**PICK PMR Insights Collective Tool**

Software Design Document

<1.0.2>

<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: | 1.0.2 |
| Indicator of Last Page in Document: | & |
| Date of Last Review: | 3/30/20 |
| Date of Next Review: | 04/02/20 |
| Target Date for Next Update: | 04/02/20 |

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 |

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 4](#_Toc226283114)

[1.1. Purpose and Intended Audience 4](#_Toc226283115)

[1.2. Scope of Product 4](#_Toc226283116)

[1.3. References 4](#_Toc226283120)

[1.4. Definitions, Acronyms, and Abbreviations 5](#_Toc226283121)

[1.4.1. Definitions 5](#_Toc226283122)

[1.4.2. Acronyms 5](#_Toc226283123)

[1.4.3. Abbreviations 5](#_Toc226283124)

[1.5. Overview 5](#_Toc226283125)

[2. Decomposition Description 7](#_Toc226283126)

[2.3. Subsystem Description 7](#_Toc226283129)

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

[4. Detailed Description of Subsystem Ingestion 21](#_Toc226283132)

[5. Detailed Description of Subsystem Management 27](#_Toc226283133)

[4. DatabaseSchema 30](#_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 |

## Overview

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

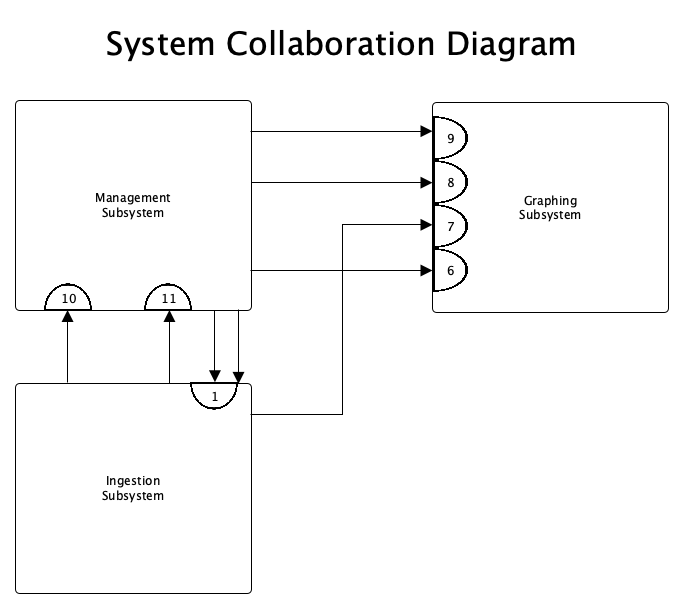
The SDD has the following sections:

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. Detailed Description of Management Subsystem
   * This section provides a detailed description of the Management Subsystem
5. 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.

## System Collaboration Diagram



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

## Subsystem and Component Descriptions

## 2.2.1. Subsystem Name: Ingestion Subsystem

|  |
| --- |
| **Classes:** Transcription handler  enforcement action report  log file  Log Entries Manager  validation splunk handler |
| **Description:** The ingestion subsystem handles the ingestion of files into the system, including all cleansing, transcription, and transformation done on the files. |
| **2. Get Log Information**  -Knows the log file name.  -Knows the log file contents  **3. Set Log Content** -Knows the cleansing status of a log file.  -Knows the validation status of a log file.  -Knows the ingestion status of a log file.  **16. Validate Log File**  -Check that the log files contain a timestamp per line  -Check that log file contain timestamps that are bounded by the start data, end date, start time, and end time specified in the event configuration.  -If the log file is of type CVS and the originator of the log file is from the white team, check that the log file contains timestamps that are within Lower limit of the range and Upper limit of the range  -Certify log files as validated log file.  -Change status of log file to pass or fail.    **17. Cleanse Log File** -Cleanse Log Files from unwanted characters if log file is of type TMUX  -Cleanse Log Files from blank rows if the log file is of type CVS  -Create cleanse certification for Log Files  **18. Convert log files into log entries**  -Convert log file to log entries |

## 2.2.2. Subsystem Name: Graphing Subsystem

|  |
| --- |
| **Classes:** Node, Graph, Icon, Vector, Relationship, Significant Log Entries |
| **Description:** The Graphing Subsystem is responsible for creating a graph composed of nodes that identify a relationship between a log entry and a vector. Nodes are represented by icons that can be connected to illustrate flow of events. |
| **Contracts:**  **4. Get Node Information**  -Knows its ID  -Knows its name  -Knows its description  -Knows its log entry reference  -Knows its log creator  -Knows its event type  -Knows its icon type  -Knows its source  -Knows its visibility  -Knows its ID visibility  -Knows its name visibility  -Knows its description visibility  -Knows its log entry reference visibility  -Knows its log creator visibility  -Knows its event type visibility  -Knows its icon type visibility  -Knows its source visibility  **5. Set Node Content**  -Set its ID  -Set its name  -Set its description  -Set its log entry reference  -Set its log creator  -Set its event type  -Set its icon type  -Set its source  -Set its visibility  -Set its ID visibility  -Set its name visibility  -Set its description visibility  -Set its log entry reference visibility  -Set its log creator visibility  -Set its event type visibility  -Set its icon type visibility  -Set its source visibility  **6. Get Vector Information**  **-**Knows its name  -Know its description  -Knows set of log entries    **7. Set Vector Content** -Set its name  -Set its description  -Add log entries    **8. Get Log Entry Information**  -Knows the path of the log file from which it originates.  -Knows the log entry number.  -Knows the timestamp.  -Knows the log entry content.  -Knows the host / IP address of creator.  -Knows the source type.    **9. Manage Graph Elements** **-**Add Nodes  -Delete Nodes  -Edit Nodes  -Move Nodes  -Add relationships  -Edit Relationships  -Delete Relationships  -Change Icon  -Add Icon  -Delete Icon  **12. Get Icon Image**  -Assign icon for individual node  **13. Get Relationship Contents**  -Know source node  -Know destination node  -Know label  -Know description  **14. Set Relationship Contents**  -Set source node  -Set destination node  -Set label  -Set description |

## 2.2.3. Subsystem Name: Management Subsystem

|  |
| --- |
| **Classes:** Version Manager, User Manager, User |
| **Description:** Limit number of users that can interact with the system and control the access for committing, pushing, and pulling graphs and vectors into the Vector Database. |
| **Contracts:**  **10. Manage User Connections**  **-**Manage the number of connected users to maximum 20 users  -Listens for incoming user connections to the server  -Knows number of connected users    **11. Distinguish The Lead**  -Differentiate the IP addresses between lead and non-lead  -Updates lead and non-lead status of users    **15. Get User Info**  -Knows the IP address of the User  -Knows if the User is a Lead |

## 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.

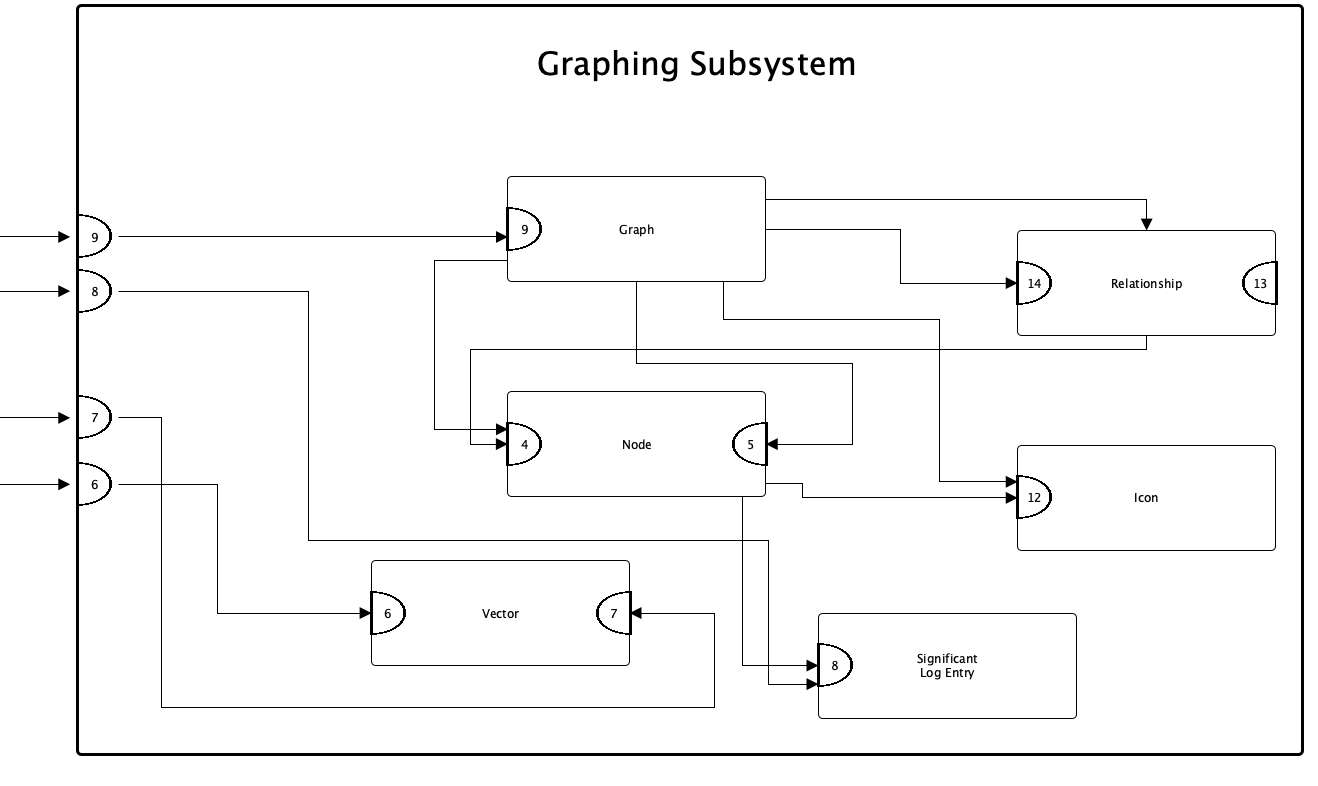
**Splunk-SDK:** The system depends on this library to ingest log files.

**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 Subsystem Graphing



The graphing component integrates the graphing function into the system and provides the analyst with the ability to manipulate the graph to create a timeline of events.

## 3.1. Class Description Node

|  |  |
| --- | --- |
| **Class Name**: Node |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 4:** Get Node Information |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows its ID 2. Knows its name 3. Knows its description 4. Knows its log entry reference 5. Knows its log creator 6. Knows its event type 7. Knows its icon type 8. Knows its source 9. Knows its visibility 10. Knows its ID visibility 11. Knows its name visibility 12. Knows its description visibility 13. Knows its log entry reference visibility 14. Knows its log creator visibility 15. Knows its event type visibility 16. Knows its icon type visibility 17. Knows its source visibility |  |
| **Contract 5:** Set Node Content |  |
| **Responsibilities** | **Collaboration** |
| 1. Set its ID 2. Set its name 3. Set its description 4. Set its log entry reference 5. Set its log creator 6. Set its event type 7. Set its icon type 8. Set its source 9. Set its visibility 10. Set its ID visibility 11. Set its name visibility 12. Set its description visibility 13. Set its log entry reference visibility 14. Set its log creator visibility 15. Set its event type visibility 16. Set its icon type visibility 17. Set its source visibility | Significant Log Entry (8)  Icon (12) |

### 3.1.1. Contract: Get Node Contents

**Protocol:** get\_node\_ID( )

**Pre-condition:** 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\_name( )

**Pre-condition:** 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\_description( )

**Pre-condition:** 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( )

**Pre-condition:** 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( )

**Pre-condition:** 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( )

**Pre-condition:** Node must not be null.

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

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

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

**Pre-condition:** 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( )

**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( )

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

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

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

**Protocol:** get\_id\_visibilty( )

**Pre-condition:** Node must not be null.

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

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

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

**Pre-condition:** Node must not be null.

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

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

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

**Pre-condition:** Node must not be null.

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

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

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

**Pre-condition:** Node must not be null.

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

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

**Protocol:** get\_log\_entry\_creator\_visibility( )

**Pre-condition:** Node must not be null.

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

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

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

**Pre-condition:** Node must not be null.

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

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

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

**Pre-condition:** Node must not be null.

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

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

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

**Pre-condition:** Node must not be null.

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

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

### 3.1.2. Contract: Set Node Contents

**Protocol:** set\_description(description)

**Pre-condition:** Node must not be null and description must be a string.

**Post-condition:** Sets description of a node.

**Description:**  This method sets the value for the description attribute of a node.

**Protocol:** set\_log\_entry\_reference(log\_entry\_reference)

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

**Post-condition:** Sets log entry reference of a node

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

**Protocol:** set\_log\_creator(creator)

**Pre-condition:** Node must not be null, asignificant log entry must exist, and creator must be string.

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

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

**Protocol:** set\_event\_type(event\_type)

**Pre-condition:** Node must not be null and event type must be a string.

**Post-condition:** Sets the event type of the node.

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

**Protocol:** set\_icon\_type(icon\_type)

**Pre-condition:** Node must not be null, icon must exist, and icon type must be a string.

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

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

**Protocol:** set\_source(source)

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

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

**Description:**  This method sets the source attribute of a node.

**Protocol:** set\_visibility(node\_visibility)

**Pre-condition:** Node must exist, and node\_visibility must be a boolean.

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

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

**Protocol:** set\_id\_visibilty( id\_visibility)

**Pre-condition:** Node must not be null and id\_visibility must be a boolean.

**Post-condition:** Sets the ID visibility of a node.

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

**Protocol:** set\_name\_visibilty(name\_visibility)

**Pre-condition:** Node must not be null and name\_visibility must be a boolean.

**Post-condition:** Sets the name visibility of a node.

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

**Protocol:** set\_description\_visibility(description\_visibility)

**Pre-condition:** Node must not be null and description\_visibilty must be a boolean.

**Post-condition:** Set the description visibility of a node.

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

**Protocol:** set\_log\_entry\_reference\_visibility(reference\_visibility)

**Pre-condition:** Node must not be null and reference\_visibility must be a boolean.

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

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

**Protocol:** set\_log\_entry\_creator\_visibility(log\_entry\_creator\_visibility)

**Pre-condition:** Node must not be null and log\_entry\_creator\_visibility must be a boolean.

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

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

**Protocol:** set\_event\_type\_visibility(event\_type\_visibility)

**Pre-condition:** Node must not be null and event\_type\_visibility must be a boolean.

**Post-condition:** Set the event type visibility of a node.

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

**Protocol:** set\_icon\_type\_visibility(icon\_type\_visibility)

**Pre-condition:** Node must not be null and icon\_type\_visibility must be a boolean.

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

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

**Protocol:** set\_source\_visibility(source\_visibility)

**Pre-condition:** Node must not be null and source\_visibility must be a boolean.

**Post-condition:** Set the source visibility of a node to the parameter.

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

## 3.2. Class Description: Vector

|  |  |
| --- | --- |
| **Class Name**: Vector |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 6:** Get Vector Information |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows its name 2. Knows its description 3. Knows set of log entries |  |
| **Contract 7:** Set Vector Contents |  |
| **Responsibilities** | **Collaborations** |
| 1. Set its name 2. Set its description 3. Add log entries |  |

### 3.2.1 Contract: Get Vector Information

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

**Pre-condition**: 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()

**Pre-condition**: 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.

**Protocol:** get\_vector\_entries()

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

**Post-condition**: A list of associated entries is returned.

**Description**: This method returns the list of log entries associated with the vector.

### 3.2.2 Contract: Set Vector content

**Protocol:** set\_vector\_name(vector\_name)

**Pre-condition:** The event configuration procedure must have been completed and vector name must be a string.

**Post-condition:** The vector name will be able to be changed.

**Description:**  This method sets the name for a vector.

**Protocol:** set\_vector\_description(vector\_description)

**Pre-condition:** The event configuration procedure must have been completed and vector\_description must be a string.

**Post-condition:** Allow the vector description to be changed.

**Description:**  This method sets the description for a vector.

**Protocol:** add\_log\_entry(log\_entry)

**Pre-condition:** The event configuration procedure must have been completed,vector must not be null, and log\_entry must not be null.

**Post-condition:** Will add given log entry to the set of entries that the vector has.

**Description:**  This method adds a log entry to the set of entries on the vector.

## 3.3. Class Description: Significant Log Entry

|  |  |
| --- | --- |
| **Class Name**: Significant Log Entry |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 8:** Get Log Entry Information |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows the path of the log file from which it originates. 2. Knows the log entry number. 3. Knows the timestamp. 4. Knows the log entry content. 5. Knows the host / IP address of creator. 6. Knows the source type. |  |

### 3.3.1 Contract: Get Log Entry Information

**Protocol:** get\_log\_entry\_info()

**Pre-condition:** The event configuration procedure must have been completed.

**Post-condition:** Returns an array containing all attributes of the significant log entry.

**Description:** This method returns all attributes of a significant log entry.

## 3.4. Class Description: Graph

|  |  |
| --- | --- |
| **Class Name**: Graph |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities**  Keep Track of Graph Actions List  Filter Nodes and Relationships  Know set of Graph actions  Know set of nodes  Know set of relationships  Export an image of the graph. |  |
| **Contract 9:** Manage Graph Elements |  |
| **Responsibilities** | **Collaborations** |
| 1. Add Nodes 2. Delete Nodes 3. Edit Nodes 4. Move Nodes 5. Add relationships 6. Edit Relationships 7. Delete Relationships 8. Change Icon 9. Add Icon 10. Delete Icon | Node (4, 5)  Vector(6, 7)  Relationship (13, 14)  Icon (12) |

### 3.4.1 Contract: Manage Graph Elements

**Protocol:** add\_node()

**Pre-condition:** A must vector exist

**Post-condition:** A text editable menu with all the properties of a node prompts the user to specify the attributes of the node to add

**Description:** This method provides the ability to add nodes by allowing the user to specify the attributes of a node under a text editable menu.

**Protocol:** delete\_node(node)

**Pre-condition:** Node must exist. Node parameter must be a Node object.

**Post-condition:** Node is removed from graph, and all relationships associated with it

**Description:**  This method removes a selected node and its relationships from the graph

**Protocol:** move\_node(node, position)

**Pre-condition:** Node exists and is selected. Node parameter must be a Node object.

**Post-condition:** The node’s position field is set to the parameter position

**Description:** This method moves a node to follow the cursor

**Protocol:** delete\_relationship(relationship)

**Pre-condition:** Relationship exists. Relationship must be a Relationship object

**Post-condition:** Relationship is removed

**Description:** This method removes a relationship between two nodes

**Protocol:** add\_relationship(node1, node2)

**Pre-condition:** Two nodes without an existing relationship must be selected

**Post-condition:** A relationship between the two selected nodes will be created. Node 1 and Node2 must be Node objects.

**Description:**  This method adds a relationship between two nodes on the graph

**Protocol:** edit\_relationship(relationship)

**Pre-condition:** Relationship exists. Relationship parameter must be a Relationship object.

**Post-condition:** Return edited relationship.

**Description:** This method edits a relationship object.

## 3.5. Class Description Icon

|  |  |
| --- | --- |
| **Class Name**: Icon |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities**   * Know path to icon image |  |
| **Contract 12:** Get icon image |  |
| **Responsibilities** | **Collaborations** |
| 1. Assign icon for individual node |  |

### 3.5.1 Contract: Get Icon Image

**Protocol:** get\_icon\_path()

**Pre-condition:** A pathname to the given icon must exist.

**Post-condition:** Returns a pathname to a given icon.

**Description:**  This method will return the path to an icon image.

**Protocol:** change\_icon(node)

**Pre-condition:** Node exists. Node must be a Node object

**Post-condition:** Node icon is replaced by selected icon

**Description:** This method changes the icon of a selected node

**Protocol:** add\_icon(icon)

**Pre-condition:** Node must not have an icon. Icon must be an Icon object

**Post-condition:** Icon is added to a selected node.

**Description:** This method adds an icon to a selected node.

**Protocol:** delete\_icon(icon)

**Pre-condition:** Node must exist. Icon must be an Icon object

**Post-condition:** Node icon will be removed from graph.

**Description:** This method removes an icon from a selected node.

## 3.6. Class Description Relationship

|  |  |
| --- | --- |
| **Class Name**: Relationship |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 13:** Get Relationship contents |  |
| **Responsibilities** | **Collaborations** |
| 1. Know source node 2. Know destination node 3. Know label 4. Know description | Node (4) |
| **Contract 14:** Set Relationship Contents | **Collaborations** |
| 1. Set source node 2. Set destination node 3. Set label 4. Set description |  |

### 3.6.1 Contract: Get Relationship Contents

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

**Pre-condition:** Relationship instance must not be null.

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

**Description:**  This method will return the source node.

**Protocol: get\_destination\_node( )**

**Pre-condition:** Relationship instance must not be null.

**Post-condition:** Return the destination node.

**Description:**  This method will return the destination node.

**Protocol: get\_relationship\_label( )**

**Pre-condition:** Relationship instance must not be null.

**Post-condition:** Return the label of the relationship.

**Description:**  This method will return the label of the relationship.

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

**Pre-condition:** Relationship instance must not be null.

**Post-condition:** Return the description of the relationship.

**Description:**  This method will return the description of the relationship.

### 3.6.2 Contract: Set Relationship Contents

**Protocol: set\_nodes(**Source Node, Destination Node**)**

**Pre-condition:** Relationship instance must not be null. Source Node and Destination Node are Node objects.

**Post-condition:** Sets the value of the source node and destination node for a relationship to the given node objects.

**Description:**  This method will set the source node and destination node at the same time.

**Protocol: set\_source\_node(**Source Node**)**

**Pre-condition:** Relationship instance must not be null. Source Node must be a Node object

**Post-condition:** Sets the value of the source node to the Source Node given.

**Description:**  This method will set the source node.

**Protocol: set\_destination\_node**(Destination Node)

**Pre-condition:** Relationship instance must not be null. Destination Node should be a Node object.

**Post-condition:** Sets the destination of a node to the Destination Node given.

**Description:**  This method will set the destination of a node.

**Protocol: set\_relationship\_label**(label)

**Pre-condition:** Relationship instance must not be null. Label parameter must be a String.

**Post-condition:** Sets the label of the relationship to the label given.

**Description:**  This method will set the label of a relationship.

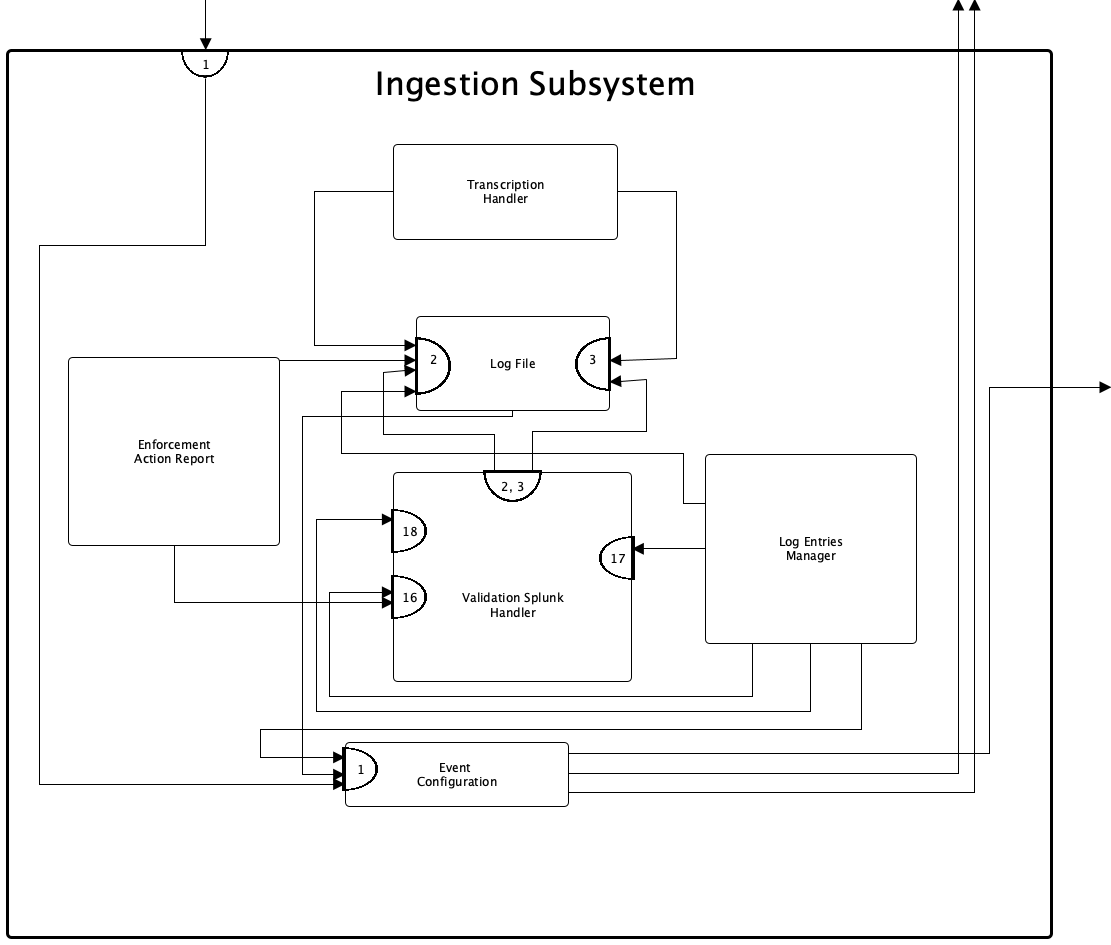
**Protocol: set\_relationship\_description**(description)

**Pre-condition:** Relationship instance must not be null. Description must be a String.

**Post-condition:** Sets the description of the relationship to the description given.

**Description:**  This method will set the description of a relationship.

# Detailed Description of Subsystem Ingestion



## 4.1. Class Description Event Configuration

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

### Contract: Get Event Data

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

**Pre-condition:** The event must not be null.

**Post-condition:** Return an events’ description.

**Description:**  This method will return the name of the event.

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

**Pre-condition:** The event must not be null.

**Post-condition:** Return an events’ description.

**Description:**  This method will return the description of an event.

**Protocol:** get\_event\_start\_timestamp()

**Pre-condition:** The event must not be null.

**Post-condition:** Return an events’ start timestamp.

**Description:**  This method will return the start time of an event.

**Protocol:** get\_event\_end\_timestamp()

**Pre-condition:** The event must not be null.

**Post-condition:** Return an events’ end timestamp.

**Description:**  This method will return the end time of an event.

**Protocol:** get\_root\_directory()

**Pre-condition:** The event must not be null.

**Post-condition:** Return an event’s root directory.

**Description:**  This method will return the path to a root directory.

**Protocol:** get\_red\_team\_folder()

**Pre-condition:** The event must not be null.

**Post-condition:** Return an event’s red team folder.

**Description:**  This method will return the path to the red team folder.

**Protocol:** get\_blue\_team\_folder()

**Pre-condition:** The event must not be null.

**Post-condition:** Return an event’s blue team folder.

**Description:**  This method will return the path to the blue team folder.

**Protocol:** get\_white\_team\_folder()

**Pre-condition:** The event must not be null.

**Post-condition:** Return an event’s white team folder.

**Description:**  This method will return the white team folder.

**Protocol:** is\_lead()

**Pre-condition:** The event must not be null.

**Post-condition:** Return whether the ip is lead..

**Description:**  This method will return whether the IP address entered is the lead IP.

**Protocol:** get\_lead\_IP()

**Pre-condition:** The event must not be null.

**Post-condition:** Return whether the ip is lead..

**Description:**  This method will return whether the IP address entered is the lead IP.

**Protocol:** get\_connections()

**Pre-condition:** The event must not be null.

**Post-condition:** Return number of established connections to lead IP address.

**Description:**  This method will return the number of connections to the lead IP address.

## 4.2. Class Description Log File

|  |  |
| --- | --- |
| **Class Name**: Log File |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities**   * Knows its acknowledgement status |  |
| **Contract 2:** Get Log Information |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows the log file name. 2. Knows the log file contents | Event Configuration (1) |
| **Contract 3:** Set Log Content |  |
| **Responsibilities** | **Collaborations** |
| 1. Knows the cleansing status of a log file. 2. Knows the validation status of a log file. 3. Knows the ingestion status of a log file. |  |

### 4.2.1. Contract: Get Log Info

**Protocol:** get\_log\_file\_name( )

**Pre-condition:**  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\_log\_file\_contents( )

**Pre-condition:**  Event must have been successfully configured.

**Post-condition:** Return log file contents.

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

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

**Pre-condition:**

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

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

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

**Pre-condition:**

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

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

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

**Pre-condition:**

**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\_acknowledgement\_status( )

**Pre-condition:**

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

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

### 4.2.2. Contract: Set Log Content

**Protocol:** set\_cleansing\_status(status)

**Pre-condition:**  A session to be created a log file is required. Status must be a Boolean.

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

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

**Protocol:** set\_ingestion\_status(status)

**Pre-condition:**  A session to be created a log file is required. Status must be a Boolean

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

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

**Protocol:** set\_validation\_status(status)

**Pre-condition:**  A session to be created a log file is required. Status must be a String.

**Post-condition:** Sets the validation status of a log file to the status.

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

**Protocol:** set\_content(content)

**Pre-condition:**  A session to be created a log file is required. Content must be a String.

**Post-condition:** Sets the content of a log file to the content given.

**Description:** This method sets the content of a log file.

## 4.3. Class Description: Enforcement Action Report

|  |  |
| --- | --- |
| **Class Name**: Enforcement Action Report |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:**   1. Display ingestion errors in log file 2. Knows where an error occurs in a log file. 3. Knows why a specific line in the log file failed the validation test. | **Collaborations**  Log File (2)  Validation Splunk Handler (16) |

## 4.4 Class Description Log Entries Manager

|  |  |
| --- | --- |
| **Class Name**: Log Entries Manager |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:**   1. Check for changes between log files in the system and log file from the server root directory | **Collaborations:**  Log File (2)  Event Configuration (1)  Validation Splunk Handler (16, 17, 18) |

**4.5 Class Description Validation Splunk Handler**

|  |  |
| --- | --- |
| **Class Name**: Validation Splunk Handler |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:**   1. Create log files with same source of ingested log files |  |
| **Contract 16:** Validate Log File |  |
| **Responsibilities** | **Collaborations** |
| 1. Check that the log files contain a timestamp per line 2. Check that log file contain timestamps that are bounded by the start data, end date, start time, and end time specified in the event configuration. 3. If the log file is of type CVS and the originator of the log file is from the white team, check that the log file contains timestamps that are within Lower limit of the range and Upper limit of the range 4. Certify log files as validated log file. 5. Change status of log file to pass or fail. | Log File (2,3) |
| **Contract 17**: Cleanse Log File |  |
| **Responsibilities** | **Collaborations** |
| 1. Cleanse Log Files from unwanted characters if log file is of type TMUX 2. Cleanse Log Files from blank rows if the log file is of type CVS 3. Create cleanse certification for Log Files | Log File (2,3) |
| **Contract 18:** Convert log files into log entries   1. Convert log file to log entries | Log File (2,3) |

### 4.5.1 Contract: Validate Log File

**Protocol:** validate\_log\_file(log\_file)

**Pre-condition:** Log File has been ingested into the system. Log\_file must be a Log File object.

**Post-condition:** Will certify log file as validated and change status of log file to pass or fail.

**Description:**  This method will validate the log file by checking that each line has a timestamp, timestamps are bounded to desire time on event configuration and if it is a type CVS, check that timestamps are within the given range. After that, it will certify log file as valid in order to be cleansed.

### 4.5.2 Cleanse Log File

**Protocol:** cleanse\_log\_file(log\_file)

**Pre-condition:** Log file has been certified as valid.

**Post-condition:** Will cleanse log file and return a log entry. Log\_file parameter must be a Log File object

**Description:**  This method will cleanse a log file by removing unwanted chars and blank rows. After that, it will create cleanse certification for the log file and return a cleansed log entry that is ready to be converted into a log entry.

**4.5.3 Convert Log File to Log Entries**

**Protocol:** convert\_log\_file(log\_file)

**Pre-condition:** Log file has been certified as validated and cleansed. Log\_file parameter must be a Log File object

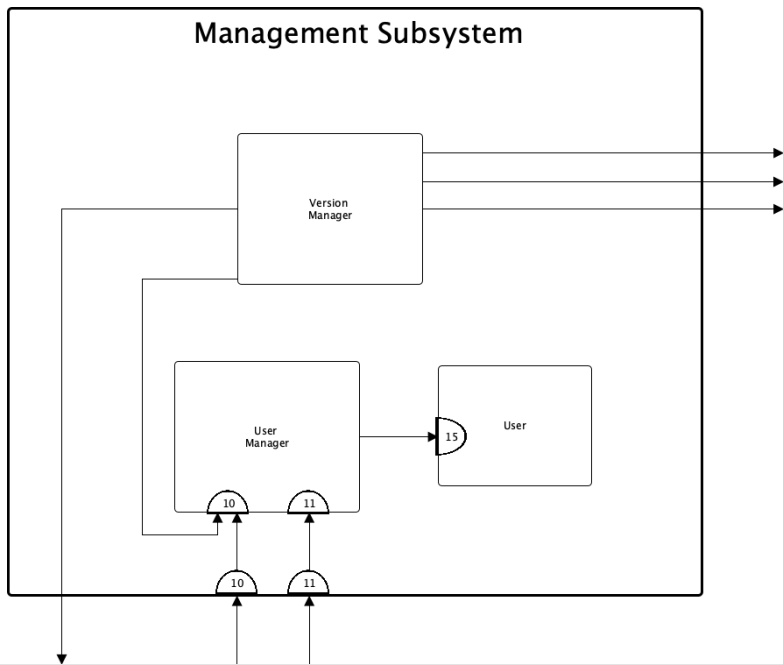
**Post-condition:** Will convert a log file and return a set of log entries.

**Description:** This method will convert a log file into a set of log entries after checking that the log file has been validated and cleansed.

**4.6 Class Description Transcription Handler**

|  |  |
| --- | --- |
| **Class Name**: Transcription Handler |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities** | **Collaborations** |
| 1. Transcribe audio Log Files into text. 2. Translate Image Log Files into text. 3. Transcribe Video Log Files into text | Log File (2,3) |

# Detailed Description of Subsystem Management



## 5.1 Class Description Version Manager

|  |  |
| --- | --- |
| **Class Name**: Version Manager |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** | **Collaborations** |
| 1. Compare Vector DB commit differences between Lead and Non-Lead 2. Load commit history changes made in the Vector DB Lead and Non-Lead 3. Download Vector and Graph from Lead Vector DB to Non-Lead Vector DB 4. Upload Vector and Graph from Non-Lead Vector DB with Lead Approval 5. Load Vector and Graph from Local Storage 6. Download copy (Saving) of Vector and Graph to Local Storage | Event Configuration (1)  User Manager (10)  Significant Log Entry (8)  Graph (9)  Vector (6) |

**5.2. Class Description User**

|  |  |
| --- | --- |
| **Class Name**: User |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities:** None |  |
| **Contract 15:** Get User Info |  |
| **Responsibilities** | **Collaborations** |
| 1. Knowsthe IP address of the User 2. Knows if the User is a Lead |  |

### 5.2.1 Contract: Get User Info

**Protocol: g**et\_ip()

**Pre-condition:** A session created and stored in the system for the user.

**Post-condition:** Returns the ip of the user.

**Description:**  This method returns the ip of the user.

**Protocol: g**et\_lead\_status()

**Pre-condition:** A session created and stored in the system for the user.

**Post-condition:** Returns whether user is lead or not with a boolean.

**Description:**  This method returns true if the user is lead or false otherwise.

**Protocol: s**et\_ip(ip)

**Pre-condition:** A session created and stored in the system for the user.

**Post-condition:** Sets the ip for a user.

**Description:**  This method sets the ip for a user.

**Protocol: s**et\_lead\_status(status)

**Pre-condition:** A session created and stored in the system for the user. Status field must be a Boolean value.

**Post-condition:** Sets the user’s isLead field to status.

**Description:**  This method sets the lead status of a user.

## 5.3 Class Description: User Manager

|  |  |
| --- | --- |
| **Class Name**: User Manager |  |
| **Superclass**: None |  |
| **Subclasses**: None |  |
| **Private Responsibilities**   * Knows current I.P. addresses connected to the system |  |
| **Contract 10:** Manage User Connections |  |
| **Responsibilities** | **Collaborations** |
| 1. Manage the number of connected users to maximum 20 users. 2. Listens for incoming user connections to the server. 3. Knows number of connected users. | Event Configuration (1) |
| **Contract 11:** Distinguish the Lead |  |
| **Responsibilities** | **Collaborations** |
| 1. Differentiates the IP addresses between Lead and Non - Lead User 2. Updates Lead and Non – Lead Status of Users | User (15) |

### 5.3.1 Contract: Manage Users Connected

**Protocol:** limit**\_**users\_connected()

**Pre-condition:** A session to be created IP of the host machine is required.

**Post-condition:** Listens to connections made to the host machine and limit the amount of connections.

**Description:**  This method will listen to connections made to the host machine and limit connections to the maximum on 20 connections.

**Protocol:** get**\_**users\_connected(host\_ip\_address)

**Pre-condition:** A session to be created IP of the host machine is required and host\_ip\_address must be a string.

**Post-condition:** Returns the number of users connected to the host machine.

**Description:**  This method will listen to connections made to the host machine and update number of users connected to the host machine.

### 5.3.2 Contract: Distinguish the Lead

**Protocol:** check\_lead(user\_ip\_address)

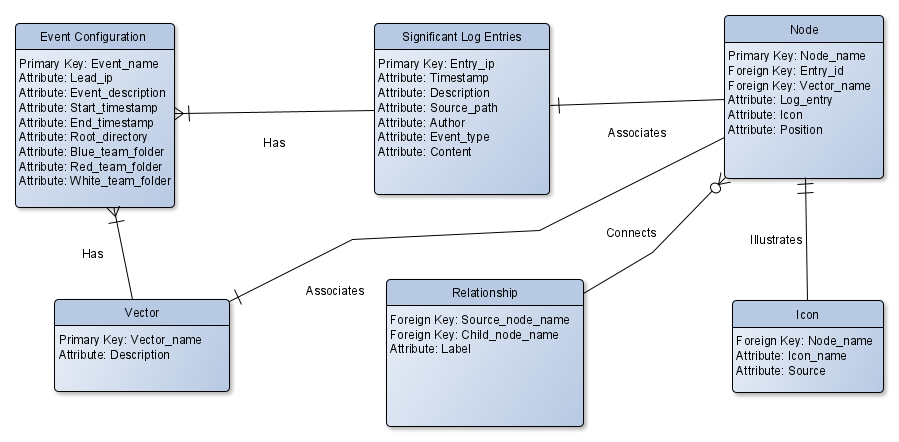
**Pre-condition:** A session to create the User IP address is required and user\_ip\_address must be a string.

**Post-condition:** Returns the boolean value based if IP connection is the Lead.

**Description:**  This method will determine is the IP given is the Lead of the system and update the status for the user.

# Database

## Database Schema



&