|  |
| --- |
| Intel |
| ISC Events Design Specifications |
| Version 1.0 |

|  |
| --- |
| Ali Bidabadi  ISC Team  3/9/2015 |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Author’s Name | Description | Date |
| 1.0 | Ali Bidabadi | First draft | 3/9/2015 |

Contents

[Introduction 4](#_Toc413833555)

[Requirements 4](#_Toc413833556)

[Technologies 5](#_Toc413833557)

[Architecture and Design Overview 6](#_Toc413833558)

[Email Notification Settings Workflow 6](#_Toc413833559)

[Alarm Definition and Alert Generation Workflow 6](#_Toc413833560)

[Components and Design Considerations 7](#_Toc413833561)

[Email Notification 7](#_Toc413833562)

[Alarm Definition and Alert Generation 7](#_Toc413833563)

[Scoping and High-Level Estimates 10](#_Toc413833564)

# Introduction

Intel Security Controller (ISC) lacks an event notification mechanism to alarm users of important failures in their ISC echo system. As implemented in many enterprise applications, the notion of an event framework is also quite crucial for ISC.

The fundamental idea is to allow users to define how and when they should be notified if something in the environment goes wrong. Without such a mechanism some major issues could arise without being noticed for a relatively long period of time which could have serious repercussions in the whole customer environment.

# Requirements

         Allow users to define alarms with the following information:

* The triggering logic, i.e. when alarm should be triggered (This could be on any ISC job failure)
* The severity of the alarm
* Actions that should be taken in the event of an alarm (In the first implementation we will most likely only support sending email notification).

         We could optionally have some pre-canned alarms in the ISC database.

         When alerts are generated based on user’s/pre-canned alarms, they should contain:

* Link to the job that caused the alert
* Severity
* Action specified in the alarm definition

         On a separate view, all the generated alerts should show up. Users would particularly be interested in the following:

* Acknowledge action should timestamp a selected alert
* Delete action will remove the alert from database
* Show all or show acknowledged alerts will filter accordingly

        Since email notification action will be supported, we need to allow users to configure their

email address and outgoing mail server (SMTP) on a separate dialog

         We will support alerts archiving by adding the alerts to CVS dump (We will not support this

feature in the first implementation)

# Technologies

In order to implement an ISC events feature, we can leverage some of the available Java projects listed below :

* Email notification will be supported by ISC therefore an email framework is required to implement this feature. To that end my initial research shows that one of the best available java project is JavaMail API. It defines classes which represent the components of a mail system. JavaMail does not implement an email server, instead it allows us to access an email server using a Java API. And that is exactly what we will need. For further details refer to the following links: <http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/javamail/javamail.html>

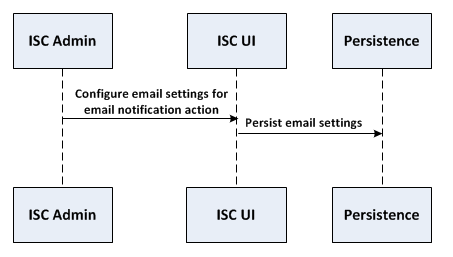
<https://javamail.java.net/nonav/docs/api/>

*Note: My initial investigation shows that different mail servers will have sometimes very different configuration that could be challenging to communicate with through one generic set of JavaMail APIs. We need to verify this later as customer environments might have very different networking/firewall setup which could impact the requirements.*

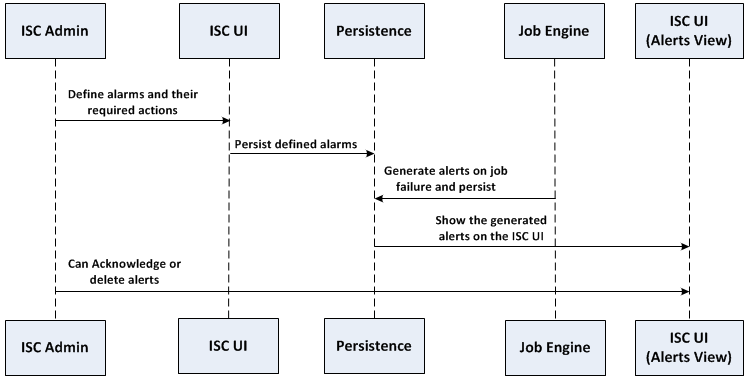
* Another important piece needed in our implementation is the notion of job listener. Fortunately ISC Job engine supports such a needed feature and we can leverage it so that we can receive alerts in the event of a job failure. The most efficient way to implement this is to have a job listener at the job engine level instead of modifying each and every job that today exists in ISC. This way at the completion of every job the listener will be invoked.

# Architecture and Design Overview

## Email Notification Settings Workflow



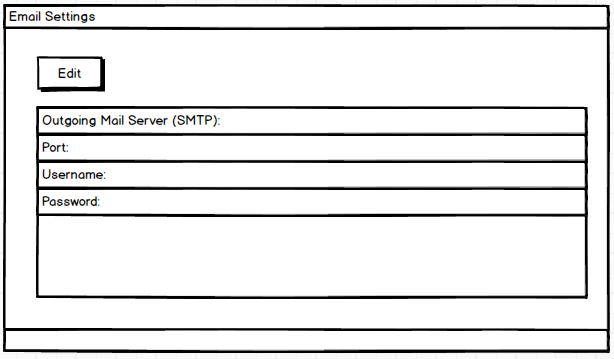
## Alarm Definition and Alert Generation Workflow



# Components and Design Considerations

## Email Notification

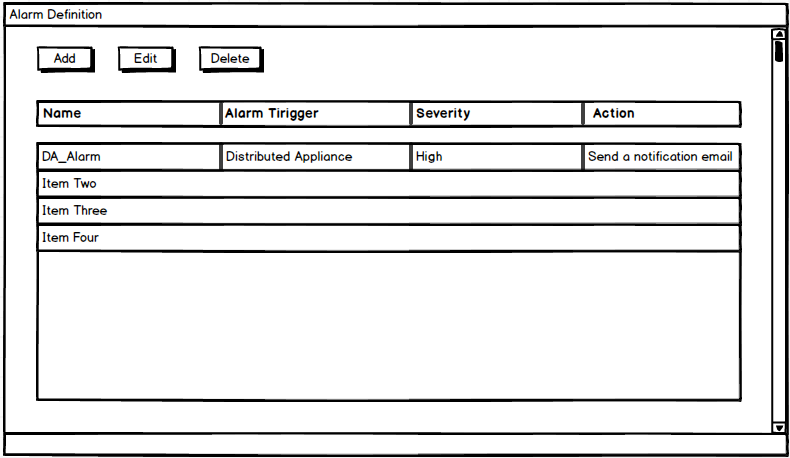
* UI Layer to allow admins to configure email settings of their organizations :
  + Outgoing mail server(SMTP) name
  + Mail server port. The traditional SMTP port is 25. However many organizations choose different ports such as 465 (SSL) , 587 (TLS), etc.
  + Username, which would be the email ID of the sender.
  + Password if required (many organizations trust all the internal domains hence no authentication is required)
  + Here is a sample email settings UI (propose this as a new tab under Manage -> Server ):

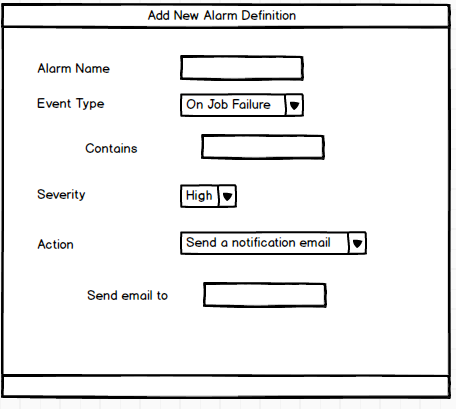


* Service Layer to invoke Persistence Layer to persist the above settings. It also needs to validate the email settings that the user entered.
* Persistence Layer: Database Schema and Data Modeling .

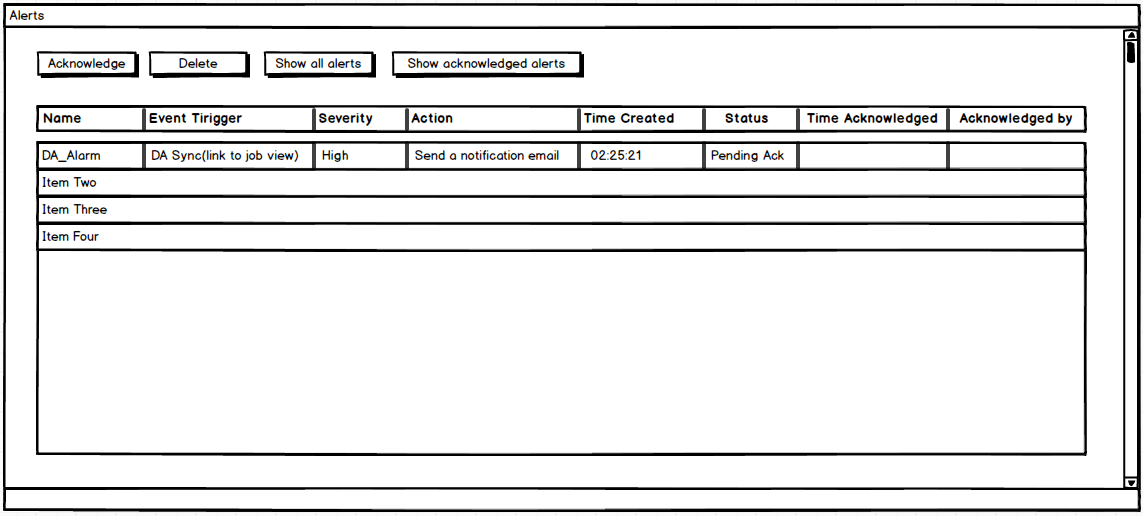
## Alarm Definition and Alert Generation

* UI Layer to allow admins to define new alarms :
  + A CRUD view to allow Add, Edit and deletion of alarms
  + Could show some pre-canned alarms
  + Each alarm should contain the triggering criteria and severity.
  + The only event type that we support for the next release would be “on job failure”.
  + Admins can define criteria with the help of RegEx.
  + A user could (must ?) associate an action to an alarm. A very common action is sending email notification. Frequency of the actions is also configurable.
  + Here is a sample alarm definition UI (propose this as a new tab under Manage):





* Service Layer to invoke Persistence Layer to persist defined alarms.
* Persistence Layer: Database Schema, Data Modeling and SQL.
* Need to introduce job listener in the Job Engine
* UI Layer to show generated alerts:
  + Allow ISC admins to either acknowledge the alerts or delete them
  + Delete means delete from database
  + Acknowledged alerts will be time stamped.
  + Order based on severity + status
  + Below is a sample Alerts UI (propose this as a new tab under Status):



* Service Layer to
  + Invoke Persistence Layer to persist generated alerts.
  + Generate and send email notifications if applicable.
* Persistence Layer: Database Schema, Data Modeling and SQL.
* Alerts archiving: Archive the generated alerts (Not for the next release)

# Scoping and High-Level Estimates

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story** | **Sub-Tasks** | **Story-Level  T-Shirt Estimate**  **(S, M, L)** | **Comment** |
| Email notifications | * UI Layer. * Service Layer. * DB Schema and Persistence Layer. | M |  |
| Alarm Definition | * UI Layer (Full CRUD) * Service Layer. * DB Schema and Persistence Layer. * Job failure listener implementation * Job Engine modification if needed | XL |  |
| Alert Generation | * UI Layer. * Service Layer. * DB Schema and Persistence Layer. * Email Infrastructure | XL | JavaMail APIs a bit non deterministic when it comes to dealing with networking/firewall settings |
| Alerts Archiving  (Not for the next release) | * Archive the generated alerts | S |  |

=== End of Document ===