|  |
| --- |
| Intel |
| ISC Jobs/Tasks Archive Design Spec |
| Version 2.0 |

|  |
| --- |
| Trinh Lee  ISC Team  12/31/2014 |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Author’s Name | Description | Date |
| 1.0 | Trinh Lee | Initial Version | 12/24/2014 |
| 2.0 | Trinh Lee | Added detailed effort estimates after review feedback. | 12/31/2014 |

Contents

[Introduction 4](#_Toc407785696)

[Technologies 4](#_Toc407785697)

[Requirements 4](#_Toc407785698)

[Architecture and Design Overview 6](#_Toc407785699)

[Components and Design Considerations 6](#_Toc407785700)

[Scheduler 6](#_Toc407785701)

[Archiving SQL and CSV File Writer 7](#_Toc407785702)

[CSV File Downloader 7](#_Toc407785703)

[Scoping and High-Level Estimates 7](#_Toc407785704)

[Detailed Effort Estimates 9](#_Toc407785705)

# Introduction

Intel Security Controller (ISC) has a job engine responsible for launching all the orchestration jobs and tasks in the backend. The ISC Job Engine also persists the states of the jobs and tasks in ISC database. As the result, the number of the jobs and tasks records will grow overtime.

In order to control such growth, we need to allow admins to be able to archive Intel Security Controller (ISC) jobs/tasks database records and purge them from our ISC database based on user-configurable schedules.

# Requirements

         Allow user to archive jobs that are older than a specific date or X days/month/year back

         Allow invocation of the above on an automated schedule

         Archive dumps jobs into a zipped-up file containing jobs/tasks only. We need to decide what would be the format of this file, but it should be import/import friendly (CSV?).

         Archived zip files are available for downloading and deletion.

~~         Consideration for later: Import archive back to db:~~

Importing Data from a CSV File into H2 Database (need to verify):

A fast way to load or import data (sometimes called 'bulk load') from a CSV file is to combine table creation with import. Optionally, the column names and data types can be set when creating the table. Another option is to use INSERT INTO ... SELECT.

CREATE TABLE TEST AS SELECT \* FROM CSVREAD('test.csv');

CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255))

AS SELECT \* FROM CSVREAD('test.csv');

Refer to this link <http://h2database.com/html/tutorial.html#csv> for more details.

# Technologies

There are some open-source projects we can leverage for our implementation. They are listed here:

* We can use Log4j framework to archive/log our jobs/tasks db records in CSV files. Log4j will automatically roll-over to the next files based on file size limit so that we don’t need to implement this roll-over mechanism.
* We can use light-weight “javacsv” open-source project, <http://mvnrepository.com/artifact/net.sourceforge.javacsv/javacsv/2.0>, to manually generate and manage CSV files ourselves. Following are sample codes:  
  <http://www.codeweblog.com/csv-format-file-java-action-javacsv-jar/>
* In addition, we can use built-in H2 CSV tools to write to a csv file as part of sql query. For example (need to verify),

Writing a CSV File from Within an H2 Database:

The built-in function CSVWRITE can be used to create a CSV file from a query. Example:

CREATE TABLE TEST(ID INT, NAME VARCHAR);

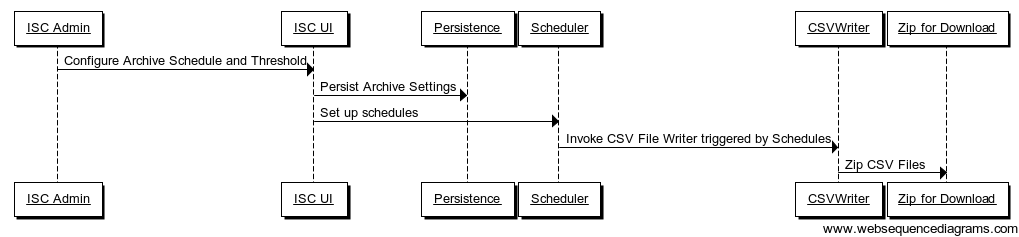
INSERT INTO TEST VALUES(1, 'Hello'), (2, 'World');

CALL CSVWRITE('test.csv', 'SELECT \* FROM TEST');

Refer to this link <http://h2database.com/html/tutorial.html#csv> for more details.

* We will use open-source Quartz, <http://www.quartz-scheduler.org/>, for scheduling purposes. Or we can use the standard built-in Java ScheduledExecutorService, <http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledExecutorService.html>, too.

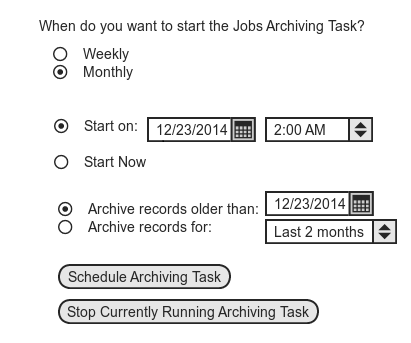
# Architecture and Design Overview



# Components and Design Considerations

## Scheduler

* UI Layer to allow admins to configure schedules and archiving thresholds:
  + Schedule Frequency: How often do we want to archive (like weekly, monthly)?
  + Schedule Start Time/Date or Start Now: When will we kick off the archive schedule?
  + Schedule Duration or Stop Now: How long do we want to run (run for 3 months, run for 4 weeks)?
  + Archive Threshold: archive records older than some specific threshold date (mm/dd/yyyy) or based on some archive period (last X days, months, and so on) or last X days/weeks/months prior to the specific threshold date.
  + Here is a sample scheduling UI:



* Service Layer to invoke Persistence Layer to query/persist the above settings.
* Persistence Layer: Database Schema, Data Modeling and E/R Design, and SQL.
* Scheduler Engine: can use open-source project “Quartz” or the standard built-in Java 1.7’s ScheduledExecutorService framework.

## Archiving SQL and CSV File Writer

* Can use Log4j, Open-Source JavaCSV, or the built-in H2 CSV Writer functions.
* Regardless of the approaches, something to keep in mind:
  + Roll-over mechanism based on size?
  + Or just a single (zipped) giant file?
  + Need to zip the files.
  + Location/folder to store the generated files.
* Need to come up with efficient SQL statements to query Jobs and Tasks database tables to extract records that older than archive threshold date. Then based on those result sets to generate CSV files.
* Need a backend job to run the above tasks.
* Need to generate “meta” file to track the db/server versions.
* The archive CSV file names will contain display date of archive creation.

## CSV File Downloader

* UI and Service to trigger CSV zipped file downloading process.

# Scoping and High-Level Estimates

|  |  |  |  |
| --- | --- | --- | --- |
| **User Story** | **Sub-Tasks** | **Story-Level  T-Shirt Estimate**  **(S, M, L)** | **Comment** |
| Scheduler | * UI Layer. * Service Layer. * DB Schema and Persistence Layer. * DB Rebuild. * Kick off schedule and schedule engine at ISC process start-up time. * Change schedule on demand at run time. * Need a backend job thread to execute schedule. | L |  |
| SQL statements to query/purge Jobs/Tasks database records | * SQL and codes to query/purge Jobs table. * SQL and codes to query/purge Tasks table. * SQL and codes to query/purge graph dependency table. | S |  |
| CSV Writer | * Query database. * Generate CSV files. * Roll-over mechanism. * Zip the files. * Purge old database records? | M to L (M+) | L if we build everything ourselves like rolling mechanism based on file size and so on…. |
| CSV Downloader | * UI Layer. * Service Layer. * Purge CSV files? | S to M (S+) |  |
| Scalability and Performance Simulation and Measurement | * Scalability Measurement Tests. * Performance Measurement Tests. | M |  |

# Detailed Effort Estimates

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task #** | **Top-Level Task** | **Area** | **Estimate  (in Weeks)** | **Comment** |
| 1 | * ISC UI views/windows to display/persist archive schedule and threshold config data. * Added new tab view to the existing ISC “Maintenance” menu. | Archive Scheduler UI Layer | 1.5 | * For now, only support archive/purge based on a specific date threshold in the past. |
| 2 | * Archive scheduler service layer, DTOs, Validation, to invoke/change scheduler engine. | Scheduler Service Layer | 0.5 |  |
| 3 | * Archive scheduler persistence layer, db schema, db upgrade/rebuild, SQLs to query/persist schedule data. | Scheduler Persistence Layer | 1 |  |
| 4 | * Create Quartz-based job and schedule trigger. * Create ISC workflow job/task and service. * Logics to init/stop/change scheduler engine. | Scheduler Engine | 1.5 |  |
| 5 | * Use H2 CSV writer. * Create SQLs to query/purge the 4 job/task related db tables. * Need to generate “meta” file for tracking ISC db/server versions. | CSV Writer | 1.5 | * If H2 CSV writer does not work as expected, we can use open-source JavaCSV. * Assume for now for simplicity, we only maintain a single zip file, i.e. new archive files will overwrite old ones (just like the current backup/restore feature) |
| 6 | * UI to trigger download. * Zip the CSV files. | CSV Zip Downloader | 1 |  |

=== End of Document ===