**Implementation in C++**

As mentioned in the Design Patterns document, we have not implemented many design patterns and instead built all of our basic functionality with a Model View Controller (MVC) design pattern in mind. MVC is a simple design pattern with emphasis on separation between the UI and the data. This has enabled us to quickly create a problem-specific solution, rather than a pattern-specific solution. With our basic functionalities in place, we are now at a stage where we can refactor code and begin implementing new design patterns that best fit the needs of our application.

**Model**

The Model consists of the Data Transfer Object (DTO) classes that are assembled from the parsed csv data. Each csv type is represented by a unique DTO which provides the necessary accessor functions for the controller it is passed to.

For example, in this snippet of the *PublicationDTO* header file we can see that the model just contains the accessible data with no connection to the view or controller classes.

**View**

The View is made up of the “.ui” User Interface xml files that describe the layout and interactivity of the elements on screen. The views contain no application logic.

**Controller**

The Controller consists of the *load\_csv.cpp*, *verify\_csv.cpp*, *analyze\_csv.cpp* classes that connect the models with the views. Each controller is connected to a “.ui” file that describe its **view**and is passed a **model** as a parameter. The role of the controller is to dictate how the **model** should impact the **view** and how the users interactions with the **view** should impact the **model**. For example, in the *analyze\_cv.cpp:*



The *\_data* parameter is a pointer to a DTO of a certain csv type and acts as this views **model**

The *ui* element is an instance of the *analyze\_csv.ui* **view**

The controller is also responsible for translating user actions into operations on the model, and so it contains all of the *button clicked* handlers.