PICK

Software Design Document

version 1.5

5/8/2020

**Document Control**

**Approval**

The Guidance Team and the customer shall approve this document.

**Document Change Control**

|  |  |
| --- | --- |
| Initial Release: | 3/2/2020 |
| Current Release: | 1.5 |
| Indicator of Last Page in Document: | **%** |
| Date of Last Review: | 5/8/2020 |
| Date of Next Review: | 5/9/2020 |
| Target Date for Next Update: | 5/9/2020 |

**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. Steve Roach, Jake Lasley

**Customer:** Army Research Lab which consists of Herandy Denisse Vazquez, Baltazar Santaella, Vincent Fonseca, Oscar Perez, and Florencia Larsen.

**Software Team Members:** Zayra Padilla, Erik Macik, Priscilla Mendoza, Jose Lujan, Michael Contreras

**Change Summary**

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 3/2/2020 | Team | Creation of Document |
| 1.2 | 3/08/2020 | Team | Finished section 2. |
| 1.3 | 3/08/2020 | Team | Finished section 3-6 |
| 1.4 | 3/08/2020 | Team | Finished section 7 |
| 1.5 | 3/29/2020 | Team | Finished Collaboration diagram |
| 1.6 | 3/30/2020 | Team | Added protocols |
| 1.7 | 6/9/2020 | Team | Updated Class Descriptions, Collaboration Diagram |

Table of Contents

[**DOCUMENT CONTROL II**](#_heading=h.30j0zll)

[Approval ii](#_heading=h.3cqmetx)

[Document Change Control ii](#_heading=h.1rvwp1q)

[Distribution List ii](#_heading=h.4bvk7pj)

[Change Summary ii](#_heading=h.2r0uhxc)

[**1.**](#_heading=h.1664s55) **INTRODUCTION 1**

[1.1.](#_heading=h.3q5sasy) Purpose and Intended Audience 1

[1.2.](#_heading=h.25b2l0r) Scope of Product 1

[*1.2.1.*](#_heading=h.kgcv8k) *Database 1*

[*1.2.2.*](#_heading=h.34g0dwd) *Notification Manager 1*

[*1.2.3.*](#_heading=h.1jlao46) *Unit Conversion 1*

[1.3.](#_heading=h.43ky6rz) References 1

[1.4.](#_heading=h.17dp8vu) Definitions, Acronyms, and Abbreviations 2

[*1.4.1.*](#_heading=h.2iq8gzs) *Definitions 2*

[*1.4.2.*](#_heading=h.xvir7l) *Acronyms 2*

[*1.4.3.*](#_heading=h.3hv69ve) *Abbreviations 2*

[1.5.](#_heading=h.1x0gk37) Overview 2

[**2.**](#_heading=h.4h042r0) **DECOMPOSITION DESCRIPTION 3**

[2.1.](#_heading=h.2w5ecyt) Scope 3

[2.2.](#_heading=h.1baon6m) Use 3

[2.3.](#_heading=h.3vac5uf) Subsystem Description 3

[2.4.](#_heading=h.2afmg28) Hierarchy Graphs 4

[*2.4.1.*](#_heading=h.pkwqa1) *Database Manager Subsystem 4*

[*2.4.2.*](#_heading=h.39kk8xu) *Database Manager (API) 4*

[*2.4.3.*](#_heading=h.1opuj5n) *Unit Conversion 5*

[*2.4.4.*](#_heading=h.48pi1tg) *Notification Manager 5*

[**3.**](#_heading=h.2nusc19) **DEPENDENCY DESCRIPTION 6**

[3.1.](#_heading=h.1302m92) Scope 6

[3.2.](#_heading=h.3mzq4wv) Use 6

[3.3.](#_heading=h.2250f4o) Collaboration Description 6

[**4.**](#_heading=h.haapch) **DETAILED DESIGN 8**

[4.1.](#_heading=h.319y80a) Scope 8

[4.2.](#_heading=h.1gf8i83) Use 8

[4.3.](#_heading=h.40ew0vw) Components 8

[*4.3.1.*](#_heading=h.2fk6b3p) *Database Manger 8*

[*Scenario 1: Perform insert, update, select, or delete query. 8*](#_heading=h.upglbi)

[*Scenario 1: Upload. 9*](#_heading=h.3ep43zb)

[*Scenario 2: Download. 9*](#_heading=h.1tuee74)

[*4.3.2.*](#_heading=h.4du1wux) *Unit Conversion 9*

[*Scenario 1: Request conversion of units. 9*](#_heading=h.2szc72q)

[*4.3.3.*](#_heading=h.184mhaj) *Notification mails 10*

[*Scenario 1: Edit data. 10*](#_heading=h.3s49zyc)

[*Scenario 2: Add new weather station. 10*](#_heading=h.279ka65)

[4.4.](#_heading=h.sqyw64) Database Schema 11

# Introduction

## Purpose and Intended Audience

The purpose of the Software Design Document (SDD) is to give the developers a clear description of the design of the system. It will provide a way for software development to go about by giving a thorough understanding of what is to be built and how to build the system. It will list enough detail of all subsystems and components that make up the system and how each responsibility collaborates with other components to fulfill the system. The intended audience of the SDD is Dr. Oscar Perez, Mr. Vincent Fonseca, Ms. Herandy Vazquez, Mr. Baltazar Santaella, Ms. Florencia Larsen and the guidance team.

## Scope of Product

The Lethality, Survivability, and HSI Directorate (LSH) recognizes the complexity and the time it takes to analyze the applicable logs, observation notes, and other artifacts gathered from an adversarial assessment from the red, blue, and white teams and generate a report that represents the events that took place during the adversarial assessment. They want a system that would aid their analysts in correlating red team’s activities to blue team’s responses and represent the events that took place during an adversarial assessment graphical.

The University of Texas at El Paso (UTEP) and LSH are collaborating to develop Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK) that will provide the ability to correlate red team’s activities to blue team’s responses and graphically represent the events that took place during an adversarial assessment.

## References

E. T. Ramirez, A. T. Gates, S. T. Salamah, S. T. Roach, and P. T. Hanson, “Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK) Software Requirements Specification .” .

## Definitions, Acronyms, and Abbreviations

### Definitions

|  |  |
| --- | --- |
| **TERM** | **DEFINITION** |
| 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. |
| Data Validation | Data validation is the process of inspecting data in the cleansed log files based on predefined data validation rules. |
| 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. |
| Significant Log Entry | A log entry selected by the user and associated with a vector. The attributes are the same as for a log entry. The system stores significant log entries. Splunk stores log entries in the normalized data files. |
| Timestamp | Denotes time in hours:minutes, date in month:date:year, and section in am/pm. |
| Significant log entry | Denotes a log entry that is associated to at least one vector. |

### Acronyms

|  |  |
| --- | --- |
| **TERM** | **DEFINITION** |
| UTEP | The University of Texas at El Paso |
| PICK | Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System |
| LSH | Lethality, Survivability, and HSI Directorate |
| PMR | Prevent, Mitigate, and Recover |
| IP | Internet Address |
| AA | Adversarial Assessment |

### Abbreviations

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

## Overview

The SDD is divided into four major sections: Introduction (Section 1), Decomposition Description (Section 2), Detailed Description of Component (Section 3 to 6) and Database (Section 7).

Section 1 includes a brief description of the system, what an SDD is and what purpose it serves, the intended audience and the organization of the document.

Section 2 is made up of three subsections. Section 2.1 provides a Wirfs-Brock Collaboration Diagram. Section 2.2 provides a description of subsystems and components in the diagrams in section 2.1. Section 2.3 describes how the component dependencies will impact development.

Section 3 to 6 is made up of two or more subsections each. Section n.1 provides the component’s description. Section n.2 and so on provides the listed class’s description and its corresponding contract.

Section 7 provides the database schema and layout as well as a brief description of all the different tables in the schema.

# Decomposition Description

## System Collaboration Diagram

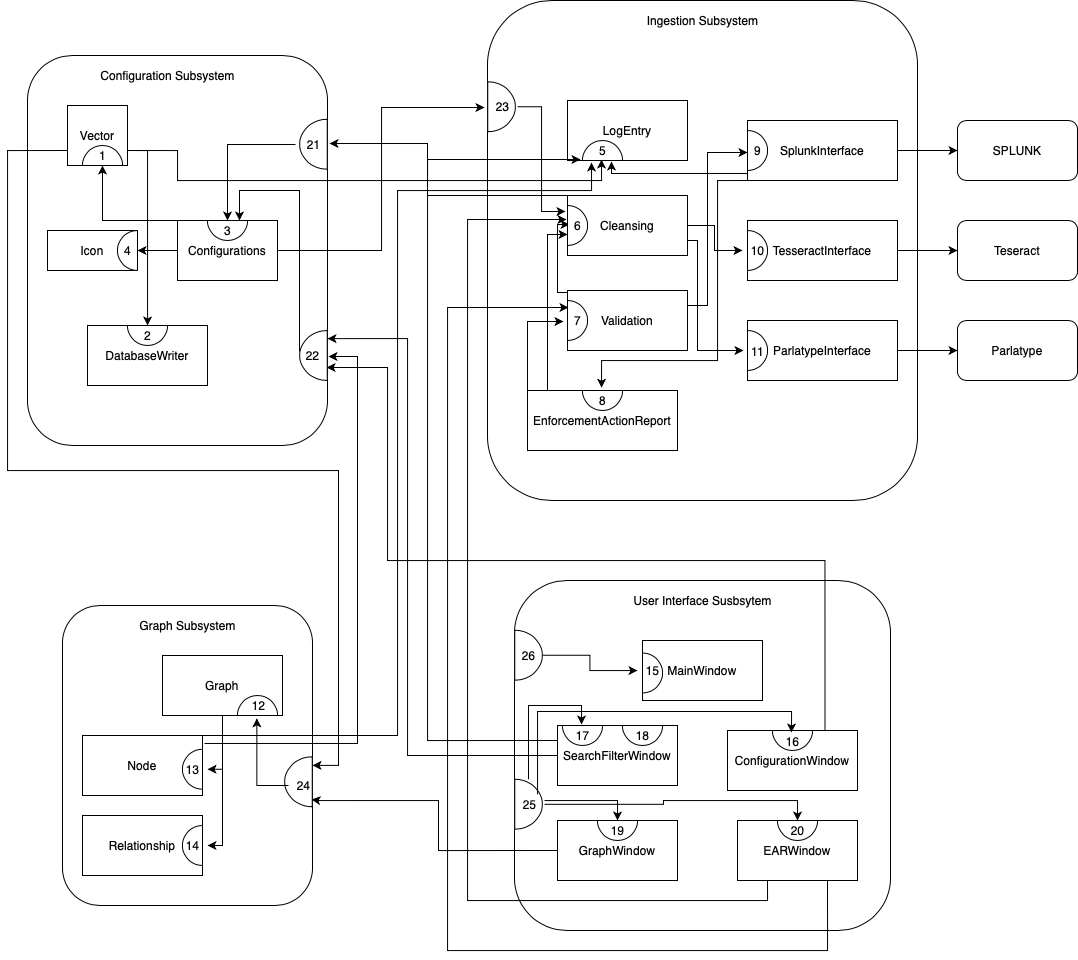


Figure 1: Wirfs-Brock Collaboration Diagram for PICK System

## Subsystem and Component Descriptions

Configuration Subsystem (Section 3)

* Purpose - Know an initial setup for the system that includes event, team, directories, icons, and vectors.
* Supported Contracts
  + 21. Provide log files
  + 22. Provide configuration details

Ingestion Subsystem (Section 4)

* Purpose - Responsible for transcribing, cleansing, and validating log files, and sending log files for Splunk ingestion.
* Supported Contracts
  + 23. Provide log entries

Graph Subsystem (Section 5)

* Purpose - Responsible for creating a graphical representation of a vector and its associations.
* Supported Contracts
  + 24. Create a graphical representation of vector

User Interface Subsystem (Section 6)

* Purpose - Responsible for displaying information to the user and receiving interactions from the user.
* Supported Contracts
  + 25. Send user interaction signals to classes that should be notified of changes.
  + 26. Launch Windows

## Dependencies

The system can be run on Linux, Mac OS and Windows 10.

Required components for system to run:

* Python 3 (version 3.7.0)
  + Python is used to create all the backend of the PICK tool
* PyQt5 (version 5.14.1)
  + This tool is used to create the GUI interface.
* MongoDB (version 4.2.6)
  + This tool is used to store important data generated by the PICK tool.
* pyMongo (version 3.10.1)
  + This library is used to interact with the MongoDB database through Python.
* pandas (version 1.0.3)
  + This tool is used as an aid for cleansing the log entries.
* Splunk-SDK (version 1.6.12)
  + This tool is used to import all the log files to the PICK tool and obtain the log entries.
* GraphViz (version 0.13.2)
  + This tool is used for creating the graph.
* Pytesseract (version 0.3.4)
  + Python-tesseract is a python wrapper for Google's Tesseract-OCR. This tool is used to transcribe image files to text.
* Parlatype (version 2.0.0)
  + This tool is used to transcribe audio files to text.

# Detailed Description of Configuration Subsystem

## Subsystem Description

**Subsystem name:** Configuration Subsystem

**Purpose of subsystem:** Know an initial setup for the system that includes event, team, directories, icons, and vectors.

**Classes contained in subsystem:** Configuration, Vector

## Class Description of Vector

|  |  |
| --- | --- |
| **Class Name**: Vector | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Know the vector ID. * Know the vector name. * Know the vector description. * Know which log entries are associated with this vector * Delete existing vectors along with all of its associations. | |
| **Contract 1:** Manipulate the Vector’s associated log entries. | |
| **Responsibilities** | **Collaborations** |
| 1. Associate log entry to vector. 2. Update vector information in the database based on changed information by the user. | * 1 -> Log Entry (5) * 2 -> DatabaseWriter (2) |
| **Protocol:** add\_log\_entry(log\_entry)   * Pre-Condition:log\_entry is a valid logentry object. * Post-Condition:This vector is now aware of this new log entry by appending to the Vector’s list of known log entries * Description:Add the given log entry to this vector. | |

## Class Description of Database Writer

|  |  |
| --- | --- |
| **Class Name**: DatabaseWriter | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities**   * Know possible mongoDB collections * Know port connection of mongoDB client | |
| **Contract 2:** Manipulate the local mongoDB database | |
| **Responsibilities** | **Collaborations** |
| 1. Write a document to a collection. 2. Retrieve documents in a collection | N/A |
| **Protocol: write\_dict\_to\_collection(dict\_to\_write, collection\_name)**   * Pre-Condition:The dict\_to\_write given is a valid Python native dictionary. The collection\_name is a string that represents a valid collection stored in the local mongoDB database, as defined by constants in the DatabaseWriter class. * Post-Condition**:** The given dictionary has been written to the local mongoDB database at the given collection * Description:Writes the given dictionary as a document to the given collection at the local mongoDB database. | |
| **Protocol: get\_all\_documents\_in\_collection(collection\_name)**   * Pre-Condition:The collection\_name is a string that represents a valid collection stored in the local mongoDB database, as defined by constants in the DatabaseWriter class. * Post-Condition:All entries in the local database collection are returned in a list. * Description:Return a list of all documents in the given collection from the local mongoDB database. | |

## Class Description of Configurations

|  |  |
| --- | --- |
| **Class Name**: Configurations | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities**   * Obtain files in known directories. * Obtain event details * Obtain team details * Obtain icon details * Update existing icon fields. | |
| **Contract 3:** Manipulate the application’s configuration details. | |
| **Responsibilities** | **Collaborations** |
| 1. Know the event’s name. 2. Know the event’s description. 3. Know the event’s start date. 4. Know the event’s start time. 5. Know the event's end date. 6. Know the event's end time. 7. Know the lead host machine where the master vector DB is stored. 8. Know the number of established connections. 9. Know icon names. 10. Know icon sources. 11. Know the directories needed to retrieve log files. 12. Delete selected icons. | 13 -> Icon (4)  14 -> Icon (4)  16 -> Icon (4) |
| **Protocol: set\_team(isLead, lead\_IP, established\_connections)**   * Pre-Condition: isLead is a boolean value that represents whether or not the user is a lead. lead\_IP represents the IP address of the lead’s machine. established\_connections represents the number of current connections to the current lead’s machine. * Post-Condition: The configuration object now knows whether or not the current user is a lead, the lead’s IP address, and the number of current connections to the lead’s machine. * Description: Set the team information for this Configuration object. | |
| **Protocol: set\_event(name, description, startTime, endTime)**   * Pre-Condition: Name and description are valid strings. startTime and endTime are valid date strings in the format MM/DD/YYYY HH:MM (AM/PM). * Post-Condition: The configuration object now knows the event name, description, start time & date, and end time & date. * Description: Set the event information for this Configuration object. | |
| **Protocol: set\_directories(root\_dir, red\_dir, blue\_dir, white\_dir)**   * Pre-Condition: All parameters are folder paths\ strings that describe the root dir, red directory, blue directory, and white directory respectively where log files are stored by the user. * Post-Condition: The configuration object now knows the directories where the log files are stored. * Description: Set the root, red, blue, and white folder directories for this Configuration object. | |
| **Protocol: add\_vector(name, description)**   * Pre-Condition: name and description are valid strings * Post-Condition: The configuration now knows this vector, given its name and description * Description: Add a new vector to this Configuration’s list of vectors. | |
| **Protocol: add\_icon(name, source)**   * Pre-Condition: name is a valid string, and source is a valid file path to an image file stored on the user’s local machine. * Post-Condition: The configuration now knows this icon, given its name and source. * Description: Add a new icon to this Configuration’s list of vectors. | |

## Class Description of Icon

|  |  |
| --- | --- |
| **Class Name**: Icon | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities**   * Load the icon image from the known icon source file path. | |
| **Contract 4:** Provide details of the Icon | |
| **Responsibilities** | **Collaborations** |
| 1. Know the icon’s name 2. Know the icon’s image source file path | N/A |
| **Protocol: set\_name(name)**   * Pre-Condition:Name is a valid string with length greater than 0 * Post-Condition**:** This icon object’s name attribute has been set to the given name * Description:Set the name of this Icon object | |
| **Protocol: set\_source(source)**   * Pre-Condition:source is a string containing the file path of an image file on the user’s local machine. * Post-Condition:The icon now knows the file path of the user’s desired icon * Description:Set the source of this icon image to the given source file path. | |
| **Protocol: get\_name(source)**   * Pre-Condition:true * Post-Condition:the Icon name has been returned * Description:Set the source of this icon image to the given source file path. | |

# Detailed Description of Ingestion Subsystem

## Subsystem Description

**Subsystem name:** Ingestion Subsystem

**Purpose of subsystem:** Responsible for transcribing, cleansing, and validating log files, and sending log files for Splunk ingestion.

**Classes contained in subsystem:** Log Entry, Cleansing, Validation, Enforcement Action Report, Splunk, OCR Feeder, Parlatype

## Class Description of Log Entry

|  |  |
| --- | --- |
| **Class Name**: Log Entry | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:** N/A | |
| **Contract 5:** Store log entry details | |
| **Responsibilities** | **Collaborations** |
| 1. Know log entry ID. 2. Know time of occurrence 3. Know description 4. Know log file path 5. Know log creator 6. Know the event type 7. Know artifact |  |
| **Protocol: set\_details(entryID, occurrenceTime, description, path, creator, eventType, artifact)**   * Pre-Condition: entryID, description, path, creator, eventType, and artifact are valid strings. occurrenceTime is a valid date string in the format MM/DD/YYYY HH:MM (AM/PM). * Post-Condition: The log object now knows the entryID, occurrenceTime, description, path, creator, eventType, and artifact. * Description: Set the log entry information for this log object. | |

## Class Description of Cleansing

|  |  |
| --- | --- |
| **Class Name**: Cleansing | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Know cleansing status. * Know cleansing errors. * Know the cleansing script. | |
| **Contract 6:** Cleanse log file according to script. | |
| **Responsibilities** | **Collaborations** |
| 1. Run cleansing script.   27. Send log files to Audio Transcription Tool.  28 Send log files to Tesseract Transcription Tool. | 26 -> Configurations (1)  27 -> ParlatypeInterface(11)  28 -> TesseractInterface(10) |
| **Protocol: cleanse(source)**   * Pre-Condition:The source is a string containing the file path of a file on the user’s machine. * Post-Condition:The cleansing class will now know the cleansing status for the source. * Description:The source will either get cleansed and sent for validation or be redirected to an audio or an ocr transcription tool. | |

## Class Description of Validation

|  |  |
| --- | --- |
| **Class Name**: Validation | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Know Validation errors. * Know Validation status. | |
| **Contract 7:** Approve/Deny cleansed file | |
| **Responsibilities** | **Collaborations** |
| 1. Validate that the cleansed log file content falls within the start date/time. 2. Validate that the cleansed log file content falls within the end date/time. 3. Validate that the cleansed log file content does not contain any empty rows. 4. Begin SPLUNK ingestion by sending the validated log files. | 27 -> Cleansing (6)  28 -> Cleansing (6)  30 -> Splunk (9) |
| **Protocol: validate\_rows(source)**   * Pre-Condition:The source is a string containing the file path of a file on the user’s machine. * Post-Condition:The validation class will now know the validation status for the source. It will know the error for that source. * Description:Checks to make sure the source does not contain any empty rows. | |
| **Protocol: validate\_time(source, startTime, endTime)**   * Pre-Condition: source is a valid string. startTime and endTime are valid date strings in the format MM/DD/YYYY HH:MM (AM/PM). * Post-Condition: The validation class will now know the validation status for the source. It will know the error for that source. * Description: Checks to make sure that the source falls with the startTime and endTime range. | |

## Class Description of Enforcement Action Report

|  |  |
| --- | --- |
| **Class Name**: Enforcement Action Report | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * References the status of log files and their error descriptions. | |
| **Contract 8:** Provide log files statuses and their error descriptions. | |
| **Responsibilities** | **Collaborations** |
| 1. Display log files that could not be ingested. 2. Display error descriptions for log files. | 31 -> Cleansing (6)  32 -> Validation (7) |
| **Protocol: get\_log\_files(source)**   * Pre-Condition:true * Post-Condition:The log files have been retrieved and displayed. * Description:Set the source of this icon image to the given source file path. | |
| **Protocol: get\_cleansing\_status(source)**   * Pre-Condition:The source is a string containing the file path of a file on the user’s machine. * Post-Condition:The Enforcement Action Report will now know the cleansing status for the source. * Description:This method gets the status of the source from the Cleansing class and displays it. | |
| **Protocol: get\_validation\_status(source)**   * Pre-Condition:The source is a string containing the file path of a file on the user’s machine. * Post-Condition:The Enforcement Action Report will now know the cleansing status for the source. * Description:This method gets the status of the source from the Validation class and displays it. | |
| **Protocol: get\_ingestion\_status(source)**   * Pre-Condition:The source is a string containing the file path of a file on the user’s machine. * Post-Condition:The Enforcement Action Report will now know the cleansing status for the source. * Description:This method gets the status of the source from the SplunkInterface class and displays it. | |
| **Protocol: get\_error\_description(source, log\_error)**   * Pre-Condition:The log\_error is a string that represents a valid error description for the selected file and get\_logFiles(source) has previously been called on this Enforcement Action Report object. * Post-Condition: Returns the log\_error for the given source. * Description:This method will obtain, store, and display the error descriptions for the log files. | |

## Class Description of SplunkInterface

|  |  |
| --- | --- |
| **Class Name**: SplunkInterface | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Connect to Splunk * Send log files to Splunk for ingestion * Transform log files to log entries. | |
| **Contract 9:** Transform log files into log entries. | |
| **Responsibilities** | **Collaborations** |
| 1. Return log entries. 2. Receive log files 3. Provide ingestion status | 33 -> LogEntry (5)  35 -> Enforcement Action Report 8 |
| **Protocol:** start\_ingestion()   * Pre-Condition:This SplunkInterface object has been initialized with lists of file path strings. Root, Red, Blue, and White lists respectively. Splunk must be running on the current user’s machine. * Post-Condition:Log Files have been uploaded to Splunk. * Description:This method will start ingestion by sending log files to Splunk so that Splunk can transform them into log entries. | |
| **Protocol:** get\_log\_entries()   * Pre-Condition: start\_ingestion() has previously been called on this SplunkInterface object. * Post-Condition: A list of log entries that were obtained from uploaded log files has been returned * Description: Get a list of log entries that have been uploaded to the splunk index. | |

## Class Description of Tesseract Interface

|  |  |
| --- | --- |
| **Class Name**: TesseractInterface | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Send files to Tesseract OCR tool. * Transcribe files into text files. | |
| **Contract 10:** Get text files from pdf files and images. | |
| **Responsibilities** | **Collaborations** |
| 1. Receive files that need OCR transcription. 2. Return the transcribed file contents. | 35 -> Cleansing(6) |
| **Protocol: get\_files(source)**   * Pre-Condition:true * Post-Condition:The files have been retrieved from the Cleansing class. * Description:This method gets the files from the Cleansing class in order to start the transcription. | |
| **Protocol: transcribe(source)**   * Pre-Condition:The TesseractInterface has been initialized. * Post-Condition:The files have been sent to Tesseract. * Description:This method will begin transcription by sending the received files to Tesseract. | |
| **Protocol:** **get\_transcribedFiles(source)**   * Pre-Condition: transcribe(source) has previously been called on this TesseractInterface object. * Post-Condition: transcribed files have been obtained from Tesseract. * Description: This method will get files that have been transcribed from Tesseract and send them back to the Cleansing class. | |

## Class Description of Parlatype Interface

|  |  |
| --- | --- |
| **Class Name**: ParlatypeInterface | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:** N/A | |
| **Contract 11:** Get text files from audio files. | |
| **Responsibilities** | **Collaborations** |
| 1. Receive files. 2. Transcribe audio files into text files. 3. Return the files back to the Cleansing class. | 37 -> Cleansing(6) |
| **Protocol: get\_files(source)**   * Pre-Condition:true * Post-Condition:The files have been retrieved from the Cleansing class. * Description:This method gets the files from the Cleansing class in order to start the transcription. | |
| **Protocol: transcribe(source)**   * Pre-Condition:The ParlatypeInterface has been initialized. * Post-Condition:The files have been sent to Parlatype. * Description:This method will begin transcription by sending the received files to Parlatype. | |
| **Protocol:** **get\_transcribed\_files(source)**   * Pre-Condition: transcribe(source) has previously been called on this ParlatypeInterface object. * Post-Condition: transcribed files have been obtained from Parlatype. * Description: This method will get files that have been transcribed from Parlatype and send them back to the Cleansing class. | |

# Detailed Description of Graph Subsystem

## Component Description

**Subsystem name:** Graph Subsystem

**Purpose of subsystem:** Responsible for creating a graphical representation of a vector and its associations.

**Classes contained in subsystem:** Graph, Node, Relationship, Commit, GraphViz

## Class Description of Graph

|  |  |
| --- | --- |
| **Class Name**: Graph | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Know graph name. * Know vector * Commit. | |
| **Contract 12: Know graph detail information** | |
| **Responsibilities** | **Collaborations** |
| 1. Know list of nodes 2. Know list of relationship | 41 -> Node (13)  42 -> Relationship (14) |
| **Protocol: add\_node(node)**   * Pre-Condition: node must be a valid Node object. * Post-Condition: This graph is now aware of this new node by appending to the Graph’s list of known nodes. * Description:Add the given node to this graph. | |
| **Protocol: add\_relationship(relationship)**   * Pre-Condition: relationship must be a valid Relationship object. * Post-Condition: This graph is now aware of this new relationship by appending to the Graph’s list of known relationships. * Description: Add the given relationship to this graph. | |
| **Protocol: remove\_node(node\_id)**   * Pre-Condition: node\_id must be a node that has already been added to the graph. * Post-Condition: The node with the given node\_id has been removed from this graph * Description: Remove the node with the given node\_id from this graph | |
| **Protocol: remove \_relationship(relationship)**   * Pre-Condition: relationship must be a valid Relationship object. * Post-Condition: The relation will now be removed from both the Graph View and Relationship Table. * Description: Remove the given relationship to this graph. | |

## Class Description of Node

|  |  |
| --- | --- |
| **Class Name**: Node | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Know node id. | |
| **Contract 13:** Provide Node details | |
| **Responsibilities** | **Collaborations** |
| 1. Know node name. 2. Know the log entry it will reference 3. Know icon id. | 44 -> Log Entry (5)  45 -> Icon(4) |
| **Protocol:** set\_details(id, name, log\_entry, icon = None)   * Pre-Condition:id is a valid int value, name is a string value, log\_entry is a valid LogEntry object, and icon is a valid Icon object. Icon is optional, as a node may or may not use a special icon. * Post-Condition:This Node now knows and has set the given id, name, and log\_entry as its details. * Description:Set the attributes of this Node to the given details. | |
| **Protocol:** get\_name()   * Pre-Condition: true * Post-Condition: the name of this Node is returned * Description: Get the name of this icon object. | |
| **Protocol:** get\_log\_entry()   * Pre-Condition: true * Post-Condition: the log\_entry object of this Node is returned * Description: Get the log\_entry of this icon object. | |
| **Protocol:** get\_icon()   * Pre-Condition: true * Post-Condition: the icon object of this Node is returned * Description: Get the icon of this icon object. | |

## Class Description of Relationship

|  |  |
| --- | --- |
| **Class Name**: Relationships | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Know relationship identity number. * Know label for the relationship | |
| **Contract 14:** Store relationship details between 2 nodes. | |
| **Responsibilities** | **Collaborations** |
| 1. Know parent node 2. Know child node 3. Provide associations. |  |
| **Protocol: set\_child\_parent(parent\_node=None, child\_node=None)**   * Pre-Condition:parent\_node and child\_node are both optional values. If a parent\_node is provided as a valid Node object, the parent node will be changed to this new node. If a parent\_node is not provided, no change to the parent node will be made. If a child\_node is provided as a valid Node object, the child node will be changed to this new node. If a child\_node is not provided, no change to the child node will be made. * Post-Condition:This Relationship’s parent\_node and child\_node are set to the new values, or unchanged if their respective values are not provided. * Description:Change the child and parent nodes of this node relationship | |
| **Protocol:** **set\_label(label)**   * Pre-Condition: label is a valid string * Post-Condition: This relationship now knows a new label * Description: Set the relationship’s label to the given label | |
| **Protocol: get\_nodes()**   * Pre-condition: true. * Post-condition: Returns a list of size 2. The 0 index contains the parent node, and the 1 index contains the child node. * Description: Method that returns a list of size 2, containing the parent and child nodes. | |

## 

# Detailed Description of User Interface Subsystem

## Class description of Main Window

|  |  |
| --- | --- |
| **Class Name**: MainWindow | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Accept User Interactions | |
| **Contract 15: Launch windows** | |
| **Responsibilities** | **Collaborations** |
| 1. Launch Configurations Window 2. Launch Enforcement Action Report Window 3. Launch Search/Filter Window 4. Launch Graph Window | 50 -> ConfigurationUI(16)  51 -> EARWindow (20)  52 -> SearchFilterWindow(17)  52 -> SearchFilterWindow(18)  53 -> GraphUI (19) |
| **Protocol: open\_configurationsWindow()**   * Pre-Condition: true * Post-Condition: The Event Configuration Window will now be displayed. * Description: Launches the Event Configuration Window | |
| **Protocol: open\_EAR()**   * Pre-Condition: true * Post-Condition: The Action Report Window will now be displayed. * Description: Launches the Action Report Window. | |
| **Protocol: open\_SearchFilterWindow()**   * Pre-Condition: true * Post-Condition: The Search and Filter Window will now be displayed. * Description: Launches the Search and Filter Window, | |
| **Protocol: open\_graphWindow()**   * Pre-Condition: true * Post-Condition: The Graph Window will now be displayed * Description: Launches the Graph Window. | |

* 1. **Class description of Configuration Window**

|  |  |
| --- | --- |
| **Class Name**: ConfigurationsWindow | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Accept User Interactions | |
| **Contract 16: Launch Configuration Tabs** | |
| **Responsibilities** | **Collaborations** |
| 1. Sets Lead IP Address 2. Sets Event Name 3. Sets Event Description 4. Sets Root Directory 5. Sets Red Team Folder 6. Sets Blue Team Folder 7. Sets White Team Folder 8. Sets Vector Name 9. Sets Vector Description 10. Sets Icon Name 11. Sets Icon Source | 54 -> Configuration(3)  55 -> Configuration(3)  56 -> Configuration(3)  57 -> Configuration(3)  58 -> Configuration(3)  59 -> Configuration(3)  60 -> Configuration(3)  61 -> Configuration(3)  62 -> Configuration(3)  63 -> Icon(4)  64 -> Icon(4) |
| **Protocol: get\_teamConfiguration()**   * Pre-Condition: true * Post-Condition: data entered will be stored in the local database * Description: The ip address will be entered by the analyst and it will be true/false if the analyst is the lead. | |
| **Protocol: get\_eventConfiguration()**   * Pre-Condition: true * Post-Condition: data entered will be stored in the local database * Description: The event name, event description, event start date and event end date will be entered by the analyst. | |
| **Protocol: get\_directoryConfiguration()**   * Pre-Condition: true * Post-Condition: data entered will be stored in the local database * Description: The root directory, red team directory, blue team directory and the white team directory will be selected by the analyst. | |
| **Protocol: get\_vectorConfiguration()**   * Pre-Condition: true * Post-Condition: data entered will be stored in the local database * Description: The vector name and the vector description will be entered by the analyst. | |
| **Protocol: get\_iconConfiguration()**   * Pre-Condition: true * Post-Condition: data entered will be stored in the local database * Description: The icon name will be entered by the analyst. The source folder for the icon will be selected by the analyst. | |

* 1. **Class description of Search & Filter Window**

|  |  |
| --- | --- |
| **Class Name**: SearchFilterWindow | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Accept user interactions * Display Splunk log entries * Display Vector’s associated log entries * Sort log entries by log id * Sort log entries by time * Sort log entries by description * Sort log entries by loge entry reference * Sort log entries by creator * Sort log entries by action type * Sort log entries by artifact | |
| **Contract 17: Associate log entry** | |
| **Responsibilities** | **Collaborations** |
| 1. Associate log entry to vector | 65 -> LogEntry (5)  65 -> Vector (1) |
| **Protocol: associate(vector\_name)**   * Pre-Condition: Vector name is a string that matches the name of a vector from configurations. * Post-Condition: The vector with the given vector name now has the selected log entries associated with it. * Description: Associate the window’s user selected log entries to the vector whose name matches the given name. | |
| **Contract 18: Filter by criteria** | |
| **Responsibilities** | **Collaborations** |
| 1. Filter by keyword 2. Filter by creator 3. Filter by event 4. Filter by time range |  |
| **Protocol: filter()**   * Pre-Condition: true * Post-Condition: The search filter window now displays log entries that match the criteria entered into the window. * Description: Display the log entries that match the given criteria in the window. | |

* 1. **Class description of GraphWindow**

|  |  |
| --- | --- |
| **Class Name**: GraphUI | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Display vector name. * Display vector description * Display graph in vertical position. * Display graph in horizontal position. * Display the graph in the selected time interval units. * Display relationships for each vector. * Display nodes for each vector. * Display node id. * Display node name. * Display node time of occurrence. * Display node long description. * Display log entry reference. * Display log creator. * Display node event type. * Display all node properties. * Hide node id. * Hide node name. * Hide node time of occurrence. * Hide node long description. * Hide log entry reference. * Hide log creator. * Hide node event type. * Hide node artifact. * Hide all node properties. * Display relationship ID on the relationship table. * Display relationship label on the relationship table. * Display parent node on the relationship table. * Display child node on the relationship table. | |
| **Contract 19: Manipulate Graph** | |
| **Responsibilities** | **Collaborations** |
| 1. Add node to graph 2. Remove node from graph 3. Add relationship to graph 4. Remove relationship from graphExport the graph 5. Open and load an existing graph 6. Create a new graph | 70 -> Graph(12)  71 -> Graph(12)  72 -> Graph(12)  73 -> Graph(12)  74 -> Graph(12)  75 -> Graph(12) |
| **Protocol: add\_node()**   * Pre-Condition: The node name is provided in the input box in the add node dialog box. The other fields are optional. * Post-Condition: The new node is created and the graph displays it. The tabular view gets populated with the new node and all its properties. * Description: Create a new node with the information given by the user on the graph window. | |
| **Protocol: rem\_node()**   * Pre-Condition: A node exists on the graph to be able to choose to remove. The user has selected the node they wish to remove. * Post-Condition: The node is deleted and removed from the graph and tabular view. * Description: Remove the chosen node from the graph window. | |
| **Protocol: add\_edge()**   * Pre-Condition: At least two nodes exist to be able to add a relationship between them. The user has dragged their mouse in a line between the two chosen nodes. * Post-Condition: The new relationship is created and added to both the graph and relationship table view. * Description: Add a new relationship between two chosen nodes on the graph window. | |
| **Protocol: rem\_edge()**   * Pre-Condition: An edge exists on the graph to be able to choose to remove. The user has selected the relationship they wish to delete. * Post-Condition: The relationship is deleted and removed from the graph and relationship table view. * Description: Remove the chosen relationship from the graph window. | |
| **Protocol: save()**   * Pre-Condition: true * Post-Condition: The current graph with all its nodes and relationship properties is exported and formatted into a CSV file on your machine. * Description: Export and save the graph with all node and relationship properties as a CSV file on your machine. | |
| **Protocol: load()**   * Pre-Condition: A CSV file containing and following the graph format exists and the user has chosen the file. * Post-Condition: The chosen CSV file is loaded and displays the nodes and relationships on both the graph and table views. * Description: Open an existing CSV file and display all the graph window properties with the information found in the file. | |
| **Protocol: new()**   * Pre-Condition: true * Post-Condition: The graph and both table views are cleared of all previously existing information. * Description: Clear all the graph window contents to start a new graph from scratch. | |

* 1. **Class description of EARWindow**

|  |  |
| --- | --- |
| **Class Name**: EARWindow | |
| **Superclass**: N/A | |
| **Subclasses**: N/A | |
| **Private Responsibilities:**   * Display log files with errors * Display log file error description * Cancel Ingestion * Validate log file and continue ingestion. | |
| **Contract 20:** Manipulate Enforcement Action Report | |
| **Responsibilities** | **Collaborations** |
| 1. Display log files that could not be ingested. 2. Display error descriptions for log files. | 31 -> Cleansing (6)  32 -> Validation (7) |
| **Protocol: get\_log\_files(source)**   * Pre-Condition:true * Post-Condition:The log files have been retrieved and displayed. * Description:Set the source of this icon image to the given source file path. | |
| **Protocol: get\_cleansing\_status(source)**   * Pre-Condition:The source is a string containing the file path of a file on the user’s machine. * Post-Condition:The Enforcement Action Report will now know the cleansing status for the source. * Description:This method gets the status of the source from the Cleansing class and displays it. | |
| **Protocol: get\_validation\_status(source)**   * Pre-Condition:The source is a string containing the file path of a file on the user’s machine. * Post-Condition:The Enforcement Action Report will now know the cleansing status for the source. * Description:This method gets the status of the source from the Validation class and displays it. | |
| **Protocol: cancel()**   * Pre-Condition: true * Post-Condition: The selected log file will cancel the ingestion process * Description: This method will cancel the ingestion for the selected log file. | |
| **Protocol: validate()**   * Pre-Condition: true * Post-Condition: The selected log file will be ingested. * Description: This method will validate the log file and allow it to be ingested.. | |

# Database

## Database Schema

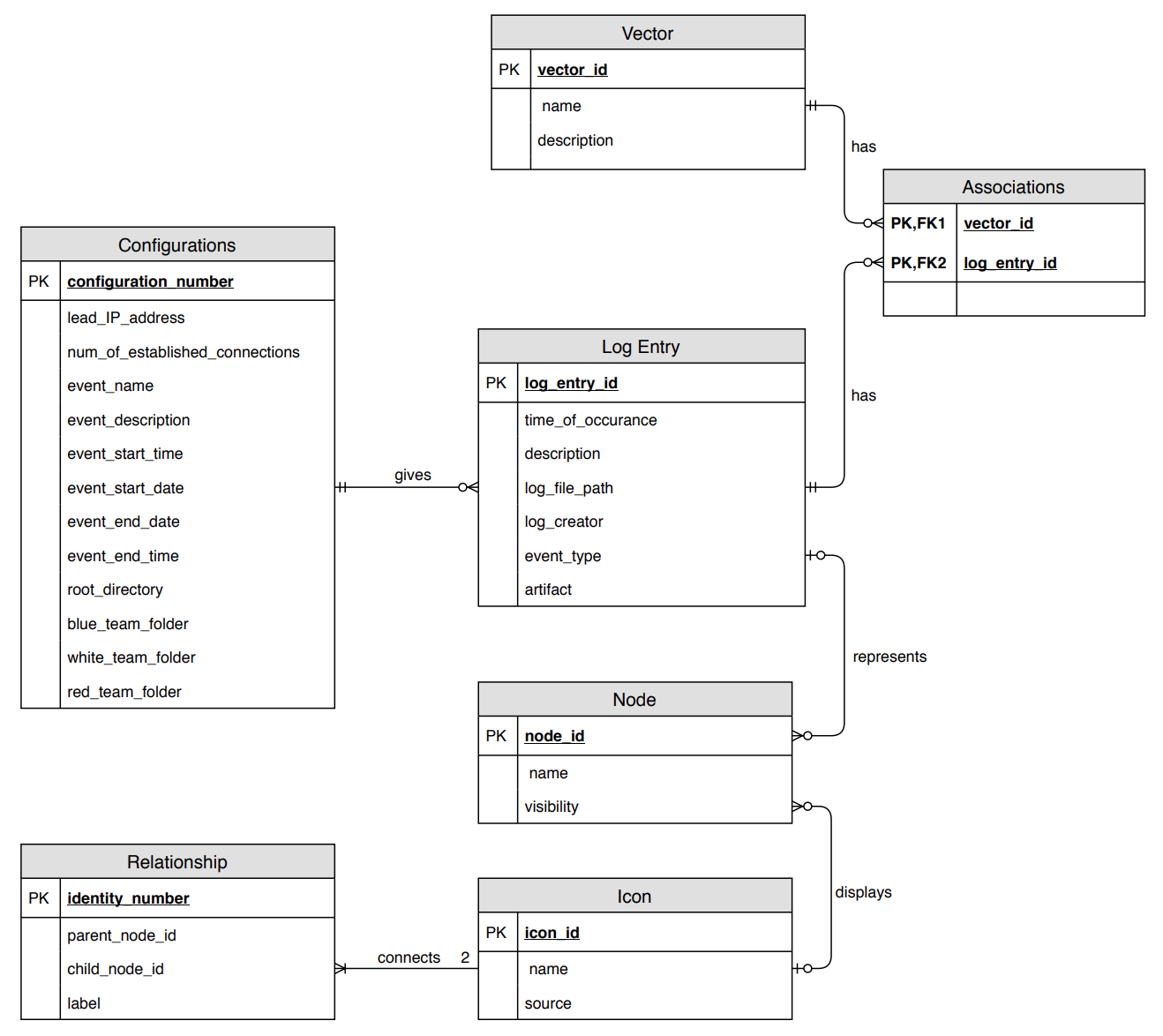


Figure 2: Entity-Relationship Diagram with Crow’s Foot notation

Configurations

The configuration table is an abstraction for all the different types of configurations. Each event will have its own corresponding configuration data which will be stored in this table and retrieved through a unique configuration number.

Log Entry

The Log Entry table will only store log entries and their corresponding attributes if they are associated with any existing vector. These log entries will be retrieved from splunk and stored the moment they are associated to any existing vector.

Vector

The vector table will store different vectors created within the system by any user. Each vector has associations to different log entries.

Associations

The associations table will store the associations between log entries and vectors by storing the primary keys of both the log entry primary key and vector primary key.

Node

The node table will store all existing nodes. They are needed to represent a log entry in a graphical view.

Icon

The icon table will store all uploaded icons that will be used to visually display different images for each node.

Relationship

The relationship table will store all relationships that exist between different nodes.

**%**