Transforming Freight Flow Data Collection

Final Report

Agreement No. 693JK41850002

Saeed Ghanbartehrani

Sara Akbar Ghanadian

Mohammad Aladailah

Hoda Rahmani Motlagh

US Department of Transportation

Office of The Assistant Secretary for Research and Technology

Bureau of Transportation Statistics

December 2019

ACKNOWLEDGEMENTS

This research was funded by the US Department of Transportation Bureau of Transportation Statistics (BTS). We would like to thank Julie Parker from BTS Office of Data Development and Standards and Medhi Hashemipour from BTS Office of Spatial Analysis and Visualization for their insights, assistance and support in this research.

# Table of Contents

[1. Introduction 1](#_Toc19709056)

[2. Data Validation Taxonomy 3](#_Toc19709057)

[2.1. Field validation 3](#_Toc19709058)

[2.1.1. Data type check 4](#_Toc19709059)

[2.1.2. Presence check 4](#_Toc19709060)

[2.1.3. Format check 4](#_Toc19709061)

[2.1.4. Field length check 4](#_Toc19709062)

[2.1.5. Invalid character check 4](#_Toc19709063)

[2.2. Range validation 4](#_Toc19709064)

[2.2.1. Range and limit check 5](#_Toc19709065)

[2.3. Code and cross consistency 5](#_Toc19709066)

[2.3.1. Lookup tables 5](#_Toc19709067)

[2.3.2. External rules and logic check 5](#_Toc19709068)

[2.3.3. Combinational lookup tables and rules 6](#_Toc19709069)

[2.3.4. Consistency check 6](#_Toc19709070)

[3. Data validation life cycle 7](#_Toc19709071)

[3.1. Design phase 7](#_Toc19709072)

[3.2. Implementation phase 8](#_Toc19709073)

[3.3. Execution phase 8](#_Toc19709074)

[3.4. Review phase 8](#_Toc19709075)

[4. Open File Formats 9](#_Toc19709076)

[4.1. Conclusion 18](#_Toc19709077)

[5. Application Architecture 20](#_Toc19709078)

[5.1. Backend 20](#_Toc19709079)

[5.1.1. Validation base (B1-1) 21](#_Toc19709080)

[5.1.2. Validation establishment (B1-2) 21](#_Toc19709081)

[5.1.3. Validation shipment (B1-3) 22](#_Toc19709082)

[5.1.4. Related resources (BR1) 22](#_Toc19709083)

[5.1.5. Variable name matching (B2-1) 23](#_Toc19709084)

[5.1.6. Related resources BR2 26](#_Toc19709085)

[5.1.7. Libraries (BL) 26](#_Toc19709086)

[5.2. Frontend (F) 26](#_Toc19709087)

[5.2.1. User interface (F1) 27](#_Toc19709088)

[5.2.2. Related resources (FR1) 27](#_Toc19709089)

[5.2.3. Front end libraries (FL) 28](#_Toc19709090)

[6. Application Graphical User Interface (GUI) 29](#_Toc19709091)

[6.1. Submitting a file for validation 29](#_Toc19709092)

[6.2. Configuring the variable names 30](#_Toc19709093)

[6.3. Validating data and data validation report 33](#_Toc19709094)

[7. References 35](#_Toc19709095)

# Introduction

Data validation is the process of ensuring the delivery of clean and clear data to the programs, applications and services. This process is a decisional procedure that leads to accepting or refusing the entered data (Wikipedia contributors, 2017). In the data validation procedure, rules are applied to data in order to validate them for the final use it is intended for (Di Zio et al., 2016).

The main objective of data validation is to guarantee a certain level of quality for the final data. Quality level consists of various dimensions including *accuracy*, *coherence and comparability*, *clarity and accessibility*, *timeliness*, *relevance*, and *completeness*. A summary of the definitions for the discussed quality dimensions are presented in the following paragraphs.

*Accuracy* is the metric to measure the difference between the target and estimated parameters which is typically caused by sampling or none-sampling errors. Sampling errors are not considered in data validation since they are the result of taking samples from a population. Non-sampling errors are caused by factors not related to taking samples and therefore are aimed in data validation process.

*Coherence and comparability* ensure that statistics are consistent internally over time, and they are comparable between regions and countries based on common standards. It is essential to maintain coherence and comparability in validation rules to detect errors, and flag inconsistencies in the data validation process.

*Clarity and accessibility* are associated with the information system formal checks to prevent any misunderstanding or misinterpretation of the collected data.

*Timeliness* can be a constraint in the data validation process. If the data validation process is time consuming, the process can be made less restrictive on the level of acceptance to meet the expected deadline for the release of the validated data (Di Zio et al., 2016).

*Relevance* is defined as the level of consistency between the content and the application area of the data (Ferraggine et al., 2009). This dimension is not considered in the data validation process since it is mostly related to the choice of the data source.

*Completeness* mans that the collected data meets the requirements for the current and future applications of the data (Ferraggine et al., 2009). This dimension is considered when compiling data requirements and selecting the data sources. Some data validation rules can also ensure that the submitted data is complete.

The data validation procedures mostly focus on accuracy and coherence and comparability dimensions as the main components related to the structure of the data. Accordingly, integrity and validity of the input data are checked in data validation process to ensure accuracy, coherence and consistency (Wikipedia contributors, 2017).

# Data Validation Taxonomy

Data validation procedures are designed specifically based on the type of the data and its intended use and implemented in multiple levels. Data validation methods or rules can be classified based on the scope, complexity, and purpose of the validation operations to be carried out (Wikipedia contributors, 2017). Table 1 presents a taxonomy of data validation methods including the most common data validation rules.

|  |  |  |
| --- | --- | --- |
| **Level** | **Classification** | **Check types** |
| 1 | Field validation | Data type check  Presence check  Format check  Field length check  Invalid character check |
| 2 | Range validation | Range and limit check (extreme value check) |
| 3 | Code and cross consistency | Lookup tables  External rules and logic check  Combinational lookup tables and rules  Cross consistency check of various fields |

Table 1. Data validation rules and methods

## Field validation

Field validation includes all validation rules in the context of the field. One common error is misinterpreting the units or meaning of a field. For example, the weight value may be required in Kilogram, ton, or pound. Looking at both the definition and actual reported values is helpful in avoiding misinterpretations that can lead to incorrect conclusions. Therefore, field validation ensures that the provided values make sense in the context of a given field (Di Zio et al., 2016). Field validation includes different types of checks such as data type check, presence check, format check, and invalid character check.

### Data type check

Data type check is a mechanism to detect if the type of input data does not match the predefined data type. For example, alphabetic characters are not acceptable values in numeric fields such as zip code, fax number, phone number, and so on.

### Presence check

Presence check is performed to check that a required data element has been provided and is not missing. For example, it is usually necessary to verify the presence of data in contact information fields.

### Format check

Format check is a mechanism to check that the input data is in the correct format. For example, the date values may be requested in a specific format such as “MMDDYYYY”.

### Field length check

Field length check ensures that the input data is not too long or too short. For example, the minimum and maximum number of characters in a name field can be set to 2 and 8 characters respectively.

### Invalid character check

Invalid character check is a mechanism to check for unusual characters which are not allowed to be used in a given field. For example, special characters such as ‘!’, ‘%’, or ‘#’ are not allowed in an email address field.

## Range validation

The second level in the proposed taxonomy is range validation which includes range and limit check which is also known as extreme value check.

### Range and limit check

Range check verifies the data is within a specified range of values while limit check, checks the data against one limit only which is specified with an upper or lower bound value. For example, the minute attribute in a time stamp field ranges between 0 and 59 which is an example of a range check. An example of a limit check is the odometer reading value a vehicle which should always be greater than or equal to the previous reading.

## Code and cross consistency

Code and cross consistency validation rules involve cross consistency checks performed on multiple fields to verify the consistency of the input data with one or more external rules or a set of pre-defined assumptions and validity constraints. Moreover, applying lookup tables is common in this kind of validation (Di Zio et al., 2016). Lookup tables, external rules and logic check, combinational lookup tables and rules, and cross consistency check of various fields are the checks performed at this level.

### Lookup tables

A lookup table check compares the input data to a valid list of entries which are kept in a table referred to as lookup table. For example, a lookup table can be used to ensure the combination of zip code, city, and state values is valid.

### External rules and logic check

External rules are used to verify the input data complies with related external rules or a set of pre-defined assumptions based on the specific data on hand (Di Zio et al., 2016). For example, the rightmost digit in a UPC barcode is a check digit that is computed based on the rest of the digits in the barcode using a special formula. The same formula can be used to re-calculate the check digit to ensure the validity of the scanned code.

### Combinational lookup tables and rules

In some cases, both lookup tables and external rules are needed to be verified as a combinational validation check. As an example, the transportation modes for a specific product is required to be verified by specific rules which are based on the type of commodity as well as possible combinations of the modes of transportation which are provided in lookup tables.

### Consistency check

Consistency check is a way to ensure that there is no semantically inflicting elements in the data (Janssen & Janssen, n.d.). In other words, it is a method to check the consistency between the data collected from different sources (Wikipedia contributors, 2017). For example, the actual number of shipments in the data file should be consistent with the total number of shipments reported during a given period according to the sampling instructions.

# Data validation life cycle

The data validation process life cycle is a set of activities to design and execute data validation. Data validation life cycle focuses on improving the performance of the entire validation process by providing clear definitions of the activities and responsibilities to guarantee the highest performance and minimize the possibility of mistakes. Data validation life cycle consists of the four phases depicted in Figure 1.

Figure 1. Data validation life cycle (Di Zio et al., 2016)

## Design phase

In the design phase, the data sets are first reviewed. Then, the quality requirements for the variables are evaluated to identify the variables and their relations. The proper level of validation rules for the identified variables in the datasets are determined based on the quality requirements. Finally, the validation rules are set up. The validation rules should be coherent, complete, and consistent. Also, Co-operation of questionnaire designers, validation and editing experts, subject matter specialists, and survey designers is necessary in this phase.

## Implementation phase

In the implementation phase, the validation rules are formally implemented in a common syntax. Then, they are applied to the test data and the test results are analyzed. The validation rules are revised based on the test results and they are documented.

## Execution phase

In the execution phase, the data is checked against the validation rules. The validation rules produce flags pointing out acceptable and unacceptable data. The results are then analyzed and summarized to be used for improving the validation rules.

## Review phase

In the review phase, the need for new design elements are recognized based on the feedback from users and stakeholders or the analysis of outcomes. The updates are then prioritized for implementation in future revisions.

# Open File Formats

The standard ways of displaying the sequence of bytes are defined by file formats (McCallum, 2013). An open format is a file format for storing digital data, which can be used and implemented by anyone without any restrictions (i.e. not proprietary). File formats with no restrictions and patents placed upon their use are known as open file formats. Open file formats are usually recognized by multiple programs. This characteristic makes open formats an ideal choice for transferring information between several software packages (Dietrich et al., 2010).

There are different types of open data file formats, which vary in terms of structure, length, application, and other characteristics. In addition, several considerations need to be taken into account when determining which format should be employed in a given project. For instance, human readability should be considered as an important criterion if the files are expected to be viewed naturally by human eyes (McCallum, 2013). A list of common open file formats is presented in Table 2.

|  |  |  |
| --- | --- | --- |
| **Format** | **Advantages** | **Disadvantages** |
| Spreadsheet | Simple integration with graphic and word processing tools  Perfect choice for storing data in tabular form  Easy to perform statistical analysis  Easy to edit table entries | Increased storage space requirement  Greater risk for data corruption or mismanagement of information  High processing/parsing overhead |
| Plain Text | Easy to parse, filter and index  Small in size  Ability to be recognized by any text editing or word processing program  Ability to store information with no special formatting | Inefficient for data sets with complex structure |
| CSV  (Comma Separated Values) | Easy to generate, implement and parse  Compatibility with a wide variety of data analysis and processing applications such as spreadsheet editors, Database Management Systems, and so on  Ability to differentiate between text or numeric values  The header row provides a straightforward information schema  Memory efficient format | All rows of data must follow the same format defined in the header |
| HTML  (Hyper Text Markup Language) | Can be used to display any type of data on a web page  Can be viewed by any web browser  Ability to develop applications adapting to different devices | Requires programming  Specifically made for web documents |
| JSON  (JavaScript Object Notation) | Flexible structure  Compatible with many data analysis and processing applications  General purpose data storage format  Easily consumed by JavaScript programming language | High storage overhead  Not easy to read by human |
| XML  (eXtensible Markup Language) | Compatible with many data analysis and processing applications  General purpose data storage format | Inefficient for exchanging relational data  High storage overhead  Not easy to read by human |
| RDF (Resource Description Framework) | Easy data exchange and interoperability  Ability to capture metadata and structure from unstructured data | Not a very common format  Special purpose data storage for metadata  Not easy to read by human |

Table 2. Common open file formats

One common open data file format which is widely accepted by users and utilized by many organizations is spreadsheet. Spreadsheet files store data in cells, which are organized into rows and columns. This file format is known as the ancestor of all data formats since it consists of columns and rows for presenting attributes and their corresponding observations, respectively (Dietrich et al., 2010). Microsoft Excel is a widely used proprietary spreadsheet program that provides tools to visualize and analyze data in workbooks and worksheets with \*.xls and \*.xlsx extensions. Figure 2 illustrates a spreadsheet file with few rows.



Figure 2. An example of a spreadsheet file

Plain text is a simple file format (such as \*.txt or \*.dat) in which data can be arranged in any desired form. The simplicity of plain text format enables it to be read by almost any software package as well as being portable across different computer platforms. However, using this file format might not be suitable for data sets with complex structure. Also, the fact that text files do not follow a predefined structure makes them prone to errors and makes the verification process difficult. Figure 3 demonstrates an example of a plain text file.

First Name Last Name Gender Country Age Date ID

Judy Morison Female United States 32 15/10/2017 1562

Jennifer Hanner Female Great Britain 25 16/08/2016 1582

Philip Gent Male France 36 21/05/2015 2587

Figure 3. An example of a plain text file

Another format is Comma Separated Values (also known as CSV), which is one of the simplest methods to store structured data. This format can be considered a structured plain text format. In fact, a CSV file contains several lines of data, with each data element (i.e. attribute) being separated from the next one by a comma delimiter. Rows in a CSV file are separated by line breaks. The first row, referred to as the header, lists the names for the data elements or columns for the data stored in the file. CSV files are used for sharing tabular data and many spreadsheet applications like Microsoft Excel and OpenOffice can import and export data in this format (“European Data Portal,” 2018). In CSV files, tabular data consisting of textual or numeric values can be stored in plain text (“Open data,” 2018). Although CSV files are a perfect choice for storing structured tabular data, they are inefficient for storing semi-structured or unstructured data. An example of a CSV file is shown in Figure 4.

First Name,Last Name,Gender,Country,Age,Date,ID

Judy,Morison,Female,United States,32,15/10/2017,1562

Jennifer,Hanner,Female,Great Britain,25,16/08/2016,1582

Philip,Gent,Male,France,36,21/05/2015,2587

Figure 4. An example of a CSV file

Hypertext Markup Language (HTML) is another type of open data file format, which is used for creating [web pages](https://en.wikipedia.org/wiki/Web_page). An HTML file consists of elements that are the building blocks of HTML pages. Perhaps, one of the easiest ways of sharing the data is through a web page accessible on the Internet (Dietrich et al., 2010). HTML file format is supported by any standard web browser application such as Mozilla Firefox, Google Chrome, Safari, and so on. The web browser executes the code in an HTML file to display the data properly. Figure 5 is an illustration of the source code in an HTML file while Figure 6 depicts the same document as it is displayed in a web browser.

<!DOCTYPE html>

<html>

<body>

<table>

<thead>

<tr>

<th>First Name</th>

<th>Last Name</th>

<th>Gender</th>

<th>Country</th>

<th>Age</th>

<th>Date</th>

<th>ID</th>

</tr>

</thead>

<tbody>

<tr>

<td>Judy</td>

<td>Morison</td>

<td>Female</td>

<td>United States</td>

<td>32</td>

<td>15/10/2017</td>

<td>1562</td>

</tr>

<tr>

<td>Jennifer</td>

<td>Hanner</td>

<td>Female</td>

<td>Great Britain</td>

<td>25</td>

<td>16/08/2016</td>

<td>1582</td>

</tr>

<tr>

<td>Philip</td>

<td>Gent</td>

<td>Male</td>

<td>France</td>

<td>36</td>

<td>21/05/2015</td>

<td>2587</td>

</tr>

</tbody>

</table>

</body>

</html>

Figure 5. An example of an HTML file (source code)



Figure 6. An example of an HTML file (viewed in a web browser)

JavaScript Object Notation (JSON) is a format that is easy to read by most programming languages, particularly JavaScript programming language. This format allows data to be represented in a structured manner with a level of data relationship. In fact, JSON is perfect for storing semi-structured and unstructured data due to its highly flexible structure. JSON is not the most efficient format for storing tabular data due to the extensive use of brackets and attribute names to display each line of data. JSON files are intuitive which makes them easy for human to read. A significant advantage of JSON data structure is its easy application in JavaScript programming language. Figure 7 displays an example JSON file.

[

{

"First Name": "Judy",

"Last Name": "Morison",

"Gender": "Female",

"Country": "United States",

"Age": "32",

"Date": "15/10/2017",

"ID": "1562"

},

{

"First Name": "Jennifer",

"Last Name": "Hanner",

"Gender": "Female",

"Country": "Great Britain",

"Age": "25",

"Date": "16/08/2016",

"ID": "1582"

},

{

"First Name": "Philip",

"Last Name": "Gent",

"Gender": "Male",

"Country": "France",

"Age": "36",

"Date": "21/05/2015",

"ID": "2587"

}

]

Figure 7. An example of a JSON file

Another common open file format is eXtensible Markup Language or XML. XML is mainly used for representing formatted text and application configuration files. Microsoft office, OpenOffice, and Apple’s iWork suites use XML based formats.

In XML files, each data element starts and ends with a tag (i.e. <tag> …</tag>). XML file format allows the developers to define tags as needed. However, the high storage overhead of the tags in each data element makes XML documents inefficient for data storage. Figure 8 shows an example XML file.

<?xml version="1.0" encoding="UTF-8" ?>

<root>

<0>

<First Name>Judy</First Name>

<Last Name>Morison</Last Name>

<Gender>Female</Gender>

<Country>United States</Country>

<Age>32</Age>

<Date>15/10/2017</Date>

<ID>1562</ID>

</0>

<1>

<First Name>Jennifer</First Name>

<Last Name>Hanner</Last Name>

<Gender>Female</Gender>

<Country>Great Britain</Country>

<Age>25</Age>

<Date>16/08/2016</Date>

<ID>1582</ID>

</1>

<2>

<First Name>Philip</First Name>

<Last Name>Gent</Last Name>

<Gender>Male</Gender>

<Country>France</Country>

<Age>36</Age>

<Date>21/05/2015</Date>

<ID>2587</ID>

</2>

</root>

Figure 8. An example of an XML file

Resource Description Framework (RDF) is an open file format which can be stored in XML and JSON-like formats as well as custom, application specific formats. By supporting the use of URLs as identifiers, RDF allows direct interconnection between several data sources on the Web (Dietrich et al., 2010).

The RDF data model is based on the idea of making statements about resources based on subject–predicate–object from which is known as triples. The *subject* denotes the resource, the *predicate* denotes traits or specifics of the resource, and expresses a relationship between the *subject* and the *object*.

RDF is a highly flexible format which makes it a perfect choice for storing unstructured data. This benefit also makes the parsing of RDF file difficult. An example of an XML-RDF file is presented in Figure 9.

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE rdf:RDF [

<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>

<!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>

<!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>

]>

<rdf:RDF xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns="http://example.org/data/input.csv#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

<rdf:Description rdf:about="http://example.org/data/input.csv#row=1/1562">

<Age rdf:datatype="&xsd;integer">32</Age>

<Country xml:lang="en-us">United States</Country>

<Date>15/10/2017</Date>

<First\_Name xml:lang="en-us">Judy</First\_Name>

<Gender xml:lang="en-us">Female</Gender>

<Last\_Name xml:lang="en-us">Morison</Last\_Name>

</rdf:Description>

<rdf:Description rdf:about="http://example.org/data/input.csv#row=2/1582">

<Age rdf:datatype="&xsd;integer">25</Age>

<Country xml:lang="en-us">Great Britain</Country>

<Date>16/08/2016</Date>

<First\_Name xml:lang="en-us">Jennifer</First\_Name>

<Gender xml:lang="en-us">Female</Gender>

<Last\_Name xml:lang="en-us">Hanner</Last\_Name>

</rdf:Description>

<rdf:Description rdf:about="http://example.org/data/input.csv#row=3/2587">

<Age rdf:datatype="&xsd;integer">36</Age>

<Country xml:lang="en-us">France</Country>

<Date>21/05/2015</Date>

<First\_Name xml:lang="en-us">Philip</First\_Name>

<Gender xml:lang="en-us">Male</Gender>

<Last\_Name xml:lang="en-us">Gent</Last\_Name>

</rdf:Description>

</rdf:RDF>

Figure 9. An example of an RDF file

## Conclusion

Since each file format has its own features, it is crucial to examine the best choice for the data file format that better meets the project functional requirements, long-term access, and preservation of the data.

Being open and widely used are the most critical factors when choosing a format. Also, the ability to view and edit the data file with a wide range of programs without having to rely on a proprietary software is another important consideration (“U.S. Geological Survey,” 2018).

When transferring data between users and applications (e.g. collecting survey data), If the chosen file format imposes too much effort on the users to compose or convert the data from the original source it comes from which could be a database, enterprise application such as an Enterprise Resource Planning (ERP) or Transaction Processing System (TSP), the choice would be costly and can result in errors or loss of the collected data.

The structure of the input data is another key factor in the choice of file format. Formats suited for structured data such as CSV lose their efficiency when used for storing semi-structured or unstructured data, while more flexible formats like JSON and XML use more storage space which can be an issue when transferring the data over the web.

Amongst the file formats introduced above, spreadsheets, CSV, XML, and JSON files are the best candidates for transferring data between users or users and applications.

CSV offers features that makes it an ideal format for collection, storage, and transfer of the data over the web. The CSV is a standard and efficient format because it can be viewed and edited by any text or spreadsheet editors like Notepad and Microsoft Excel as well as their free and open source alternatives such as Notepad++ and OpenOffice. Most databases and enterprise applications offer importing and exporting data in CSV file format. CSV offers the minimal storage overhead for storing large tabular datasets.

In comparison, JSON is one of the best alternatives for storing semi structured and unstructured data. The big advantage of JSON is its minimal processing overhead in JavaScript programming language which results is a more responsive application. These features make JSON the perfect choice for the data stored internally in the validation platform for validation purposes such as lookup tables and application configuration files.

# Application Architecture

The validation platform consists of two main components, referred to as the *backend* and *frontend*. The frontend includes functions and libraries related to what the user sees on the user interface and interacts with directly while the backend contains functions, libraries, and related resource (related data) that support the frontend and the overall operation of the platform.

## **Backend**

Modules and resources in the backend are depicted in Figure 10. The backend consists of four modules: data validation (B1), related resources (BR1 and BR2), variable name matching (B2), and libraries (BL).



Figure 10. Data validation platform architecture: Backend

Modules that contain functions (i.e. code) are displayed in boxes with solid lines and labeled with B (Backend), while boxes representing the related resources (i.e. configuration files, lookup tables, and data files) as well as libraries (i.e. open source JavaScript libraries) have dashed lines and labeled with BR (Backend Resources) and BL (Backend Libraries) respectively. Resources for each module are connected with an arrow to its corresponding module. The libraries module contains the libraries that support all modules in the backend.

### Validation base (B1-1)

The validation base sub-module consists of a set of functions that support the basic operations in the data validation process. For example, the “check allowed\_char()” function, is a function that is used to check for the allowed characters of an input. This operation needs to be performed in several validation functions in both validation\_establishment.js and validation\_shipment.js sub-modules and therefore, it is included in validation\_base.js to be used several times in other sub-modules. This function along with other functions contained in validation\_base.js perform the basic data validation operations such as checking for allowed, invalid, and required characters, field length, allowed range check, look up checks to find an element from a table and auto fill check. All functions in this sub-module are discussed in great details in the technical documentation.

### Validation establishment (B1-2)

validation\_establishment.js sub-module contains a set of functions that are used to validate the establishment data file. This module has a main function titled “verify\_est()” that calls all validation functions in this sub-module to validate the establishment data. Functions in this sub-module validate shipping and mailing company names, address, zip code, city, and state attributes. The validation involves checking the presence of input for the required fields, the existence of allowed characters in the input, and ensuring the input does not contain any of the disallowed characters. In addition, attributes like state are checked against the valid list if inputs (i.e. states) as well as the consistency of the input with related inputs such as city and zip code based on lookup tables.

### Validation shipment (B1-3)

The validation shipment sub-module contains a set of functions that are used to validate the shipment data file. This sub-module has a main function titled “verify\_shipment()” that calls all validation functions in this sub-module to validate the shipment data attributes. The validation process involves checking shipment ID, date, quarter, value, weight, SCTG code, SCTG description, temperature control, UN or NA numbers, destination address, city, state, zip code, transportation mode, export, and export city and country name. Checking the presence of input for required attributes, data type check, range and value check, and validity and cross consistency checks are among the tests performed in this module.

### Related resources (BR1)

Related resources BR-1 contains the configuration files, lookup tables and data files supporting the validation functions in B1. All resources in BR-1 are stored in JSON objects so that they can be easily used in JavaScript code without additional coding. Also, the data and configuration are kept separate from the source code so that all validation parameters such as allowed, not allowed, and required characters, lookup tables, error flags and their corresponding priorities and so on can be modified without any programming knowledge.

The configuration file ‘config.js’ contains a list of variable names and data types that have constraints on the value, type, range of the inputs. For example, the ‘numeric’ data type in this table, it is applied to variables with values limited to digits (i.e. 0-9). Other variable types may have a set of allowed, not allowed, and required characters. ‘flags.js’ configuration file contains a list of all error flags along with flag numbers, a brief description of the error, flag value, and priority. Five look up table files (i.e. ‘lkup\_\*.js’ files) store acceptable input values or combination of values to check the validity and consistency of the input data. For example, ‘lkup\_1.js’ contains all SCTG codes and their associated lower and upper bounds for value to weight ratios.

### Variable name matching (B2-1)

This module includes a set of functions that match the fields in the user uploaded CSV data file with the list of variable names stored in the validation platform (i.e. variables.js configuration file). This feature is referred to as user configurable variable name mapping since it allows the user to view and modify the mapping between the fields and variable names if needed. The mapping process is done by calculating similarity scores for each variable name to all fields from the user data file. “variables.js” configuration file in BR2 contains the list of variable names along with the list of keywords associated with each variable name. Figure 11 shows a sample entry for Establishment\_ID variable.

1: {

name: 'Establishment\_ID',

keys: ['est','company','id','num'],

required: false,

hint: 'Establishment unique identifier.'

}

Figure 11. Sample variable name entry in variables.js

Similarity score is simply the number of keywords that match a field from the user data file based on the substring function. For example, keywords ‘est’ and id’ from variable name ‘Establishment\_ID’ both match ‘est\_id’ field from the user data file since both ‘est’ and ‘id’ are substrings of ‘est\_id’ which results in similarity score of 2. It is worth mentioning that fields from the user data file are converted to lower case and all spaces are removed when calculating similarity scores to make the process insensitive to spaces and upper case characters. It is recommended that keywords in variables.js provided in all lower case letters and not have common substrings (e.g. ‘est’ and ‘estid’ both have ‘est’ in common and should not be used as keywords at the same time).

For each variable name in ‘variables.js’, the similarity scores to all fields form the user data file are calculated. Variable names for shipment and establishment data files are stored in separate sections in ‘variables.js’, so only variable names from the appropriate section/file is considered in the mapping process.

The calculated similarity scores for each variable name are then sorted in descending order. If there is only one field from the user data file that has the highest similarity score (i.e. the next similarity score in the list of sorted scores for a variable is less than the highest score), the variable name is marked as a potential match, otherwise (i.e. there is more than one field with the highest score in the list of sorted scores for a variable), the variable name is marked for manual user configuration. Also, if all similarity scores for a field from the user data file are zero (i.e. field does not match any of the keywords listed under all variable names), the field is marked for manual user configuration.

Once potential matches for all variable names are determined, all variable names are checked again for possible double matches (i.e. if two variable names have the same potential field match). All variable names with a unique field match are marked as matched while the variable names with the same potential field match are marked for manual user configuration.

Once the matching process is complete, for each variable name, a combo box with the list of fields from the user data file sorted by their similarity score is generated. Figure 12 shows the list of fields from the user data file sorted by their similarity score for ‘Shipping\_Company\_Name’ variable. Also, combo boxes for variable names marked for manual configuration will display ‘Make a Selection’ and are highlighted in red. ‘Establishment\_ID’ variable name is an example of this case displayed in Figure 12.

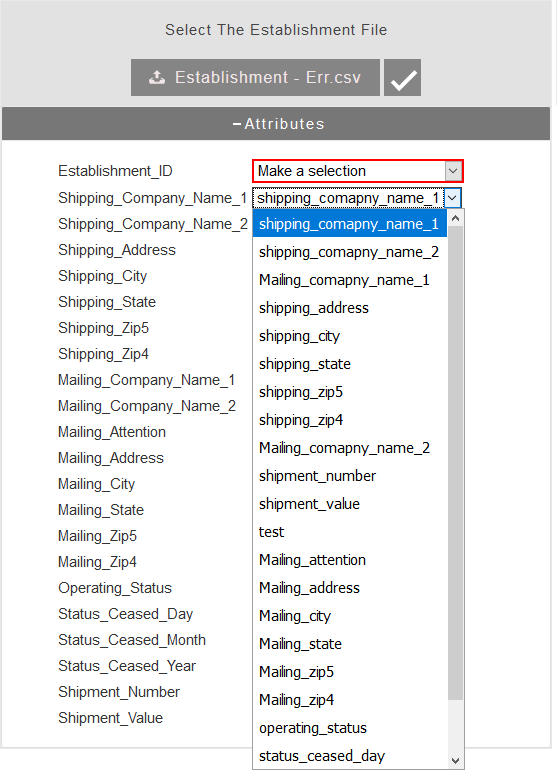


Figure 12. Combo boxes generated for establishment file variable name

The value for attribute ‘required’ in variables.js’ determines whether a variable is required to be present in the user data file or not. When set to ‘false’, ‘Not available’ option is added at the end of the combo box for the variable. The user can proceed with the data validation without configuring variable names that are not required even if they are set for manual configuration. All required variables need to be configured properly for the validate button (i.e. button with the check mark icon next to the file selection button) to appear.

### Related resources BR2

This sub-module includes ‘variables.js’ configuration file which contains a list of all variable names, their associated keywords, whether they are required or not, and the description displayed on the tooltip (on the combo boxes on the user interface). The keywords are used by functions in ‘var\_match.js’ in the variable mapping process.

### Libraries (BL)

This sub-module contains the open source JavaScript libraries used in the backend. The two libraries are jquery and papaparse. Papaparse is a library for parsing large CSV files, and the jquery is a general purpose library adding modern features including HTML/DOM manipulation, CSS manipulation, HTML event methods, effects and animations, AJAX, and so on.

## Frontend (F)

The frontend consists of three sub-modules: User interface module (F1), related resources module (FR1) and libraries module (FL). Figure 12 shows the overall architecture of the validation platform frontend. Conventions similar to the backend module are used for naming (i.e. F, FR, FL) and representing (i.e. solid and dashed lines) the frontend sub-modules. The user interface (F1) sub-module uses resources in FR1 as well as the three JavaScript libraries in FL.



Figure 13. Data validation platform architecture

### User interface (F1)

This sub-module contains the set of functions that are related to the user interface. These functions recieve the user submitted data files, process the uploaded files, format and control the Graphical User Interface (GUI), and implement the interactive GUI features such as hiding and unhiding controls and tooltips on mouse hover or click.

### Related resources (FR1)

This module consists of one configuration file containing the settings for the colors associated with the error flag priorities as well as the display of the error flags in the reports. The first variable (colors) includes the three color values associated with error flags with priorities 1, 2, and 3. It is worth mentioning that this variable controls the colors on the GUI as well as the Excel reports.

The second variable (filters) controls the level(s) for the error flags displayed on each of the shipment and establishment reports. Both filters are currently set to 3 which means error flags with priorities of 3 or higher (i.e. 1,2, and 3) are displayed.

var colors = {

1:{

color: "rgba(255, 0, 0, 0.2)" // #ffcccc rgb(255,204,204)

},

2:{

color: "rgba(255,153,0,0.4)" // #ffd699 rgb(255,214,153)

},

3:{

color: "rgba(255, 255, 0, 0.2)" // #ffffcc rgb(255,255,204)

}

}

var filters = {

shipTable\_filter: 3,

estTable\_filter: 3

}

Figure 14. ‘colors.js’ configuration file

### Front end libraries (FL)

This module includes the three libraries of datatables, jquery, and jquery-ui all supporting the frontend module. ‘ datatables’ library presents the error reports in tabular form and provides features such as sorting, paging, searching, and exporting (i.e. CSV, Excel, PDF). The jquery-ui is a JavaScript library providing advanced styling features (e.g. animations, icons, etc.) for the datatables. ‘jquery’ is also required by datatables and jquery-ui libraries.

# Application Graphical User Interface (GUI)

The design elements and features in Validation Platform GUI are discussed in this chapter. Figure 15 depicts the application home page. Before submitting the data files, the two ‘Select a File’ buttons are the only controls displayed to the user. The left button is for submitting the establishment data file, and the right buttons is for the shipment data file submission.

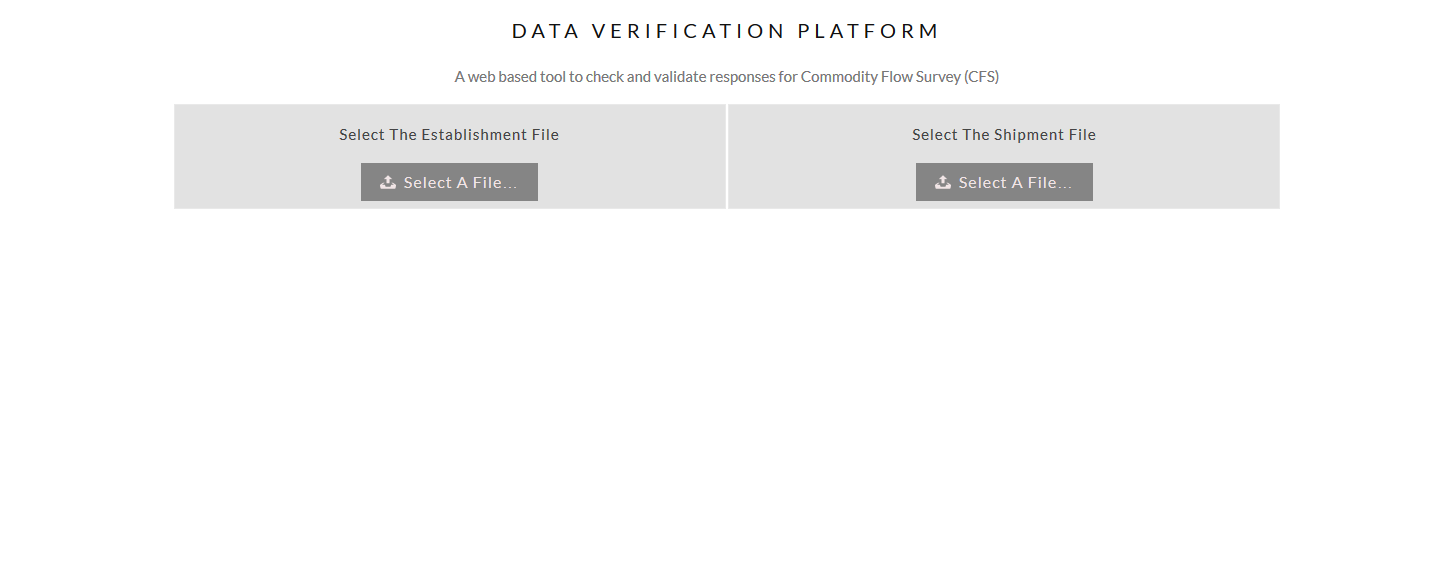


Figure 15. Validation Platform home page

## Submitting a file for validation

Validation Platform accepts standard CSV files with headers. Files can be submitted by clicking on ‘Select A File’ buttons. A ‘file upload’ window allows the user to select and submit the file from their computer. The file upload window appearance varies based on the users’ Operation System (OS) (e.g. Windows, Mac OS, Linux, etc.). Figure 16 shows the file upload window on Windows OS. If the correct file type and format is submitted, the ‘Attributes’ pane in Figure 17 will be appear under the file submission section.

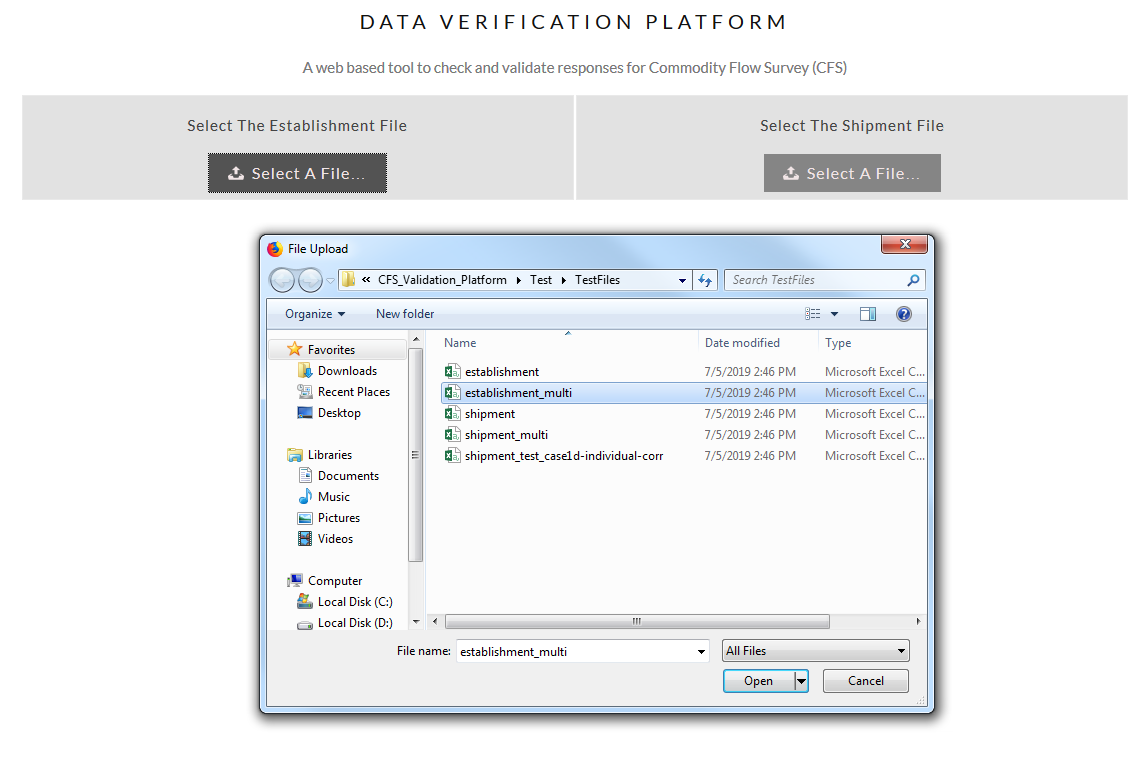


Figure 16. ‘File Upload’ Window appearance in Windows OS

## Configuring the variable names

After submitting an appropriate CSV data file, the ‘Attributes’ pane displayed in Figure 17 will be appended to the user interface. The list of expected variable names for each file is displayed on the left side of the attributes pane (highlighted with a red rectangle) while the list of fields (attributes) in the user uploaded data file are listed in the combo boxes next to each variable name. The process of mapping the variable names to the fields in the user submitted data file is discussed in detail in section ‎5.1.5. The fields from the user data file are sorted by their calculated similarity score to each of the variables in each combo box to facilitate the manual configuration (if needed). A tooltip with the definition of each variable pops up when the cursor is pointed on each combo box as shown in Figure 18.

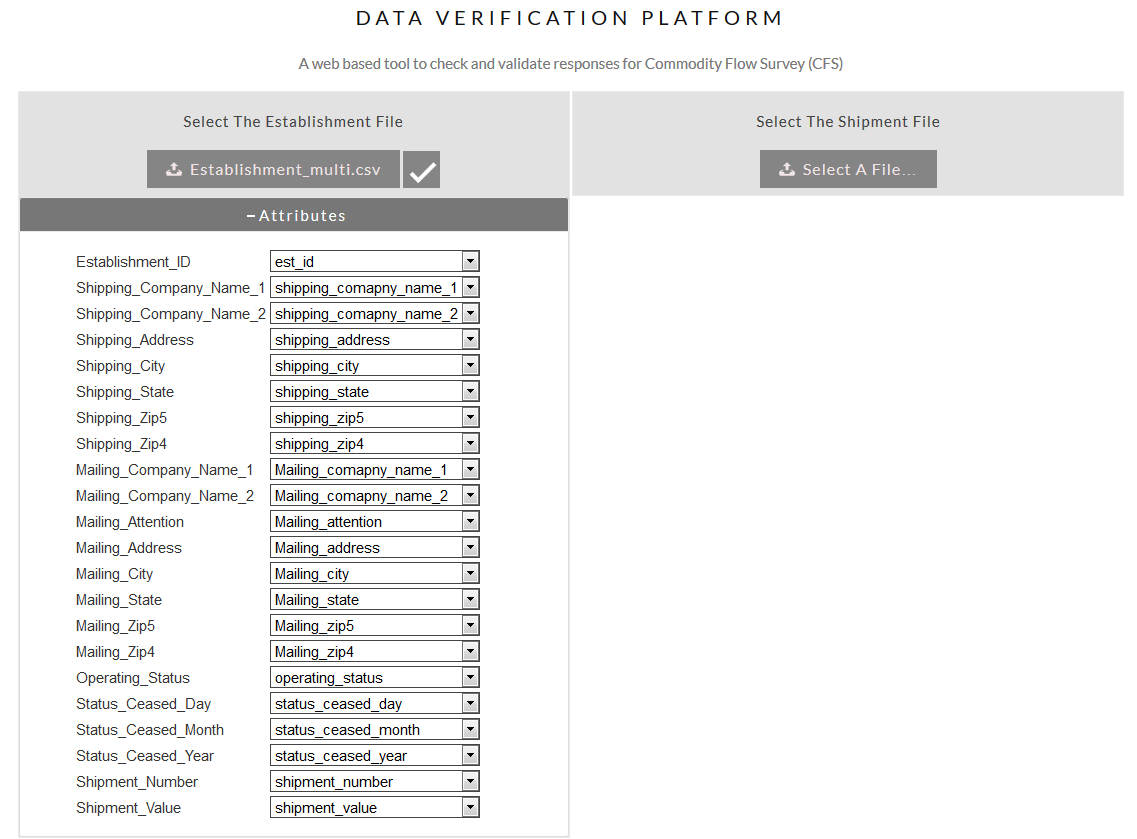


Figure 17. Data attributes configuration screen

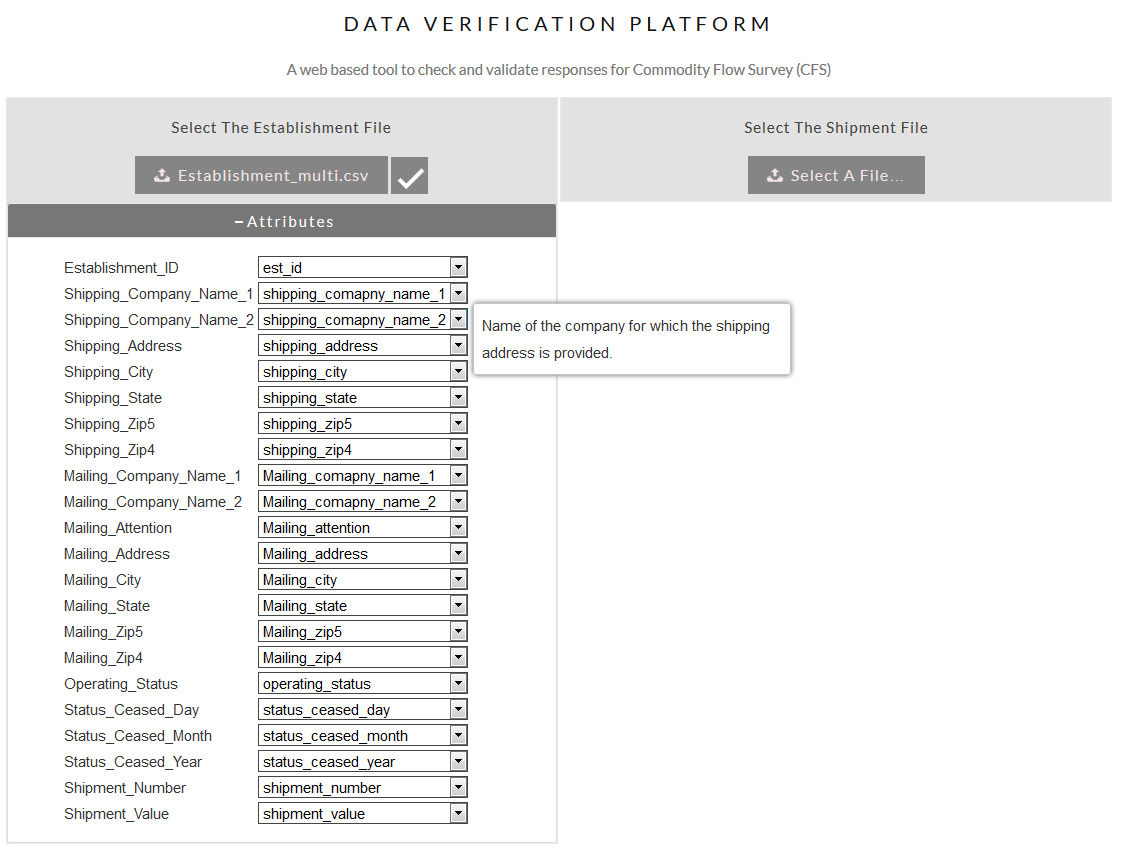


Figure 18. Attributes pane and variable definition tooltip

When any of the combo boxes is modified, selected fields for all variables are verified to ensure all required variables are mapped to a field from the user submitted data file no filed is mapped to more than one variable name. Figure 19 shows an example scenario in which ‘shipping\_zip4’ filed from the user submitted data file is mapped to two variable names. All combo boxes that require manual configuration are highlighted in red. Once all variables are properly configured, red highlights disappear immediately and the verify button with the check icon on it appears which allows the user to proceed with the data validation as shown in Figure 18.

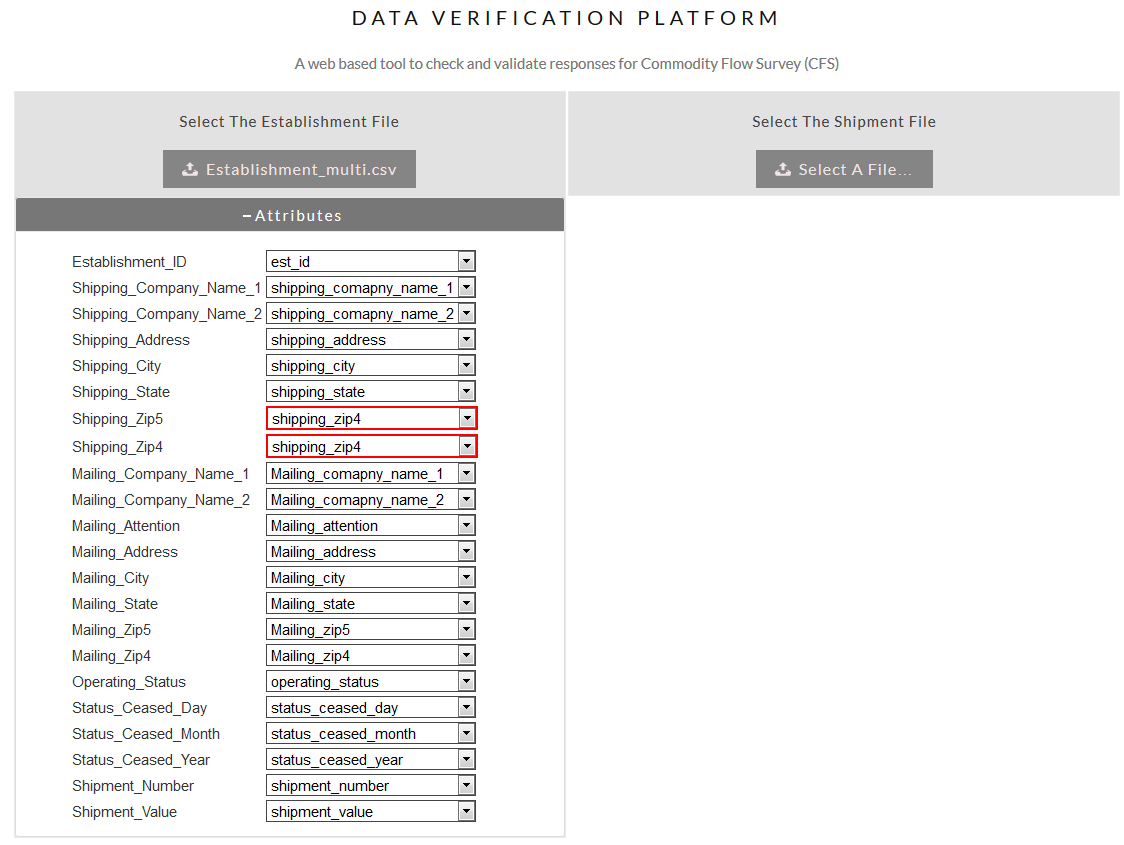


Figure 19 error in data attributes configuration due to attribute duplication

## Validating data and data validation report

Clicking on the button with the check mark runs the data validation process. The ‘Attributes’ pane is collapsed to free up space on the screen to display the data validation report. The ‘Attributes’ pane can be expanded again if needed by clicking on it. The data validation report corresponding with each data file is displayed on a separate tab on the lower section of the user interface as shown in Figure 20.

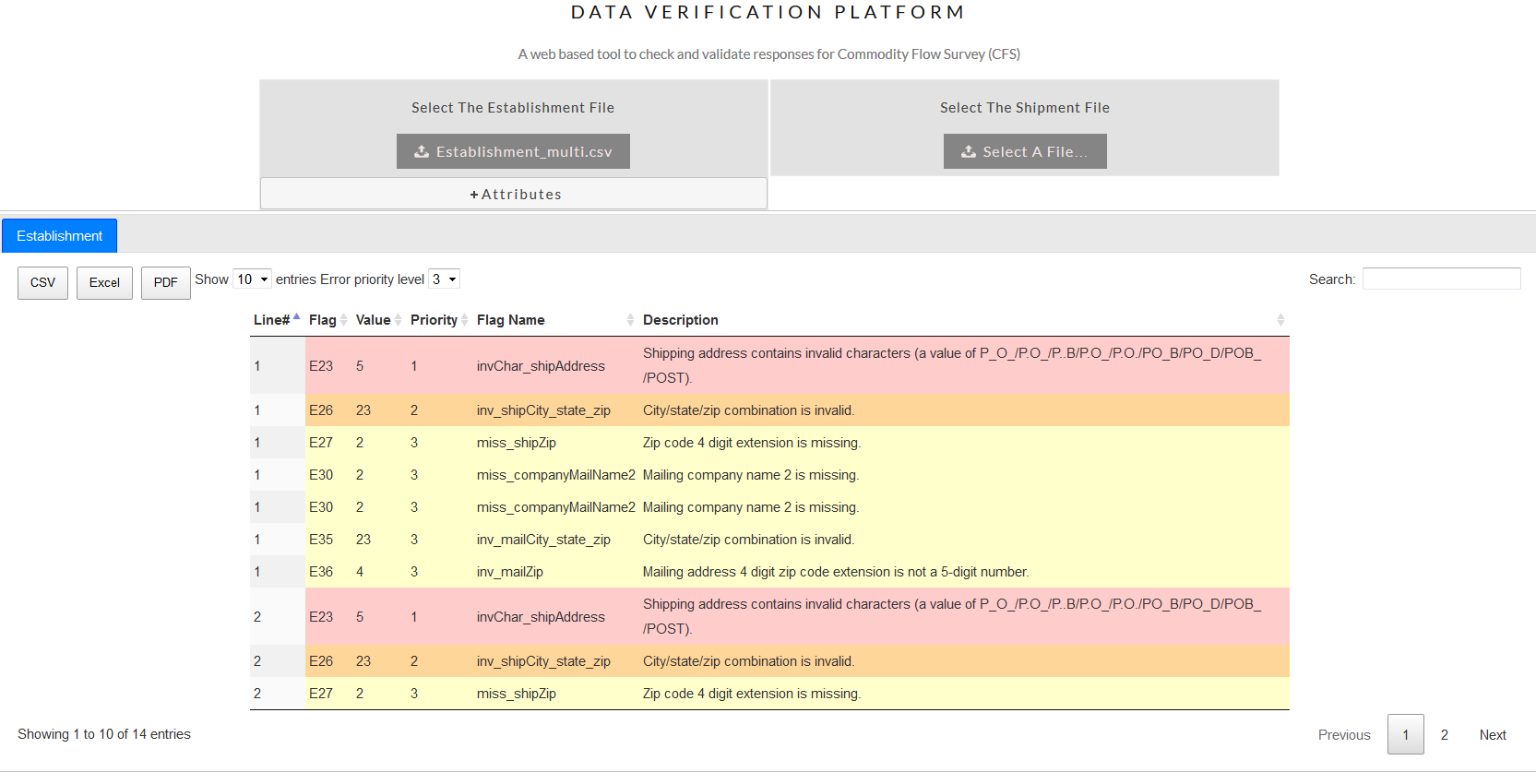


Figure 20 data validation results screen

Validation reports are presented in a tabular form similar to the traditional spreadsheet software (i.e. Excel). Rows in the table can be sorted based on the values in any of the columns by clicking on the (up and down) arrow shaped icons on the column headings. Values in the first column on the report present the line number, which is the line in the user submitted data file in which the error is detected. The second column is the error flag number, showing the number associated with the detected error flag. Multiple error flags can be displayed for each line in the user submitted data file depending on the number of errors detected. The third and fourth columns displays the error flag value and the priority associated with the error flag. Error flag name and description are displayed in the next columns. All details associated with error flags (i.e. name, number, value, description, and priority) can configured in ‘flags.js’ configuration filed discussed in section ‎5.1.4.

Another feature in data validation report tables is the ability to export the report in CSV, Excel, or PDF file format by clicking on the corresponding button at the top left of the table. Aside from the export feature, rows can be selected and copy-pasted into other software such as MS Excel or Word.

The number of rows (records) displayed in each page can also be configured using the combo box next to ‘show’ label. Errors can also be filtered by their priority using the ‘Error Priority Level’ combo box. The search box on the top right corner allows searching freely in all columns of the error report. Navigation between the report pages can be performed using ‘previous’, ‘next’ or the numbered buttons on the bottom right of the report.

# References

Di Zio, M., Fursova, N., & Gießing, S. (2016). *Methodology for data avalidation* (p. 76).

Dietrich, D., Poikola, A., Villum, C., Gray, J., Tait, J., Denis, J., … Goëta, S. (2010). Open Data Handbook. Retrieved from http://opendatahandbook.org

European Data Portal. (2018). Retrieved July 4, 2018, from https://www.europeandataportal.eu/elearning/en/#/id/co-01

Ferraggine, V. E., Doorn, J. H., & Rivero, L. C. (Eds.). (2009). *Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends*. https://doi.org/10.4018/978-1-60566-242-8

Janssen, D., & Janssen, C. (n.d.). What is Data Validation? Retrieved February 16, 2018, from Techopedia.com website: https://www.techopedia.com/definition/10283/data-validation

McCallum, E. (2013). *Bad Data Handbook*. o’Reilly.

Open data. (2018, February 20). Retrieved from https://en.wikipedia.org/w/index.php?title=Open\_data&oldid=826742486

U.S. Geological Survey. (2018). Retrieved July 4, 2018, from USGS Data Management website: https://doi.org/10.5066/F7MW2G15.

Wikipedia contributors. (2017). Data validation. In *Wikipedia*. Retrieved from https://en.wikipedia.org/w/index.php?title=Data\_validation&oldid=811833945