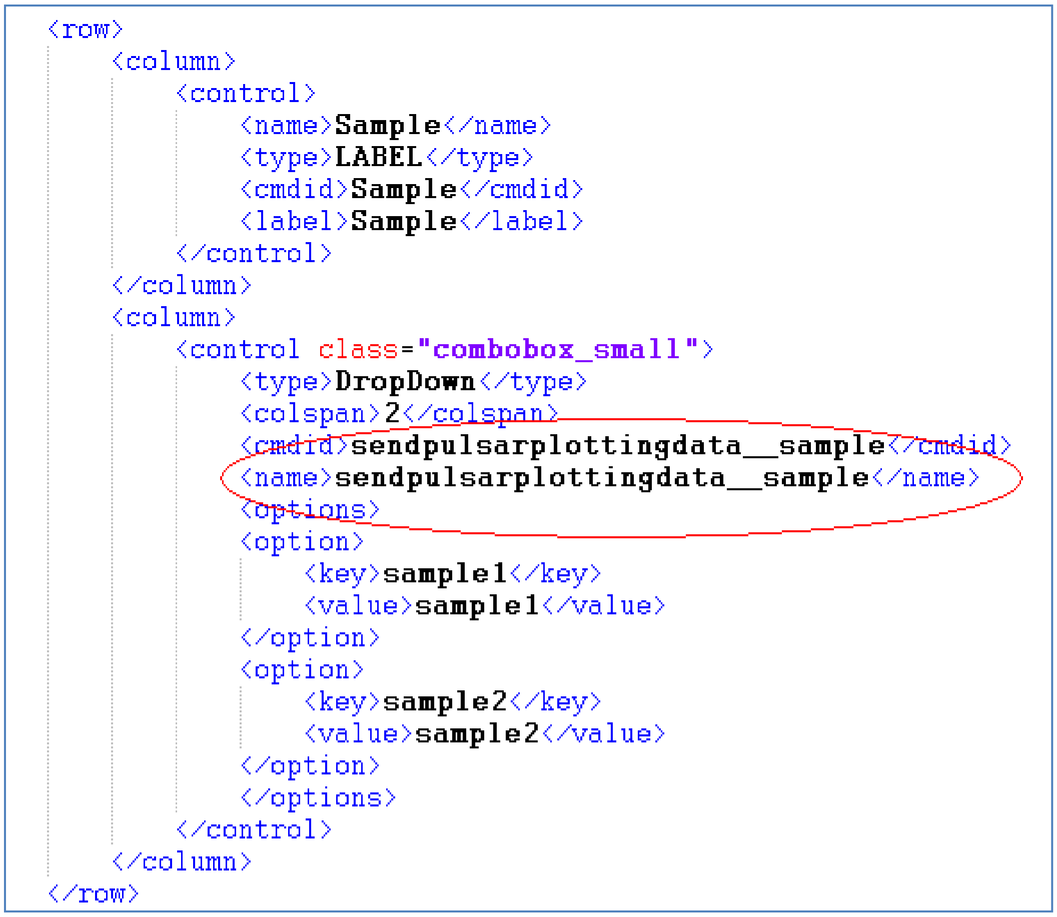
e. **<options>** : Specifies the options that will be shown in the dropdown.

f. **<option>** : User needs to specify the key and value for each option in option tag.

g. **<key>** : Key is the one that is displayed to user on the dropdown.

h. **<value>** : Value is the one that is sent to backend after a particular option in dropdown is selected.

```
Example : <options>
                <option>
                        <key>sample1</key>
                        <value>sample1</value>
                </option>
        </options>
```



```
<row>
    <column>
        <control>
            <name>Sample</name>
            <type>LABEL</type>
            <cmdid>Sample</cmdid>
            <label>Sample</label>
        </control>
    </column>
    <column>
        <control class="combobox_small">
            <type>DropDown</type>
            <colspan>2</colspan>
            <cmdid>sendpulsarplottingdata__sample</cmdid>
            <name>sendpulsarplottingdata__sample</name>
            <options>
            <option>
                <key>sample1</key>
                <value>sample1</value>
            </option>
            <option>
                <key>sample2</key>
                <value>sample2</value>
            </option>
            </options>
        </control>
    </column>
</row>
```

Figure 26

<u>Note & Assumptions:</u>

- Name and cmdid must start with "**sendpulsarplottingdata __**".
- Meta-data section fields are all read-only and they correspond to Global parameter" bean data. User can change this data through "Expert" or "Tune Receiver"

8 Batch Functionality

This section contains the details idea about Batch functionality for the CMS application.

8.1 Batch Script

8.1.1 Batch Script Validation

1. User should be able to upload and validate any batch script that supports .pl or .txt extension
2. In case of any invalid command or syntax error in command appropriate message should be displayed to user
3. The supported common cms commands supported in batch script are as follows:
    1. command – executes any valid subsystem command.
       Supported syntax is :
       command(“ <subsystem-name>,<command-name>,<command parameters(comma-seperated)>“);

    2. error - displays the error message on console.
       Supported syntax is :
       error(“<error-msg>”);

    3. info - displays the informational message on console.
       Supported syntax is :
       info(“<msg>”);

    4. getCommandStatus – returns the command status of a given command.
       Supported syntax is :
       getCommandStatus (“<seq-num of command>”);
       Status messages are provided are as below:
       TIMED-OUT - (indicating Command Timed-out)
       ERROR - (indicating Command Error)
       CONN-REFUSED - (indicating connection refused by wrapper)
       WRAPPER-ACK- (indicating wrapper acknowledge)
       WRAPPER-NAK- (indicating wrapper command refusal)
       ACK- (indicating subsystem acknowledgement)
       NAK- (indicating subsystem negative acknowledgement)
       SUCCESS- (indicating successful execution of command by subsystem)
       FAIL- (indicating failure of command at subsystem)
       QUEUED- (indicating command is queued)
       READY TO SEND- (indicating command is DE queued any ready to be sent to wrapper)
       In Progress- (indicating command execution is in progress)
       Unknown Status- (indicating command status is not known)

    5. getCommandStatusInt – returns the integer status of a given command.
       Supported syntax is :
       getCommandStatusInt (“<seq-num of command>”);

Integer Status of command are provided as below:
-1 - (Indicating Command Timed-out)
-2 - (Indicating Command Error)
-3 - (Indicating connection refused by wrapper)
1- (Indicating wrapper acknowledge)
2- (Indicating wrapper command refusal)
3- (Indicating subsystem acknowledgement)
4- (Indicating subsystem negative acknowledgement)
5- (Indicating successful execution of command by subsystem)
6- (Indicating failure of command at subsystem)
7-(Indicating subsystem intermediate response received)
9- (Indicating command is queued)
11- (Indicating command is DE queued any ready to be sent to wrapper)
0- (indicating command execution is in progress)

6. getParamValue – return the parameter value received from subsystem/wrapper as response.
   Supported syntax is :
   getParamValue (<seq-num of command>**,**”<param-name>”);

7. startTime – starts the script after specified time in hh:mm:sec
   Supported syntax is :
   startTime (“<hh:mm:ss>”);

8. stop – stops the batch script
   Supported syntax is :
   stop ();

9. waitforCmdCompletion - wait for command completion
   Supported syntax is :
   waitforCmdCompletion (“<seq-num>, <sleepTime[sec](optional)>, <time_out[sec](optional)>”);

10. restore – restore the last monitoring parameter value from database
    Supported syntax is :
    restore(“ <subsystem-name>,<param-name>);

11. getMonitoringParam – restore the recent monitoring parameter value
    Supported syntax is:
    getMonitoringParam (“ <subsystem-name>,<param-name>”);

12. gts – gets the catalog object
    Supported syntax is :
    gts(“ <source-name>“);

13. updateBatchStatus – updateBatchStatus updates the batch status. Valid integer values are 2(fail) and 5(Success).
    Supported syntax is: updateBatchStatus(<batchstatus>);

14. checkSubSystemStatus – check the sub system status, either it is active or de-active.
    Supported syntax is: checkSubSystemStatus ("<subsystem-name>");

4. The supported ncra specific commands supported in batch script are as follows:
    1. loadProperty – Updates the Global property value. The property has to be pre-configured in globalparameter.properties.
       Supported syntax is:
       loadProperty(" <property-name>,<property-value>");

       The below mentioned batch functions are for CMS internal use, so these are not available as commands in Expert Tab and Engineering GUI. These are implemented to be used only in a batch script. These functions if used inappropriately may change the CMS internal settings.

    2. getProperty – Gets the Global property value.

Supported syntax is:
getProperty("<property-name>");

3. updateTrackingAttributes – Updates the Tracking attributes such as source, ra_mean, dec_mean etc. on firing track command.

Supported syntax is:
updateTrackingAttributes();

4. clearTrackingAttributes – Clears the Tracking attributes. **This is called when the track parameter is received as false in servo monitoring data.**

Supported syntax is:

5. updateDataAcqProgress – Updates the data acq progress. The acq-percentage and

Supported syntax is:
updateDataAcqProgress("<acqPercentage>,<acqStatusMessage>");

6. updateDataAcqHeaderStatus – Updates the data acq progress in CMS Header.

Supported syntax is:
updateDataAcqHeaderStatus("<acqStatus>");

7. enableStartDataAcq– Enables the Start Data Acquisition button on Manual mode tab. And disables the Stop Data Acquisition button.

Supported syntax is:
enableStartDataAcq ();

8. disableStartDataAcq– Disables the Start Data Acquisition button on Manual mode tab. And enables the Stop Data Acquisition button.

Supported syntax is:
disableStartDataAcq ();

updateBandCenterFrequency – Updates the setup frequency values in header in section 1.

Supported syntax is:
updateBandCenterFrequency('<arg>')

Following are valid values for the 'arg'

0 –	Command to set frequency executed successfully
2 –	Error in command while setting frequency values

5. Sleep supports the Functions specified in Sleep Manual

   Note: The variables when specified in a batch script should be preceded or succeeded by a white space. For e.g if $x is to be used in a function info it should be used as follows:
   Info(" x value = $x ");

8.1.2 Batch Script Execution

1. User should be able to upload and run any batch script that supports .pl or .txt extension
2. In case of any invalid command or syntax error in command appropriate message should be displayed to user.
3. The user should be able to stop batch command at any point of time. The batch will eventually stop after the current command gets executed completely. This may take time depending upon the time taken by the last command to execute.
4. Once started, the start button would be disable and stop button would be enabled

   On Stop, the stop button will be disabled and start button will be enabled

8.2 Batch Commands

8.2.1 Batch Command Validation

1. User should be able to specify a batch command in the subsystem commands.xml file. For e.g for servo subsystem the batch command can be specified in servo_commands.xml in the following format:

```
<batchcommand>
            <name>batch1(init-position)</name>
            <id>412</id>
            <syntax>z</syntax>
            <sample>2</sample>
            <params>
                  <param required="true">
                        <paramname>z</paramname>
                        <type>integer</type>
                  </param>
            </params>
            <filepath>E:\\batch1.txt</filepath>
</batchcommand>
```

2. If user makes a mistake in the xml syntax the application should display an error message in the log file mentioning the cause.
3. If the batch file path is incorrect, then on batch command execution the error message of "Empty or missing file path or script tag should be displayed" or similar such error message should be shown to user
4. The batch command is validated similar to the normal command, i.e. the syntax of the input parameters is checked, and the associated batch script validation is done only during batch command execution. So the user is expected to test the batch script using batch console before adding it as a batch command.

   The batch command can be inserted in a batch script in the similar way as the normal command.

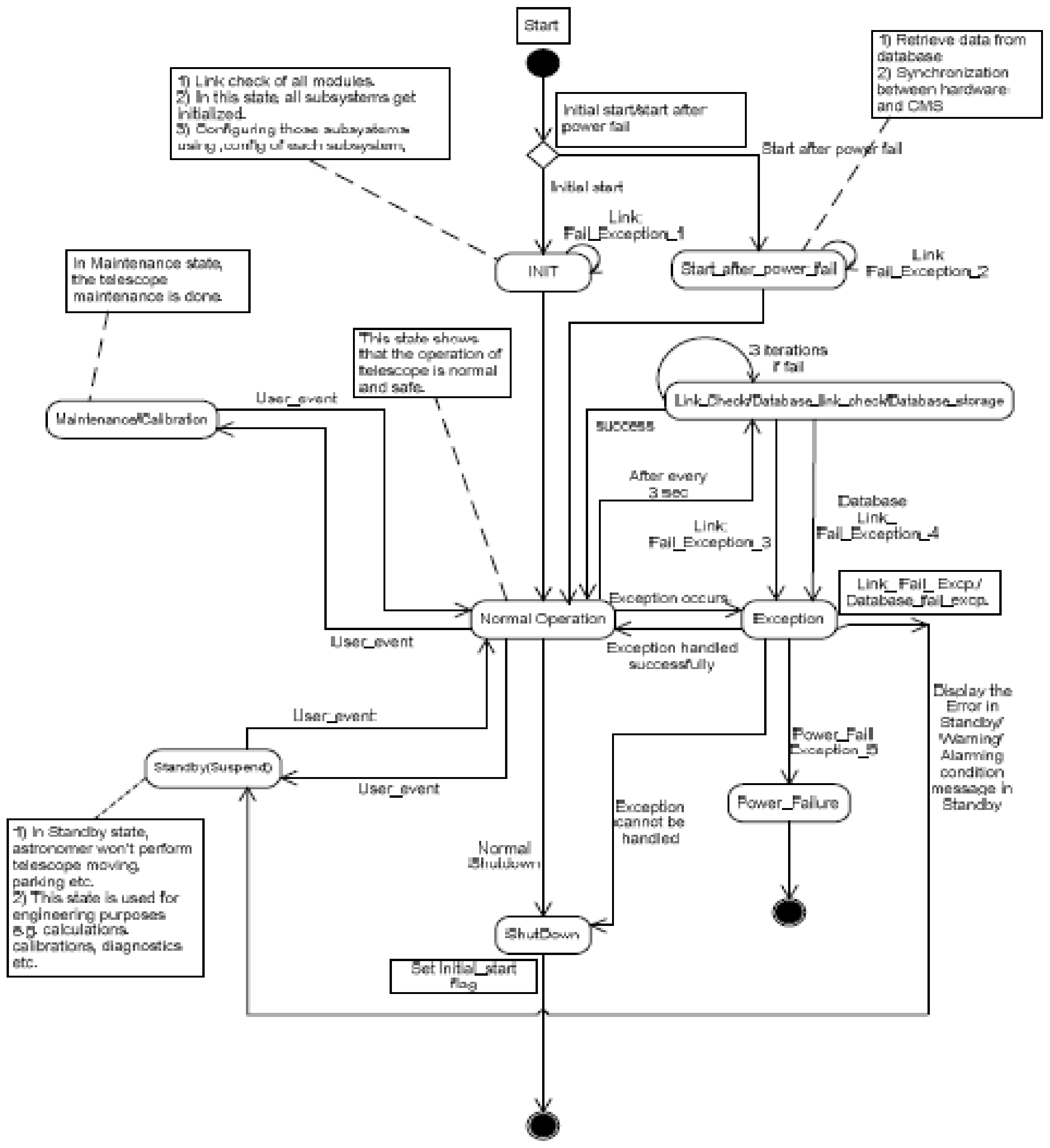
8.2.2 Batch Command Execution

1. The batch command can be executed in following ways:
   a. Through expert tab where it appears like a normal command
   b. By adding it to the batch script
2. The batch execution status can be viewed in message console when the batch command is fired from Expert Tab
3. The batch execution can be viewed in batch console when the batch command is fired from a batch script

9 State Machine

This section describes the component and features of the State Machine and provides the guide line for the rule configuration.

9.1 Control diagram



Start
1) Link check of all modules.
2) In this state, all subsystems get initialized.
3) Configuring those subsystems using .config of each subsystem.
Initial start/start after power fail
1) Retrieve data from database
2) Synchronization between hardware and CMS
Start after power fail
Initial start
Link Fail_Exception_1
INIT
Start_after_power_fail
Link Fail_Exception_2
In Maintenance state, the telescope maintenance is done.
This state shows that the operation of telescope is normal and safe.
3 iterations if fail
Link_Check/Database_link_check/Database_storage
success
User_event
Maintenance/Calibration
After every 3 sec
Link Fail_Exception_3
Database Link_ Fail_Exception_4
Link_ Fail_ Excp./ Database_fail_excp.
Exception occurs
Normal Operation
Exception
User_event
Exception handled successfully
Display the Error in Standby/ Warning/ Alarming condition message in Standby
User_event
Power_Fail Exception_5
Standby(Suspend)
User_event
Power_Failure
1) In Standby state, astronomer won't perform telescope moving, parking etc.
2) This state is used for engineering purposes e.g. calculations, calibrations, diagnostics etc.
Exception cannot be handled
Normal Shutdown
ShutDown
Set Initial_start flag

9.2 States

CMS will have the following states

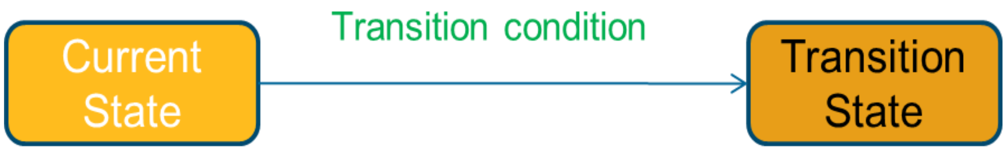
1. **START** – this state is the starting point for state-machine
2. **INIT** – CMS will enter into init state from START state. In this state CMS will do connectivity check for all subsystems. On successful connections CMS will run an init batch script. This script will include the initialization commands. On successful execution of init script, CMS will move Normal State otherwise move in to SUSPEND state
3. **NORMAL** – In this state the normal CMS operations can proceed.
4. **DRIFT** – This is an NCRA specific state where the servo subsystem would be assumed to be down and no command would be allowed to be sent to SERVO. The Engineer/Expert would have to manually change the CMS state to DRIFT mode.
5. **MAINTENANCE** - This state can be achieved through user driven operation in CMS.
6. **STANDBY** – This state can be achieved through user driven operation in CMS.
7. **EXCEPTION** – CMS will enter this state when there is an exception.
8. **INIT_ON_POWERFAILURE** – CMS will enter this state from START state when the CMS comes up after power failure or when CMS has not been shutdown properly
9. **SUSPEND** - CMS will enter this state if any high level alarm is raised by system.
10. **SHUTDOWN** – This state can be achieved through user driven operation in CMS.

9.3 Transitions & Actions

9.3.1 Transitions

Transitions indicate the steps involved for moving from one state to another.

To transition from one state to another the condition to enter the particular state needs to be defined. State transition occurs upon positive evaluation of condition.



9.3.2 Actions:

Actions are tasks done on entering a state.

1. When CMS enters a state it can perform some actions associated with that state. For e.g. in INIT state the initialization scripts need to be executed.

2. When a rule condition evaluates to true, the corresponding action specified by the rule will be executed. The details of conditions and actions supported in Rules-engine are specified in the Rules section

9.4 Rules

This section contains the details idea about rule configurations for the alarms management.

9.4.1 Configuring Rules

The Rules can be configured by making an entry in CMSRules.drl file present in the /usr/ncra/lib directory.

A typical rule looks like this:

```
rule "domon time out"
 no-loop
 lock-on-active true
 dialect "java"
 when
 alarm : Alarm(name == "domon_timeout")
 state : State(current != "SUSPEND" )
 then
 System.out.println("changing state to SUSPEND");
 stateManager.changeState("SUSPEND","domon time out for the sub
 system"+alarm.getSubsystem(),"rule-engine");
 state.setRuleApplied(true);
 update(state);
 end
```

`"domon time out"` – rule name

`no-loop` – indicates when the Rule's consequence modifies a fact it may cause the Rule to activate again, causing recursion. Setting no-loop to true means the attempt to create the Activation will be ignored.

`lock-on-active true` – indicates that when the current rule get activated no other rule will get activated due to the consequence of changes done by current rule.

`dialect "java"` – indicates the scripting language supported

`when` – this indicates start of condition for this rule to get executed

`alarm : Alarm(name == "domon_timeout")` – this indicates that one of the condition required for this rule is alarm name should be `domon_timeout`

`state : State(current != "SUSPEND" )` – this indicates that one of the condition required for this rule is that current state should not be SUSPEND state

`then` – this indicates the end of condition and start of the actions to be taken when the rule condition is true

```
stateManager.changeState("SUSPEND","domon time out for the sub
system"+alarm.getSubsystem(),"system");
```

- this is a java api which changes the state to SUSPEND

`state.setRuleApplied(true);` – this is a java api which indicates that the rule has got applied.

`update(state)` – this is a java api which updates the state variable so that state-machine can determine that the rule has got applied.

`end` – indicates end of rule

On Similar lines a rule can be defined to run a batch command when a state-machine enters into one of the states:

For e.g. on entering MAINTENANCE state state-machine will execute the `initServo` script

```
rule "MAINTENANCE State"
no-loop
lock-on-active true
dialect "java"
when
state : State(current == "MAINTENANCE")
then
System.out.println("In MAINTENANCE");
stateManager.runBatchCommand("servo","initServo",null);
state.setRuleApplied(true);
update(state);
end
```

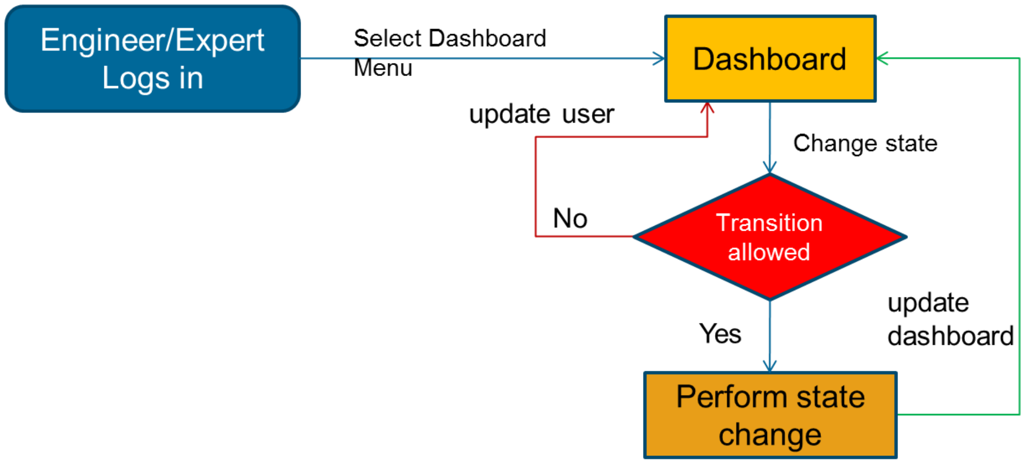
9.4.2 Rule API

The following are the API'S that can be called from Rules engine for configuring the rule functionality.

9.4.2.1 Change State

```
stateManager.changeState("SUSPEND","domon time out for the sub
system"+alarm.getSubsystem(),"system");
```

1st argument – state to which CMS should transition
2nd argument – reason for state change, this gets logged in database
3rd argument – the user that will perform the state change



9.4.2.2 Run Batch

For calling a batch it must be pre-configured as a batch command, and then this configured batch command can be called from rules engine as below:

```
stateManager.runBatchCommand("servo","handleException",null);
```

1st argument – subsystem name
2nd argument – batch command name
3rd argument – input parameters for the batch if any

For exception handler to be called it must be pre-configured as a batch command.

9.4.2.3 Block Command to a subsystem

For blocking all command to a subsystem use the below mentioned api :

```
stateManager.blockAllCommand("servo","*");
```

1st argument – subsystem name
2nd argument – batch command name or “*” indicating all commands

9.4.2.4 Unblock Command to a subsystem

For unblocking all command to a subsystem use the below mentioned api :

```
stateManager.unblockAllCommand("servo","*");
```

1st argument – subsystem name
2nd argument – batch command name or “*” indicating all commands

9.4.3 Exception Handling through Rules

As mentioned in above section any batch command can be run in event of a critical alarm. So if exception handling is to be done a batch command can be configured to handle the exception.

In case of a critical alarm if no rule is configured in CMSRules.drl file state-machine will automatically enter the SUSPEND state.

9.4.4 Subsystem State and DoMon

When CMS is initializing the Subsystem state is displayed as NOT CONNECTED. During initialization CMS tries to connect to individual wrapper till the connectivityTimeOut period specified in cms.properties. If the connection is not established and the connectivityTimeOut time elapses, the Subsystem state is displayed as NOT CONNECTED. If the wrapper connection gets established the Subsystem state is displayed as CONNECTED.

Once doMon command for each active subsystem is executed in initialization the CMS starts receiving monitoring data. The subsystem state gets updated as per the value specified in **state** monitoring parameter.

If the subsystem wrapper gets disconnected in between the doMon command for that particular subsystem will time out after the monitoringTimeout period (this is configurable in cms.properties). On doMon timeout the Subsystem state is displayed in NOT CONNECTED. Once wrapper is up it sends RESET in state monitoring parameter and the Subsystem state displays the state as RESET. On receiving RESET CMS executes the RESET script for that particular subsystem.

After RESET the subsystem state displays the value received in state monitoring parameter.

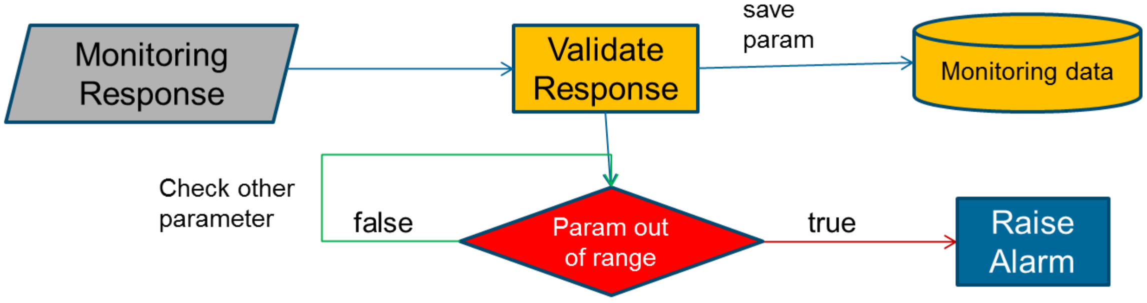
9.5 Alarms

9.5.1 Alarms & Monitoring Parameters

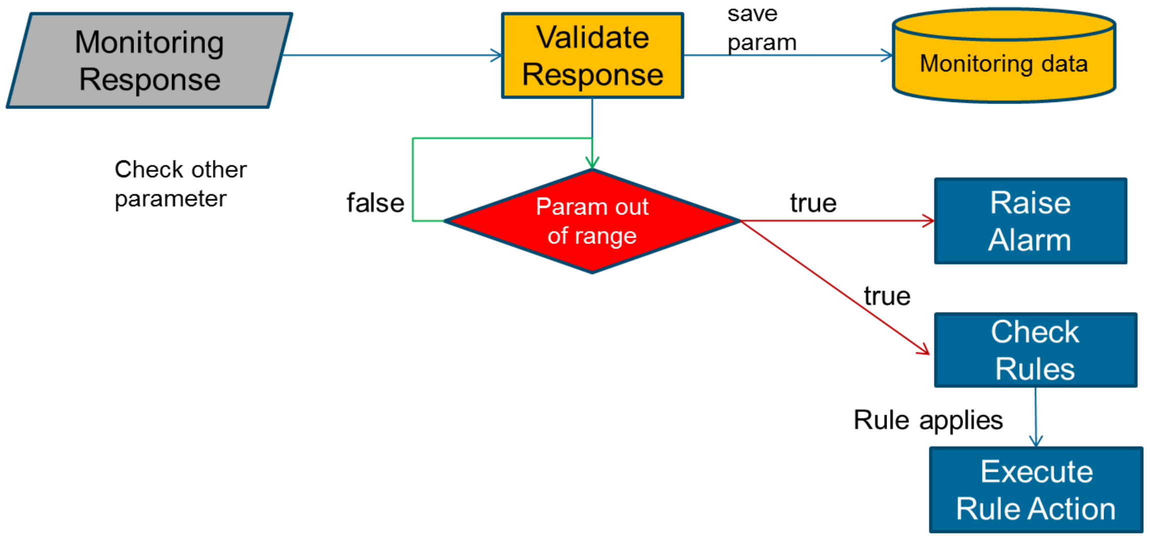
1. Alarms can be raised by subsystem by sending it as part of Response
2. Alarm is raised when a monitoring parameter goes out of range.
3. The Alarm associated with a monitoring parameter is specified in the <supportedalarms> section of the commands xml. The syntax to be followed for alarm name is alarm_<monitoring param name> . For e.g. the name for wind_vel1 parameter the alarm name would be alarm_wind_vel1
4. The alarm is also raised directly from monitoring response please refer section Monitoring Response Format
5. Each subsystem will send monitoring parameters at specific interval. If the monitoring parameter is not received at specific interval of time and if the monitoring is not disabled then an alarm is raised and the subsystem will be assumed to be in exception state.
6. The monitoring parameter will be validated against the validation specified in the <supportedresponses> section.

9.5.2 Alarm work flow

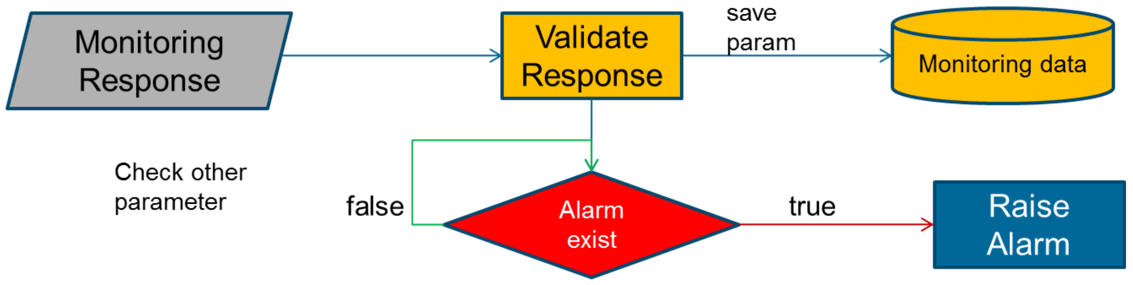
9.5.2.1 *Raise alarm when monitoring parameter out of range*



9.5.2.2 *Apply rule when monitoring parameter is out of range*



9.5.2.3 *Raise alarm from the monitoring response*



9.6 Features

1. Monitor each monitoring parameter and raise alarm when parameter goes out of range. The range and alarm has to be pre-defined in CMS.
2. Change state when an alarm is raised. The rule for state change needs to be defined.
3. Change to Suspend state when a critical alarm is raised.
4. Change State though UI
5. Raise an alarm for any command failure, if the particular rule is defined.
6. Take an action when an alarm is received. The actions will have to be specified through batch script. The alarms will have to be pre-defined and the batch script to be called also needs to be pre-defined in the rules-engine.
7. Raise an Alarm for any command failure. The rule for this will have to be defines in rules-engine.
8. Raise alarm when the Antenna/Telescope goes off source after it was on source. This will be determined if the Alt or Az or both are consistently out of the allowed limits after being on source.
9. Block commands (one or many) to a subsystem when state machine is in particular state. This rule has to be defined as to which commands need to be blocked
10. Unblock command (one or many) to a subsystem when state machine is in particular state. This rule has to be defined as to which commands need to be unblocked

9.7 State Machine Challenges and Limitations

1. States are not fully configurable, since the transitions and actions associated with a State requires coding. So States having some complex actions associated with it cannot be added through XML.
2. The Rules are defined using JBoss Drools, so the Expert/Engineer need to get acquainted with the syntax of Drools http://www.jboss.org/drools/drools-expert.html
3. Transitions need to be pre-configured.
4. Currently dynamic changing the rules is not supported i.e. the rules have to be configured and a tomcat restart would be required for the rules to get applied. We are exploring the part of configuring the rules when state machine is up and running.
5. The actions mentioned in rules are limited to the functionality supported by state machine. The supported actions are as mentioned below:
    - Change state
    - Block some or all Commands to particular subsystem
    - Run a pre-defined batch script
    - Raise an Alarm

10 Miscellaneous

10.1 Pre-requisite

10.1.1 Header Section

"**cms.properties**" file should be configured properly as mentioned in CMS Configuration Section.

10.1.2 Catalog & 2D Plot

The catalog object file should follow the NCRA standard format. Sample catalog file included in Release build is ncra15m.catalog

10.1.3 Polar Plot

1. To display the polar plot CMS should receive the below mentioned monitoring parameters from wrapper:
   a) az_cp
   b) el_cp

2. az_target and el_target are required to display the target position. In order to display the target object user should execute the track command from the expert tab.

3. The refresh interval of Polar plot needs to be configured to the time after which polar plot refresh is desired. Refer to CMS Configuration section configuration related to polarRefreshInterval parameter.

4. The number of tracking points to be viewed can also be configured to the desired value so that the plot doesn't look cluttered. For this refer to CMS Configuration section for configuration related to polarPlotPointsLimit parameter.

10.1.4 Tracking Status

1. globalparameter.properties needs to be correctly configured to show the default parameters in Tracking status, Please refer global parameter configuration section

2. Wrapper should send monitoring information for Az, EL in monitoring parameter az_cp and el_cp respectively.

10.1.5 Tune Receiver

1. All the command configuration files are configured correctly for corresponding subsystems

2. All the UI configuration xmls are configured correctly

10.1.6 Batch Mode

The user should be aware of the Sleep 2.1 syntax for writing a valid batch script

10.1.7 Manual Mode

Following are the pre-requisites for manual mode which must be specified in “**cms.properties**” file

1. “**manualmodeSubsystem**” : specifies the manual mode subsystem to which manual mode commands will be sent for execution.
   i. **Example**: manualmodeSubsystem = servo

2. “**manualmode.file**”: specifies the property file which will contain the key value pairs for manual mode.

   i. manualmode.file=manualmode.properties

   ii. **Example**: TRACK=trackObject

   iii. The value “trackObject” is used against the key “TRACK” in “manualmode.properties” where “trackObject” is a command specified in “servo_commands.xml”.

3. “**dataAcqSubsystem**” : specifies the backend command subsystem to which backend commands will be sent for execution.
   i. **Example**: dataAcqSubsystem=backend

4. “**dataAcq.file**”: specifies the property file which will contain the key value pairs for manual mode.

   i. dataAcq.file=dataAcq.properties

   ii. **Example**: STARTDATAACQ=START

   iii. The value “START” is used against the key “STARTDATAACQ” in “dataAcq.properties” where “START” is a command specified in “backend_commands.xml”.

10.1.8 Data Monitor Tab

ChartRecorder.xml, SpectralLineDisplay.xml, PulsarDisplay.xml files are configured properly, these xml's are used for the dynamic UI generation, for more information about dynamic UI generation please refer “Dynamic UI Generation Configuration” section.

10.1.9 Expert Tab

All the command configuration files '*_commands.xml' (e.g. servo_commands.xml) are configured correctly.

10.1.10 Command Log

Following properties must be specified in "**cms.properties**" file.

1. commandlogfilename=file.xls
   commandlogfilename specifies the default name of the file used for downloading the logs.

10.1.11 Meta Data

MetaData.xml file is configured properly, this xml's are used for the dynamic UI generation, for more information about dynamic UI generation please refer "Dynamic UI Generation Configuration" Section

10.1.12 Engineering UI

1. The below mentioned configuration files must be present in the lib directory.

   a. ncra-subsystemconfig.xml

   It contains subsystem related information along with subsystemname_engineering.XML file used to generate UI from it.
   Syntax:
   Servo subsystem: Under subsystem tag user can mention configuration about subsystem. Tag <**engXML**> specifies name of xml file used to generate engineering UI.

```
<subsystems>
        <subsystem>
                <name>servo</name>
                <connectionurl>127.0.0.1:7775</connectionurl>
                <commandfile>servo_commands.xml</commandfile>
                <version>1</version>
                <engXML>servo_engineering.xml</engXML>
        </subsystem>
        <subsystem>
                <name>sentinal</name>
                <connectionurl>127.0.0.1:7775</connectionurl>
                <commandfile>sentinal_commands.xml</commandfile>
                <version>1</version>
                <engXML>sentinal_engineering.xml</engXML>
        </subsystem>
</subsystems>
```

b. subsystemname_commands.XML (e.g. servo_commands.xml)

It contains all the supported commands and supported responses with validation. It also contains Monitoring parameter with validation. Refer to servo_commands.xml for a sample subsystem command, response and monitoring parameter configuration.
The command issued from UI is validated against command mentioned in this xml. And the responses received from monitoring simulator are validating against responses mentioned in this xml.

c. subsystemname_engineering.XML (e.g. servo_engineering.xml)

It contains four section Status param, Monitoring param, Basic commands and detailed commands for particular sub system. Engineering UI is loaded from this xml during CMS initialization. This XML defines contents of the Engineering UI.

2. The user should have permission to view the individual subsystem Engineering GUI. For e.g. to view Servo Engineering GUI user should be granted permission of SERVO ENGINEERING.

3. For details of configuring Engineering GUI refer to **Engineering UI Configuration NCRA** document.

User can add a status parameter or Monitoring parameter to UI through this engineering xml. The newly added parameter must be pre-configured as monitoring parameter in subsystemname_commands.XML

User can add a command to Basic Commands or Detailed commands section of subsystemname_engineering.XML. The command added should have been pre-configured in the subsystemname_commands.XML. If command is not specified in subsystemname_commands.xml then UI shows “Command not found” error message

10.1.13 State Machine

The below mentioned configurations should be present for state-machine. For additional details of this configuration refer to CMS Configuration and Deployment [NCRA]

1. ncra-subsystemconfig.xml should be configured to indicate the active subsystems. Alarms will be raised only for active subsystems.

2. connectivityTimeout determines the maximum time for which state-machine waits for all the wrappers to get connected. If all wrappers are connected before connectivityTimeout is reached CMS will proceed for the subsystem initialization.

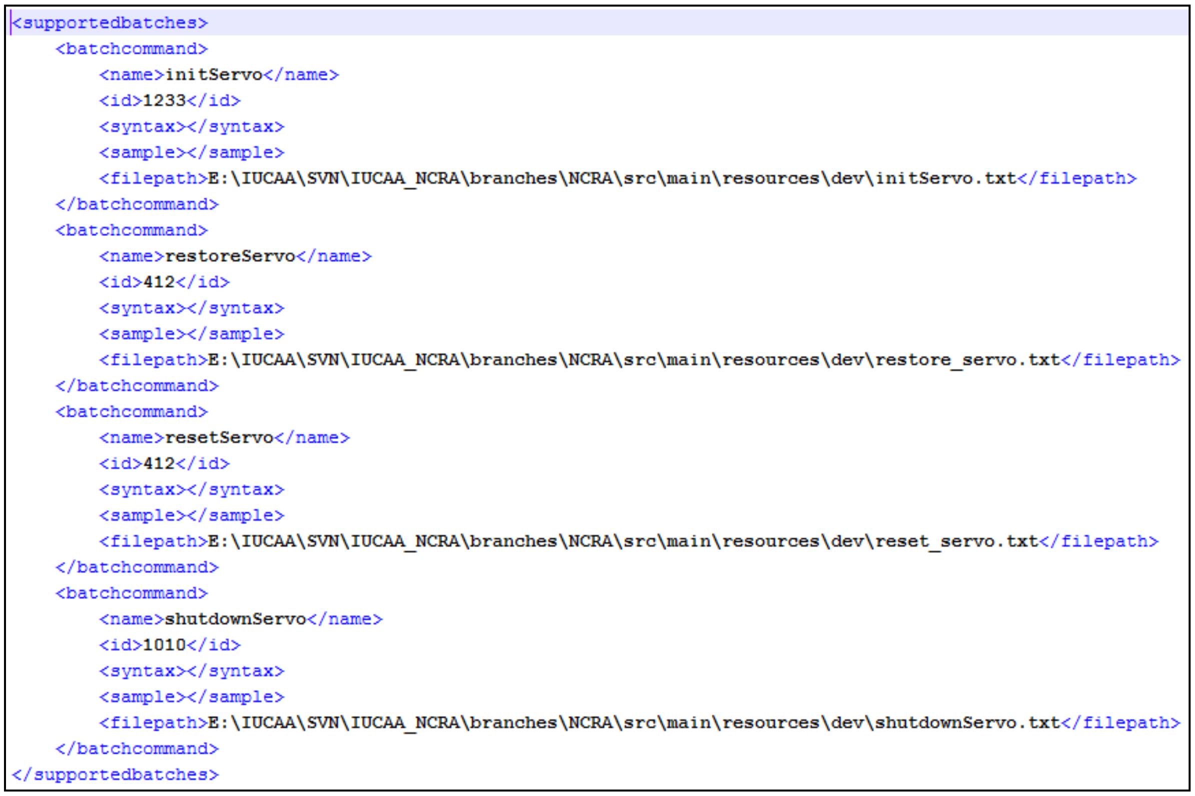
3. connectivityDelay determines the wait time between the ping requests to the unconnected wrappers.
4. timeIntervalOfAlarm property determines the minimum time interval between two identical alarms after which the newly received alarm can be saved in database by state machine. This prevents incessant of the same alarm, if the alarm is raised quite frequently.

5. Monitoringfrequency determines the time interval between two monitoring responses from the same wrapper

6. Initialization configuration i.e. configuring the initialization script for each subsystem and CMS initialization script configuration

7. Shutdown configuration i.e. configuring the shutdown script for each subsystem and CMS shutdown configuration.

8. Init and shutdown scripts must be configured as a batch command in “supportedbatches” in corresponding *_command xmls.
   For example: “initServo” and “shutdownServo” must be configured in servo_commands.xml as follows:



```
<supportedbatches>
    <batchcommand>
        <name>initServo</name>
        <id>1233</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>E:\IUCAA\SVN\IUCAA_NCRA\branches\NCRA\src\main\resources\dev\initServo.txt</filepath>
    </batchcommand>
    <batchcommand>
        <name>restoreServo</name>
        <id>412</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>E:\IUCAA\SVN\IUCAA_NCRA\branches\NCRA\src\main\resources\dev\restore_servo.txt</filepath>
    </batchcommand>
    <batchcommand>
        <name>resetServo</name>
        <id>412</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>E:\IUCAA\SVN\IUCAA_NCRA\branches\NCRA\src\main\resources\dev\reset_servo.txt</filepath>
    </batchcommand>
    <batchcommand>
        <name>shutdownServo</name>
        <id>1010</id>
        <syntax></syntax>
        <sample></sample>
        <filepath>E:\IUCAA\SVN\IUCAA_NCRA\branches\NCRA\src\main\resources\dev\shutdownServo.txt</filepath>
    </batchcommand>
</supportedbatches>
```

9. Alarm configurations i.e. configuring the alarm to be raised on init failure, shutdown failure, monitoring parameter going out of range, command failure and for command timeout. If the alarm is not configured for above mentioned failure scenarios, dummy alarm with info level will be raised for the same.

10. **criticalalarmemailalias** property from cms.properties should contains valid email alias as mails will be sent out to users in this alias when a critical alarm is raised in CMS or received by CMS.