**PICK PMR Insights Collective Tool**

Software Design Document

<1.0.3>

<03/30/2020>

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

|  |  |
| --- | --- |
| Initial Release: | 1.0.0 |
| Current Release: | 1.0.3 |
| Indicator of Last Page in Document: | & |
| Date of Last Review: | 05/09/20 |
| Date of Next Review: | 05/13/20 |
| Target Date for Next Update: | 05/14/20 |

Distribution List

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

**Guidance Team Members:**

Jake Lasley

Steven Roach

**Clients:**

Dr. Oscar Perez

Vincent Fonseca

Herandy Denisse Vazquez

Baltazar Santaella

Florencia Larsen

Erick De Nava

**Software Team Members:**

Jay James

Adrian Sosa

Cristian Molina

Sergio Nogami

Jesus Gomez

Change Summary

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 3/8/20 | ALL | Creation of Document |
| 1.0.1 | 3/25/20 | ALL | Divided work and completed document. |
| 1.0.2 | 03/30/20 | ALL | Completed Database Schema, Collaboration Diagrams and Documentation |
| 1.0.3 | 5/9/2020 | ALL | Made SDD match the final product |

Table of Contents

[Document Control ii](#_Toc226283109)

[Approval ii](#_Toc226283110)

[Document Change Control ii](#_Toc226283111)

[Distribution List ii](#_Toc226283112)

[Change Summary ii](#_Toc226283113)

[1. Introduction 1](#_Toc226283114)

[1.1. Purpose and Intended Audience 1](#_Toc226283115)

[1.2. Scope of Product 1](#_Toc226283116)

[1.3. References 1](#_Toc226283120)

[1.4. Definitions, Acronyms, and Abbreviations 2](#_Toc226283121)

[1.4.1. Definitions 2](#_Toc226283122)

[1.4.2. Acronyms 2](#_Toc226283123)

[1.4.3. Abbreviations 2](#_Toc226283124)

[1.5. Overview 2](#_Toc226283125)

[2. Decomposition Description 4](#_Toc226283126)

[2.3. Subsystem Description 9](#_Toc226283129)

[3. Detailed Description of Subsystem Graphing 11](#_Toc226283131)

[4. Detailed Description of Subsystem Ingestion 16](#_Toc226283132)

[5. Detailed Description of Subsystem Management 24](#_Toc226283133)

[4. DatabaseSchema 26](#_Toc226283139)

# Introduction

## Purpose and Intended Audience

The purpose of the Software Design Document (SDD) is to describe the protocols that will be used in the PICK PMR Insight Collective Knowledge system, as well as to identify the different components of the system, collaborations, responsibilities, and contracts. This document also defines how we shall delegate implementations to programmers and will show how we shall derive the tests for pre and post-conditions. The SDD gives us an improved perspective on the design of the system.

The intended audience for the SDD is the following:

**Guidance Team Members:**

Jake Lasley

Steven Roach

**Customer:**

Dr. Oscar Perez

Vincent Fonseca

Herandy Denisse Vazquez

Florencia Larsen

Erick De Nava

**Software Team Members:**

Jay James

Adrian Sosa

Cristian Molina

Sergio Nogami

Jesus Gomez

## Scope of Product

The Lethality Survivability & Human Systems Integration Directorate (LSH) works with the Department of Defense (DOD) to perform operational testing on technology systems to provide secure, resilient capabilities in the expected operational environment. The LSH or White team will analyze the flow of events that occurred during adversarial attacks between the Red and Blue teams to validate what happened during the attack. The Red team performs the attacks and the Blue team tries to mitigate them. The scope of PICK PMR Insights Collective tool will cover the need of the LSH to draw relationships between the events that occurred between the Red and Blue teams. The PMR tool will not draw the correlation of events itself but rather ease the process of drawing the correlations for the analyst.

## References

[1] ￼Elsa Tai (2018), “Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK)

Software Requirements Specification ” UTEP, Sept 2020.

[2] O. Perez et al, Requirements Definition Document, Lethality, Survivability and HSI Directorate, 2019.

[3] “Components and Containers in AWT”. Internet: [https://www.cs.utexas.edu/~mitra/csSpring2009/cs313/lectures/GUIComponents.html](about:blank), 2009 [Jan. 28, 2019]

## Definitions, Acronyms, and Abbreviations

Table 1 shall provide the definitions for terms that are used throughout this document and our project overall.

These are terms that were also mentioned in the course as well as the SRS document.

### Definitions

|  |  |
| --- | --- |
| **Class** | A representation of an object (i.e. a template definition of the methods and variables on an object) |
| **Responsibility** | What an object knows or does |
| **Contract** | Set of cohesive responsibilities |
| **Protocol** | Set of signatures for methods to be implemented |
| **Collaboration** | Set of cohesive responsibilities |
| **Subsystem** | Set of Cohesive classes that collaborate among themselves to assist a set of contracts |
| **Pre-Condition** | Capture the conditions that must be true for the method to execute correctly |
| **Post-Condition** | Section that states what is true when the method completes execution |

### Acronyms

|  |  |
| --- | --- |
| **SDD** | Software Design Document |
| **PICK** | PMR Insight Collective Knowledge |
| **SRS** | Software Requirement Specification document |
| **ARL** | Army Research Lab |
| **OCR** | Optical Character Recognition software |
| **GUI** | Graphical User Interface |

### Abbreviations

|  |  |
| --- | --- |
| **e.g.** | For example |
| **i.e.** | That is |
| **TBD** | To be determined |
| **DB** | Database |

## Overview

The SDD will describe the decomposition of the system, into its components and subsystems and identify collaborations, responsibilities, and contracts.

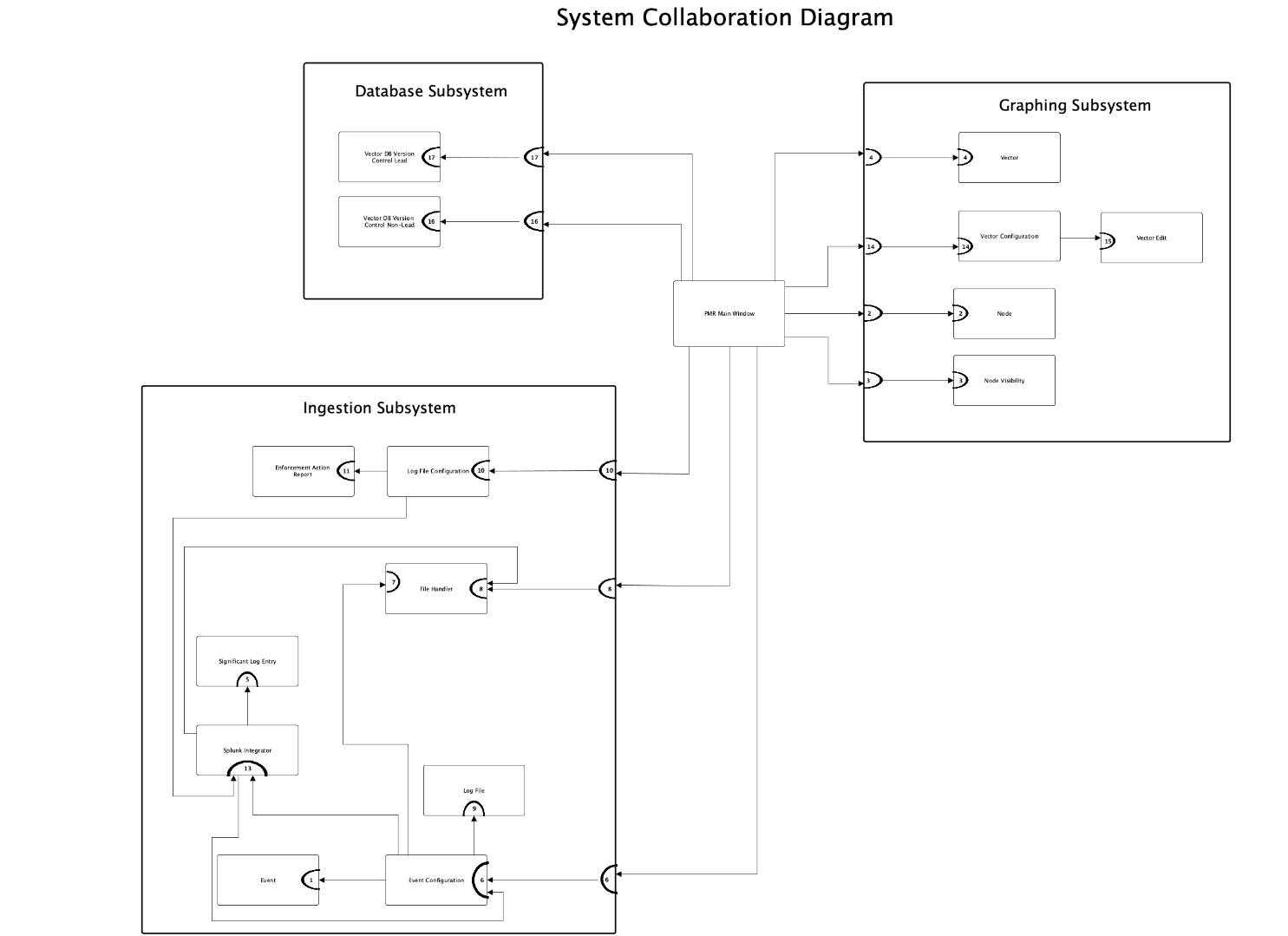
The SDD has the following sections:

1. Decomposition Description
   * This section provides the Subsystem Collaboration Graph and a detailed description of the classes and contracts that contribute to its execution.
2. Detailed Description of Graphing Subsystem
   * This section provides a detailed description of the System Collaborations Graph and their components.
3. Detailed Description of Ingestion Subsystem
   * This section provides a detailed description of how the system ingest, cleanses and validates log files. It also describes how log files are turned into a set of log entries after validation and cleansing.
4. Database
   * This section provides an outline to the database schema used to save the components of the system.

# Decomposition Description

Our system was designed following these steps: first, we scanned the SRS looking for nouns and verbs. This gave us an idea of what classes and responsibilities would be needed in the system and we came up with a mapping from nouns to classes and verbs to responsibilities. Then we decided what responsibilities would belong to which classes. Each class description gives a summary of what their responsibilities consist of and whether there were any instances where the classes needed help from other classes. However, the design we came up using this procedure was scrapped. The actual design seen in this document was achieved through analyzing our code, figuring out all the responsibilities, collaborations and contracts of each class. Then, after having how each class relates to each other, we started thinking about how to group classes into subsystems. We realized that we have three main subsystems: the ingestion, graphing and database. Ingestion subsystem takes care of ingestion, cleansing and validation of log files. The graphing subsystem takes care of displaying the graph information such as adding and editing nodes. The database subsystem is responsible for helping with system persistence.

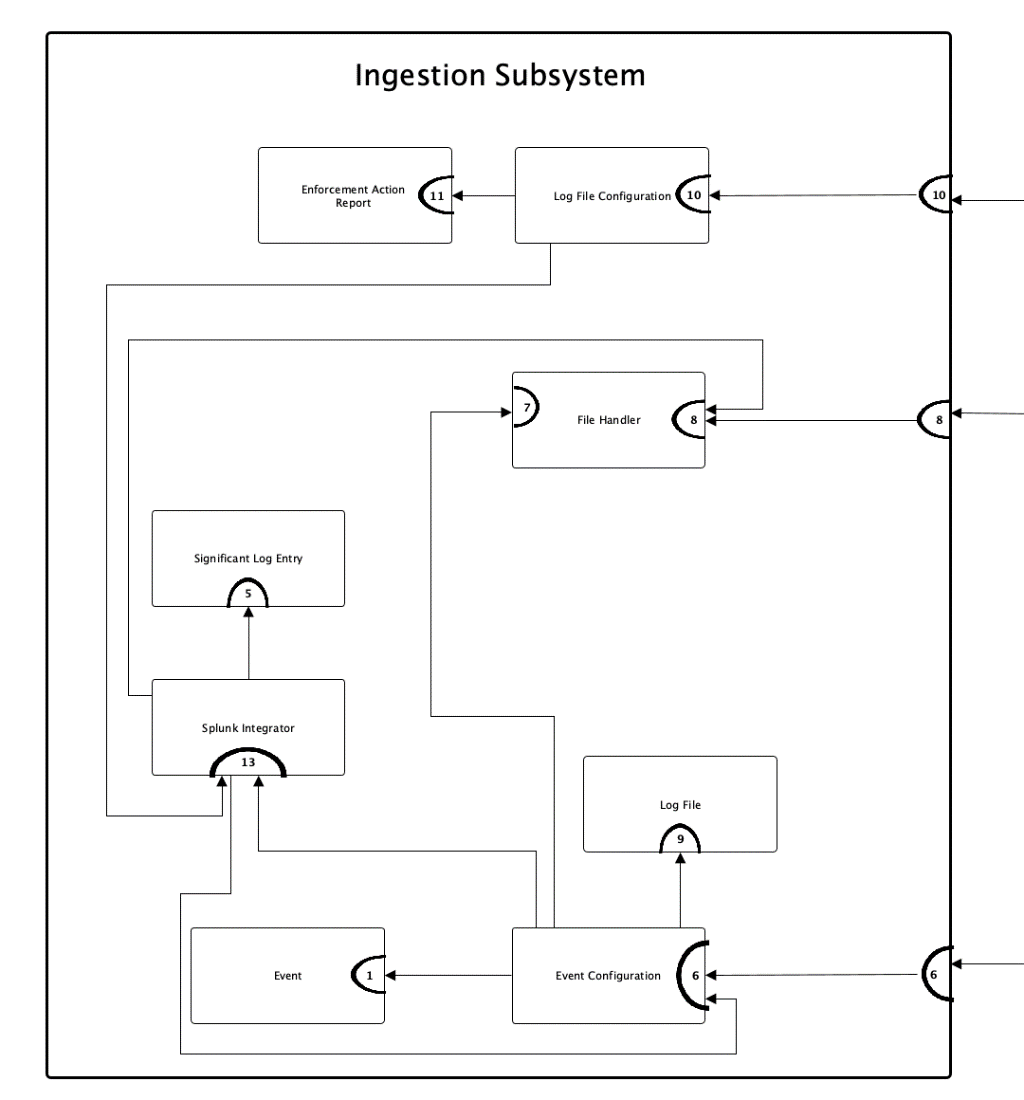
## System Collaboration Diagram



The picture above is the system collaboration graph with the major subsystems present. The major subsystems are Ingestion Subsystem, Graphing Subsystem, and Database Subsystem.

## Subsystem and Component Descriptions

## 2.2.1. Subsystem Name: Ingestion Subsystem

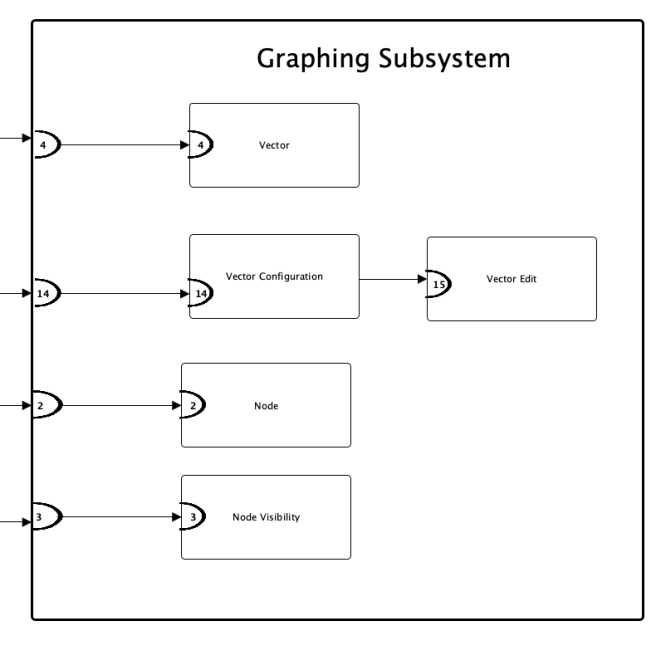


|  |
| --- |
| **Classes:**   * Log File Configuration * Enforcement Action Report * File Handler * Log File * Significant Log Entry * Splunk Integrator * Event * Event Configuration |
| **Description:** The ingestion subsystem handles the ingestion of files into the system, including all cleansing, transcription, and validation done on the files. |
| **Contracts:**  **1. Get Event Information**   * Knows the event name * Knows the event description * Knows the event start timestamp * knows the event end timestamp * Knows the root directory * Knows the red team folder * Knows the white team folder * Knows the blue team folder * Knows the host machine * Knows the lead IP address * Knows the number of established connections   **5. Get Log Entry Information**   * Knows the log entry number * Knows the timestamp. * Knows the log entry content. * Knows the host / IP address of creator. * Knows the source of an entry (the name and location from which a particular activity originates) * Knows the source type.   **6. Validate**   * Validate credentials * Begin ingestion   **7. Convert File Format**   * Convert file from audio to text * Convert file from video to audio * Convert file from image to text   **8. Cleanse and Validate**   * Cleanse files * Validate files   **9. Get Log Attributes**   * Knows the log file name. * Knows the log file cleansing status * Knows the log file validation status * Knows the log file ingestion status * Knows the log file acknowledgment status |
| **10. Log Files Report**   * Create table of log files with validation, ingestion, and cleansing status.   **11. Enforcement Action Report**   * Create Enforcement Action Report Table   **12. Share Log File Path**   * Share Log File Path |

**13. Manage Log Files**

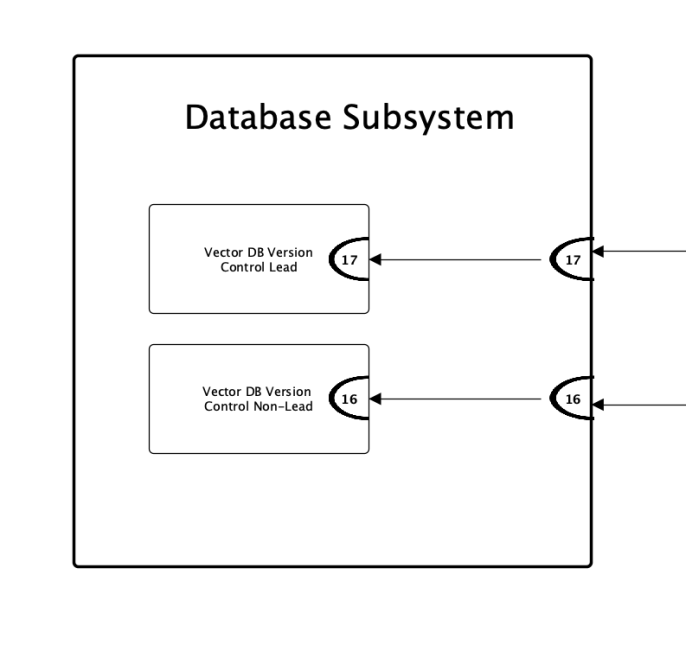
* Upload Log File
* Download Log Files
* Cleanse Log File
* Validate Log File

## 2.2.2. Subsystem Name: Graphing Subsystem



|  |
| --- |
| **Classes:**   * Vector * Vector Configuration * Vector Edit * Node * Node Visibility |
| **Description:** The Graphing Subsystem is responsible for creating a graph composed of nodes that identify a relationship between a log entry and a vector. |
| **Contracts:**  **2. Get Node Information**   * Knows its ID * Knows its name * Knows its timestamp * Knows its description * Knows its log entry reference * Knows its log creator * Knows its icon type * Knows its source * Knows its visibility     **3. Get Node Visibility Information**   * Know the node visibility * know the node id visibility * know the name visibility * Know the timestamp visibility * Know the description visibility * Know the log entry reference visibility * know the log creator visibility * know the event type visibility * know the icon type visibility * know the source type visibility   **4. Get Vector Information**   * Knows its name * Knows its description     **14. Manage Vector**   * Add Vector * Delete Vector * Check Vector * Sort Vector * Edit Vector   **15. Edit Vector**   * Edit Vector name * Edit Vector description * Save Vector Changes |

## 2.2.3. Subsystem Name: Database Subsystem



|  |
| --- |
| **Classes:**   * Vector DB Version Control Non-Lead * Vector DB Version Control Lead |
| **Description:** Configure the connection to the system, the file directory structure, the Splunk integration, and control the access for committing, pushing, and pulling graphs and vectors into the Vector Database. |
| **Contracts:**  **16. Manage Local DB**   * Connect to Mongo DB * Push object into Mongo DB * Pull object from Mongo DB   **17. Commit to Database**   * Connect to MongoDB * Load commits from MongoDB * Commit pushed changes into the MongoDB |

## Dependencies

A dependency is a piece of software(s) that the PICK-PMR Insight Collective Knowledge relies on to function as it is expected. The following are all the dependencies of the PICK system.

**OS:** The system depends on Kali Linux because it is a requirement for the system to run on Kali Linux

**Tesseract Optical Character Recognition Software (OCR):** The system depends on two pieces of software to translate text to speech: tesseract (a popular OCR for python)

**Ffmpeg:** the system depends on ffmpeg to transcribe audio and video to text

**Enterprise Splunk:** The system depends on Enterprise Splunk because this software aids on the ingestion, cleansing and validation processes. See Detailed Description of Subsystem Ingestion for more detailed information on how Splunk aids in these topics.

**Python Version**: Python 3.6 and up is needed because the library PyQt5 only works with Python 3.6 and up.

**Python Libraries:**

**PyQt5:** The system is dependent on this library because this library builds the Graphical User Interface.

**SpeechRecognition:** The system depends on this library to convert audio to text

**Datefinder:** The system depends on this library to extract timestamps from text for log file validation

**Moviepy:** The system depends on this library to convert video into audio.

**PocketSphinx:** The SpeechRecognition library depends on this library to convert audio to text offline.

**Pydub:** Converts mp3 to audio formats that SpeechRecognition can use.

**Pytesseract:** The system depends on this library to extract text from images.

**Python-dateutil:** The system depends on this library to convert strings into datetime formats.

**Regex:** The system depends on this library to find any unwanted characters during the cleansing operations.

**Splunk-SDK:** The system depends on this library to ingest log files.

**Package Version Latest Version**

|  |  |  |
| --- | --- | --- |
| PyQt5 | 5.14.1 | 5.14.1 |
| SpeechRecognition | 3.8.1 | 3.8.1 |
| datefinder | 0.7.0 | 0.7.0 |
| moviepy | 1.0.2 | 1.0.2 |
| pocketsphinx | 0.1.15 | 0.1.15 |
| pydub | 0.23.1 | 0.23.1 |
| pytesseract | 0.3.3 | 0.3.3 |
| python-dateutil | 2.8.1 | 2.8.1 |
| regex | 2020.2.20 | 2020.2.20 |
| splunk-sdk | 1.6.12 | 1.6.12 |

Dependencies on Splunk, OCR, Transcription Software, and Graphviz. Component dependencies will impact development by ensuring that these components of the system are able to be integrated in their current version.

# Detailed Description of Graphing Subsystem

The graphing component integrates the graphing function into the system and provides the analyst with the ability to manipulate the graph.

## 3.1 Class Description: Node

|  |  |
| --- | --- |
| **Class Name**: Node |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 2:** Get Node Information |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows its ID 2. Knows its name 3. Knows its timestamp 4. Knows its description 5. Knows its log entry reference 6. Knows its log creator 7. Knows its icon type 8. Knows its source 9. Knows its visibility | PMR (8) |

### 3.1.1. Contract: Get Node Contents

**Protocol:** get\_node\_id(self)

**Pre-condition:** self must be of type Node. Node must not be null.

**Post-condition:** Return the ID of a node.

**Description:**  This method returns the ID attribute of a node.

**Protocol:** get\_node\_name(self)

**Pre-condition:** self must be of type Node. Node must not be null.

**Post-condition:** Return the name of a node.

**Description:**  This method returns the name attribute of a node.

**Protocol:** get\_node\_timestamp(self)

**Pre-condition:** self must be of type Node. Node must not be null.

**Post-condition:** Return the timestamp of a node.

**Description:**  This method returns the timestamp attribute of a node.

**Protocol:** get\_description(self)

**Pre-condition:** self must be of type Node. Node must not be null.

**Post-condition:** Return the description of a node.

**Description:**  This method returns the description attribute of a node.

**Protocol:** get\_log\_entry\_reference(self)

**Pre-condition:** self must be of type Node. Node must not be null and a significant log entry must exist.

**Post-condition:** Return the log entry reference of a node.

**Description:**  This method returns the log entry reference attribute of a node.

**Protocol:** get\_log\_creator(self)

**Pre-condition:** self must be of type Node. Node must not be null and a significant log entry must exist.

**Post-condition:** Return the log entry creator of a node.

**Description:**  This method returns the log entry creator attribute of a node.

**Protocol:** get\_event\_type(self)

**Pre-condition:** self must be of type Node. Node must not be null and type must exist.

**Post-condition:** Return the type of the event on the node.

**Description:**  This method returns the event type attribute of a node.

**Protocol:** get\_icon\_type(self)

**Pre-condition:** self must be of type Node. Node must not be null and icon must exist.

**Post-condition:** Return the icon type visibility of a node.

**Description:**  This method returns the icon type attribute of a node.

**Protocol:** get\_source(self)

**Pre-condition:** Node must not be null and significant log entry must exist.

**Post-condition:** Return the source of a node.

**Description:**  This method returns the ID attribute of a node.

**Protocol:** get\_visibility(self)

**Pre-condition:** Node must exist.

**Post-condition:** Return the visibility of a node.

**Description:**  This method returns the visibility attribute of a node.

## 3.2 Class Description: Node Visibility

|  |  |
| --- | --- |
| **Class Name**: Node Visibility |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities: None** |  |
| **Contract 3:** Get Node Visibility Information | **Collaborations** |
| **Responsibilities:**   1. Know the node visibility 2. know the node id visibility 3. know the name visibility 4. Know the timestamp visibility 5. Know the description visibility 6. Know the log entry reference visibility 7. know the log creator visibility 8. know the event type visibility 9. know the icon type visibility 10. know the source type visibility | PMR (8) |

### 3.2.1. Contract: Get Event Information

**Description:** We don’t have getters and setters, variables in this class are public.

## 3.3 Class Description: Vector

|  |  |
| --- | --- |
| **Class Name**: Vector |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 4:** Get Vector Information |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows its name 2. Knows its description | PMR (7) |

**Note:** this class is deprecated and does not serve a purpose in the final product of the system. This file should be deleted but due to time constraints we are leaving it as it is.

### 3.3.1 Contract: Get Vector Information

**Protocol:** get\_vector\_name(self)

**Pre-condition**: self must be a Vector. The event configuration procedure must have been completed and vector must not be null.

**Post-condition**: Returns the name of the vector.

**Description**: This method returns the name of a vector.

**Protocol:** get\_vector\_description(self)

**Pre-condition**: Self must be a Vector. The event configuration procedure must have been completed and vector must not be null.

**Post-condition**: Returns the description of the vector

**Description**: This method returns the description of a vector.

## 3.4 Class Description: Graph

|  |  |
| --- | --- |
| **Class Name**: Graph |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities**  Knows how to select nodes  Knows how to select an edges  Knows how to invoke a node  Knows how to invoke an edge  knows how to remove a node  Knows how to remove an edge  Knows how to build itself  Knows how to move nodes  Knows how to save the graph  Know how to make a new graph  Knows how to load a new graph  Knows how to add nodes  Knows how to remove nodes  Knows how to add edges  Knows how to filter  Knows how to export the graph as an image |  |

This Graph class is not part of the collaboration diagram because it only serves as a GUI representation of the graphing subsystem. It portrays no collaborations.

## 3.5 Class Description: Vector Configuration

|  |  |
| --- | --- |
| **Class Name**: Vector Configuration |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 14:** Manage Vector |  |
| **Private Responsibilities:**   1. Add Vector 2. Delete Vector 3. Check Vector 4. Sort Vector 5. Edit Vector | PMR (5)  PMR (7) |

### 3.5.1 Contract: Manage Vector

**Protocol:** update\_vector(self)

**Pre-condition:** A vector must exist. Self must be a vector object.

**Post-condition:** The given vector will get updated.

**Description:** This method updates the vector.

**Protocol:** delete\_vector(self)

**Pre-condition:** A vector must exist. Self must be a vector object.

**Post-condition:** The given vector gets deleted from the configuration.

**Description:** this method deletes vectors

**Protocol:** add\_vector(self)

**Pre-condition:** A vector must exist. Self must be a vector object.

**Post-condition:** a new vector will be added.

**Description:**this method lets the user add vectors

**Protocol:** edit\_vector(self)

**Pre-condition:** A vector must exist. Self must be a Vector object.

**Post-condition:** the vector editor will open and let the user modify the vector fields.

**Description:**this method lets the user edit vectors

## 3.6 Class Description: Vector Edit

|  |  |
| --- | --- |
| **Class Name**: VectorEdit |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** |  |
| **Contract 15:** Edit Vector |  |
| **Responsibilities** | **Collaborations** |
| 1. Edit Vector name 2. Edit Vector description 3. Save Vector Changes | Vector Configuration (14) |

### 3.6.1 Contract: Edit Vector

**Protocol:** initUI(self)

**Pre-condition:** self must be a VectorEdit object. The Vector Configuration table has been created.

**Post-condition:** The name and description of the table are replaced by the user input. GUI Components are updated.

**Description:**  This method will allow the user to edit a Vector’s name and description in the Vector Configuration.

**Protocol:** submitted(self)da

**Pre-condition:** self must be a VectorEdit object. Table cell has been selected and the button “Submit” has been clicked. User has edited the vector name or description in the Vector Configuration.

**Post-condition:** String in variable Name and Desc are copied into the table as input.

**Description:**  This method will insert the edited string in the table cell in Vector Configuration into the table as input.

# Detailed Description of Subsystem Ingestion

The ingestion subsystem allows the PICK-PMR project to ingest mp3, image, video and text files into the system. It also cleanses and validates such log files before ingesting them.

## 4.1 Class Description: Significant Log Entry

|  |  |
| --- | --- |
| **Class Name**: Significant Log Entry |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 5:** Get Log Entry Information |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows the log entry number 2. Knows the timestamp. 3. Knows the log entry content. 4. Knows the host / IP address of creator. 5. Knows the source of an entry (the name and location from which a particular activity originates) 6. Knows the source type. | Splunk Integrator (13) |

### 4.1.1 Contract: Get Log Entry Information

**Protocol:** get\_log\_entry\_number(self)

**Pre-condition:** self must be a Significant Log Entry

**Post-condition:** returns the log entry number (an integer) of the log entry specified by self

**Description:** This method returns the unique identifier of a log entry

**Protocol:** get\_log\_entry\_timestamp(self)

**Pre-condition:** self must be a Significant Log Entry

**Post-condition:** returns the log entry timestamp (a string) of the log entry specified by self

**Description:** This method returns the time and date when the activity described by the log entry took place.

**Protocol:** get\_log\_entry\_content(self)

**Pre-condition:** self must be a Significant Log Entry

**Post-condition:** returns the log entry content (a string) of the log entry specified by self

**Description:** This method returns the content (i.e. a description of the activity) of the significant log entry specified by self.

**Protocol:** get\_host(self)

**Pre-condition:** self must be a Significant Log Entry

**Post-condition:** returns the IP address (a string) associated with the significant log entry defined by self.

**Description:** This method returns the IP address of the specified significant log entry

**Protocol:** get\_source(self)

**Pre-condition:** self must be a Significant Log Entry

**Post-condition:** returns the source (a string; the name and location from which a particular activity orginates) specified by self

**Description:** This method returns the source of a significant log entry

**Protocol:** get\_source\_type(self)

**Pre-condition:** self must be a Significant Log Entry

**Post-condition:** returns the source type (a string referring to how splunk processes the incoming data) of the log entry specified by self

**Description:** This method returns the source type of a log entry

## 4.2 Class Description: Event

|  |  |
| --- | --- |
| **Class Name**: Event |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities: None** |  |
| **Contract 1:** Get Event Information |  |
| **Responsibilities:**   * 1. Knows the event name   2. Knows the event description   3. Knows the event start timestamp   4. knows the event end timestamp   5. Knows the root directory   6. Knows the red team folder   7. Knows the white team folder   8. Knows the blue team folder   9. Knows the host machine   10. Knows the lead IP address   11. Knows the number of established connections | Event Configuration (6) |

### 4.2.1. Contract: Get Event Information

**Description:** We don’t have getters and setters, variables in this class are public.

## 4.3. Class Description: Event Configuration

|  |  |
| --- | --- |
| **Class Name**: Event Configuration |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:**   1. Validate root structure 2. Validate timestamp |  |
| **Contract 6:** Validate |  |
| **Responsibilities** | **Collaborations** |
| 1. Validate credentials 2. Begin ingestion | Splunk Integrator (13)  PMR (2)  PMR (3)  PMR (10)  PMR (11) |
| **Contract 12:** Share Log File Path |  |
| Responsibilities | Collaborations |
| 1. Share Log Files Path |  |

### 4.3.1 Contract: Get Event Data

**Protocol:** validate\_timestamp(self)

**Pre-condition:** self must be an EventConfigurationWindow object. Event details must have been entered and save button must have been pressed.

**Post-condition:** The chosen timestamps are validated. If validation fails, exceptions are returned.

**Description:**  The method will validate the timestamp ranges entered in the event configuration to ensure the start timestamp should be less than the end timestamp.

**Protocol:** validate\_credentials(self)

**Pre-condition:** self must be an EventConfigurationWindow object. The splunk configuration file must be configured.

**Post-condition:** Returns exceptions for IP, port, or analyst type entered.

**Description:**  This method opens a file dialog and allows a user to select the directory containing the files they want to ingest.

**Protocol:** open\_file(self)

**Pre-condition:** self must be an EventConfigurationWindow object. The user must have previously clicked the folder icon within the Event Configuration window

**Post-condition:** Saves the path of the root file and if there are folders corresponding to the teams, it will automatically put their names into the text fields.

**Description:**  This method opens a file dialog and allows the user to select a directory.

**Protocol:** validate\_root\_structure(self)

**Pre-condition:** A directory must be selected for the root directory.

**Post-condition:** The method validates the entered root directory and returns any exceptions or allows the user to begin ingesting if there are none.

**Description:**  This method will validate whether the chosen root directory contains at least three folders which contains the names 'red', 'blue', and 'white' respectively.

**Protocol:** begin\_ingestion(self, count)

**Pre-condition: C**ount must be an integer. Root structure must be validated.

**Post-condition:** The method ingests log files into the system.

**Description:**  This method ingests log files into the system, populating the log entry table. The count given is the number of entries.

### 4.3.2 Contract: Share Log File Path

**Protocol:** download (self, files, client, start, end, logs)

**Pre-condition:** The parameters provided must not be null and Ingestion process has initiated. Client must be a SplunkIntegrator object, start must be a string, end must be a string, logs must be a list.

**Post-condition:** The status of log file ingestion is illustrated in a table at real time. Throws Authentication Error exception if not logged in.

**Description:**  This method will call cleansing and validation methods for each file, and upload successful files, and illustrate the user the log file configuration results in a table as ingestion is happening.

## 4.4 Class Description: File Handler

|  |  |
| --- | --- |
| **Class Name**: File\_Handler |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities: N**one |  |
| **Contract 7: Convert File Format** |  |
| **Responsibilities** | **Collaborations** |
| 1. Convert file from audio to text 2. Convert file from video to audio 3. Convert file from image to text | Event Configuration (12) |
| **Contract 8: Cleanse and Validate File** |  |
| **Responsibilities** | **Collaborations** |
| 1. Cleanse files 2. Validate files | Splunk Integrator (13)  PMR (10) |

### 4.4.1 Contract: Convert File Format

**Protocol:** convert\_audio\_to\_text(self, audio\_file)

**Pre-condition:**  self must be a fileConverter object ,audio\_file must be the path of the audio file to be converted to text.

**Post-condition:** The file specified by audio\_file is converted to a text form

vert an audio file to text.

**Protocol:** convert\_image\_to\_text(self, image\_file)

**Pre-condition:** self must be a fileConverter object, image\_file must be the path of the audio file to be converted to text.

**Post-condition:** returns the path to the converted audio file.

**Description:** The method will use tesseract to convert the image to a text format.

**Protocol:** convert\_video\_to\_audio(self, video\_file)

**Pre-condition:**  self must be a fileConverter object, video\_file must be the path of the audio file to be converted to text.

**Post-condition:** The file is converted to text format and returned.

**Description:** The method will convert the video file to audio, then call convert\_audio\_to\_text( ) and return the result.

### 4.4.2 Contract: Cleanse and Validate File

**Protocol:** cleanse\_file(self, file)

**Pre-condition:**  file must be ingested.

**Post-condition:** The file is cleansed from unwanted characters as well as empty lines for every line in the file and it gets return.

**Description:** The method will remove any unwanted characters and empty lines from a file and return True if successful.

**Protocol:** validate\_file(self, file)

**Pre-condition:**  file must be ingested. Self must be a FileValidator object and not null.

**Post-condition:** The file is checked for empty lines, missing timestamps and ensuring all timestamps in the file are within range of the start and end dates provided in the event configuration. It generates an enforcement action report for the file and stores it, then it returns the enforcement action report.

**Description:** The method will validate the file to ensure the file follows the timestamp range, and that it can be ingested.

## 4.5 Class Description: Log File

|  |  |
| --- | --- |
| **Class Name**: Log File |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities: N**one |  |
| **Contract 9:** Get Log Attributes |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows the log file name. 2. Knows the log file cleansing status 3. Knows the log file validation status 4. Knows the log file ingestion status 5. Knows the log file acknowledgment status | Event Configuration (12) |

### 4.5.1 Contract: Get Log Attributes

**Protocol:** get\_name(self)

**Pre-condition:** self must be of type LogFile and an Event must have been successfully configured.

**Post-condition:** Returns the name of a log file.

**Description:** This method returns the name of a log file.

**Protocol:** get\_cleansing\_status(self)

**Pre-condition:**  self must be of type LogFile and an Event must have been successfully configured.

**Post-condition:** Return the cleansing status of a log file.

**Description:** This method returns the cleansing status of a log file.

**Protocol:** get\_validation\_status(self)

**Pre-condition:**  self must be of type LogFile and an Event must have been successfully configured.

**Post-condition:** Return a string containing the validation status of a log file.

**Description:** This method returns the validation status of a log file.

**Protocol:** get\_ingestion\_status(self)

**Pre-condition:**  self must be of type LogFile and an Event must have been successfully configured.

**Post-condition:** Return the ingestion status of a log file.

**Description:** This method returns the ingestion status of a log file.

**Protocol:** get\_acknowledgement\_status(self)

**Pre-condition:**  self must be of type LogFile and an Event must have been successfully configured.

**Post-condition:** Return the acknowledgment status of a log file.

**Description:** This method returns the acknowledgement status of a log file.

## 4.6 Class Description: Log File Configuration

|  |  |
| --- | --- |
| **Class Name**: LogFileConfigurationWindow |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:**   1. Display enforcement action reports 2. Sort log file table in ascending/descending order. 3. Ingest log files despite errors 4. Revalidate log files 5. Create UI for the Log File Configuration Window |  |
| **Contract 10:** Log files Report |  |
| **Responsibilities** | **Collaborations** |
| 1. Create table of log files with validation, ingestion, and cleansing status. | PMR (3)  PMR (10) |
|  |  |

### 4.6.1 Contract: Log Files Report

**Protocol:** populate\_table (self, log\_files)

**Pre-condition:** self must be a LogFileConfigurationWindow. log\_files must be a list containing log files. The parameters provided are not null. Ingestion, Validation, and Cleansing process have been completed.

**Post-condition:** A table will be created with the following attributes: Log File Name, Source, Cleansing Status, Validation Status, and Ingestion.

**Description:**  This method will create and display a table with log files with their status results for validation, ingestion, and cleansing.

## 4.7 Class Description: Enforcement Action Report

|  |  |
| --- | --- |
| **Class Name**: Enforcement Action Report |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** |  |
| **Contract 11:** Enforcement Action Report |  |
| **Responsibilities** | **Collaborations** |
| 1. Create Enforcement Action Report Table | Log File Configuration (10) |

### 4.7.1 Contract: Enforcement Action Report

**Protocol:** populate\_table (self, er\_report)

**Pre-condition:** self must be an EnforcementActionReport; er\_report must be a string. The parameters provided are not null. Ingestion, Validation, and Cleansing process have been completed.

**Post-condition:** A table is created with index errors and key errors for an individual log file.

**Description:**  This method will create a table with index and key errors for each log file that flagged an error.

## 4.8 Class Description: Splunk Integrator

|  |  |
| --- | --- |
| **Class Name**: SplunkIntegrator |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:**   1. Create Splunk Index 2. Connect to Splunk 3. Find Source File 4. Find Event Source |  |
| **Contract 13:** Manage Log File |  |
| **Responsibilities** | **Collaborations** |
| 1. Upload Log File 2. Download Log Files 3. Cleanse Log File 4. Validate Log File | Event Configuration (12)  Log File Configuration (10)  Event Configuration (6) |

### 4.8.1 Contract: Manage log File

**Protocol:** upload\_file (self, index)

**Pre-condition:** Self must be a string and index an integer.

**Post-condition:** The file gets uploaded to Splunk.

**Description:**  This method will get a file by using the file path and will upload the file into Splunk at the indicated index.

**Protocol:** download\_log\_files (count)

**Pre-condition:** Count must be an integer.

**Post-condition:** List is created of significant log entries and they are printed.

**Description:**  This method downloads significant log entries from Splunk and puts them into a list. The list’s significant log entries source is printed.

**Protocol:** cleanse\_file (self, file)

**Pre-condition:** self must be a FileCleanser object; file must be a string. The parameters provided are not null.

**Post-condition:** A true value is returned if SRS cleansing conditions are met.

**Description:**  This method cleanses an individual log file and returns the cleansing status of the log file.

**Protocol:** validate\_file (self, file)

**Pre-condition:** self must be a SplunkIntegrator object; file must be a string. The parameters provided are not null.

**Post-condition:** A true value is returned if the SRS validation conditions are met.

**Description:**  This method will validate all log files, return the validation status, and update the enforcement action report for the given log file.

## 4.9 Class Description: PMR Main Window

|  |  |
| --- | --- |
| **Class Name**: PMR |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:**   1. Create System Window Configuration 2. Populate Log Entries 3. Populate Log Files 4. Populate Error Reports 5. Update Log Entry Vectors 6. Update Non Lead Vector DB 7. Update Lead Vector DB 8. Update Nodes 9. Connect Mongo DB 10. Revalidate Log Files 11. Revert Log Entry Table 12. Event Configuration Toolbar Control |  |

# Detailed Description of Database Subsystem

## 5.1 Class Description: Vector DB Version Control Non-Lead

|  |  |
| --- | --- |
| **Class Name**: VectorDBConfigurationNonLead |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 16:** Manage Local DB |  |
| **Responsibilities:** | **Collaborations** |
| 1. Connect to Mongo DB 2. Push object into Mongo DB 3. Pull object from Mongo DB | PMR (7) |

### 5.1.1 Contract: Manage Local DB

**Protocol: connect**(self)

**Pre-condition:** self must be a VectorDBConfigurationNonLead object.

**Post-condition:** the caller will be connected to the Mongo Client. If the connection is unsuccessful, an error message stating " a connection could not be established at this time"

**Description:**  This method allows the user to connect to the Mongo Client

**Protocol:** pushed(self)

**Pre-condition:** self must be a VectorDBConfigurationNonLead object.

**Post-condition:** changes made to the object will be pushed to the lead database

**Description:**  This method pushes changes to the lead database

**Protocol:** pull(self)

**Pre-condition:** self must be a VectorDBConfigurationNonLead object. A change must have been made in the lead vector’s database for this method to run.

**Post-condition:** changes made to the lead database will be pulled to the non-lead database

**Description:**  This method pulls changes from the lead database

## 5.2 Class Description: Vector DB Version Control Lead

|  |  |
| --- | --- |
| **Class Name**: VectorDBConfigurationLead |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 17:** Commit to Database | **Collaborations** |
| **Responsibilities:**   1. Connect to MongoDB 2. Load commits from MongoDB 3. Commit pushed changes into the MongoDB | PMR (6) |

### 5.2.1 Contract: Edit Vector

**Protocol:** load\_commits(self)

**Pre-condition:** self must be a database object.

**Post-condition:** if no error exceptions are thrown, commits will be loaded. Otherwise, it can generate an exception for error configuration, error connection to the database and key error.

**Description:** This method will load commits from the given database.

**Protocol:** commit\_to\_database(self)

**Pre-condition:** self must be a database object.

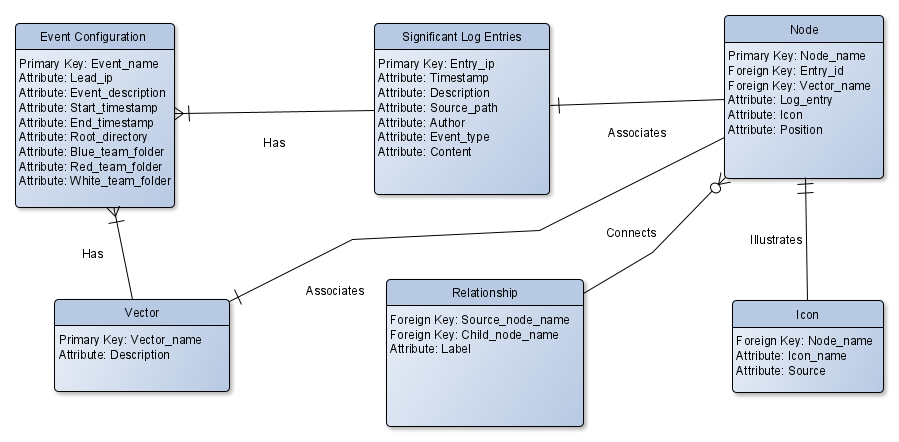
**Post-condition:** if no error exceptions are thrown, commits will be done. Otherwise, it can generate an exception for error configuration and error connection to the database.

**Description:**  This method will commit to the database.

# Database

The following is the database design pattern created for the Mongo DB that interacts with the system. Currently the Mongo DB in the system does not reflect the Database Schema. That section of the system is not in function.

## Database Schema



&