IDM UID
9KBPN8
VERSION CREATED ON / VERSION / STATUS
01 Feb 2013 / 2.0/ Signed
EXTERNAL REFERENCE

Report

CODAC Alarm Handling System - Test Plan

Test Plan of CODAC Alarm Handling System

Approval Process			
	Name	*Action*	*Affiliation*
Author	**Utzel N.**	**01-Feb-2013:signed**	**IO/DG/DIP/CHD/CSD/CDC**
CoAuthor			
Reviewers			
Approver			
Document Security: level 1 (IO unclassified) *RO: Di Maio Franck*			
Read Access	**LG: SOPRA extra, AD: ITER, AD: External Collaborators, AD: Division - Control System Division, AD: Section - CODAC, project administrator, RO**		

DISCLAIMER : UNCONTROLLED WHEN PRINTED – PLEASE CHECK THE STATUS OF THE DOCUMENT IN IDM

Change Log				
Title (Uid)	***Version***	***Latest Status***	***Issue Date***	***Description of Change***
CODAC Alarm Handling System - Test Plan (9KBPN8_v2_0)	v2.0	Signed	01 Feb 2013	Introduction of a Pump Control System Use Case to demonstrate the use of the Alarm System
CODAC Alarm Handling System - Test Plan (9KBPN8_v1_2)	v1.2	Signed	10 Oct 2012	Minor changes on how to start manually the alarm server and notifier. Updated screenshot of the new version of the OPI Probe Tool.
CODAC Alarm Handling System - Test Plan (9KBPN8_v1_1)	v1.1	Signed	06 Jun 2012	Some additional details regarding the workspace setup
CODAC Alarm Handling System - Test Plan (9KBPN8_v1_0)	v1.0	Signed	04 Jun 2012	

DISCLAIMER : UNCONTROLLED WHEN PRINTED – PLEASE CHECK THE STATUS OF THE DOCUMENT IN IDM

china eu india japan korea russia usa

CODAC Alarm Handling System

Software Test Plan (STP)
Based on QA Template Version <1.0>

This document describes the tests that should be performed for CODAC Alarm Handling System in order to be installed as part of Core System release. Different test cases are described, as well as and test pass-fail criteria.

Contents

1 Introduction 4
1.1 Purpose 4
1.2 Scope 4
1.3 System/Software overview and key features 5
1.4 References 5
2 Details of the Testing Process 6
2.1 Definition of test levels 6
2.2 Test administration 6
2.2.1 Anomaly resolution and reporting 6
2.2.2 Test reporting requirements 6
2.2.3 Test deliverables 6
3 Component Test Plan 7
3.1 Scope 7
3.1.1 Test items and their identifiers 7
3.1.2 Features to be tested 7
3.1.3 Features not to be tested 7
3.2 Approach 7
3.2.1 Testing Methods 7
3.2.2 Item pass/fail criteria 8
3.3 Environment / Infrastructure 8
3.4 Component Test Procedures 8
3.4.1 Alarm Configuration Import 8
3.4.2 Alarm Configuration Export 9
3.4.3 Alarm Server Startup 11
3.4.4 Alarm PV Tree 12
3.4.5 Alarm Related Display – Operator Interface 13
3.4.6 Alarm Related Display – Data Plot 14
3.4.7 Alarm Notification 16
3.4.8 Alarm Guidance 17
3.4.9 Latched Alarm 20
3.4.10 Display Limit Alarms 20
3.4.11 JMS Monitor 22
3.4.12 Message History 23
3.4.13 Automated Actions 24

3.5 Component Test Log 27
3.5.1 Alarm Configuration Import 27
3.5.2 Alarm Configuration Export 27
3.5.3 Alarm Server Startup 27
3.5.4 Alarm PV Tree 27
3.5.5 Alarm Related Display – Operator Interface 27
3.5.6 Alarm Related Display – Data Plot 27
3.5.7 Alarm Notification 28
3.5.8 Alarm Guidance 28
3.5.9 Latched Alarm 28
3.5.10 Display Limit Alarms 28
3.5.11 JMS Monitor 28
3.5.12 Message History 28
3.5.13 Automated Actions 28
Software Test Plan Checklist 30

1 INTRODUCTION

1.1 Purpose

This document describes the tests that should be performed for CSS BEAST - Best Ever Alarm System Toolkit - in order to be installed as part of CODAC Core System. These tests will ultimately compare the capabilities of BEAST against these described in CODAC System Requirement (SRD) Document [IDM 28C2HL].

Particular functions to be tested are the alarm configuration, alarm notification and alarm graphical user interface (GUI) – i.e. the main components of the alarm system as shown on Figure 1-I - BEAST Architecture, except the reporting and annunciation functions not yet part of CODAC Core System.

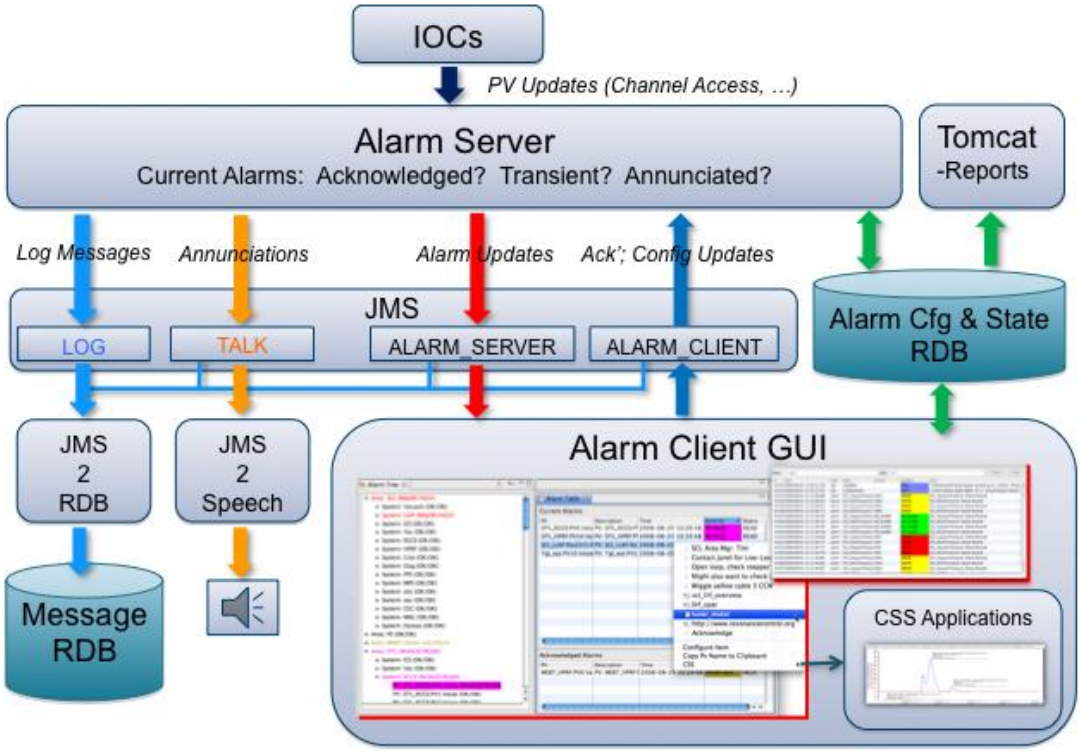


Figure 1-I - BEAST Architecture

1.2 Scope

The test items are:

- The operational version of BEAST,
- The data, including all the configuration data needed to run the alarm system,
- The documentation, including the online help and the release notes.

The installation and uninstallation of the components are not part of this test plan.

1.3 System/Software overview and key features

Best Ever Alarm System Toolkit (BEAST) - is a distributed alarm system consisting of:

- Alarm Server that monitors alarm triggers in the control system,
- Relational Database for configuration and logging,
- CSS user interface for viewing current alarms as a table or hierarchical tree.

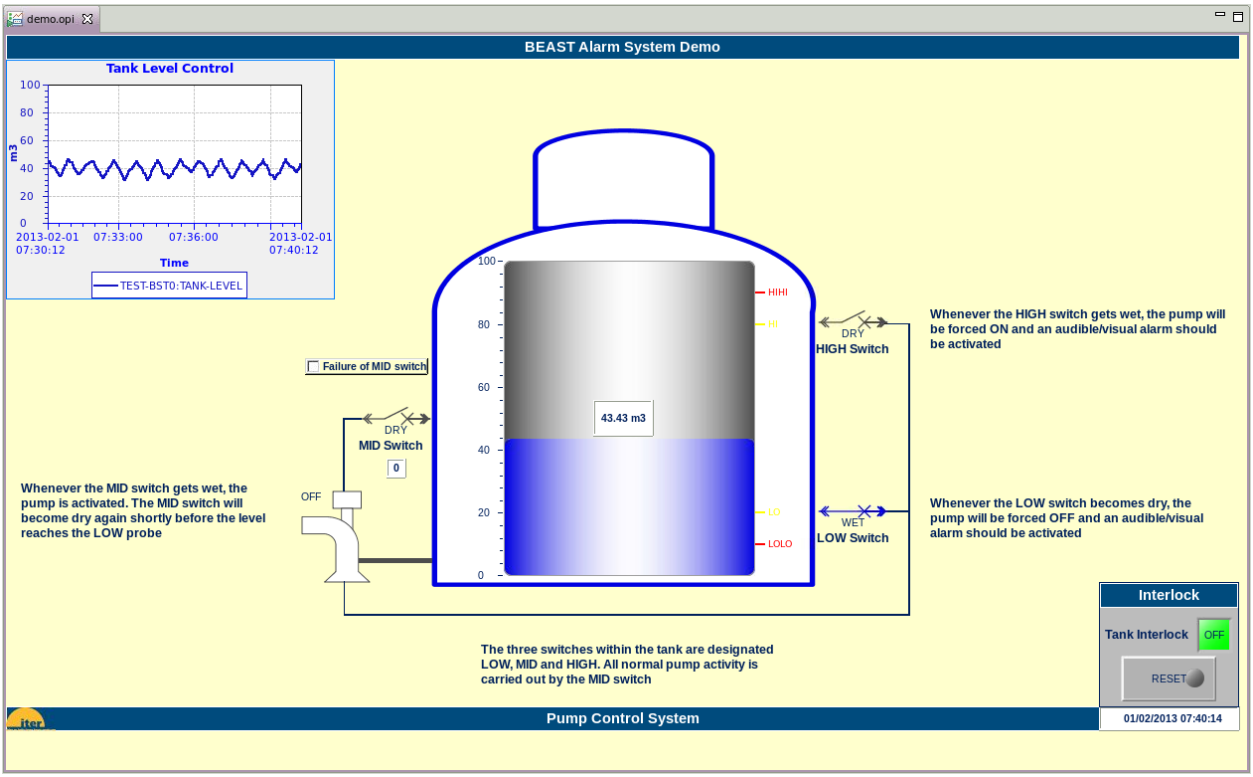
A classic pump control process which manages the level of liquid in a tank will be used to simulate abnormal situations that require the operator attention. When the level in the tank rises to an appropriate level, the pump must be activated. When a lower level is reached the pump must stop.

Three switches within the tank are designed: **LOW**, **MID** and **HIGH**. All normal pump activity is carried out by the **MID** switch. The **LOW** and **HIGH** switches provide the necessary redundancy and supervision to the installation.

Whenever the **MID** switch gets wet, the pump is activated. After some time, the **MID** switch will become dry again. At that point a timer of 25 seconds is started in order to stop the pump shortly before the level reaches the LOW probe.

Whenever the **HIGH** switch gets wet, the pump will be forced **ON** and an auditable / visual alarm will be activated for this abnormal situation.

Whenever the **LOW** switch becomes dry, the pump will be forced **OFF** and an auditable / visual alarm should be activated.



1.4 References

CODAC Quality assurance plan - https://user.iter.org/?uid=6J7RW4

2 DETAILS OF THE TESTING PROCESS

2.1 Definition of test levels

The described component tests will focus on the desired features of CODAC Alarm Handling System.

Following test levels are defined in this test plan to organize the testing activity.

Alarm Configuration Component Test	**CFG**
Test of the different archiving modes import configuration	
Alarm Server Component Test	**SRV**
Test of archiving overrun and monitoring	
Alarm Display Component Test	**DSP**
Test of CSS Alarm Graphical Interface	
Alarm History Component Test	**HST**
Test of JMS Alarm Log	

2.2 Test administration

2.2.1 Anomaly resolution and reporting

Anomaly Reports shall be submitted in Bugzilla.

2.2.2 Test reporting requirements

The test logs shall be generated to record the outcome of test procedures as described in section *.4 and *.5 of the level test plans.

2.2.3 Test deliverables

The test deliverables include:

- Component Test Logs / Reports
- Anomaly Reports with Bugzilla bug references.

Test input data are registered in SVN code repository.

No other test tool is needed.

The test reports may be submitted on ITER IDM.

3 COMPONENT TEST PLAN

3.1 Scope

3.1.1 Test items and their identifiers

CODAC Alarm Handling System is included the following unit:

- m-css organised in 2 main products:
  - org.csstudio.iter.alarm.beast.configtool.product/
  - org.csstudio.iter.alarm.beast.server.product/

  And 4 utilities

  - org.csstudio.iter.alarm.beast.annunciator.product/
  - org.csstudio.iter.alarm.beast.notifier.product/
  - org.csstudio.iter.jms2rdb.product/
  - org.csstudio.iter.utility.jmssendcmd.product/

CODAC Alarm Handling System mainly depends on different features:

- org.csstudio.iter.alarm.beast.configtool.app.feature/
- org.csstudio.iter.alarm.beast.server.app.feature/
- org.csstudio.iter.alarm.beast.ui.feature/
- org.csstudio.iter.alarm.beast.annunciator.app.feature/
- org.csstudio.iter.alarm.beast.notifier.app.feature/
- org.csstudio.iter.jms2rdb.app.feature/
- org.csstudio.iter.utility.jmssendcmd.app.feature/

3.1.2 Features to be tested

The main CODAC Alarm Handling System features to be tested are:

- Alarm configuration import/export
- Alarm Server startup
- Alarm Notification
- Limit Alarms
- JMS Monitor
- Message History

3.1.3 Features not to be tested

The annunciator and reporting functions are not part of CODAC Alarm Handling System for now.

3.2 Approach

3.2.1 Testing Methods

The overall approach for the level of testing is the Black box method to test the functionality of CODAC Alarm Handling System.

3.2.2 Item pass/fail criteria

Each major anomaly found determines whether each test item has passed or failed testing.

3.3 Environment / Infrastructure

Core System in its development role version should be installed on a CODAC standard machine. Access to SVN is required.

3.4 Component Test Procedures

CFG-01	**3.4.1 Alarm Configuration Import**	

Prerequisite	In a Linux console, create a working directory, download and start a demo IOC:
	1.$ mkdir test
	2.$ cd test
	3.$ svn co https://svnpub.iter.org/codac/iter/codac/dev/units/m-css/trunk/products/ITER/products/org.csstudio.iter.alarm.beast.server.product/demo/
	`A ???`
	`Checked out revision xxx.`
	4.$ cd m-TEST-BEAST
	5.$ softIoc -s -d src/main/epics/TEST-BST0App/Db/PSH0-TEST-BST0.db
	`Starting iocInit`
	`############################################################################`
	`## EPICS R3.14.12.3-rc1 $Date: Mon 2012-12-03 16:39:27 -0600$`
	`## EPICS Base built Dec 10 2012`
	`############################################################################`
	`iocRun: All initialization complete`
	`epics> dbl`
	`TEST-BST0:HIGH-SWITCH`
	`TEST-BST0:LOW-SWITCH`
	`TEST-BST0:MID-SWITCH`
	`TEST-BST0:MID-SWITCH-FAIL`
	`TEST-BST0:PUMP`
	`TEST-BST0:TANK-MODE`
	`TEST-BST0:MID-TIMER`
	`TEST-BST0:TANK-LEVEL`
	`TEST-BST0:HIGH-CHECK`
	`TEST-BST0:LOW-CHECK`
	`TEST-BST0:MID-CHECK`
	`TEST-BST0:TANK-HIGH`
	`TEST-BST0:TANK-INTERLOCK`
	`TEST-BST0:TANK-LOW`
	`TEST-BST0:TANK-MID`
	`TEST-BST0:TIMER-CHECK`
	`TEST-BST0:TIMER-START`
	`TEST-BST0:IOC-TIME`
	`epics>`	

Test Cases	1. Positive confirmation of the alarm configuration loaded	
Procedure	In a new Linux console, import the alarm configuration for a “demo” alarm server: 1.$ cd test/m-TEST-BEAST	

	2.$ alarm-configtool -root demo -import -file src/main/beast/TEST-BST0-beast.xml Check that the Demo Archive Engine is configured 3. $ alarm-configtool -list	

Pass Criteria	2. The output of the command should be:
	`Alarm Config Tool <current Core System version>`
	`2013-01-23 13:33:36.332 INFO [Thread 19]`
	`org.apache.activemq.transport.failover.FailoverTransport (doReconnect) -`
	`Successfully connected to tcp://localhost:61616`
	`Reading RDB configuration of 'demo'`
	`Deleting existing RDB configuration for 'demo'`
	`Importing configuration 'demo' from src/main/beast/TEST-BST0-beast.xml`
	`Loading /demo/TEST-BST0`
	`Loading /demo/TEST-BST0/TEST-BST0`
	`Loading /demo/TEST-BST0/TEST-BST0/TEST-BST0:HIGH-SWITCH`
	`Loading /demo/TEST-BST0/TEST-BST0/TEST-BST0:LOW-SWITCH`
	3. The output of the command should contain the following declaration:
	`Alarm Config Tool <current Core System version>`
	`2013-01-23 13:35:09.030 INFO [Thread 19]`
	`org.apache.activemq.transport.failover.FailoverTransport (doReconnect) -`
	`Successfully connected to tcp://localhost:61616`
	`'demo'`
	…	

CFG-02	**3.4.2 Alarm Configuration Export**	
Prerequisite	1. Alarm Configuration Imported successfully	
Test Cases	1. Positive confirmation of the alarm configuration export	
Procedure	In the previous Linux console, export the Demo Alarm Server configuration: 1.$ alarm-configtool -root demo -export -file src/main/beast/export-beast.xml Check the exported configuration: 3. $ gedit src/main/beast/export-beast.xml& After the check, close gedit.	

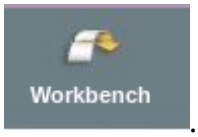
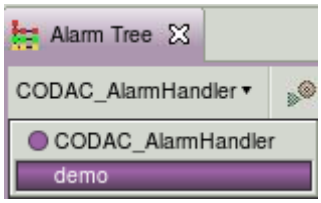
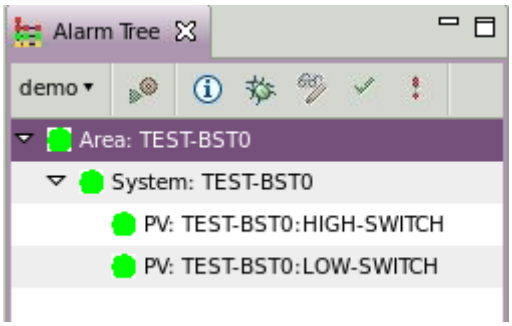
Pass Criteria	1. The output of the command should be:
	`archive-configtool -engine demo -export -config src/main/beauty/export-beauty.xml`
	`Exporting config for engine demo to src/main/beauty/export-beauty.xml`
	`2013-01-07 13:50:18.700 INFO [Thread 19]`
	`org.apache.activemq.transport.failover.FailoverTransport (doReconnect) -`
	`Successfully connected to tcp://localhost:61616`	

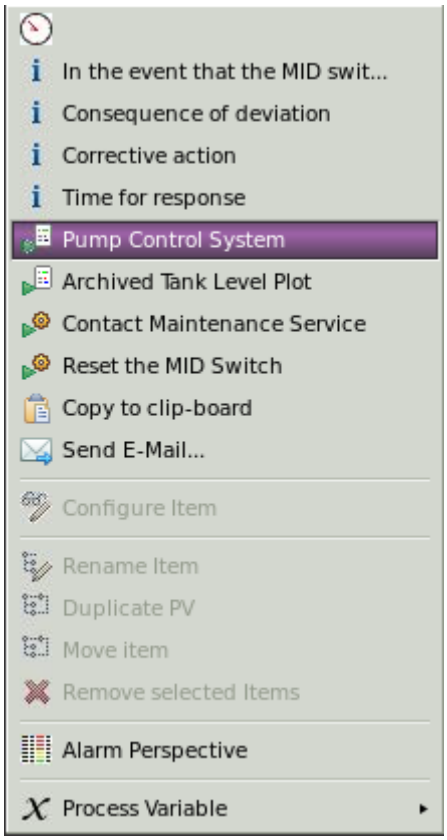
2. The xml configuration should be:

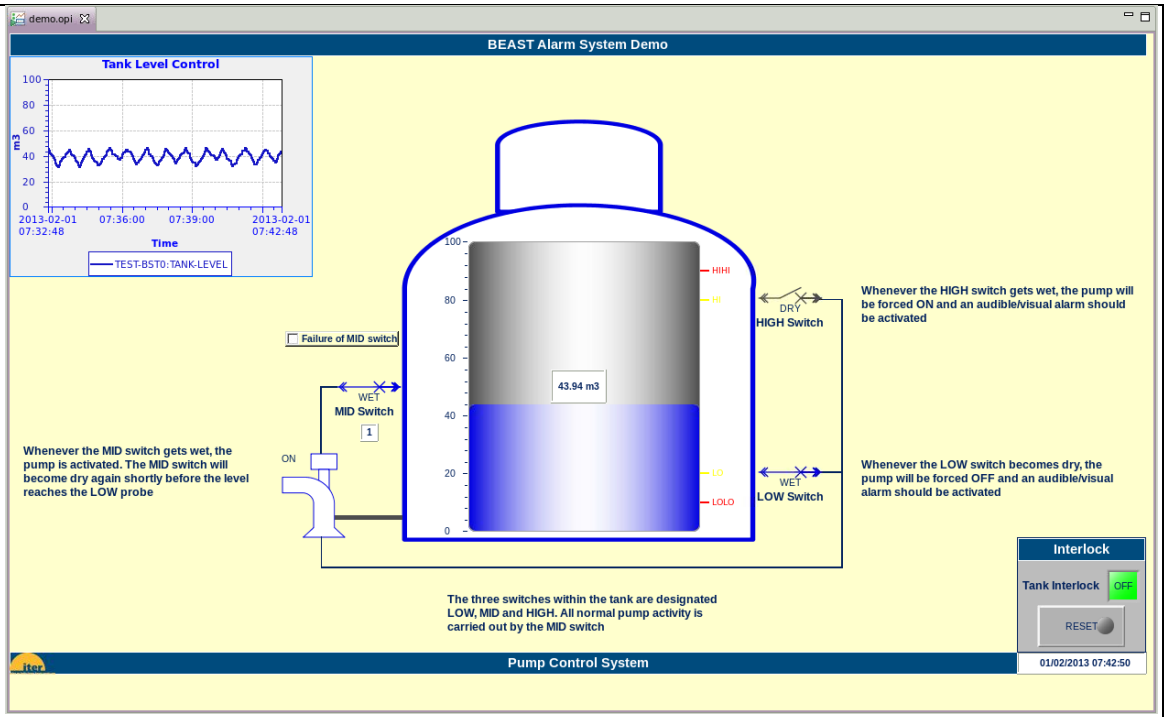
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--
     Alarm configuration snapshot Wed Jan 30 08:58:28 UTC 2013
     URL : jdbc:postgresql://localhost/css_alarm_3_0_0
     Root: demo
  -->
<config name="demo">
    <component name="TEST-BST0">
        <component name="TEST-BST0">
            <pv name="TEST-BST0:HIGH-SWITCH">
                <description>Whenever the HIGH switch gets wet, the pump will be forced ON</description>
                <latching>true</latching>
                <annunciating>true</annunciating>
                <guidance>
                    <title>In the event that the MID switch has failed</title>
                    <details>The tank level has continued to increase and has reached the HIGH level generating a MAJOR
alarm</details>
                </guidance>
                <guidance>
                    <title>Consequence of deviation</title>
                    <details>The MID switch fails to operate: defective switch, blocked switch or damaged
wiring.&#10;As a consequence the pump remaining dormant, the tank will overflow, causing inconvenience and damage to
property.</details>
                </guidance>
                <guidance>
                    <title>Corrective action</title>
                    <details>Reset the MID Switch (if needed)</details>
                </guidance>
                <guidance>
                    <title>Time for response</title>
                    <details>The tank will overflow if the pump fails to be forced ON in 10 minutes</details>
                </guidance>
                <display>
                    <title>Pump Control System</title>
                    <details>/m-TEST-BEAST/src/main/boy/demo.opi</details>
                </display>
                <display>
                    <title>Archived Tank Level Plot</title>
                    <details>file:/m-TEST-BEAST/src/main/databrowser/tank-level.plt</details>
                </display>
                <command>
                    <title>Reset the MID Switch</title>
                    <details>caput TEST-BST0:MID-SWITCH-FAIL OFF</details>
                </command>
                <automated_action>
                    <title>Contact Maintenance Service</title>
                    <details>mailto:someone@iter.org,someone_else@iter.org?subject=*MID Switch fails to maintain the
tank level&amp;amp;body=*{0} alarm - Tank Switch is {1}</details>
                    <delay>5</delay>
                </automated_action>
            </pv>
            <pv name="TEST-BST0:LOW-SWITCH">
                <description>Whenever the LOW switch becomes dry, the pump will be forced OFF</description>
                <latching>true</latching>
                <annunciating>true</annunciating>
                <guidance>
                    <title>In the event that the MID switch has failed</title>
                    <details>The tank level has continued to decrease and has reached the LOW level generating a MAJOR
alarm</details>
                </guidance>
                <guidance>
                    <title>Consequence of deviation</title>
                    <details>The MID switch fails. The outcome is simple, the pump will drain the tank until it is
empty, whereupon it will try to pump air indefinitely and burn out.&#10;Unfortunately, every time a MID switch fails
for too long, a new pump may have to be installed</details>
                </guidance>
                <guidance>
                    <title>Corrective action</title>
                    <details>Change the MID Switch and check if the pump has already burned out</details>
                </guidance>
                <guidance>
                    <title>Time for response</title>
                    <details>The pump will burn out in 30 minutes</details>
                </guidance>
                <display>
                    <title>Pump Control System</title>
                    <details>/m-TEST-BEAST/src/main/boy/demo.opi</details>
                </display>
                <display>
                    <title>Archived Tank Level Plot</title>
                    <details>file:/m-TEST-BEAST/src/main/databrowser/tank-level.plt</details>
                </display>
                <command>
                    <title>Reset the MID Switch</title>
                    <details>caput TEST-BST0:MID-SWITCH-FAIL OFF</details>
                </command>
                <automated_action>
                    <title>Contact Maintenance Service</title>
                    <details>mailto:someone@iter.org</details>
                    <delay>0</delay>
                </automated_action>
            </pv>
        </component>
    </component>
</config>
```

SRV-01	**3.4.3 Alarm Server Startup**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration Imported successfully	
Test Cases	1. Positive confirmation of the demo alarm server and notifier started	
Procedure	In the previous Linux console, start the “demo” alarm server: 1.$ alarm-server -root demo& 2. $ alarm-notifier -root demo& 3. [optional - only if the machine has a sound card and speakers] alarm-annunciator&	

Pass Criteria	1. The output of the command should be:
	`$ 2012-05-31 18:45:37.847 INFO [Thread 10]`
	`org.csstudio.alarm.beast.server.Application (start) - Alarm Server <current`
	`Core System version> started for 'demo' configuration`
	`Alarm Server <current Core System version>`
	`Configuration Root: demo`
	`JMS Server Topic: demo_SERVER`
	`JMS Client Topic: demo_CLIENT`
	`JMS Talk Topic: demo_TALK`
	`JMS Global Topic: GLOBAL_SERVER`
	`<… many Info messages …>`
	`Read 2 PVs in xxx seconds: yyy PVs/sec`
	`<… many Info messages …>`
	2. The output of the command should be:
	`$ 2012-10-10 14:34:46.591 INFO [Thread 1]`
	`org.csstudio.alarm.beast.notifier.Application (start) - Alarm Notification`
	`<current Core System version> started for 'demo' configuration`
	`Alarm Notification <current Core System version>`
	`Configuration Root: demo`
	`JMS Server Topic: demo_SERVER`
	`JMS Client Topic: demo_CLIENT`
	`JMS Global Topic: GLOBAL_SERVER`
	`Notifier timer threshold: 100`
	`Notifier thread threshold: 100`
	`… org.csstudio.alarm.beast.notifier.AlarmNotifier (start) - Alarm Notifier started`
	`<… many other messages …>`	

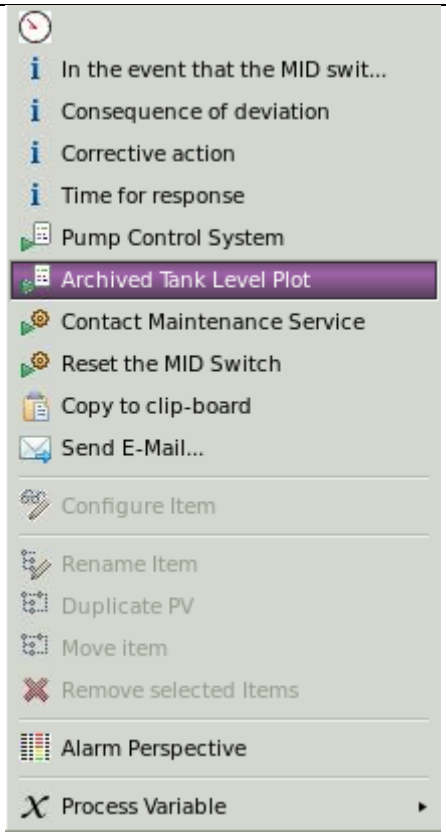
DSP-01	**3.4.4 Alarm PV Tree**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration imported 3. Alarm server started	
Test Cases	1. Positive confirmation of Demo alarm configuration and state	
Procedure	In the previous Linux console, start the Operator Interface to monitor the alarms triggered: 1.$ css& 2. Browse to select the working directory test 3. Close the Welcome screen by clicking on Workbench icon:  . Import the demo project into the Workspace from the Navigator View: 4. Right-click and select the option Import… and then General -> "Existing Projects into Workspace". Click on Next button. To select the root directory, click on Browse button, select m-TEST-BEAST and click OK. To import the selected project, click on Finish 5. Open the Alarm Perspective: menu Window -> Open Perspective -> Other… and select Alarm 6. Change the root element of the Alarm Tree View using the arrow near CODAC_AlarmHandler and select demo: 	
Pass Criteria	6. The Alarm Tree should reflect the demo server structure and current alarms state: 	

DSP-02	**3.4.5 Alarm Related Display – Operator Interface**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration imported 3. Alarm server started 4. Alarm Operator Interface started	
Test Cases	1. Positive confirmation of OPI related display	
Procedure	From CSS Alarm Tree View: 1. Right-click on the PV: TEST-BST0:HIGH-SWITCH   2. Select the option Pump Control System and check that the main monitoring operator interface is opened. In order to have a better view, you can switch to the dedicated perspective OPI Runtime: menu Window -> Open Perspective -> Other… and select OPI Runtime.	
Pass Criteria	2. The Pump Control System Operator Interface is displayed and you can monitor the tank level - All normal pump activity is carried out by the MID switch:	

		
DSP-03	**3.4.6 Alarm Related Display – Data Plot**	

Prerequisite	1. Demo IOC running
	2. Alarm Configuration imported
	3. Alarm server started
	4. Alarm Operator Interface started
	5. [optional only if you want some archived data]
	$ archive-configtool -engine demo -description 'Demo Test Engine' -port 5812 -import -config src/main/beauty/TEST-BST0-beauty.xml -replace_engine
	$ archive-engine -port 5812 -engine demo&	

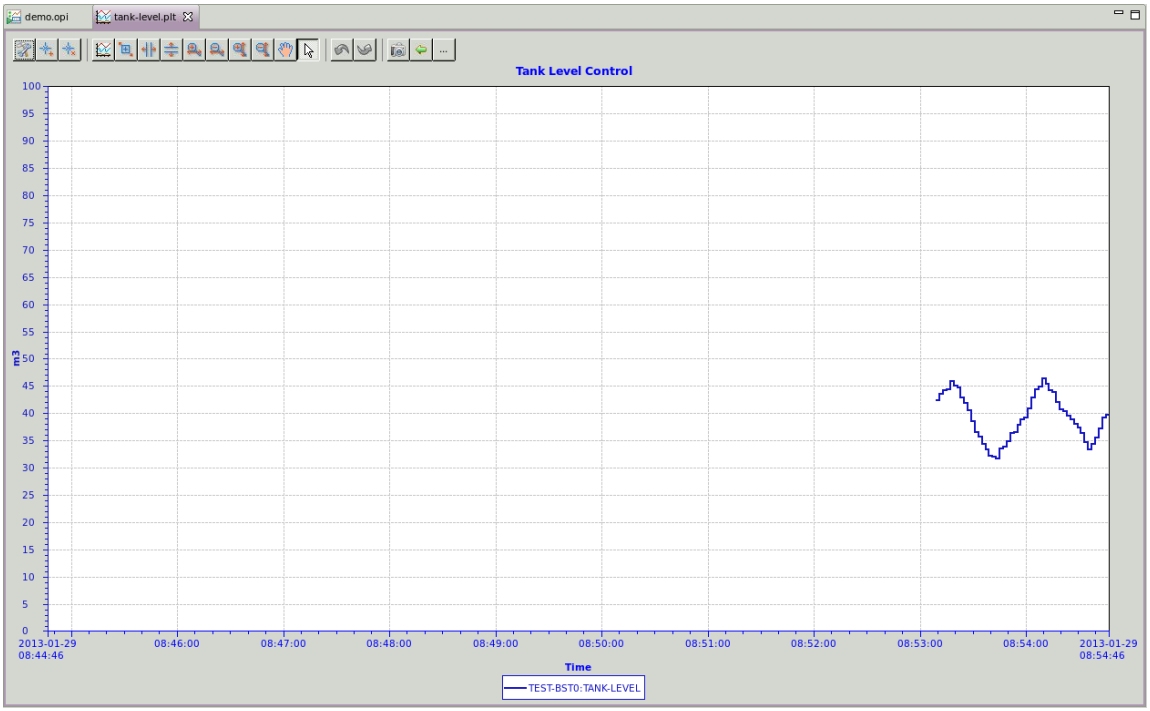
Test Cases	1. Positive confirmation of DataBrowser plot related display	
Procedure	From CSS Alarm Perspective - Alarm Tree View: 1. Right-click on the PV: TEST-BST0:HIGH-SWITCH 	

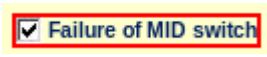
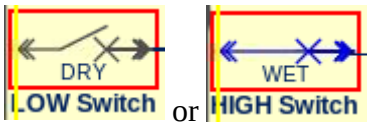
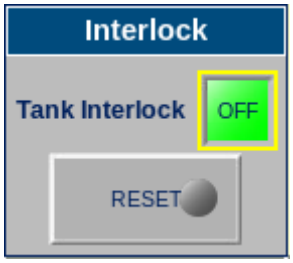
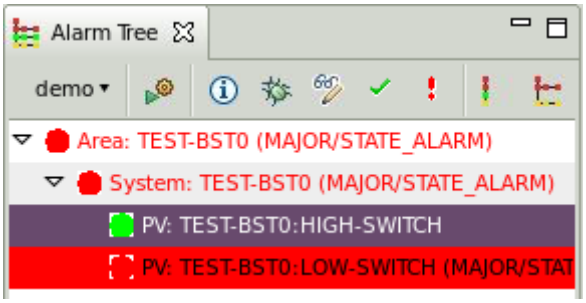


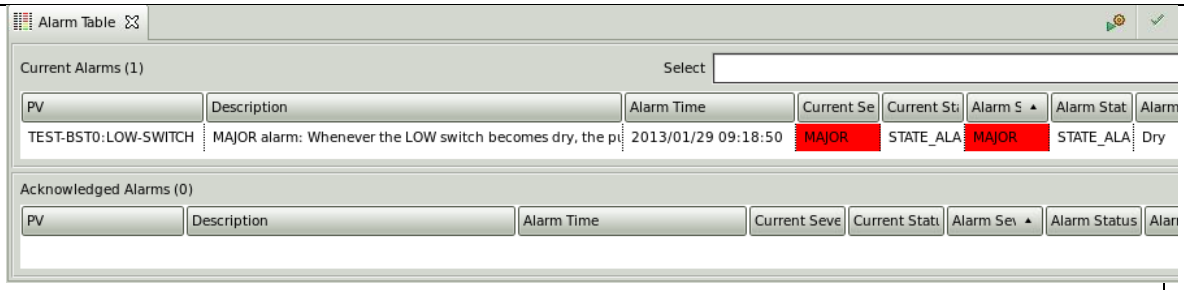
2. Select the option Archived Tank Level Plot and check that the plot is opened

Pass Criteria

2. By default, the plot is configured to retrieve – 10 minutes from the Archive Database and then it subscribes directly to the EPICS PV to have live data. After some minutes, the tank level should oscillate under the mid-level of 50 m3:

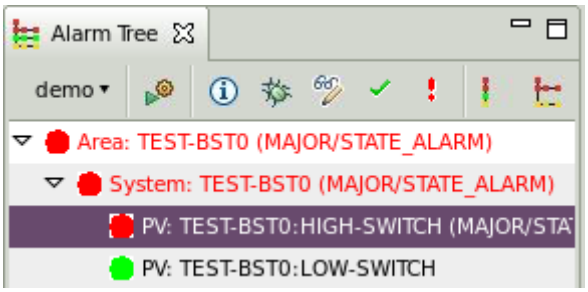


SRV-02	**3.4.7 Alarm Notification**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration imported 3. Alarm server started 4. Alarm Operator Interface started	
Test Cases	1. Positive confirmation of an alarm triggered	
Procedure	From the Operator Interface demo.opi, put the MID switch in error mode: 1. Check the box Failure of Mid switch  2. Wait for LOW switch to be dry (tank level less than LOW level of 20 m3) or the HIGH switch to be wet (tank level more than HIGH level of 80 m3):  Then switch to Alarm Perspective to check the current status 3. Click on Interlock Reset button on the bottom right of the operator interface  and confirm the action.	
Pass Criteria	2. If the pump was activated when the fault was triggered, the lower lever is reached after some time and a MAJOR alarm is generated on TEST-BST0:LOW-SWITCH:  The LOW alarm is also reflected in the Alarm Table View:	

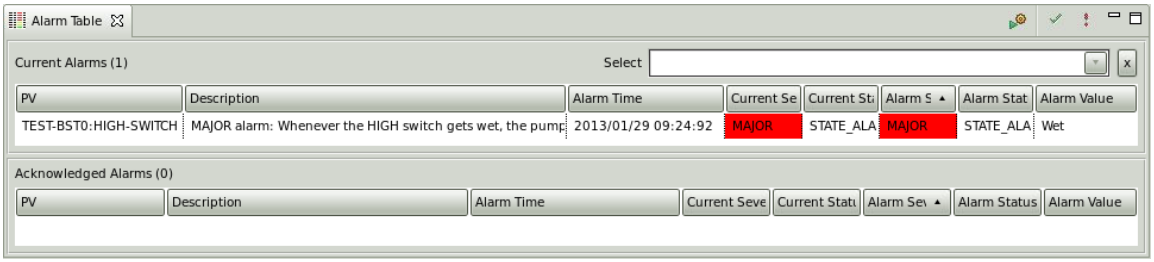


When resetting the system, the pump is stopped and the tank level starts to increase.

On the other hand, if the pump was stopped when the fault was generated on MID switch, the tank will overflow and generate a MAJOR alarm on PV: TEST-BST0:HIGH-SWITCH:

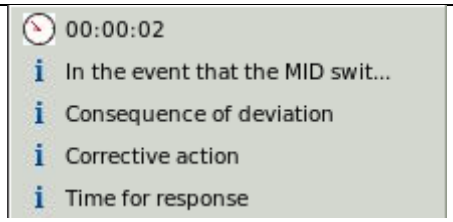
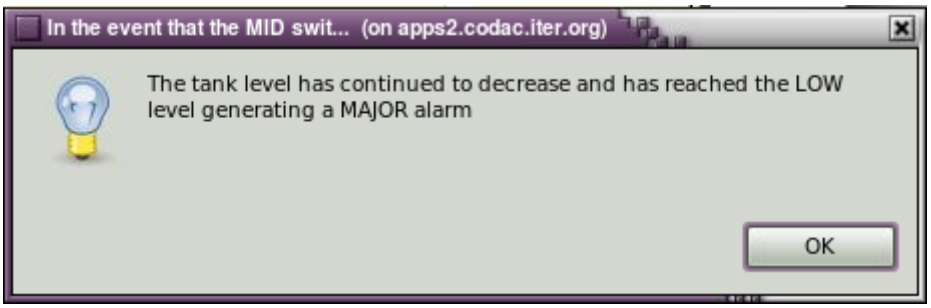
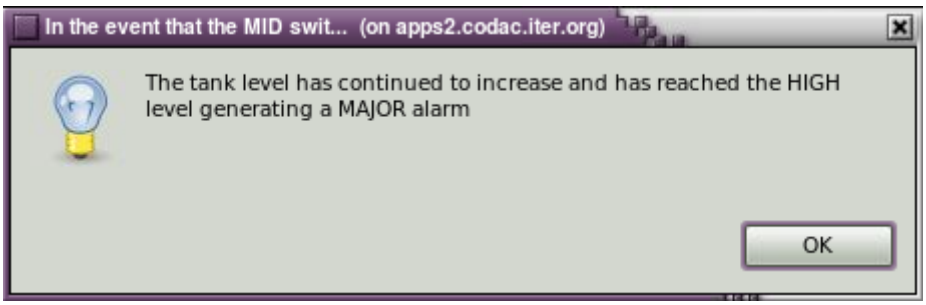
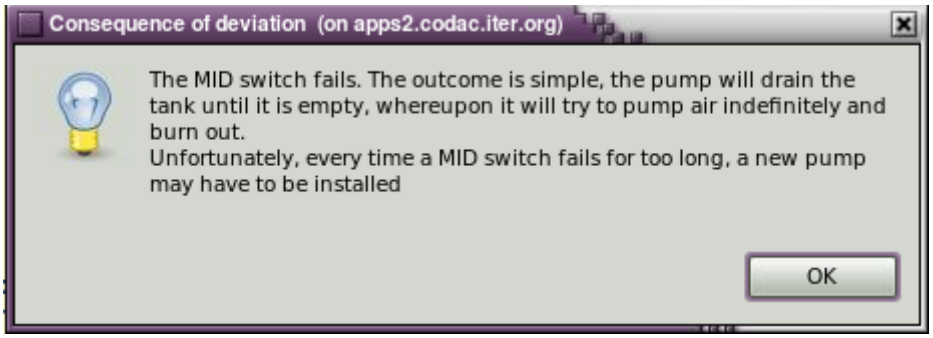


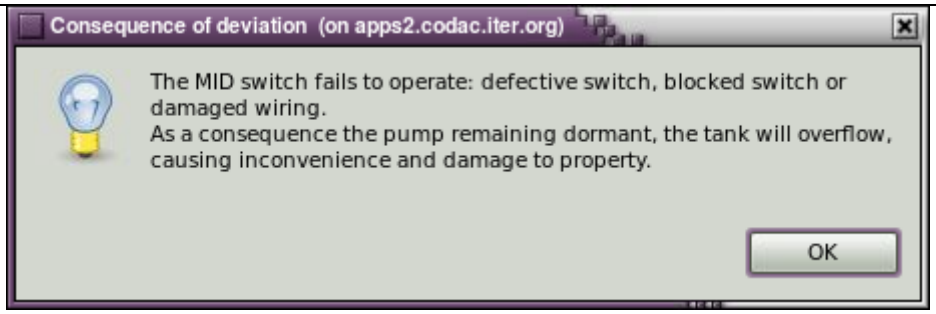
The HIGH alarm appears also in the Alarm Table View:



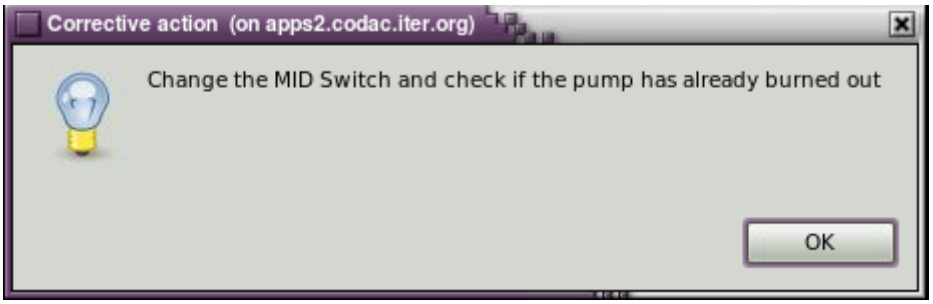
When resetting the system, the pump is started and the tank level starts to decrease.

DSP-04	**3.4.8 Alarm Guidance**
Prerequisite	1. Demo IOC running 2. Alarm Configuration imported 3. Alarm server started 4. Alarm Operator Interface started
Test Cases	1. Positive confirmation of alarm guidance
Procedure	From the Alarm Perspective, right-click on the MAJOR alarm:

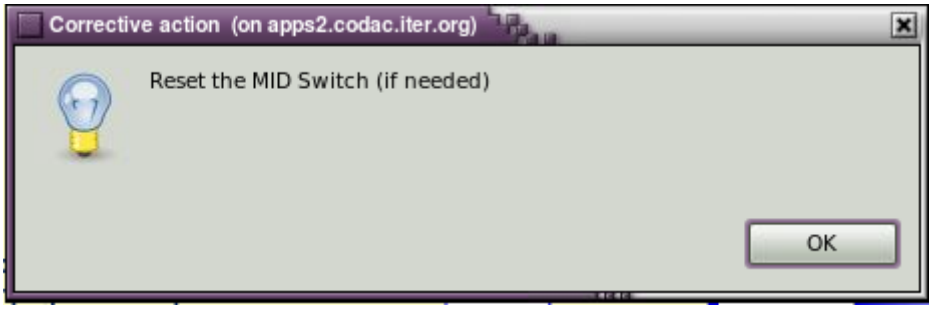
	 1. Check when (clock icon) the alarm has been triggered 2. Click on the first Guidance message “In the event that the MID switch…”. Click OK 3. Then click on the other Guidance messages: “Consequence of deviation”, “Corrective action”, “Time for response”	
Pass Criteria	2. Depending on the LOW or HIGH MAJOR alarm, the guidance is different. Guidance for TEST-BST0:LOW-SWITCH:  Guidance for TEST-BST0:HIGH-SWITCH:  3. “Consequence of deviation” for TEST-BST0:LOW-SWITCH:  “Consequence of deviation” for TEST-BST0:HIGH-SWITCH:	



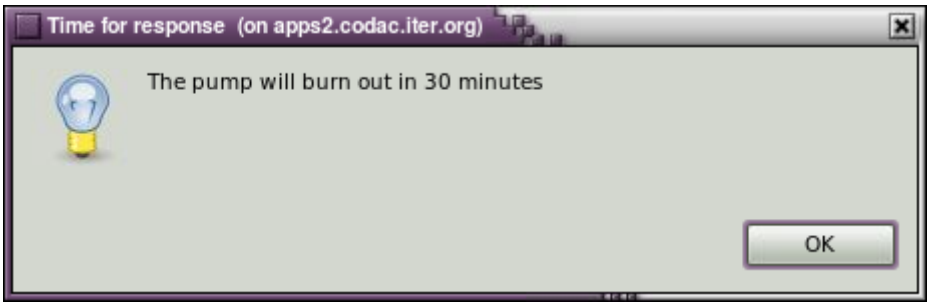
"Corrective action" for TEST-BST0:LOW-SWITCH:



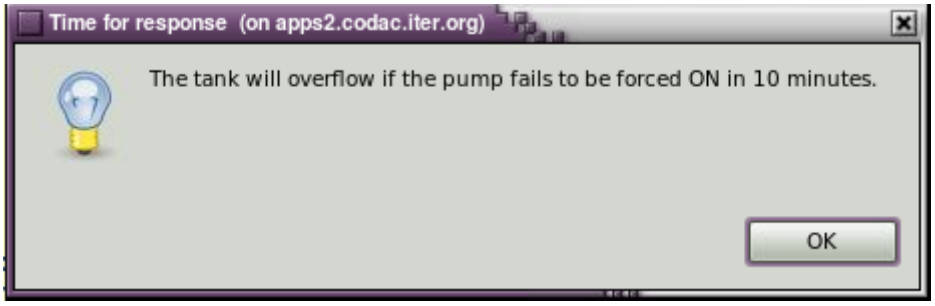
"Corrective action" for TEST-BST0:HIGH-SWITCH:

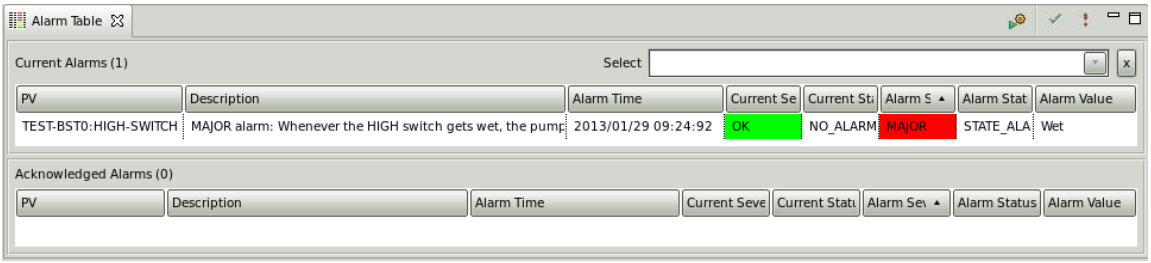
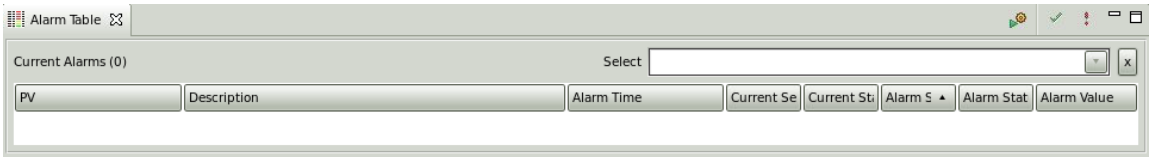
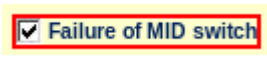


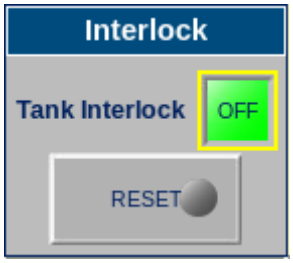
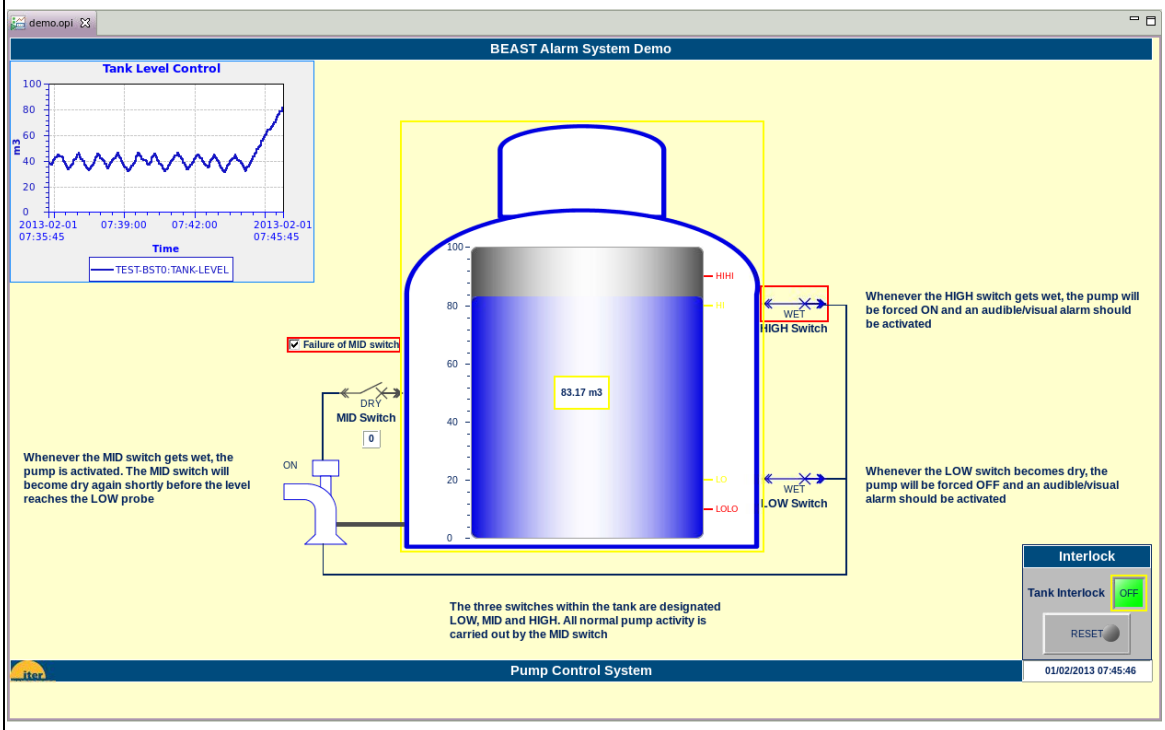
"Time for response" for TEST-BST0:LOW-SWITCH:

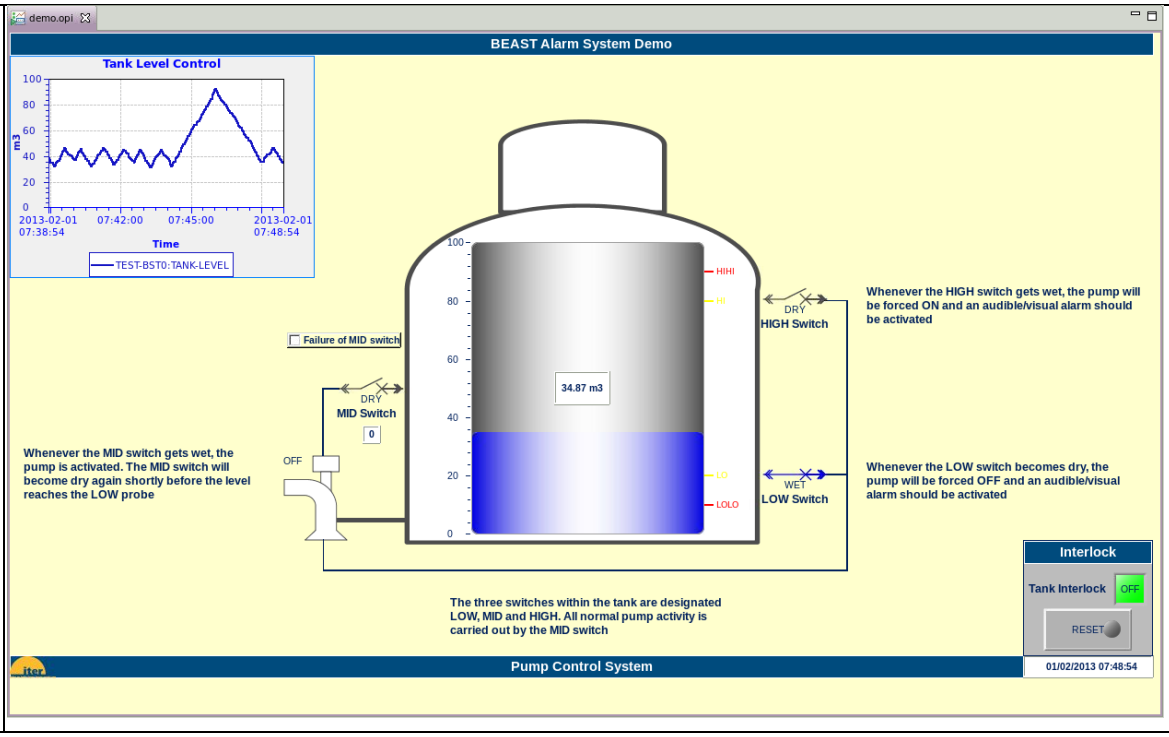


"Time for response" for TEST-BST0:HIGH-SWITCH:



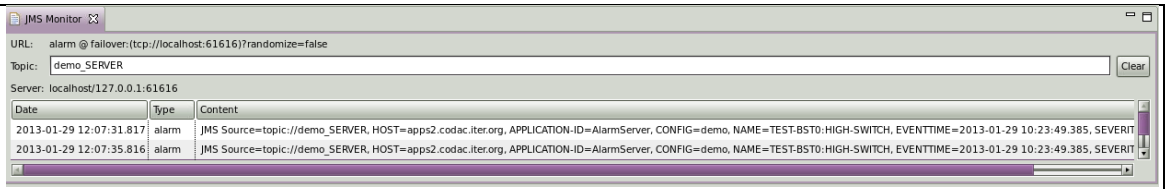
DSP-05	**3.4.9 Latched Alarm**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration imported 3. Alarm server started 4. Alarm Operator Interface started	
Test Cases	1. Positive confirmation of latched alarm	
Procedure	From the Alarm Perspective, even when the alarm has recovered (Current Severity is OK), the alarm should be still present in the Alarm Table and require to be acknowledged:  1. Right-click on the recovered alarm and select the option Acknowledge: 	
Pass Criteria	1. The recovered and acknowledged alarm is suppressed from the Current Alarms Table: 	
DSP-05	**3.4.10 Display Limit Alarms**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration imported 3. Alarm server started 4. Alarm Operator Interface started	
Test Cases	1. Positive confirmation of display limit alarms	
Procedure	In CSS, when the tank level reaches High / High-High limits or Low / Low-Low limits, the tank border reflects the display alarm status: 1. Check the box Failure of Mid switch  2. Wait for the tank level to reach an alarm limit	

	3. Reset the tank interlock by clicking on the button:  And confirm the action.	
Pass Criteria	2. According to the alarm severity, the tank border should be displayed in MINOR or MAJOR color:  After the Interlock Reset, all normal pump activity is carried out again by the MID switch:	



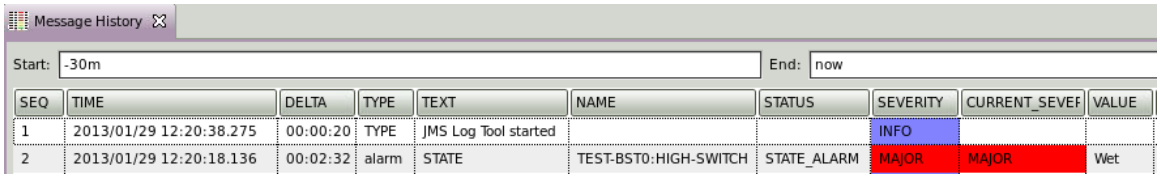
SRV-04	**3.4.11 JMS Monitor**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration Imported successfully 3. Alarm server started 4. Alarm Operator Interface started	
Test Cases	1. Positive confirmation of JMS notification messages sent	
Procedure	In CSS, monitor JMS messages sent by the “demo” alarm server: 1. Menu CSS -> Debugging -> JMS Monitor 2. In the Topic field of the JMS Monitor View, enter demo_SERVER and press <Enter> Generate an alarm by setting manually the tank level outside its limits. In the previous Linux console: 3. $ caput TEST-BST0:TANK-LEVEL 82 4. Check for new JMS messages in CSS JMS Monitor View 5. Close JMS Monitor View	

Pass Criteria	3. `caput TEST-BST0:TANK-LEVEL 82`
	`Old : TEST-BST0:TANK-LEVEL xx.xxx`
	`New : TEST-BST0:TANK-LEVEL 82`
	4. IDLE messages and the new HIGH Switch alarm notification message:	



HST-01	**3.4.12 Message History**
Prerequisite	1. Demo IOC running 2. Alarm Configuration Imported successfully 3. Alarm server started 4. Alarm Operator Interface started
Test Cases	1. Positive confirmation of alarm message history

Procedure	From the previous Linux console, edit a new jms plugin customisation ini file and enter the following org.csstudio.jms2rdb parameters:
	1. $ gedit src/main/beast/jms.ini&
	`org.csstudio.jms2rdb/httpd_port=4914`
	`org.csstudio.jms2rdb/jms_url=failover:(tcp://localhost:61616)`
	`org.csstudio.jms2rdb/jms_topic=demo_SERVER, demo_CLIENT, demo_TALK, LOG, TALK, WRITE`
	`org.csstudio.jms2rdb/rdb_url=jdbc:postgresql://localhost/css_log_3_0_0?user=log&password=$log`
	`org.csstudio.jms2rdb/rdb_schema=`
	Save the file in the editor and from the previous Linux console, start jms2rdb to log JMS messages in the RDB:
	2. $ jms2rdb -pluginCustomization src/main/beast/jms.ini&
	Generate an alarm by setting manually the tank level outside its limits. In the previous Linux console:
	3. $ caput TEST-BST0:TANK-LEVEL 82
	In CSS, view logged last 30 minutes JMS messages from RDB:
	4. CSS -> Alarm -> Message History
	5. In the Start field of the Message History View, enter -30m and press <Enter>
	6. Check that the MAJOR alarm was stored in the history

Pass Criteria	6. The output of the command should shows all the message history since 30 minutes: 

SRV-05	**3.4.13 Automated Actions**	
Prerequisite	1. Demo IOC running 2. Alarm Configuration Imported successfully 3. Alarm server and notifier started 4. Alarm Operator Interface started	
Test Cases	1. Configure an alarm to executed automated actions when triggered 2. When configured alarm is triggered, an email is automatically sent	
Procedure	In CSS, log in as codac-dev to be able to modify dynamically the alarm configuration: 1. File -> Log in… 2. Enter codac-dev as User Name and type the Password. Click on OK to validate 3. In the Alarm Tree view, right-click on PV: TEST-BST0:HIGH-SWITCH and select the option Configure Item  4. In the Automated Actions section, modify the predefined Automated Action “Contact Maintenance Service”. In the Detail field, specify your email address in place of someone@iter.org:  Click on OK to validate the new configuration in the Alarm RDB. Generate an alarm by setting manually the tank level outside its limits. In the previous Linux console: 5. $ caput TEST-BST0:TANK-LEVEL 92 6. Wait for ~5 seconds and check your inbox	
Pass Criteria	6. The email automatically sent when the configured alarm is triggered :	

The output in the Linux console looks like that:

```
xxx INFO [Thread 9663] org.csstudio.alarm.beast.notifier.WorkQueue (add) -
TEST-BST0:HIGH-SWITCH: Contact Maintenance Service => SCHEDULED: 5s
xxx INFO [Thread 9664]
org.csstudio.alarm.beast.notifier.ExecuteActionTask$ExecuteActionThread (run) -
TEST-BST0:HIGH-SWITCH: Contact Maintenance Service => EXECUTED
```

To terminate the tests, stop the demo IOC, the demo alarm server and jms2rdb. Close css:

1. $ epics> exit

2. $ ps -ef|grep demo|grep <user>

```
user    <PID1>  xxx  0 11:12 pts/14   00:00:00 /bin/bash /usr/bin/alarm-server -root demo
user    <PID2>  xxx  0 11:12 pts/14   00:00:00 /opt/codac-3.1/css/alarm-server/alarm-server -
root demo -pluginCustomization /tmp/plugin_customization.ini.5994
user    <PID3>  xxx  0 11:12 pts/14   00:00:09 /usr/bin/java -Djava.awt.headless=true -Xms64m -
Xmx256m -Declipse.exitdata= -jar /opt/codac-3.1/css/alarm-
server/plugins/org.eclipse.equinox.launcher_1.2.0.v20110502.jar …
user    <PID1>  xxx  0 11:12 pts/14   00:00:00 /bin/bash /usr/bin/alarm-notifier -root demo
user    <PID2>  xxx  0 11:12 pts/14   00:00:00 /opt/codac-3.1/css/alarm-notifier/alarm-notifier
-root demo -pluginCustomization /tmp/plugin_customization.ini.5994
user    <PID3>  xxx  0 11:12 pts/14   00:00:09 /usr/bin/java -Djava.awt.headless=true -Xms64m -
Xmx256m -Declipse.exitdata= -jar /opt/codac-3.1/css/alarm-
notifier/plugins/org.eclipse.equinox.launcher_1.2.0.v20110502.jar …
```

3. $kill -9 <PID1> <PID2> <PID3>

4. $ ps -ef|grep jms2rdb|grep <user>

```
user    <PID1>  xxx  0 13:18 pts/14   00:00:00 /bin/bash /usr/bin/jms2rdb -pluginCustomization
jms.ini
user    <PID2>  xxx  0 13:18 pts/14   00:00:00 /opt/codac-3.1/css/jms2rdb/jms2rdb -
pluginCustomization /tmp/plugin_customization.ini.9273
user    <PID3>  xxx 16 13:18 pts/14   00:00:03 /usr/bin/java -Declipse.exitdata= -jar
/opt/codac-3.1/css/jms2rdb/plugins/org.eclipse.equinox.launcher_1.2.0.v20110502.jar -os linux -
ws gtk -arch x86_64 -showsplash -launcher /opt/codac-3.1/css/jms2rdb/jms2rdb -name Jms2rdb --
launcher.library /opt/codac-
3.1/css/jms2rdb/plugins/org.eclipse.equinox.launcher.gtk.linux.x86_64_1.1.100.v20110505/eclipse
_1407.so -startup /opt/codac-
3.1/css/jms2rdb/plugins/org.eclipse.equinox.launcher_1.2.0.v20110502.jar --
launcher.overrideVmargs -exitdata 888059 -pluginCustomization
/tmp/plugin_customization.ini.9273 -vm /usr/bin/java -vmargs -Declipse.exitdata= -jar
/opt/codac-3.1/css/jms2rdb/plugins/org.eclipse.equinox.launcher_1.2.0.v20110502.jar
user    xxx  xxx  0 13:18 pts/14   00:00:00 grep jms2rdb
```

5. $ kill -9 <PID1> <PID2> <PID3>

6. In CSS, use the menu

File -> Exit

7. Delete the alarm configuration in the database:

$ alarm-configtool -root demo -delete /demo

```
Alarm Config Tool 1.0.0.codac_core_xxx
2013-01-24 09:41:10.921 INFO [Thread 19]
org.apache.activemq.transport.failover.FailoverTransport (doReconnect) - Successfully connected
to tcp://localhost:61616
Reading RDB configuration of 'demo'
Deleting existing RDB configuration for '/demo'
```

3.5 Component Test Log

CFG-01	**3.5.1 Alarm Configuration Import**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
CFG-02	**3.5.2 Alarm Configuration Export**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
SRV-01	**3.5.3 Alarm Server Startup**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
DSP-01	**3.5.4 Alarm PV Tree**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
DSP-02	**3.5.5 Alarm Related Display – Operator Interface**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
DSP-03	**3.5.6 Alarm Related Display – Data Plot**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		

SRV-02	**3.5.7 Alarm Notification**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
DSP-04	**3.5.8 Alarm Guidance**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
SRV-03	**3.5.9 Latched Alarm**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
DSP-05	**3.5.10 Display Limit Alarms**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
SRV-04	**3.5.11 JMS Monitor**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
HST-01	**3.5.12 Message History**	[PASS / FAIL]
[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		
SRC-05	**3.5.13 Automated Actions**	[PASS / FAIL]

[Bug ID]	[Bug title to briefly describe the anomaly]	
Remarks		

Software Test Plan Checklist

For Assessment of:	
Agency Name	
Project Name	
Document Name	
Date	

Criteria	Yes / No / NA
DOCUMENT STANDARDS COMPLIANCE	
1 Have standards/guidelines been identified to define the work product?	
2 Does the work product format conform to the specified standard/guideline (Template)?	
3 Has the project submitted any request for deviations or waivers to the defined work product?	
4 Have the following areas been addressed completely:	
4a Approval authority?	
4b Revision approval?	
4c Revision control?	
TECHNICAL REFERENCE	
5 Is there evidence that the work product was reviewed by all stakeholders?	
6 Have acceptance criteria been established for the work product?	
7 Does the work product have a clearly defined purpose and scope?	
8 Are references to policies, directives, procedures, standards, and terminology provided?	
9 Does the work product identify any and all constraints/limitations?	
S/W TEST PLAN CONTENTS	
10 Does the S/W Test Plan address the following required information:	
10a Test levels?	
10b Test types (e.g., unit testing, software integration testing, systems integration testing, end-to-end testing, acceptance testing, regression testing)?	
10c Test classes?	
10d General test conditions?	
10e Test progression?	
10f Data recording, reduction, and analysis?	
10g Test coverage (breadth and depth) or other methods for ensuring sufficiency of testing?	
10h Planned tests, including items and their identifiers?	

Criteria	Yes / No / NA
10i Test schedules, Requirements traceability (or verification matrix)?	
10j Qualification testing environment, site, personnel, and participating organizations?	
11 Does the S/W Test Plan identify the environmental exposure as well as requirements for comprehensive, functional, aliveness, end-to-end, and mission simulation testing?	
12 Does the S/W Test Plan provide a System Overview that describes the unique complexities of the system?	
13 Does the S/W Test Plan address user guide, operations / maintenance validation?	
16 Does the S/W Test Plan identify any elements that will not be tested according to the test plan (e.g., externally developed software)?	
17 Does the S/W Test Plan address software architecture in terms of which software components will be based on heritage and which will be mostly or entirely new developments?	
18 Does the S/W Test Plan identify any software reuse? If so, is the extent of reuse or the anticipated modification described?	
S/W TEST ENVIRONMENT	
19 Does the S/W Test Plan include a figure of each system test environment? If so, does it reflect the system hardware approach, simulators, and special development?	
20 Does the S/W Test Plan identify specific test hardware and simulators for each external interface?	
TEST TOOLS	
21 Does the S/W Test Plan address test execution tools?	
TEST PROBLEM REPORTING & CORRECTIVE ACTION	
22 Does the S/W Test Plan provide a description of the problem reporting system to be used by the test team to report problems and/or recommended changes cited during the test activities?	
TEST PROGRESS PLANNING & TRACKING	
23 Does the S/W Test Plan describe the routine test progress reporting approach?	
24 Does the S/W Test Plan describe the Build Test verification methodology? If so, does the description address build verification test level objectives, environment, roles & responsibilities, entry/exit criteria, general guidelines, build test planning, build test scenario development, build test procedure preparation & dry run, build test execution, reporting, and archiving?	