Data Visualization System (DVS)

Design Document

Version 1.5

December 7, 2020

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

|  |  |
| --- | --- |
| Initial Release: | Version 1.0 |
| Current Release: | Version 1.5 |
| Indicator of Last Page in Document: | $ |
| Date of Last Review: | Dec 8, 2020 |
| Date of Next Review: | Dec 9, 2020 |
| Target Date for Next Update: | Dec 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. Salamah

Software Team Members:

Stephanie Medina

Dima AbdelJaber

Briana Sanchez

Rocio Cardona

Luisana Clarke

Bianca Alvarez

Change Summary

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 12/1/20 | Dima AbdelJaber | Creation of Document |
| 1.1 | 12/7/20 | Rocio Cardona | Subsystems |
| 1.2 | 12/8/20 | Luisana Clarke | Wireshark modified files section |
| 1.3 | 12/8/20 | Stephanie Medina | Threading Description |
| 1.4 | 12/8/20 | Briana Sanchez | Sync description & diagram |
| 1.5 | 12/8/20 | All | Review of final document |

Table of Contents

[Document Control ii](#_Toc58325374)

[Approval ii](#_Toc58325375)

[Document Change Control ii](#_Toc58325376)

[Distribution List ii](#_Toc58325377)

[Change Summary ii](#_Toc58325378)

[1. Introduction 4](#_Toc58325380)

[1.1. Purpose and Intended Audience 4](#_Toc58325381)

[1.2. Scope of Product 4](#_Toc58325382)

[1.3. Definitions, Acronyms, and Abbreviations 4](#_Toc58325383)

[1.3.1. Definitions 4](#_Toc58325384)

[1.3.2. Acronyms 4](#_Toc58325385)

[1.4. Overview 5](#_Toc58325386)

[2. Decomposition Description 6](#_Toc58325387)

[2.1. System Collaboration Diagram 6](#_Toc58325388)

[2.2. Subsystem and Component Descriptions 7](#_Toc58325389)

[2.3. Dependencies 7](#_Toc58325390)

[3. Detailed Description of the Packet View Subsystem 8](#_Toc58325391)

[3.1. Packet View Subsystem Description 8](#_Toc58325392)

[4. Detailed Description of the Timeline View Subsystem 10](#_Toc58325393)

[4.1. Timeline View Subsystem Description 10](#_Toc58325394)

[5. Detailed Description of DVS Main 15](#_Toc58325395)

[5.1. Main Class Description 15](#_Toc58325396)

# Introduction

## Purpose and Intended Audience

Team 1 will develop the Data Visualization System (DVS), a software tool that will provide the user with an integrated view of the data generated by [the Evaluator Centric and Extensible Logger daemon (ECELd)](https://github.com/ARL-UTEP-OC/eceld-netsys) and the ability to tag and modify the data associated with a capture taken by ECELd. The intended audience of this document is the guidance team, Dr. Salamah, the client, Dr. Acosta and the members of Team 1.

The purpose of creating the Software Design Document (SDD) is to provide a description of the Data Visualization System (DVS) that will allow for a comprehensive system design. This document will allow to process with a software development and implementation with an understanding of what is to be build and how it is expected to be built. The SDD will identify all the subsystems, classes and components pertaining to DVS.

## Scope of Product

Data is a critical asset in cybersecurity; it helps researchers and practitioners develop novel technologies based on real past incidents. However, this asset is lacking, and even more, tools that specifically focus on the analysis of data are far and few between. The Evaluator Centric and Extensible Logger daemon (ECELd) is a tool that attempts to reduce this gap. The tool allows users to collect data for the purpose of analysis. Collected data include network data, keystrokes, system calls, and screenshots, among others. Additionally, the ECELd includes a Wireshark component which allows users to annotate the network data using the popular packet sniffer application. Both the ECELd tool and the ECELd-wireshark component are integrated into the ECELd-netsys.

## Definitions, Acronyms, and Abbreviations

The definitions in this section are given in the context of the product being developed. The intention is to assist the user in their understanding of the document.

### Definitions

Table 1: Definition of terms used in the report

|  |  |
| --- | --- |
| Dataline | Horizontal display of the data associated with an ECELd capture |
| Tag | Customized piece of information that can be added to an entry of datalines |

### Acronyms

This section lists the acronyms used in this document and their associated definitions.

Table 2: Acronyms

|  |  |
| --- | --- |
| SDD | Software Design Document |
| UTEP | The University of Texas at El Paso |
| DVS | Behavior Extraction and Analysis Tool |
| GUI | Graphical User Interface |
| ECELd | Evaluator Centric and Extensible Logger daemon |
| PID | Process Identifier |

## Overview

The Software Design Document is comprised of the following sections: Decomposition Description, Detailed Description of Component and Database.

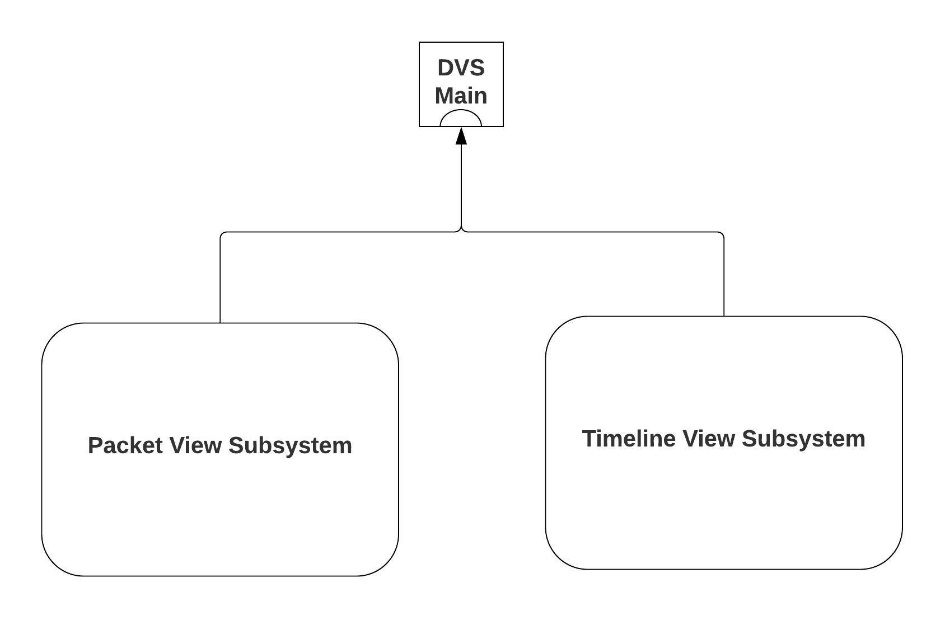
The Decomposition Description provides a description of how the component descriptions can be used by designers and maintainers. It will identify major design entities, for purposes such as determining which entity is responsible for specific functions and tracing requirements to design entities.

The Detailed Description of Component will provide a section with a detailed design description of each of the components listed in section 2.2, Subsystem and Component Description.

# Decomposition Description

This section will describe the main subsystems of the DVS. These subsystems enclose groups of classes that collaborate with one another in order to support the functionality of the main system.

## System Collaboration Diagram



*Figure 1: Subsystems Collaboration Diagram*

Figure 1 depicts the complete subsystems collaboration diagram with the major subsystems. The major subsystems identified are: Packet View Subsystem and the Timeline View Subsystem.

*Diagram

Description automatically generated*

*Figure 2:* *Subsystems Detailed Diagram*

Figure 2 depicts Subsystems Detailed Diagram with all the subsystems present and the packages that each subsystem depends on. The subsystems identified are: Packet View Subsystem and the Timeline View Subsystem.

## Subsystem and Component Descriptions

The subsystems identified in section 2.1 and shown in Figures 1 and 2, are described as follows:

**Subsystem Name:** Packet View Subsystem

**Subsystem Purpose:** The purpose of this subsystem is to analyze the packet data associated with a capture taken by ECELd through eceld-wireshark.

Detailed information about this subsystem is discussed in section 3

**Subsystem Name:** Timeline View Subsystem

**Subsystem Purpose:** The purpose of this subsystem is to provide a GUI for the user to analyze, modify and tag the data associated with a capture taken by ECELd.

Detailed information about this subsystem is discussed in section 4.

## Dependencies

The coupling of the different subsystems is depicted in Figure 1 and it shows the complete system with the main components.

In order for DVS to perform properly, additional software requirements must be fulfilled. Regarding third party software, the system is coupled to the specific versions of external software required for its execution. The system requirements for the above-mentioned software are as follows:

Third party software:

[eceld-wireshark](https://github.com/ARL-UTEP-OC/eceld-wireshark)

Languages:

Python 3.x

Python Libraries:

wheel

PyQt5==5.15.1

pandas

plotly

Flask

cachetools

dash

PyQtWebEngine

flask\_caching

dash\_daq

dash-bootstrap-components

numpy==1.19.3

tkcalendar

Operating System:

Linux

Windows

# Detailed Description of the Packet View Subsystem

This section will describe in detail the Packet View Subsystem.

## Packet View Subsystem Description

**Subsystem Name:** Packet View Subsystem

**Subsystem Purpose:** The purpose of this system is to analyze the packet data associated with a capture taken by ECELd through [eceld-wireshark.](https://github.com/ARL-UTEP-OC/eceld-wireshark)

**Description:** This subsystem is composed of an existing third-party software used in DVS. This subsystem is composed of the existing system as well as the modifications to specific files done by Team 1 members in order to complete the required functionality of DVS. This subsystem performs the standard functionality of Wireshark as well as syncing of packets to the different datalines, importing the DVS color profile to match those of the Timeline View datalines, as well as displaying the Suricata Alerts collected during an ECELd capture. For simplicity, we are only depicting the files Team 1 modified, which can be found in the Wireshark Modified Files folder in the [DVS GitHub repository](https://github.com/smedina7/DVS). Note that, although these files were modified, they were kept in the original location. Below is a complete list of these files.

**packet\_list.cpp**

This file was modified to achieve the sync functionality between the Wireshark View and the Timeline View. It leverages the internal timer and reads/writes to the current selected timestamp file, to which the Timeline View data lines also have access. It also writes the packet number selected (onClick) to a file within the DVS System for the DVS to sync with what is selected in the Wireshark View. Please refer to the diagram in sync\_helper.py for a detailed description of how the synchronization works.

**main\_window.ui**

This file was modified to display a more appropriate text for when the user hovers over the colorization button. The hover over text now reads “Draw packets using DVS coloring rules”

**main\_window\_slots.cpp**

This file was modified to achieve a more dynamic colorization of Wireshark packets using the color profiles generated by DVS. It now includes the functionality to import and update the DVS color profile with the push of the colorization button and a CTRL-R (reload) action without creating an instance of the graphical dialog box and/or manually searching and selecting the color filters.

**coloring\_rules\_dialog.cpp**

This file, along with its header definition (**coloring\_rules\_dialog.h)** was modified to achieve the actions triggered by enabling the colorization button. They now include functions to read from files generated by DVS and properly apply the rules and filters.

**CMakeOptions.txt**

This file was modified to ensure the successful build of the Qt components. It now includes the option “option(BUILD\_qtshark ”Build qtshark” )ON”

**proto\_tree.h**

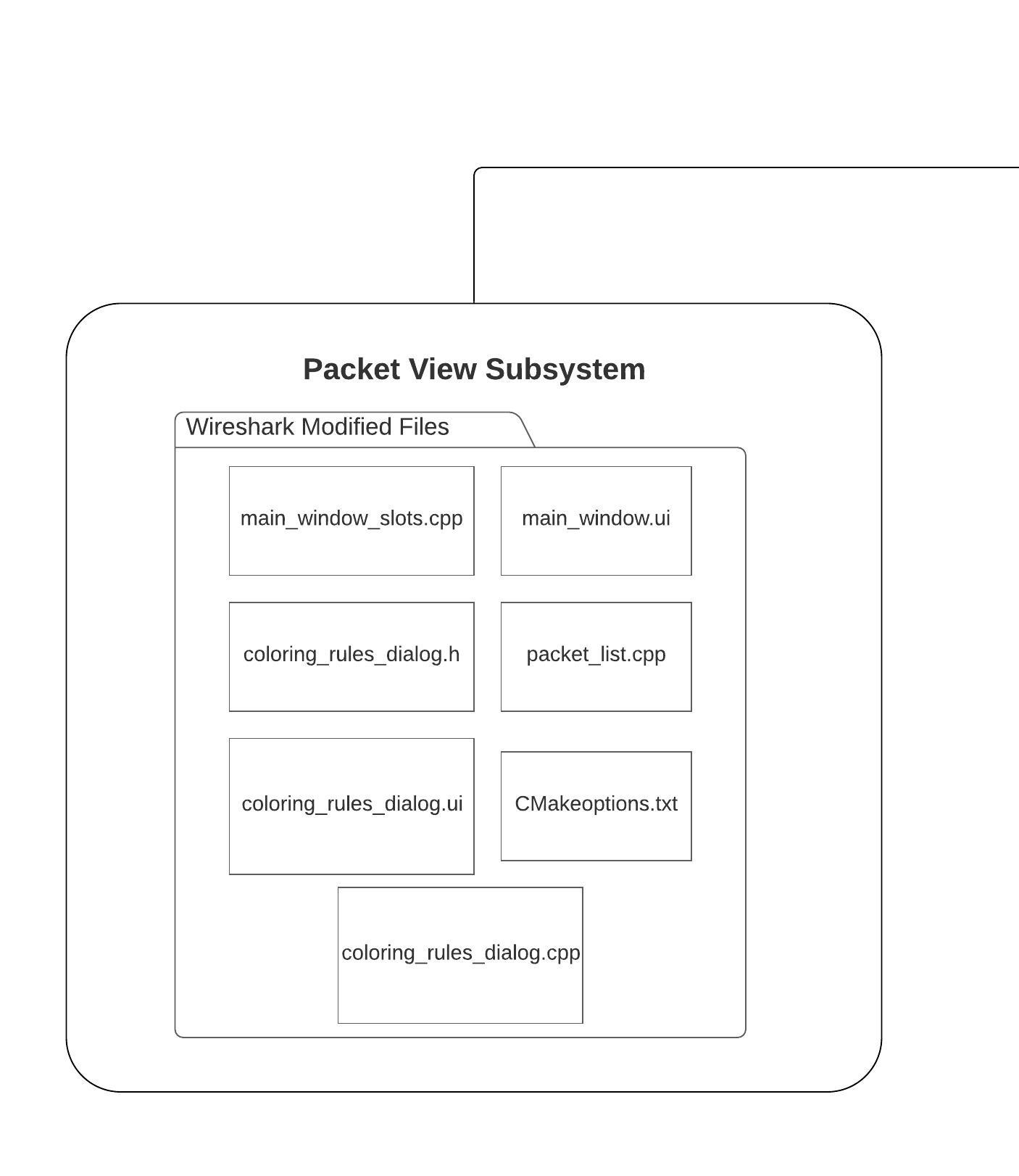
This file was modified to add the header definition for the function described below.

**proto\_tree.cpp**

This file was modified in order to display Suricata Alerts that are captured and parsed with the ECELd tool. In order to implement this functionality, an additional function was added to the class: getSuricataAlerts( ). This function is very similar to the existing functions for the keypresses and system calls logs.

**Detailed Graph:**

For a detailed graph of this subsystem please see Figure 3.



*Figure 3: Packet View Subsystem Diagram*

# Detailed Description of the Timeline View Subsystem

This section will describe in detail the Timeline View Subsystem.

## Timeline View Subsystem Description

**Subsystem Name:** Timeline View Subsystem

**Subsystem Purpose:** The purpose of this subsystem is to provide a GUI for the user to analyze, modify and tag the data associated with a capture taken by ECELd.

**Description:** This subsystem is composed of multiple packages that together give DVS its functionality. The Timeline View subsystem provides a visualization of the parsed ECELd collected data using a horizontal layout. In this subsystem, each specific set of data is considered as a dataline, (e.g. mouseclicks dataline will contain different information than what a keypresses dataline contains, which will result in specific features for each). Moreover, with different datasets, there must be a way to be able to distinguish between each of them, so each dataline will have an assignable color in order to make it easier to tell them apart. This color assignment shall be displayed in both the timeline view and the packet view. The user is able to edit, tag and modify the information in each dataline. Finally, the most important feature of this system will be synchronization. The synchronization feature shall be enabled and disabled depending on the user’s preference. The main purpose for this feature is to allow the user to use both timeline view and packet view at the same time and be able to see the data that is correlated to where the user is currently analyzing.

Below is a description of each package and the classes and module they contain.

**Packet View:**

This package contains the necessary classes and modules required for DVS to control eceld-wireshark (Packet View Subsystem). Below is a list of the classes and modules that belong to the Packet View package.

**close\_ws.py**

This module is responsible to close the Packet View Subsystem (eceld-wireshark). It does so by finding the eceld-wireshark PID’s and killing it.

**Manager.py**

This class is responsible for the finding and loading the necessary files that are needed to run DVS and eceld-wireshark.

**WiresharkColorFilters.py**

This class is responsible for generated the color profile that is imported by eceld-wireshark in order for the colors associated to each packet to match the colors for each dataset/dataline.

**WiresharkRunner.py**

This class is responsible for opening eceld-wireshark loading the PCAP file associated to the project as well as loading custom dissectors from the GeneratedDissectors folder.

**Widgets:**

This package contains the necessary classes and modules to create each dataline widget, as well as the main window that contains all datalines. It also contains the classes and modules required for the synchronization functionality. Below is a list of the classes and modules that belong to the Widgets package.

**commentsParser.py**

This class is responsible for extracting the packet comments from each packet in eceld-wireshark using tshark. Additionally, this class also parses the data and generates the necessary files that will later be displayed as a dataline.

**HomeWindow.py**

This class is responsible to display the main window of the Timeline View subsystem. This class also provides the key functionality of showing/hiding datalines, enable syncing between eceld-wireshark and DVS, edit and tag data from the datalines and save the changes made.

**Mouseclicks.py**

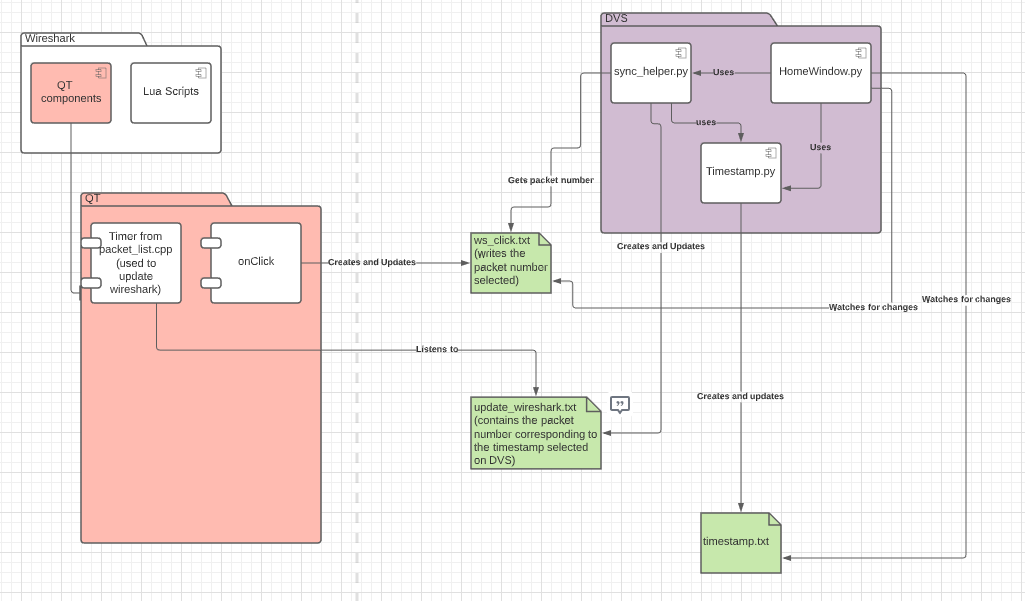
This class is responsible of the creation of the Mouseclicks dataline by reading the MouseclicksJSON file from the ParsedLogs folder in the project selected. It also provides the functionality of enlarging the image of the mouse click when clicked.

**save.py**

This class is responsible for the save functionality of the project.

**sync\_helper.py**

This class contains the methods needed to enable the synchronization of packets between eceld-wireshark and DVS. Figure 4 will describe how the sync\_helper.py interacts with other modules in the DVS System and eceld-wireshark.



*Figure 4: sync\_helper.py Collaboration Diagram*

**textdataline.py**

This class is responsible of the creation of the Keypresses, System Calls, Suricata, Packet Comments default datalines by reading the corresponding JSON file from the ParsedLogs folder in the project selected. It is also responsible for creating new datalines that do not contain images from a JSON chosen by the user.

**throughput.py**

This class is responsible of the creation of the Throughput dataline by reading the networkDataXY.JSON file from the parsed/tshark folder in the project selected.

**TimedScreenshots.py**

This class is responsible of the creation of the Timed Screenshots dataline by reading the TimesScreenshots.JSON file from the ParsedLogs folder in the project selected. It also provides the functionality of enlarging the image of the screenshot when clicked.

**Timestamp.py**

This class is responsible to help the syncing functionality, which is done through the timestamp of each row of each dataline table. Please refer to the diagram in sync\_helper.py for a detailed description of how the synchronization works.

**WebEngineView.py**

This module is responsible to open, display and update the throughput dataline.

**src:**

This package contains the source code for the startup page of DVS and the file directory picker. Below is a list of the classes that belong to the src package.

**DVSStartUpPage.iu**

This file contains the ui format of the DVSStartUpPage.

**fileDirectory.py**

This class opens the file directory window so the user can choose the desired project directory.

**Dash:**

This package contains the classes and modules which start the Dash and Web Engine services required to open the Throughput dataline. It also contains the module that shuts down the server that starts when the Throughput dataline starts. Below is a list of the classes that belong to the Dash package.

**RunWebEngine.py**

This class manages the Web Engine server that is required for the Throughput dataline.

**shutdown\_dash\_server.py**

This class kills the server needed for the Throughput dataline to avoid future conflicts with the server if it’s started again.

**Dialogs:**

This package contains the different dialogs that DVS utilizes. Below is a list of the classes that belong to the Dialogs package.

**AddTag.py**

This class provides a pop-up window that open up when the user wants to add or edit a tag to an entry of the dataline tables.

**DateTimePicker.py**

This class provides a widget that allows the user to select a date and a time to be converted to the required format from the datalines, so it is easier for the user to edit the timestamp of an entry of the dataline tables.

**EditTextDialog.py**

This class provides a widget that allows the user to modify the content of the “Content” column in the dataline tables.

**ExportDialog.py**

This class provides a widget that allows the user to choose where they want to export the project files.

**NewProjectDialog.py**

This class provides a widget that allows the user to choose whether they want to import a project, open an existing one or creating a new project.

**ProgressBarDialog.py**

This class provides a widget that displays a progress bar that indicates the project files are being loaded.

**Settings.py**

This class provides a widget that allows the user to customize the following DVS settings: syncing ON or OFF, syncing margin (0 seconds or 1 second)

**Package Manager:**

This package contains the class required to perform the packaging and unpackaging of project folders when importing and exporting projects (e.g. unzip, zip, copy files, etc.). Below is a list of the classes that belong to the Package Manager package.

**PackagerManager.py**

This class provides methods for the following functionality:

* unzip files for the import functionality
* zip files for the export functionality
* get the project files
* copy files to new project directory

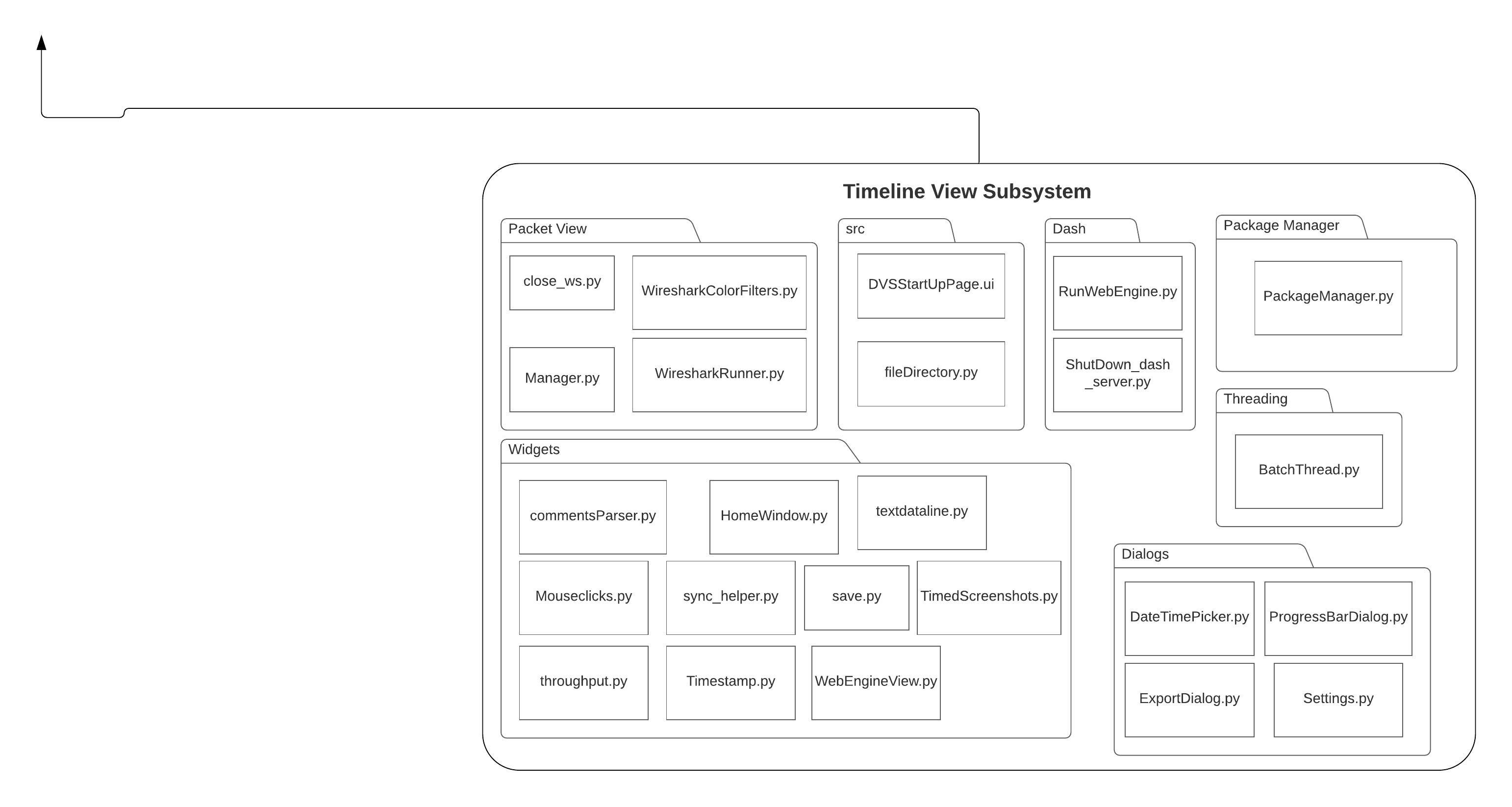
**Threading:**

This package only contains one class. This was derived from the eceld-netsys repository to help us with the progress bar. This is mainly used for the features that requires a certain set of actions in order to accomplish what is being requested. Since this is connected to the progress bar, the progress bar will disappear when all the functions have completed execution.

**BatchThread.py**

This class provides methods for the following functionality:

* Add functions to have a list of actions/functions that needs to be executed
* Get load count which just returns the length of the list mentioned above
* Run function is what is triggered once the system is ready to execute the function list



*Figure 5: Analysis Subsystem Diagram*

# Detailed Description of DVS Main

This section will describe in detail the DVS Main class.

## Main Class Description

**Class Name:** Timeline View Subsystem

**Description:** This class is responsible for initializing the startup page of DVS where the user will select whether to open an existing project or creating a new project by importing a zip folder.

&