Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK)

Software Design Document

Version 1.6

05/02/2020

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

|  |  |
| --- | --- |
| Initial Release: | 1.0 |
| Current Release: | 1.6 |
| Indicator of Last Page in Document: | & |
| Date of Last Review: | 05/02/2020 |
| Date of Next Review: | 05/12/2020 |
| Target Date for Next Update: | 05/13/2020 |

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:

Dr. Gates

Dr. Salamah

Dr. Roach

Elsa Tai Ramirez

Peter Hanson

Customer:

Dr. Oscar Perez

Vincent Fonseca

Herandy Denisse Vazquez

Baltazar Santaella

Florencia Larsen

Erick De Nava

Software Team Members:

Hector Dozal

Victor Vargas

Eduardo Lara

Irvin Bosquez

Gerardo Armenta

Change Summary

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 03/08/2020 | Hector Dozal | Creation of Document |
| 1.1 | 03/11/2020 | Hector Dozal, Victor Vargas, Eduardo Lara,  Irvin Bosquez, Gerardo Armenta | Finished section 1 |
| 1.2 | 03/11/2020 | Hector Dozal, Victor Vargas, Eduardo Lara,  Irvin Bosquez, Gerardo Armenta | Added sections for multiple subsystems, added section 2 and subsystems |
| 1.3 | 03/26/2020 | Victor Vargas, Irvin Bosquez | Added classes for subsystems |
| 1.4 | 03/29/2020 | Victor Vargas, Irvin Bosquez | Added contracts and protocols |
| 1.5 | 03/31/2020 | Victor Vargas | Reviewed |
| 1.6 | 05/02/2020 | Victor Vargas & Gerardo Armenta | Added protocols, revised and corrected |

Table of Contents

Contents

[Document Control 1](#_Toc36561372)

[Approval 1](#_Toc36561373)

[Document Change Control 1](#_Toc36561374)

[Distribution List 1](#_Toc36561375)

[Change Summary 1](#_Toc36561376)

[1.1. Purpose and Intended Audience 5](#_Toc36561377)

[1.2. Scope of Product 5](#_Toc36561378)

[1.3. References 5](#_Toc36561379)

[1.4. Definitions, Acronyms, and Abbreviations 5](#_Toc36561380)

[1.4.1. Definitions 5](#_Toc36561381)

[1.4.2. Acronyms 6](#_Toc36561382)

[1.4.3. Abbreviations 6](#_Toc36561383)

[1.5. Overview 6](#_Toc36561384)

[1.5.1. Introduction 6](#_Toc36561385)

[1.5.2. Decomposition Description 6](#_Toc36561386)

[1.5.3. Detailed Description of Components 6](#_Toc36561387)

[1.5.4. Database 6](#_Toc36561388)

[2. Decomposition Description 7](#_Toc36561389)

[2.1. System Collaboration Diagram 7](#_Toc36561390)

[2.2. Subsystem and Component Descriptions 8](#_Toc36561391)

[2.3. Dependencies 9](#_Toc36561392)

[3. Detailed Description of Visual Subsystem 10](#_Toc36561393)

[3.1. Component Description 10](#_Toc36561394)

[3.2. Class Description Node 10](#_Toc36561395)

[3.2.1. Contract 1: Node Information Operations 11](#_Toc36561396)

[3.3. Class Description Icon 11](#_Toc36561397)

[3.3.1. Contract 2: Provide Image to Display 11](#_Toc36561398)

[4. Detailed Description of Real Time Actualization Subsystem 12](#_Toc36561399)

[4.1. Class Description Graph 12](#_Toc36561400)

[4.1.1. Contract 3: Get Information from Graph 13](#_Toc36561401)

[4.1.2. Contract 4: Modify Graph 13](#_Toc36561402)

[4.2. Class Description Table 14](#_Toc36561403)

[4.2.1. Contract 5: Gets node information from Node Class 14](#_Toc36561404)

[4.2.2. Contract 6: Modify Table 14](#_Toc36561405)

[5. Detailed Description of Analysis Subsystem 16](#_Toc36561406)

[5.1. Class Description Log File 16](#_Toc36561407)

[5.1.1. Contract 9: Log File Information 16](#_Toc36561408)

[5.2. Class Description Enforcement Action Report 17](#_Toc36561409)

[5.2.1. Contract 10: Provide Enforcement Information 17](#_Toc36561410)

[6. Detailed Description of Configuration Subsystem 18](#_Toc36561411)

[6.1. Class Description Event Configuration 18](#_Toc36561412)

[6.1.1. Contract 11: Event Information Operations 18](#_Toc36561413)

[6.2. Class Description Directory Configuration 19](#_Toc36561414)

[6.2.1. Contract 12: Directory Information Operations 19](#_Toc36561415)

[6.3. Class Description Team Configuration 20](#_Toc36561416)

[6.3.1. Contract 13: Provide Connection Status 20](#_Toc36561417)

[7. Detailed Description of Documentation Subsystem 21](#_Toc36561418)

[7.1. Class Description Log Entry 21](#_Toc36561419)

[7.1.1. Contract 14: Log Entry Details Operations 21](#_Toc36561420)

[7.1.2. Contract 15: Modify Log Entry 22](#_Toc36561421)

[7.2. Class Description Vector 22](#_Toc36561422)

[7.2.1. Contract 16: Database and Vector Operations 23](#_Toc36561423)

[8. GUI Subsystem 24](#_Toc36561424)

[8.1. Class Description PICK GUI 24](#_Toc36561425)

[8.2. Class Description Directory GUI 24](#_Toc36561426)

[8.2.1. Contract 17: Directory Operations 24](#_Toc36561427)

[8.3. Class Description Event GUI 25](#_Toc36561428)

[8.3.1. Contract 18: Event Configuration Operations 25](#_Toc36561429)

[8.4. Class Description Team Configuration GUI 25](#_Toc36561430)

[8.4.1. Contract 19: Team Configuration Operations 25](#_Toc36561431)

[9. Detailed Description of Splunk Class 26](#_Toc36561432)

[9.1.1. Contract 20: Splunk Enterprise Operations 26](#_Toc36561433)

[10. Detailed Description of Main Class 27](#_Toc36561434)

[11. Database 28](#_Toc36561435)

11.1. [Database Schema 28](#_Toc36561436)

**12.** **APPENDIX……................................................................................................................................... 30**

12.1. CLEANSER FILE ………………………………………………………………………………….. 30

Introduction

Section 1 will introduce the concept of a software design document, it will provide a scope for the PICK project, give a comprehensive list of definitions, acronyms and abbreviations, and an overview of the entire document.

## Purpose and Intended Audience

Software design is the process in which requirements are translated into a concrete representation of the software incorporating classes, interfaces, and data necessary for the proper implementation. This document will serve as a central guidance which will be used to aid both LSH when using the final completed product and us; Team 13, for implementing new features. As such the intended audience of this document encompasses the clients (Dr. Oscar Perez, Mr. Vincent Fonseca, Ms. Herandy Vazquez, Mr. Baltazar Santaella, Ms. Florencia Larsen,) Team 13 (Eduardo Lara, Irvin Bosques, Gerardo Armenta, Hector Dozal, and Victor Vargas,) and the UTEP guidance team who will be providing input and constructive criticism on the overall structure of this document.

## Scope of Product

We will be creating a PMR Insight Collective Knowledge (PICK) tool to facilitate the process of writing reports by the White Team (LSH) about the ability of the blue team to defend against cyber-attacks by the red team. This software’s primary goal will consist of facilitating the job of the white team by employing different tools including: Sorting by chronological order based on the date of ingestion, automatic creation of a graphs to better represent when an attack has occurred, creation of vectors to organize all log files that pertain to a singular event, as well as other requirements described within this document. Our system shall focus firstly on improving the analysis of log files, our system shall not use the internet in any way to ensure the security of the system. The success of this project will be determined by the benefits given to the White team upon using this system.

## References

[1] E. Tai-Ramirez & S. Roach, SRS\_v7. Internet: https://github.com/CS4311-spring-2020/pick-tool-team06-team-404/blob/master/doc/SRSv7.pdf, 2020 (Jan. 30, 2020).

## Definitions, Acronyms, and Abbreviations

### Definitions

|  |  |
| --- | --- |
| **Log Entry** | Splunk takes the validated log files and convert them into normalized data. The normalized data are called log entries. Users of the system can filter and edit log entries. |
| **Data Validation** | Data validation is the process of inspecting data in the cleansed log files based on predefined data validation rules. |
| **Data Cleansing** | Data cleansing is the removal of unwanted characters from uncleansed TMUX log file; removal of blank rows from uncleansed excel log file; and removal of blank lines from uncleansed log file. |

### Acronyms

|  |  |
| --- | --- |
| **SDD** | Software Design Document |
| **LSH** | Lethality, Survivability, and HSI Directorate |
| **PICK** | Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System |
| **PMR** | Prevent, Mitigate, and Recover |
| **UTEP** | University of Texas at El Paso |
| **ERD** | Entity/Relationship diagram |

### Abbreviations

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

## Overview

The Software Design Document is divided into 4 major sections each with various subsections. The 4 main sections of the SDD are the following:

### Introduction

This will introduce the entire SDD as well as all the different subsections within it. The main subsections in section 1 are the following: Purpose of the document, Scope of the product, References and Definitions, Acronyms, and Abbreviations.

### Decomposition Description

This section will describe different components that will be involved in the design of the system. A UML component diagram will also be provided in this section.

### Detailed Description of Components

For each component or subsystem listed in section 3 a detailed description will also be provided. The description will also contain all the different contracts for those specific components.

### Database

In this section we will explore the database system that will be used to store information pertaining to the system. An entity-relationship schema will also be provided.

# Decomposition Description

## System Collaboration Diagram

A close up of text on a white background

Description automatically generated

## Subsystem and Component Descriptions

This section will provide a description of each subsystem and component of the system, specified in the previous section. It will talk about main things of each subsystem indicate where the classes are described in following sections.

**Subsystem Name:** Visual subsystem and Real Time Actualization Subsystem

**Subsystem Purpose:** The purpose of this system is to display the log entries in a visual representation with Nodes and Icons.

**Subsystem** **Contract:** Contracts 1,2 more details in section 3

**Subsystem Name:** Realtime Actualization subsystem

**Subsystem Purpose:** The purpose of this system is to correlate the data between the table and graph

**Subsystem Contract:** Contracts 3,4,5,6,7,8 more details in section 4.

**Subsystem Name:** Analysis Subsystem

**Subsystem Purpose:** The purpose of this system is to ingest, cleanse and validate log files and send them to Splunk class for uploading.

**Subsystem Contract:** Contracts 9,10 more details in section 5.

**Subsystem Name:** Configuration Subsystem

**Subsystem Purpose:** The purpose of this system is to contain all the configuration of the system for that current project.

**Subsystem Contract:** Contracts 11,12,13 more details in section 6

**Subsystem Name:** Documentation Subsystem

**Subsystem Purpose:** The purpose of this subsystem is to provide the multiple configuration information for other subsystems including, directory, event and team configuration.

**Subsystem Contract:** Contracts 14,15,16 more details in section 7.

**Subsystem Name:** GUI Subsystem.

**Subsystem Purpose:** This subsystem corresponds to the user interface aspect of the system, shows the elements and displays information.

**Subsystem Contract:** Contracts 17,18,19 more details in section 8.

**Subsystem Name:** Splunk Subsystem

**Subsystem Purpose:** Establish a connection with Splunk Enterprise and communicate with it

**Subsystem Contract:** Contract 20 more details in section 9

**Subsystem Name:** Main

**Subsystem Purpose:** Establish main flow of program and trigger events by user.

**Subsystem Contract:** No contracts

## Dependencies

This section will talk about the dependencies that the program has, by subsystem and how this can impact development.

The Splunk subsystem will depend on the Splunk Enterprise Software and Splunk Python SDK, this can influence the development of the system if the main Splunk software or the SDK are updated and deprecate any of their methods or make significant changes to the API. Any big change can lead to a delay on the development of the PICK system.

The Graph Subsystem, specifically the graph class will be dependent on the library that is chosen to provide that view. Given that Maltego does not have an SDK that delays the graphing aspect of the software until an appropriate external tool can be found. Same as with Splunk, if the chosen tool undergoes a great change, the development time will increase.

The Log subsystem and the Configuration subsystem will interact with a local database. The tool that is used for this will increase the development time, depending on its complexity and the changes the tool undergoes.

Python: 3.7

Splunk Enterprise and SDK Support: 8.0

OCR:

Pytesseract (Library) 0.3.4

Pillow 7.1.2

Tesseract-OCR 3.05.02

Speech Recognition:

SpeechRecognition 3.8.1

Pocketsphinx Python 0.1.15

SphinxBase 5prealpha

PocketSphinx 5prealpha

Cleansing:

FTFY 5.5.2

Unidecode 1.1.1

MongoDB:

PyMongo 3.10.1

Mongo DB Community Server 4.2.6

Graphing:

PyQTGraph 0.10

Graphviz 2.44

Pydot 1.4.1

# Detailed Description of Visual Subsystem

Each component or subsystem listed in section 2.2 are described with a detailed design description for clarification. They all include their respective classes with their responsibilities, collaborations, contracts, and protocols.

## Component Description

|  |  |
| --- | --- |
| **Component Name:**  Visual | **Purpose:**  It is the component that encompasses the graphical view of the data and the tabular view of the data. The data presented come from the log entries chosen by the user to be part of the vector. |
| **Classes:**   1. Node 2. Icon 3. Real Time Actualization Subsystem | |

## Class Description Node

|  |  |
| --- | --- |
| **Class Name**: Node | |
| **Description:** Visual/Tabular representation of a log entry | |
| **Superclass**: Log Entry | |
| **Subclasses**: None | |
| **Private Responsibilities**   * + Knows Node Visibility   + Knows At least one graph   + Knows Node Id Visibility   + Knows Node Name Visibility   + Knows Node Timestamp Visibility   + Knows Node Description Visibility   + Knows Log Entry Reference Visibility   + Knows Log Creator Visibility   + Knows Event Type Visibility   + Knows Icon Type Visibility   + Knows Source Visibility   + Knows Node Visibility   + Knows Relationship ID   + Knows Parent ID   + Knows Child ID   + Knows Label   + Can Make a relationship between a parent and a child node. | |
| **Contract 1**: Node Information Operations | |
| **Responsibilities** | **Collaborations** |
| Provide Node Details  Modify Node Details | Graph  PICK GUI |

### Contract 1: Node Information Operations

Purpose of this contract is to request the icon to display a certain information and image.

Protocol: Get Team

Method Name: getTeam()

Type of Output: Returns color of the team (e.g. Red, Blue, White)

Type of Input: None

Pre-Conditions: Request team color

Post-Conditions: Team color associated to node

Protocol: Display Team Icon

Method Name: displayTeamIcon()

Type of Output: None, displays icon referring to the team

Type of Input: None

Pre-Conditions: Node has a team associated to it

Post-Conditions: Node icon is displayed

Protocol: Get Node Details

Method Name: get\_details(node\_id)

Type of Output: Returns a dictionary representing the information of the Node where the key is the field and the value is the value of that field, all strings.

Type of Input: An integer representing the id of the Node

Pre-Conditions: Node exists

Post-Conditions: Node information is returned

Protocol: Make Node Relationship

Method Name: add\_Relationship(parent, child)

Type of Output: Returns a relationship information formed between a parent and a child Node.

Type of Input: Two integers representing the ids of the Nodes.

Pre-Conditions: Nodes exists

Post-Conditions: Relationship information between two Nodes is returned.

## Class Description Icon

|  |  |
| --- | --- |
| **Class Name**: Icon | |
| **Description:** Visual image of a node | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * + Knows Icon Name   + Knows File Path   + Can Store the attributes it knows | |
| **Contract 2:** Provide Image to Display | |
| **Responsibilities** | **Collaborations** |
| Provide Image to display as Icon | Node |

### Contract 2: Provide Image to Display

Contract responsible for displaying an icon depending on the information provided by node.

Protocol: Get Image

Method Name: getImage()

Type of Output: Image Resource, to represent the icon.

Type of Input: None

Pre-Conditions: Request a change of image.

Post-Conditions: Image displayed as icon.

# Detailed Description of Real Time Actualization Subsystem

|  |  |
| --- | --- |
| **Component Name:**  Real Time Actualization | **Purpose:**  The representation of all nodes in a vector, either in a graph or a table; any changes made on either representation will be shown on the corresponding view type. |
| **Classes:**   1. Graph 2. Table | |

## Class Description Graph

|  |  |
| --- | --- |
| **Class Name**: Graph | |
| **Description:** Class that will show a visual representation of significant log entries in graphical format | |
| **Superclass**: None | |
| **Subclasses**: Table | |
| **Private Responsibilities**   * Knows Export Format * Knows Orientation * Knows Interval Units * Knows Interval * Knows Position of Nodes * Knows Position of Relationships * Knows The size of the current graph/table * Can display an error when the user attempts to delete a relationship when one is not selected. * A display error is shown when the user attempts to delete a node when non is selected. * When filtering is complete, * The node table shall display nodes that meet the criteria. * The relationship table shall display relationships that meet the criteria. * The graph shall display nodes that meet the criteria. * Can export graph in selected format | |
| **Contract 3:** Get information from Graph class | |
| **Responsibilities** | **Collaborations** |
| * Provides information graph class | Table |
| **Contract 4**: Modify Graph | |
| **Responsibilities** | **Collaborations** |
| * + Can add new nodes.   + Can add relationships to nodes.   + Can delete relationships.   + Undo any changes to graph since last commit   + Commit any changes made to graph and save it   + Change icon type of node   + Do “Add Icon” operation   + Do “Delete Icon” operation   + Allow nodes to be repositioned in graph   + Display nodes in graphical view   + Display nodes in table view   + Can delete a node.   + Toggle node visibility | PICK GUI (4) |

### Contract 3: Get Information from Graph

Purpose of this contract is to obtain information from the Graph class to sync with table.

Protocol: Get Class Information

Method Name: getClassInfo()

Type of Output: List of maps that represent the nodes and their values of the graph

Type of Input: None

Pre-Conditions: There must be a vector created. There must be log entries in the system.

Post-Conditions: Graph data is received.

### Contract 4: Modify Graph

Purpose of this contract is to modify the graph by allowing user to edit nodes, move location of nodes, and create new nodes.

Protocol: Add Node

Method Name: addNode(List<String> nodeInfo)

Type of Output: Void, it applies changes to graph.

Type of Input: List of Strings, information to create the Node to be Displayed.

Pre-Conditions: The added node must be linked with a log entry.

Post-Conditions: A node is added to the graph.

Protocol: Change Icon

Method Name: iconChange(Node node)

Type of Output: void, icon change is applied to a node.

Type of Input: Node, the node to which the change is going to be applied.

Pre-Conditions: Node change action is triggered.

Post-Conditions: Icon of a Node is changed.

Protocol: Move Node

Method Name: moveNode(int x, int y)

Type of Output: None

Type of Input: Node location on graph

Pre-Conditions: Node must be created and in vector of graph

Post-Conditions: Node location is changed

## Class Description Table

|  |  |
| --- | --- |
| **Class Name**: Table | |
| **Description:** Class that will show a visual representation of significant log entries in tabular format | |
| **Superclass**: Graph | |
| **Subclasses**: None | |
| **Private Responsibilities:**   * Knows Node’s relationships. * Knows Node’s information. | |
| **Contract 5:** Gets information from table class | |
| **Responsibilities** | **Collaborations** |
| * Requests Information of table class | Graph |
| **Contract 6:** Modify Table |  |
| **Responsibilities** | **Collaborations** |
| * Change the visibility status of a node. * Order nodes in ascending and descending order * Update row information. | PICK GUI |

### Contract 5: Gets node information from Node Class

This contract is responsible for obtaining the necessary information to display in the table view.

Protocol: Get Node Information

Method Name: getNodeInfo(List<String> strings)

Type of Output: List of Strings

Type of Input: List of Strings

Pre-Conditions: There must be a vector created. There must be log entries in the system.

Post-Conditions: Shows up to date node information.

### Contract 6: Modify Table

Purpose of this contract is to modify the table by allowing user to edit nodes, and create relationships within the table view.

Protocol: Edit Row

Method Name: editRow(List<String> fieldToChange, List<String> newData, int row)

Type of Output: Void

Type of Input: List of String, List of String, Int. Row to be edited in integer, and two lists of strings, one representing the fields to change and one to represent the new data.

Pre-Conditions:

Post-Conditions: Row has been edited.

Protocol: Set Column Visibility

Method Name: ColumnShow()

Type of Output: None

Type of Input: None

Pre-Conditions:

Post-Conditions: The table show the column

# Detailed Description of Analysis Subsystem

|  |  |
| --- | --- |
| **Component Name:**  Data Processing | **Purpose:**  Handles the ingestion, cleansing, and validation of the files uploaded to PICK. Here the log files are generated prior to being sent to Splunk. |
| **Classes:**   1. Log File 2. Enforcement Action Report | |

## Class Description Log File

|  |  |
| --- | --- |
| **Class Name**: Log File | |
| **Description:** File from where entries are going to be extracted | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * + Knows Log File Name   + Knows Cleansing Status   + Knows Validation Status   + Knows Ingestion Status   + Knows Acknowledgement Status   + Can perform data transformation operation of a log file   + Can perform data cleansing operation of log file   + Can perform data validation operation of log file   + Performs the data ingestion operation once data ingestion operation is complete   + Can update ingested log file   + Can do search operation by keyword   + Can perform different search mechanisms     - Logical searching     - Wildcard searching | |
| **Contract 7:** Log File Information | |
| **Responsibilities** | **Collaborations** |
| * Provide Validation and Cleansing Status * Provide Cleansing Issues * Provide Ingestion status | PICK GUI  Enforcement Action Report |

Table 2

### Contract 9: Log File Information

Protocol: Get Validation Status

Method Name: getValStat()

Type of Output: String specifying validation status

Type of Input: None

Pre-Conditions: The log file has a validation status

Post-Conditions: Log file validation status set to pass or fail

Protocol: Get Cleansing Status

Method Name: getCleanStat()

Type of Output: String specifying cleansing status

Type of Input: None

Pre-Conditions: The log file has a cleansing status

Post-Conditions: Log file cleansing status set to pass or fail

## Class Description Enforcement Action Report

|  |  |
| --- | --- |
| **Class Name**: Enforcement Action Report | |
| **Description:** Provides insight to errors during cleansing and validation | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows Line Number * Knows Error Message * Knows Log File Validation Status * Generates an enforcement action report and set the validation status of the log file to “fail” of data validation fails | |
| **Contract 8:** Provide Enforcement Information | |
| **Responsibilities** | **Collaborations** |
| Provide information of the enforcement report | PICK GUI |

Table 3

### Contract 10: Provide Enforcement Information

Purpose of this contract is to provide an enforcement action report regarding the status of all log files during the cleansing and validation process.

Protocol: Provide Enforcement Information

Method Name: getEnforcmentInfo()

Type of Output: String specifying any errors that were found in either cleansing or validating log files

Type of Input: None

Pre-Conditions: Log files go through cleansing and/or validating

Post-Conditions: Enforcement report shows any errors with log files pertaining to cleansing and validating

# Detailed Description of Configuration Subsystem

|  |  |
| --- | --- |
| **Component Name:**  Project Configuration | **Purpose:**  Anything that has to do with the system’s setting regarding the event configuration for log ingestion, path directories for the location of the original files to be uploaded to the system are managed here. |
| **Classes:**   1. Event Configuration 2. Directory Configuration 3. Team Configuration | |

## Class Description Event Configuration

|  |  |
| --- | --- |
| **Class Name**: Event Configuration | |
| **Description:** Class that will contain all the event configuration for the current project | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows Event Name * Knows Event Description * Knows Event Start Timestamp * Knows Event End Timestamp * Verifies if timestamp is correct. | |
| **Contract 9:** Event Information Operations | |
| **Responsibilities** | **Collaborations** |
| Provide Event details  Modify Event configuration | Enforcement Action Report  PICK GUI |

### Contract 11: Event Information Operations

Purpose of this contract is to allow user to create a new event and enter specific event configuration details.

Protocol: Set Event Name

Method Name: setEventName(String eventName)

Type of Output:

Type of Input: String specifying event name

Pre-Conditions: New event, or edit event, has been selected

Post-Conditions: Event name is created, or updated

Protocol: Set Event Description

Method Name: setEventDescription(String eventDescription)

Type of Output:

Type of Input: String specifying event description

Pre-Conditions: New event, or edit event, has been selected

Post-Conditions: Event description is created, or updated

Protocol: Set Event Date

Method Name: setEventDate(int date)

Type of Output:

Type of Input: Integer specifying event date

Pre-Conditions: New event, or edit event, has been selected

Post-Conditions: Event date is created, or updated

Protocol: Set Event Time

Method Name: setEventTime(int time)

Type of Output:

Type of Input: Integer specifying event time

Pre-Conditions: New event, or edit event, has been selected

Post-Conditions: Event time is created, or updated

## Class Description Directory Configuration

|  |  |
| --- | --- |
| **Class Name**: Directory Configuration | |
| **Description:** Class that will contain all the directory paths and configuration | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows directory Structure * Validates Directory Structure * Validates Directories and their paths | |
| **Contract 10:** Directory Information Operations | |
| **Responsibilities** | **Collaborations** |
| Provide Directory path of a team folder  Modify Directory Configuration | Enforcement Action Report  PICK GUI |

Table 1

### Contract 12: Directory Information Operations

Protocol: Get Directory Path

Method Name: getDirPath(String team)

Type of Output: String representing the path of a team

Type of Input: String specifying which team to get the path

Pre-Conditions: Folder structure established

Post-Conditions:

Protocol: Reset Directory Configuration

Method Name: resetDirConfig()

Type of Output: None

Type of Input: None

Pre-Conditions:

Post-Conditions: Directory Configuration GUI is started

## Class Description Team Configuration

|  |  |
| --- | --- |
| **Class Name**: Team Configuration | |
| **Description:** Class that will contain the team configuration | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows Individual IP * Knows Team connection values * Knows Connection Status * Validates connection values * Updates connection status | |
| **Contract 11:** Provide Connection Status | |
| **Responsibilities** | **Collaborations** |
| 1. Provide connection status | PICK GUI |

Table 2

### Contract 13: Provide Connection Status

Protocol: Connection Status

Method Name: getConnectionStatus()

Type of Output: String representing current connection status (“Connected”, “Disconnected”, “Reconnecting”, “Connecting”)

Type of Input: None

Pre-Conditions: None

Post-Conditions: None

# Detailed Description of Documentation Subsystem

|  |  |
| --- | --- |
| **Component Name:**  Documentation | **Purpose:**  All the log entries that have been created from the log files; all the significant log entries corresponding to a certain vector. |
| **Classes:**   1. Log Entry 2. Vector | |

## Class Description Log Entry

|  |  |
| --- | --- |
| **Class Name**: Log Entry | |
| **Description:** Represents a single piece of data in an event | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows Log Entry Number * Knows Log Entry Timestamp * Knows Log Entry Content * Knows Host * Knows Source * Knows Source Type * Knows Significant * Marks a log entry as significant or not | |
| **Contract 12:** Log Entry Details Operations | |
| **Responsibilities** | **Collaborations** |
| Provide Log Entry Information | PICK GUI, Node |
| **Contract 13:** Modify Log Entry | |
| **Responsibilities** | **Collaborations** |
| Modify Log entry number, timestamp or content  Mark Log entry as significant. | PICK GUI  PICK GUI |

Table 5

### Contract 14: Log Entry Details Operations

Purpose of this contract is to get log entry data.

Protocol: Get Log Entry Information

Method Name: getLogEInfo()

Type of Output: Map object with string to string pairs where key is log entry field and value is value of that field.

Type of Input: None

Pre-Conditions: None

Post-Conditions: None

### Contract 15: Modify Log Entry

Purpose of this contract is to be able to modify log entry details and mark an entry as significant.

Protocol: Modify Log Entry Number

Method Name: setLogENum(int logEntryReference)

Type of Output: None

Type of Input: Integer, unique integer to refer to the log entry

Pre-Conditions:

Post-Conditions: Log entry number changed

Protocol: Modify Log Entry Timestamp

Method Name: setLogETime(String timestamp)

Type of Output: None

Type of Input: String representing a timestamp in Zulu Time

Pre-Conditions: None

Post-Conditions: Timestamp of log entry modified

Protocol: Modify Log Entry Content

Method Name: setLogEContent(String logEntryContent)

Type of Output: None

Type of Input: String representing the new log entry content

Pre-Conditions: None

Post-Conditions: Content of log entry modified

Protocol: Mark Entry as Significant

Method Name: setSignificant(boolean significant)

Type of Output: None

Type of Input: Boolean representing if entry is significant

Pre-Conditions: None

Post-Conditions: Entry is marked significant

## Class Description Vector

|  |  |
| --- | --- |
| **Class Name**: Vector | |
| **Description:** Objective that is defined by the user | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows Vector Name * Knows Vector Description * Can Save the changes to permanent storage * Establishes connection between the lead and the user | |
| **Contract 14:** Database and Vector Operations | |
| **Responsibilities** | **Collaborations** |
| Requests information from Database.  Allow user to do “Pull” operation  Allow user to do “Push” operation  Add Vector to DB  Add log entry to vector | Log Entry  PICK GUI  PICK GUI  PICK GUI  PICK GUI |

Table 4

### Contract 16: Database and Vector Operations

This contract is to requests or update information of a vector based on the database status.

Protocol: Set Vector Name

Method Name: setVectorName(String vectorName)

Type of Output: Void, sets a property of the class

Type of Input: String, represents vector name

Pre-Conditions: name of vector not set.

Post-Conditions: Name of vector is set.

Protocol: Save to Database

Method Name: saveDB()

Type of Output: Void, performs an action onto a database

Type of Input: Void, performs with fields in class.

Pre-Conditions: Save Vector action triggered.

Post-Conditions: Vector and information is saved to Database.

Protocol: Add Log Entry

Method Name: addLogEntry(LogEntry logEntry)

Type of Output: Void, performs action on class fields

Type of Input: Log Entry, received an instance of the log entry to be added to the vector.

Pre-Conditions: Log Entry is not in Vector already

Post-Conditions: Log entry added to vector.

Protocol: Pull from Database

Method Name: pulFromDB()

Type of Output: Void, modifications done on class

Type of Input: Void

Pre-Conditions: Updates are possible

Post-Conditions: Vector updated from database.

Protocol: Push to database

Method Name: pushToDB()

Type of Output: Void

Type of Input: Void, information obtained from vector itself

Pre-Conditions: Changes made to Vector

Post-Conditions: Database updated.

# GUI Subsystem

|  |  |
| --- | --- |
| **Component Name:**  GUI Subsystem | **Purpose:**  This component provides a user interface to connect the user and the system and to establish a way for the user to interact with it. |
| **Classes:**   1. PICK GUI 2. Directory GUI 3. Event GUI 4. Team Configuration GUI | |

## Class Description PICK GUI

|  |  |
| --- | --- |
| **Class Name**: PICK GUI | |
| **Description:** Represents the main user interface that provides the user with the interactions for the main flow of events. | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows User triggered events * Knows Information inputted in text boxes | |
| **Contract 15:** Main Operations | |
| **Responsibilities** | **Collaborations** |
| Trigger other classes events | Main |
|  | |

Table 3

## Class Description Directory GUI

|  |  |
| --- | --- |
| **Class Name**: Directory GUI | |
| **Description:** Represents the interface that provides the user with a way to establish the directories of the teams and the root directory. | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows User triggered events * Knows Information inputted in text boxes | |
| **Contract 16:** Directory Operations | |
| **Responsibilities** | **Collaborations** |
| Provide directory paths | Directory Configuration |

Table 4

### Contract 17: Directory Operations

Protocol: Get Directory Paths

Method Name: getDirPaths()

Type of Output: String Array where each element is a directory path in the order of Root, Red Team, Blue Team, White Team

Type of Input: None

Pre-Conditions: Save Directory Path event triggered

Post-Conditions: Can start Data Ingestion.

## Class Description Event GUI

|  |  |
| --- | --- |
| **Class Name**: Event GUI | |
| **Description:** Represents the interface with which the user establishes the main event configuration items. | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows User triggered events * Knows Information inputted in text boxes | |
| **Contract 17:** Event Configuration Operations | |
| **Responsibilities** | **Collaborations** |
| Provides event configuration information | Event |

Table 5

### Contract 18: Event Configuration Operations

Protocol: Get Event Configuration

Method Name: getEventConfig()

Type of Output: String to String Dictionary where key is event configuration field and value is value of that field.

Type of Input: None

Pre-Conditions: Save Event Configuration Event Triggered

Post-Conditions:None

## Class Description Team Configuration GUI

|  |  |
| --- | --- |
| **Class Name**: Team Configuration GUI | |
| **Description:** Represents the interface with which the user establishes the team configuration items. | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows User triggered events * Knows Information inputted in text boxes | |
| **Contract 18:** Team Configuration Operations | |
| **Responsibilities** | **Collaborations** |
| Provide Team Configuration Information | Team Configuration |

Table 6

### Contract 19: Team Configuration Operations

Protocol: Get Team Configuration

Method Name: getTeamConfig()

Type of Output: Dictionary of String to String, where key is team configuration field and value is value of that field.

Type of Input: None

Pre-Conditions: Save Team Configuration event triggered

Post-Conditions: None

# Detailed Description of Splunk Class

|  |  |
| --- | --- |
| **Component Name:**  Splunk | **Purpose:**  This component is the connection between the PICK application and Splunk. This will allow for log files to be uploaded to Splunk for filtering and generating the log entries. |
| **Classes:**   1. Splunk | |

|  |  |
| --- | --- |
| **Class Name**: Splunk | |
| **Description:** Represents the connection to Splunk Enterprise | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   * Knows Connection details to Splunk Enterprise * Establishes a connection to Splunk Enterprise * Can Upload Files to Splunk Enterprise | |
| **Contract 19:** Splunk Enterprise Operations | |
| **Responsibilities** | **Collaborations** |
| Search log entries from Splunk Enterprise  Upload Log Files to Splunk Enterprise | Log Entry  Log File |

Table 8

### Contract 20: Splunk Enterprise Operations

Protocol: Search Log Entry

Method Name: searchEntries(Array searchParams)

Type of Output: Log entries represented by the log entry class

Type of Input: Filter and Search Parameters represented as an array of strings

Pre-Conditions: Log Files were uploaded to Splunk Enterprise

Post-Conditions: Log Entries are shown in the PICK GUI

Protocol: Upload Log files

Method Name: uploadFiles(String filePath)

Type of Output: None

Type of Input: Path of files to be uploaded as a String

Pre-Conditions: Files not uploaded

Post-Conditions: Files uploaded to Splunk

# Detailed Description of Main Class

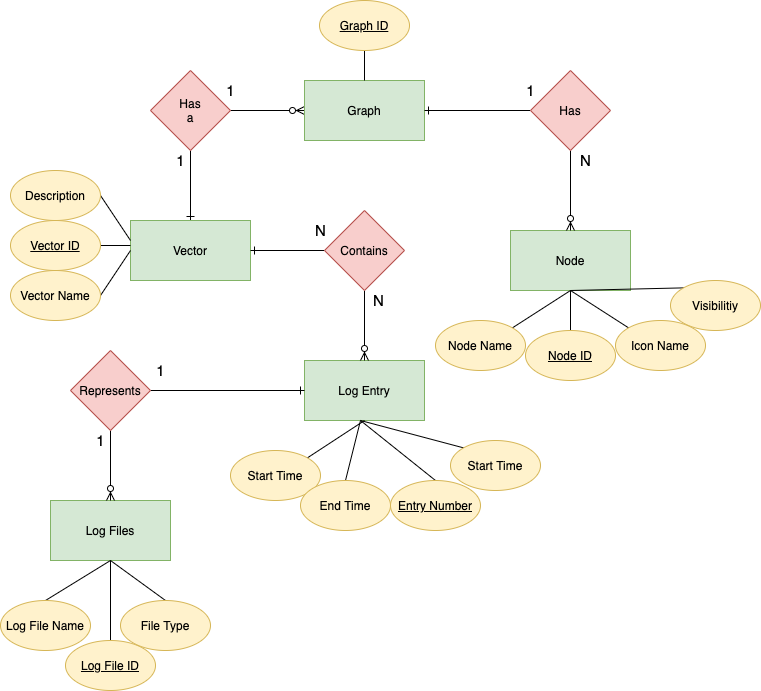
|  |  |
| --- | --- |
| **Component Name:**  Main | **Purpose:**  This component controls the main workflow of the program with the GUI and the other classes. |
| **Classes:**   1. Main | |

|  |
| --- |
| **Class Name**: Main |
| **Description:** Represents the main instance of the program |
| **Superclass**: None |
| **Subclasses**: None |
| **Private Responsibilities**   * Knows information from GUI * Redirects and trigger user events to respective actions * Controls flow of program * Contains instances of classes necessary for the main flow of the program |

Table 8

# Database

## 11.1 Database Schema



# Appendix

## In this section we will be dealing with lose files, their behavior and size not being big enough to be considered its own class or to be given a section. Their behavior will be briefly explained for maintenance purposes.

## Cleanser File

This file contains the behavior of the cleansing algorithm for the Log File class to use. The file contains multiple methods, of which only the file\_cleanse method is important and directly used by Log File class. It requires two main arguments, the name of the file to clean and the name of the output file.

Protocol: Cleanse File

Method Name: file\_cleanse(String filename, String output\_file)

Type of Output: A cleansed file.

Type of Input: Two strings representing the filename and the name of the output file

Pre-Conditions: File Exists

Post-Conditions: File is overwritten clean

## DB Manager File

This file contains the behavior to connect to the database, it also contains the behavior to insert elements (such as vector information and event information) and retrieve the information to open saved projects. It is imported as a file and used directly by the Main class and Event Configuration class, with some of the methods being used publicly. Here are the protocols that are publicly used by the Main class and Event Configuration Class:

Protocol: Insert Event to DB

Method Name: insert\_event(data)

Type of Output: An integer representing the id of the event inserted.

Type of Input: A list object containing information about the vent in the form of strings. (Name, Description, Start Date and End Data)

Pre-Conditions: An event has been created

Post-Conditions: Event is inserted into database and its id is returned.

Protocol: Update Event from DB

Method Name: update\_or\_event(document\_id,data)

Type of Output: A Boolean that represents an acknowledgment of the update with True being acknowledged and False as an error.

Type of Input: An integer that represents the id of the event to update and a list object with the information to update the event (Name, Description, Start Date, End Date)

Pre-Conditions: The event exists already

Post-Conditions: Event is updated in database.

Protocol: Get Single Event

Method Name: get\_single\_event(document\_id)

Type of Output: List object representing the information of the Event in Strings (Name, Description, Start Date, End Date)

Type of Input: Integer representing the id of the document to retrieve.

Pre-Conditions: The event exists in the database.

Post-Conditions: Event is returned.

Protocol: Get All the Events in the Database

Method Name: get\_multiple\_events

Type of Output: A list in which each element is a list of strings representing the information of an event (Name, Description, Start Date, End Date)

Type of Input: None

Pre-Conditions: There is at least one event in the DB

Post-Conditions: List of events returned.

Protocol: Remove Event from DB

Method Name: remove\_event(document\_id)

Type of Output: A boolean representing an acknowledgement of the removal of the event, True representing a successful removal and False a failed one.

Type of Input: An integer representing the id of the event to remove.

Pre-Conditions: Event exists in the Database

Post-Conditions: Event removed from DB

Protocol: Insert Directory Information

Method Name: insert\_directory(rootFolder,whiteFolder,blueFolder,redFolder,id)

Type of Output: An integer representing the id of the inserted directory structure

Type of Input: Four strings representing the paths of the directories and an integer representing the id given to the structure.

Pre-Conditions: The directory information is not in the database already

Post-Conditions: Data inserted into the database

Protocol: Get A Single Directory Structure

Method Name: get\_single\_directory(document\_id)

Type of Output: A dictionary containing the four folders as Strings (with key value as “rootFolder: path” for each folder)

Type of Input: An integer representing the id of the directory structure we want to retrieve

Pre-Conditions: Directory structure exists

Post-Conditions: Directory structure is retrieved

&