

# Automatic Data Standardization

April 18<sup>th</sup>, 2024

**Title** Automatic Data Standardization – User Facing Documentation

**Version** 0.4.7

**Description** This user facing documentation for the automatic data standardization package is meant to help familiarize oneself with how to navigate the GUI's, make function calls, and what specific outputs to expect when either modifying metadata or standardizing a file.

**Depends** R ( $\geq 4.0.0$ )

**Built On** R 4.3.2

**Encoding** UTF-8

**Language** en-CA

**Author** Cole Chuchmach [aut/cre]  
Barret Monchka [aut/ctb]

**Maintainer** Cole Chuchmach

**Date/Publication** 2024-04-18 13:54:10 CST

**R topics documented:**

Installing and Loading the Data Standardization Package	<a href="#"><u>3</u></a>
data_standardization_script.R	<a href="#"><u>4</u></a>
create_standardizing_options_lookup()	<a href="#"><u>7</u></a>
metadata_ui.R	<a href="#"><u>9</u></a>
Home	<a href="#"><u>10</u></a>
View Standardization Rules	<a href="#"><u>10</u></a>
Add and Update Databases	<a href="#"><u>11</u></a>
Enabled and Disable Databases	<a href="#"><u>13</u></a>
Add Source Fields	<a href="#"><u>13</u></a>
Update and Delete Source Fields	<a href="#"><u>15</u></a>
Add Compound Format	<a href="#"><u>16</u></a>
Update Compound Format	<a href="#"><u>19</u></a>
Add to Categorical Field	<a href="#"><u>20</u></a>
Update and Delete Categorical Field Values	<a href="#"><u>22</u></a>
Add to Record Priority	<a href="#"><u>23</u></a>
Update and Delete Record Priority Values	<a href="#"><u>25</u></a>
Add Categorical Values	<a href="#"><u>26</u></a>
Add Numeric Formats	<a href="#"><u>27</u></a>
Update Numeric Formats	<a href="#"><u>28</u></a>
Add Destination Field	<a href="#"><u>29</u></a>
Add Standardizing Module	<a href="#"><u>30</u></a>
data_standardization_ui.R	<a href="#"><u>32</u></a>
Upload Source Data File	<a href="#"><u>32</u></a>
Given Name and Surname Fields Flag Options	<a href="#"><u>33</u></a>
Location Fields Flag Options	<a href="#"><u>34</u></a>
Sex Flag Options	<a href="#"><u>34</u></a>
Output Flag Options	<a href="#"><u>35</u></a>
Chunked Reading Flag Options	<a href="#"><u>35</u></a>
Standardize Data	<a href="#"><u>36</u></a>

## Installing and Loading the Data Standardization Package

### Details

The data standardization package used to process and clean source data is hosted on and can be directly installed from GitHub. To do so, follow the steps as described:

### Step 1

Install and load the **devtools** library in **R** such that you can gain access to the **install\_github()** function:

```
# install.packages("devtools")
library(devtools)
```

### Step 2

Once the **devtools** package has been installed and loaded, use the **install\_github()** function to install the package from GitHub.

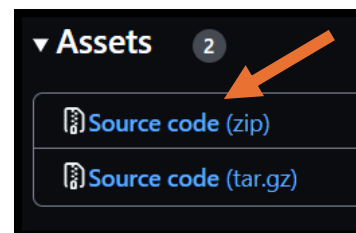
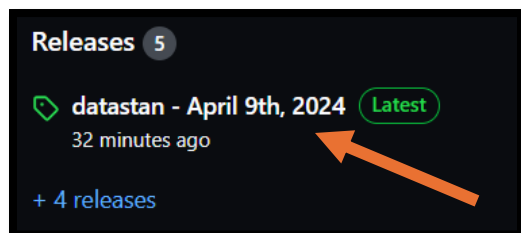
During installation, you will need to install or update other **R Libraries** that are used by the data standardization package.

Once all libraries have been successfully installed or updated, the data standardization package should successfully appear in your list of packages and can be enabled for use. Giving you access to all scripts and applications.

```
devtools::install_github("CHIMB/datastan")
```

### Step 3

If installing the **datastan** package using **install\_github()** does not work, you may also install the package locally, by first selecting the most recent release of the package and downloading the source zip file, then using **install\_local()** in **R Studio**:



```
path_to_pkg <- file.choose() # Select the unzipped package you downloaded from GitHub.
devtools::install_local(path_to_pkg)
```

---

## data\_standardization\_script.R

---

### Description

data\_standardization\_script.R is the script that will take in a unclean data file, pre-processing it and outputting a file that is cleaned and standardized. The script contains **four** main functions that handle the preprocessing and error handling.

### Details

The data standardization script is paired with the **metadata.sqlite** file to pull the standardization information for fields and requires a valid entry in the metadata database to be enabled.

To standardize a dataset programmatically, call the **standardize\_data()** function and supply four input parameters, one for the **file path**, which what the function will be reading from in chunks. One for the **dataset code** which is used in a query to determine which active dataset to choose from, an error will be thrown here if there is no active dataset enabled when passed the code. One for a lookup table consisting of flag values and their corresponding flag codes, this will determine how the flag standardizing script will further process certain fields like names, addresses, and output format. One for the path of the **output folder**, this is where the clean data file will be stored. Lastly one for the **metadata file** which is the sqlite file that contains all the users dataset information.

Additionally, the **standardize\_data()** function will return a data frame of the cleaned dataset so long as it is not too large for memory, in code, a data frame will be returned from the function if the size of the clean file is < **1GB**, otherwise an empty data frame is returned. The user may always open the clean file as a data frame afterwards, but the return check is meant to help with memory allocation.

Lastly, the error handling function working alongside these functions is to serve as a helpful tool for informing the user on what specific checks went wrong, cluing the user to either make changes to the metadata following incorrect metadata information such as the input file missing a header, or an incorrect number of columns.

### Arguments

input_file_path	A path to a file.
input_dataset_code	A dataset_code, used in the query statement within the function which will get the desired dataset code.
	File names meant to be pre-processed should contain a prefix containing the dataset code kept in the metadata, separated by something like a hyphen, underscore, etc.

input_flags	A lookup table containing manually set or default values used by the flag script which will use the values of the flags to determine how to pre-process and output the health data.
output_folder	An output folder that the successfully cleaned data ready for linkage will be placed in once pre-processing is complete.
standardization_rules_metadata	A path to the desired metadata file.

## Flag Values

Input flags are constructed using a lookup table created by the user, the available codes and associated values are as follows:

flag_codes	flag_options	Flag Codes and Options for Names:
convert_name_case	upper lower original	<b>Convert Case:</b> Convert names in the dataset to upper-case, lower-case, or keep it's original case.
compress_name_whitespace	yes no	<b>Compress Whitespace:</b> Keep whitespace between given names or remove it.
remove_name_punctuation	yes no	<b>Remove Punctuation:</b> Remove any kind of symbols from the name such as slashes, periods, commas, apostrophes, hyphens, etc.
list_all_curr_given_names	yes no	<b>All Given Names:</b> In an additional column, list all primary, and secondary given names together.
list_all_curr_surnames	yes no	<b>All Surnames:</b> In an additional column, list all primary, and secondary surnames together.
list_all_curr_names	yes no	<b>All Current Names:</b> In an additional column, list all primary, secondary given names and surnames together.
convert_name_to_ascii	yes no	<b>Convert to ASCII:</b> Convert name to ascii and remove any accents or diacritics of a person's name.

flag_codes	flag_options	Flag Codes and Options for Sex:
impute_sex	yes no	<b>Impute Sex:</b> Impute missing sex records?
impute_sex_type	default custom internal	<b>Impute Sex Type:</b> Impute sex using the <b>R gender</b> package, using the U.S. Social Security Administration 1932-2012 method. Or use a custom file in conjunction with the <b>chosen_sex_file</b> parameter, or impute <b>sex</b> internally using information from the file?
chosen_sex_file	<file_path>	<b>Chosen Sex File:</b> The custom file to impute sex, should have at least two columns, consisting of one column titled <b>primary_given_name</b> , with the cooresponding sexes under the name <b>sex</b> .

flag_codes	flag_options	Flag Codes and Options for Locations Fields:
convert_location_case	upper lower original	<b>Convert Case:</b> Convert addresses in the dataset to upper-case, lower-case, or keep it's original case.
compress_location_whitespace	yes no	<b>Compress Whitespace:</b> Keep whitespace between given locations or remove it.
remove_location_punctuation	yes no	<b>Remove Punctuation:</b> Remove any symbols, accents, or punctuation from a location string, or keep it as is.
convert_location_to_ascii	yes no	<b>Convert to ASCII:</b> Convert locations to ascii and remove any accents or diacritics.
extract_postal_code	yes no	<b>Extract Postal Code:</b> Some fields may have a persons postal code concatenated alongside things like addresses, cities, and countries, selecting this option will attempt to extract postal codes and append them to a new column called "alt_postal_code".

flag_codes	flag_options	Flag Codes and Options for Standardized Output and Processing:
file_output	csv rdata sqlite xlsx	<b>File Output:</b> Choose whether to output the standardized dataset as a <i>csv</i> , <i>rds</i> , or <i>sqlite</i> .
output_non_linkage_data	yes no	<b>Output Health and Program Data:</b> Remove any identifying information from the standardized dataset, keeping only the non-linkage variables and putting it all in a separate file.
chunk_size	10k - 1M	<b>Chunk Size:</b> This will be the row size of each chunk read and processed when handling a source file.
debug_mode	off on	<b>Debug Mode:</b> If enabled, additional messages will print to the console about query returns, timings, and standardized values.
max_file_size_output	Integer > 1	<b>Max File Size Output:</b> What is the max file size that a file can be before a data frame isn't returned? Input is an integer representing the max number of <b>megabytes</b> .
read_mode	path cmd	<b>Read Mode:</b> Use the default reading mode for functions like <b>fread()</b> which may perform a bit quicker, or use <b>shell commands</b> to lessen the workload on memory.

## Example

```
> source_file <- file.choose()
> dataset_code <- "fvotes"
> flags <- create_standardizing_options_lookup(convert_name_case = "upper", max_file_size_output = 100)
> output_folder <- choose.dir()
> metadata <- file.choose()
> df <- standardize_data(source_file, dataset_code, flags, output_folder, metadata)
[1] "Rows Read: 30300"
[1] "Can't send notification without a UI."
[1] "Finished Reading!"
[1] "Time From Inside the Chunk Function:"
  user  system elapsed
 3.75   0.07   4.36
>
```

---

**create\_standardizing\_options\_lookup()**

---

**Description**

Using this function the user can supply any number of flag options they want (not all are required, missing entries will be given a default value) which will be used as additional standardization rules.

A lookup table consisting of chosen flags and the assigned default options for bad inputs or undefined choices is returned to the user.

**Arguments**

convert_name_case	Convert the capitalization of a person's name. <b>(Options: "upper", "lower", "default")</b>
convert_name_to_ascii	Remove diacritics of a person's name. <b>(Options: "yes", "no")</b>
remove_name_punctuation	Remove any symbols or punctuation from a person's name. <b>(Options: "yes", "no")</b>
compress_name_whitespace	Replace name white space with an empty string symbol. <b>(Options: "yes", "no")</b>
list_all_curr_given_names	Combine all given names of a person into an additional column. <b>(Options: "yes", "no")</b>
list_all_curr_surnames	Combine all surnames of a person into an additional column. <b>(Options: "yes", "no")</b>
list_all_curr_names	Combine all names of a person into an additional column. <b>(Options: "yes", "no")</b>
impute_sex	Impute missing values in sex fields. <b>(Options: "yes", "no")</b>
impute_sex_type	How should the sex imputation take place? <b>(Options: "default" - Must have the gender and genderdata packages installed, "custom" - Must be a .csv file with two columns [primary_given_name] &amp; [sex], "internal" - Use sex values from the source data set)</b>
chosen_sex_file	If the custom type is chosen, supply an input file.
compress_location_whitespace	Replace location-based fields white space with an empty string symbol. <b>(Options: "yes", "no")</b>
remove_location_punctuation	Remove any symbols or punctuation from location-based fields. <b>(Options: "yes", "no")</b>
convert_location_case	Convert the capitalization of a location-based field. <b>(Options: "upper", "lower", "default")</b>

convert_location_to_ascii	Remove diacritics of a location based field. (Options: "yes", "no")
extract_postal_code	Attempt to extract postal codes from location based fields and place an additional column. (Options: "yes", "no")
file_output	What file output format is desired. (Options: "csv", "rds", "sqlite")
output_non_linkage_data	Output non-linkage fields in a separate file? (Options: "yes", "no")
chunk_size	How many rows should be read at a time when reading the source file in chunks? (Options: 10000-1000000)
debug_mode	Print additional information to the console in case of potential bugs? (Options: "on", "off")
max_file_size_output	What is the max file size that a data frame can be before it isn't returned? (Options: integer in Mega-bytes)
read_mode	Read the input file using normal file reading methods ( <i>path</i> ) or use shell commands to lessen the load on memory ( <i>cmd</i> ).

### Example

```
flags <- create_standardizing_options_lookup(convert_name_case = "upper",
impute_sex = "yes", chunk_size = 15000, max_file_size_output = 200)
```

### Important Note

When imputing sex using the default values, the **gender** and **genderdata** packages have been known to have issues installing directly from R Studio. To allow for this type of sex imputation, before standardizing takes place there is an attempt to load those two packages, and on fail will encourage the user to either install those packages locally (*via the gender and genderdata GitHub*) or select an alternative gender imputation method.



---

## metadata\_ui.R

---

### Description

The metadata GUI is to be used as an easy and accessible way to make changes to source dataset standardization rules by having queries run INSERT, UPDATE and DELETE queries on the backend while the user only needs to worry about the front end.

The metadata UI uses and make changes to the same sqlite file that is to be used with the **standardize\_data()** function in the data standardization script as information such as source field order, standardizing modules, categorical values, and destination fields are pulled from the metadata to determine how to specifically handle the source fields of interest from the source file.

You can run this app by running **startMetadataUI()** and providing a file path to an existing SQLite metadata file. If one does not exist, you may use the function **create\_new\_metadata(file\_name, output\_folder)** to create and pre-populate a new metadata file with some basic information.

### Details

The available specifications/pages are:

- Home
- View Standardization Rules
- Add or Update Databases
- Enable or Disable Databases
- Add Source Fields
- Update or Delete Source Fields
- Add Compound Formats
- Update Compound Formats
- Add to Categorical Fields
- Update or Delete Categorical Fields
- Add to Record Priority Fields
- Update or Delete Record Priority Fields
- Add Categorical Values
- Add Numeric Formats
- Update Numeric Formats
- Create New Destination Field
- Create New Standardizing Module

---

## Home

---

### Description

The home page provides users with a descriptive summary of what each page may allow the user to view, add, update or delete and allows them to transition between pages without any hassle.

Throughout the entirety of the time in the program, the **blue “Home” button** at the top will always return the user to the home page so that they may transition to another page.

---

## View Standardization Rules

---

### Description

The view standardization rules page allows the user to view the metadata in its entirety.

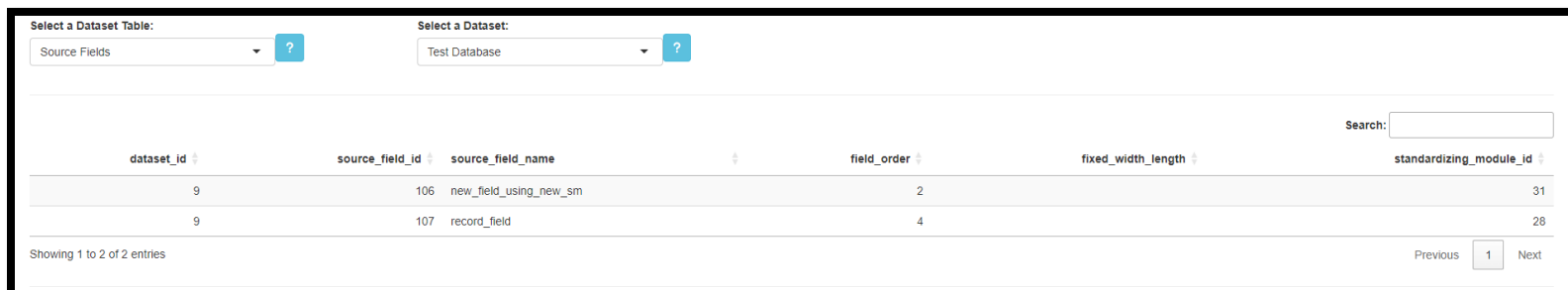
From the two drop downs, the user may select a dataset table from the left-most select input, this includes information such as **All Datasets, Source Fields, Compound Formats, Standardizing Modules**, etc.

For some tables, the user may also need to select an option from the right-most select input, which includes every currently created dataset in the metadata, for tables such as **Source Fields, Categorical Fields, Record Priority**, etc., then a dataset selection must be used to obtain results.

Once a selection has been made, a **data table** appears below, containing 10 results per page, with the option to change pages if a table has more than 10 results. Provided is a search bar to search for specific entries, and each column can be used to sort the results.

### Example

Consider wanting to view the source fields of a database you created, which you called *Test Database*, you would select the table **Source Fields** and then under the dataset selection, you would choose the option **Test Database**, you would then get an output like:



The screenshot shows the 'View Standardization Rules' interface. At the top, there are two dropdown menus: 'Select a Dataset Table:' with 'Source Fields' selected, and 'Select a Dataset:' with 'Test Database' selected. Below these is a table with 6 columns: dataset\_id, source\_field\_id, source\_field\_name, field\_order, fixed\_width\_length, and standardizing\_module\_id. The table contains 2 entries. At the bottom right, there is a search bar and pagination controls showing 'Previous', '1', and 'Next'.

dataset_id	source_field_id	source_field_name	field_order	fixed_width_length	standardizing_module_id
9	106	new_field_using_new_sm	2		31
9	107	record_field	4		28

Showing 1 to 2 of 2 entries

---

## Add and Update Databases

---

### Description

The add or update databases page allows users to create new empty databases that can have source fields added to it, or they may select an existing database and modify any pre-existing information about it.

### Details

There are **four inputs** that the user must fill before either adding or updating.

The first input field is the **dataset code**, this code is used when reading files to determine which dataset entry to look for so that the correct standardization rules can be applied, this should be the prefix of the beginning input/source file that the user would use when standardizing. A dataset code is unique and with databases that use the same code, only one may be enabled at a time.

The second input field is the **dataset name**, this serves as a longer descriptive name and the user should put versioning information here to help identify the database easier when making future or current changes to any standardizing rules.

The third input field is a select input to whether or not the source file has a **header row**, this helps the standardizing script in knowing whether or not to read the first line.

The last input field is the input **file format**, the input types are **.csv**, **.txt**, **.sas7bdat** and **fixed-width files**, however it should be noted that fixed-width files can also be of type **.txt**, yet differ due to the fact that each field has a set character count it must abide by, for databases of this format, extra information is required when adding source fields to ensure reading is successful.

### Example – Adding a New Database

Consider we have a new file we want to standardize, called **adsb\_employees\_2024.csv**, and after inspecting the file, we see that it has a row at the top containing the column names. To add this new database, we'd enter the following information into the fields and then press the **“Create Database”** button to submit the field information, we should receive a success notification near the top of the program afterwards:

Select a Row Below To update OR Create a New Entry Below

dataset_code	dataset_name	has_header	file_extension_code
No data available in table			

Showing 0 to 0 of 0 entries Previous Next

Dataset Code:  ?

Dataset Name:  ?

Does File Have a Header Row?  ?

File Format:  ?

[Create Database](#)

### Example – Updating an Existing Database

Consider that this very real business database changes during the year to a new format like **.sas7bat**, we'd select from the list of existing databases, and choose the database with the code **adsb**. This will prepopulate the update inputs, what we'll do is switch the header row input to no since the data standardization for reading **SAS** files will take the first as header, then we may switch the format to **.sas7bdat** instead:

Select a Row Below To update OR Create a New Entry Below

dataset_code	dataset_name	has_header	file_extension_code
adsb	Automatic Data Standardization Business Employee Dataset - 2024	1	csv

Showing 1 to 1 of 1 entries Previous 1 Next

[Manage Newly Created Database](#) ?

Update Dataset Code:  ?

Update Dataset Name:  ?

Does File Have a Header Row?  ?

Updated File Format:  ?

[Update Database](#)

---

## Enable and Disable Databases

---

### Description

The enable or disable databases page allows to either disable currently enabled databases or enable currently disabled databases.

### Details

When a database is enabled, that version of the database, including the standardization rules will be used during the automatic data standardization, you may switch an enabled database out for a disabled one by first disabling the current version and then re-enable an old version.

When a database is disabled, that version of the database is not considered when going through the automatic data standardization process.

You may still modify and change the source fields of a disabled dataset, but they would not be considered unless that database is enabled. Furthermore, no two datasets sharing the same dataset code may be enabled at a time, one must be disabled before the other is enabled.

### Example

To enable and disable databases, you must select from the data table to either enable or disable and press the large button below the table to confirm it.

Select a Dataset to Disable:		Select a Dataset to Enable:	
dataset_code	dataset_name	dataset_code	dataset_name
adsh	Automatic Data Standardization Business Employee Dataset - 2024	No data available in table	

Showing 1 to 1 of 1 entries      Previous      1      Next      Showing 0 to 0 of 0 entries      Previous      Next

**Disable Database**

---

## Add Source Fields

---

### Description

The add source fields page allows you to enter what columns to standardize and choose what standardization rules those columns/fields will specifically follow.

### Details

From the list of all databases, you may select the database you want to add source fields to, this will bring up another table beside the original containing the current source fields that are being standardized in the data set.

The GUI provides you with **four inputs**, one is the **source field name** of the column/field being standardized, another is the **field order** of the column which is information about what order that field appears in, another is the **fixed width length** which is the character width for fixed width files (*this field does not a value entered if the file is not fixed width*), and lastly, the user must select what type of standardizing module the field being processed goes through.

When selecting a standardizing module, additional information is pasted below it which explains what types of fields it would expect to process. Note that some standardizing modules like compound formats, categorical fields, numerical dates, and record priorities require additional information for standardization.

### Example

Consider that our file currently contains 4 source fields, **employee\_fav\_drink**, **employee\_name**, **employee\_fav\_snack**, and **employee\_gender**. When adding these source fields to standardize, we should only need to enter the fields of interest, but adding all fields is completely up to the user's discretion.

To add the name, we would set the source field name to be the column name, the field order set to **2** as it's the second column in the dataset, and the standardizing module we'd choose is treating it as a **primary\_given\_name**:

The screenshot shows the Metadata GUI interface with the following configuration:

- Source Field Name:** employee\_name
- Field Order:** 2
- Fixed Width Length:** (empty)
- Select Standardizing Module:** primary\_given\_name

Below the input fields, a description for the selected module is displayed: **primary\_given\_name:** The field containing a persons Primary Given name.

At the bottom, there is a blue button labeled "Enter Source Field".

Now to add gender, we'd set the source field name to be **employee\_gender**, and have the order set to **4** as it's the fourth column in the dataset, the standardizing module we'd select is gender, which acquires additional information. For our purpose, consider that the gender options in the dataset are **Ma**, **Fe**, and **Ot** for the available options. To enter this, into the metadata, we should enter that we are entering **3 categorical values** in this field, and then enter what the source value appears as in the dataset, and what standardized value it should map to. This would look like:

Source Field Name:  ?
Field Order:  ?
Fixed Width Length:  ?
Select Standardizing Module:  ?

gender: The field containing the gender of a person. This will require additional information in the form of categorical fields.

Number of Categorical Fields in the Dataset:  ?

Source Categorical Value 1:

Source Categorical Value 2:

Source Categorical Value 3:

Standardized Categorical Value 1:

Standardized Categorical Value 2:

Standardized Categorical Value 3:

Enter Source Field

After both fields are added, our current standardization rules should look like:

Current Dataset Source Fields:				
source_field_id	source_field_name	field_order	fixed_width_length	standardizing_module_name
1	employee_name	2		primary_given_name
2	employee_gender	4		gender

Showing 1 to 2 of 2 entries

Previous
1
Next

## Update and Delete Source Fields

### Description

The update or delete source fields page allows you to modify the information of any existing source field or delete the source field from the standardization rules so that the field will no longer be standardized the next time the standardizing script function is used.

### Details

From the list of all databases, you may select the database you want to update or delete the source fields of, this will bring up another table beside the original containing the current source fields that are able to be selected to update or delete.

This page is similar to the add page in that it has the same inputs as before.

## Example

Consider that our current database changes during the year, and the employee's name field changes to **employee\_surname**. We will select the original source field in the dataset where it said **employee\_name**, this will pre-populate the inputs with the current metadata information, and we may change both the name and standardizing module to use and then press the update source field button, this would become:

Source Field Name:  
employee\_surname ?

Field Order:  
2 ?

Fixed Width Length (if required):  
 ?

Select Standardizing Module:  
primary\_surname ?

primary\_surname: The field containing a persons Primary Surname.

Update Source Field

Delete Source Field

Afterwards, the current metadata information for this table would be:

Select a Source Field to Update:				
source_field_id	source_field_name	field_order	fixed_width_length	standardizing_module_name
1	employee_surname	2		primary_surname
2	employee_gender	4		gender

Showing 1 to 2 of 2 entries

Previous 1 Next

## Add Compound Format

### Description

The add compound format page allows you to create formats to identify fields that require data to split apart to obtain their data, this is common for fields like birthdates, acquisition dates, and in some cases, people's names.

### Details

A table is provided on the left-hand side of the screen which shows all current compound fields that have been added.

The right-hand side contains the input fields for adding the format information, selecting the split method, the number of separators or indexes in the format, and then finally the inputs based on split method.



## Example – Separator

Consider our database is expected to change in the future and we want to prepare, employee\_name will contain both first and last name, with a **comma** separating these names, this information is kept all in one field in the database.

In our compound format, we'll use general shortened terms to help identify what type of abstract format these employee names fall under, and we will enter a short description of what this compound format is in the context of our dataset, making sure to include separators.

We will then choose **Separators** as our split method, and since we're only splitting on a singular comma, then the number of separators is **one**.

Additional inputs will appear below based on the number of separators, and since we chose one separator, we are given two destination field inputs with a text box for what the separator symbol is, in our case, we would select **primary\_given\_name** and **primary\_surname** as the destination fields, with the [ , ] symbol in the separator input field. Before we hit submit, our format input and separator inputs would look like:

**Compound Format:**

**Choose Split Method:**

**Compound Format Description:**

**Number of Separators:**

**Destination Field 1:**

**Separator 1:**

**Destination Field 2:**

Afterwards, the table containing the compound formats should look like:

compound_field_format_id	compound_format	format_description
1	FN, LN	Employee compound name, consists of first and last name, separated by a single comma.

Showing 1 to 1 of 1 entries

Previous 1 Next

Separator Example Index Example

## Example – Indexes

Consider that our database is expected to change in the future, and we want to prepare, we will be getting an **employee\_birthdate** column that has birth day, month, and year all together.

In our compound format, we'll use general shortened terms to help identify what type of abstract format these employee names fall under, and we will enter a short description of what this compound format is in the context of our dataset, making sure to include what kind of split method we're using.

We will then choose **Indexes** as our split method, and since we're splitting birthdate into **three pieces**, we'll choose **3** as the number of indexes.

Additional inputs will appear below based on the number of indexes, and since we chose three indexes, we are given three destination field inputs with a numeric input beside each destination field, which will be the length of index that we record. Since birth day in the format has a length of **2** we enter a **2** into index 1, then another **2** for index 2 since month also has a length of **2**, and finally a **4** into index 3 since year has a length of **4**, then for the destination fields, we will enter **birth\_day**, **birth\_month** and **birth\_year** respectively:

**Compound Format:**  
DDMMYY  
YYY

**Choose Split Method:**  
Indexes

**Compound Format Description:**  
Employee compound birthdate, consists of birth day, month, and year, separated using indexing.

**Number of Indexes:**  
3

**Destination Field 1:**  
birth\_day

**Index 1:**  
2

**Destination Field 2:**  
birth\_month

**Index 2:**  
2

**Destination Field 3:**  
birth\_year

**Index 3:**  
4

Afterwards, the table containing the compound formats should look like:

compound_field_format_id	compound_format	format_description
1	FN, LN	Employee compound name, consists of first and last name, separated by a single comma.
2	DDMMYYYY	Employee compound birthdate, consists of birth day, month, and year, separated using indexing.

Showing 1 to 2 of 2 entries

Previous 1 Next

Separator Example Index Example

---

## Update Compound Format

---

### Description

The update compound format page allows you to update the formats used to identify fields that require data to split apart to obtain their data, this is common for fields like birthdates, acquisition dates, and in some cases, people's names.

### Details

A table is provided on the left-hand side of the screen which shows all selectable compound formats that are available to be updated, formats that appear here are used **zero** times by source fields.

The right-hand side contains the input fields for adding the format information, selecting the split method, the number of separators or indexes in the format, and then finally the inputs based on split method. The inputs are initially pre-populated with the metadata information that can be changed during the update.

### Example – Separator

Using the existing employee compound name, say that we got our information wrong and instead last name will come before the first name, in this case we can change the format to resemble the following and submit it to be updated:

The screenshot displays the 'Update Compound Format' interface with the following configuration:

- Compound Format:** A text box containing 'LN, FN' with a blue question mark icon to its right.
- Choose Split Method:** A dropdown menu set to 'Separators' with a blue question mark icon below it.
- Number of Separators:** A text box containing the value '1'.
- Compound Format Description:** A text box containing 'Employee compound name, consists of first and last name, separated by a single comma.' with a blue question mark icon to its right.
- Destination Field 1:** A dropdown menu set to 'primary\_surname'.
- Separator 1:** A text box containing a comma character ','.
- Destination Field 2:** A dropdown menu set to 'primary\_given\_name'.

## Example – Indexes

Using the existing birthdate compound date, say that we also got this information wrong and instead month comes first, but is instead has a length of **3** because it's an abbreviation of the month name. We can change the format to resemble what our new field will look like:

<b>Compound Format:</b> <div>MMMD YYYY</div>	<b>Choose Split Method:</b> <div>Indexes</div>	<b>Compound Format Description:</b> <div>Employee compound birthdate, consists of birth day, month, and year, separated using indexing.</div>
	<b>Number of Indexes:</b> <div>3</div>	
<b>Destination Field 1:</b> <div>birth_month</div>		<b>Index 1:</b> <div>3</div>
<b>Destination Field 2:</b> <div>birth_day</div>		<b>Index 2:</b> <div>2</div>
<b>Destination Field 3:</b> <div>birth_year</div>		<b>Index 3:</b> <div>4</div>

## Add to Categorical Fields

### Description

The add to categorical fields page allows users to add additional values that may appear in categorical fields such as gender, provinces, months, etc., so that the values can map to a standardized output common to all clean datasets.

### Details

The user may first select a dataset that contains the categorical field that they want to add to, a second table appears which contains all categorical fields of the dataset, the user may select which categorical field they want to add to from this table.

After this, another table will appear below alongside some input fields. This table shows all current categorical values in this field and allows the user to add more to this.

## Example

Consider that the gender column in our database gets additional choices that an employee can choose from, in this case suppose the dataset adds the option for non-binary and gender-neutral choices.

The number of categorical fields we will be adding is thus **2** so two text fields and an accompanying select input field will appear for what the value should map to.

The user may select the standardized categorical value that these two inputs map to as **X** which also contains the **Ot** value, otherwise, we may transition to the **Add Categorical Values** page to add options for this and use these as selections (*see Add Categorical Values for instructions*).

Before we add, the inputs should look like:

**Number of Categorical Fields to Add:**

?

**Categorical Value 1:**

**Standardized Categorical Value 1:**

NB
▼

**Categorical Value 2:**

**Standardized Categorical Value 2:**

GN
▼

Afterwards, our table will resemble:

source_field_id	source_value	standardized_value
2	Fe	F
2	Gender-neutral	GN
2	Ma	M
2	Non-binary	NB
2	Ot	X

Showing 1 to 5 of 5 entries

Previous
1
Next

---

## Update and Delete Categorical Fields

---

### Description

The update or delete categorical fields page allows users to modify the values that may appear in categorical fields such as gender, provinces, months, etc., so that the updated values can map to a standardized output common to all clean datasets.

### Details

The user may first select a dataset that contains the categorical field that they want to add to, a second table appears which contains all categorical values in that dataset, the user may select a categorical value from the dataset that they would like to update.

After this, an input appears below and pre-populate the input field with the metadata information, the user may then change either the source categorical value or the standardized categorical value it maps to is.

### Example

Consider that the gender column in our dataset changes the short-hand name of **Ma**, **Fe**, and **Ot** to its full name counterpart like the other categorical values we just previously added, we may then select each source categorical value and update the name that it checks for in the dataset. This would look like:

Source Categorical Value:	Select Standardized Categorical Value:
<input type="text" value="Male"/>	<input type="text" value="M"/>
<input type="text" value="Female"/>	<input type="text" value="F"/>
<input type="text" value="Other"/>	<input type="text" value="X"/>

After updating the categorical field values, our table should look like:

### Select The Categorical Field To Update:

source_field_id	source_field_name	source_value	standardized_value
2	employee_gender	Male	M
2	employee_gender	Female	F
2	employee_gender	Other	X
2	employee_gender	Non-binary	NB
2	employee_gender	Gender-neutral	GN

Showing 1 to 5 of 5 entries

Previous **1** Next

## Add to Record Priority Fields

### Description

The add to record priority fields page allows users to add additional source values and their corresponding priority values to a priority field such that tie breaks could be handled during the linkage process.

Record priority works by using the priority values to determine which record is more valid when dealing with duplicate records, a record priority of **1** has more priority than a record with the priority **3**, meaning that **1** is the highest priority, and as that number increases, the priority of that record will become less and less.

### Details

The user may first select a dataset that contains the record priority field that they want to add to, a second table appears which contains all record priority fields of the dataset, the user may select which record priority field they want to add to from this table.

After this, another table will appear below alongside some input fields. This table shows all current record priority source values and their priority mappings in this field and allows the user to add more to this using the input fields to the right of the table.

## Example

Firstly, let's say that during the year we'd like to increase the linkage rate of our employee dataset with another unnamed dataset, to do so we'd like to add a field with record priority to help with de-duplication. We can first transition to the **Add Source Fields** page and add the new field **employee\_work\_rank** that measures an employee's rank in the company.

Thus, we'll fill in the input fields for the previous source fields, selecting **record\_priority** as the standardizing module this time. And similar to categorical fields, we will choose how many values are in this dataset. For this example, say we have two values **A** and **B**, which map to the priorities **1** and **2** exclusively, so we'll fill in those values and submit the new field.

The screenshot shows the 'Add Source Fields' form. At the top, there are four input fields: 'Source Field Name' (employee\_work\_rank), 'Field Order' (5), 'Fixed Width Length' (empty), and 'Select Standardizing Module' (record\_priority). Below these is a text box explaining 'record\_priority'. Further down, there is a section for 'Number of Record Priority Fields in the Dataset' with a value of 2. Below this, there are two rows of input fields for mapping source field values to record priorities. The first row shows 'Source Field Value 1' as 'A' and 'Record Priority 1' as '1'. The second row shows 'Source Field Value 2' as 'B' and 'Record Priority 2' as '2'.

Now say we missed entering a source field value earlier, and there is a value of **C** that would map to the priority **3**. To add this, we would transition to the **Add to Record Priority Fields** page and select the dataset and source record priority field we would be adding to.

Like adding to categorical fields, we may enter how many extra record priorities we're entering, which will give us that many source input fields and their mapping priorities. In this case, we are only adding **1**, so for our source field value and record priority we will write a **C** and **3** into the inputs respectively. Before submitting the record priorities, our input should look like:

The screenshot shows the 'Add to Record Priority Fields' form. At the top, there is a section for 'Number of Extra Record Priorities' with a value of 1. Below this, there are two input fields: 'Source Field Value 1' (C) and 'Record Priority 1' (3).



Afterwards, our record priority table to the left of the input fields should resemble:

Current Record Priorities of Selected Source Field:		
source_field_id	source_value	priority
3	A	1
3	B	2
3	C	3

Showing 1 to 3 of 3 entries

Previous 1 Next

## Update and Delete Record Priority Fields

### Description

The update or delete record priority fields page allows users to modify the source and priority values of record priorities in case certain values in the dataset change or alter priority over time.

### Details

The user may first select a dataset that contains the record priority field that they want to add to, a second table appears which contains all record priority source values, along with their priority mappings in that dataset, the user may select a record priority value from the dataset that they would like to update.

After this, an input appears below and pre-populate the input field with the metadata information, the user may then change either the source record priority field value or the priority ranking that it maps to.

### Example

Consider that during the year, the values of the dataset change, and instead of using **A**, **B**, and **C**, they are replaced by the values **X**, **Y** and **Z**. Like a categorical field value, we will select the values in this dataset and modify the source field values. Each update should look like:

Source Field Value:	Record Priority:
X	1

Source Field Value:	Record Priority:
Y	2

Source Field Value:	Record Priority:
<input type="text" value="Z"/>	<input type="text" value="3"/>

After updating each record priority value, our table should resemble:

Select The Record Priority To Update:				
source_field_id	source_field_name	source_value	priority	
3	employee_work_rank	X	1	
3	employee_work_rank	Y	2	
3	employee_work_rank	Z	3	

Showing 1 to 3 of 3 entries

Previous  Next

## Add Categorical Values

### Description

The add categorical values page allows users to add to a list of standardized values that source categorical values may map to when adding source fields, adding to existing categorical fields, or updating existing categorical fields.

### Details

The user will be presented with a list of standardized categorical values with a singular user input below the table which will allow the user to enter a new usable categorical value.

### Example

Consider that we want to add a categorical value that provinces can map to. If in the employee dataset, when we get a new field containing the provinces of each employee, we'd like the source values to map to a standardized value.

If we'd like **Manitoba** to map to **MB**, then we can prepare by creating a new categorical value of **MB**. Before submitting, our input would look like:

New Categorical Value:
<input type="text" value="MB"/>

After submitting, our table of all categorical values should resemble:

Current Standardized Categorical Values:	
standardized_value_id	standardized_value
1	M
2	F
3	X
4	NB
5	GN
6	MB

Showing 1 to 6 of 6 entries

Previous 1 Next

## Add Numeric Format

### Description

The add numeric format page allows you to create formats to identify fields that require handling dates kept in a numeric format by calculating the distance from an origin date.

### Details

A table is provided which contains all currently added numeric formats along with three available inputs that a user can add to.

The first input is a format label which is used to help make choosing what specific format to use on a date would be best, almost acting as a description of sorts. The second input is the origin date, this is used to calculate the date by using the source integer input and counting from that origin date. The last input is the units of measurement that a single integer acts as in the dataset, the user may select a unit of **1** being equal to a day, hour or second from the origin date.

### Example

Consider that during the year the dataset we are using is planning on adding a field which contains the date the record was captured on, titled **employee\_capture\_date**. This uses a SAS/Stata numeric date format which means it stores dates as an integer.

To add this format, we'd describe this label as **SAS/Stata** as both of those date time values will use this same format. Further, SAS date time origin begins on January 1<sup>st</sup>, 1960, so in the origin date input field, we may manually type in this date as **1960-01-01**, or we may click inside the field box and pick the date that way. Lastly, SAS's unit of measurement for numeric date times is kept as days, meaning a value is calculated as "*the number of days since Jan 1<sup>st</sup>, 1960*". Before submitting, our fields should look like:

Numeric Date Format Label:	Origin Date:	Unit of Measurement:
SAS/Stata ?	1960-01-01 ?	Days ?

Afterwards, the table of formats should resemble:

Current Numeric Date Formats:				
numeric_date_format_id	numeric_date_format_label	origin_date	units_label	
1	SAS/Stata	1960-01-01	Days	
Showing 1 to 1 of 1 entries				
			Previous	1 Next

## Update Numeric Format

### Description

The update numeric format page allows you to update the formats used to calculate a date when given a singular numeric value so long as the format is not currently being used by a source field.

### Details

A table is provided which shows all selectable compound formats that are available to be updated, formats that appear here are used **zero** times by source fields.

Selecting a format will pre-populate the three input fields with the information currently stored in the metadata and allow the user to make changes to any current information.

### Example

Consider that we were planning on using a unique format called the **Employee Capture Date Format** for our dataset, which calculates a capture date by taking the number of days since January 1<sup>st</sup>, 2000. But right before we get our dataset, we learn that it stores the dates as seconds from 2000, not days.

What we can do is select the employee date from the table, and update the input values as follows:

Select a Numeric Format to Update:					
numeric_date_format_id	numeric_date_format_label	origin_date	units_label	Number_Of_Usages	
1	SAS/Stata	1960-01-01	Days	0	
2	Employee Capture Date Format	2000-01-01	Days	0	
Showing 1 to 2 of 2 entries					
			Previous	1	Next
Numeric Date Format Label:	Origin Date:	Unit of Measurement:			
Employee Capture Date Format ?	2000-01-01 ?	Seconds ?			

Afterwards, the table of formats should resemble:

Select a Numeric Format to Update:				
numeric_date_format_id	numeric_date_format_label	origin_date	units_label	Number_Of_Usages
1	SAS/Stata	1960-01-01	Days	0
2	Employee Capture Date Format	2000-01-01	Seconds	0

Showing 1 to 2 of 2 entries

Previous 1 Next

## Add Destination Fields

### Description

The add destination fields page is mainly used for developer purposes, this page allows you to add a new field and standardized field name that specific source fields and formats can output to.

This is used in conjunction with the **Add Standardizing Modules** page as creating a custom module involves selecting a destination field to output to.

### Details

The user will be presented with a list of all destination fields currently usable in the metadata, additionally, there are two input fields below the table to add a destination field name for which the column name will standardize to, and a short description of what the destination field contains have standardization takes place.

### Example

Consider we want to create our own standardizing module for this dataset, we want to take the **employee\_favourite\_food** column and use an online test to see what animal resembles the employee the most, to begin writing this standardizing module, we'll first create a destination field by filling out the input fields as follows:

<b>Destination Field Name:</b> <input type="text" value="employee_animal_resemblance"/>	<b>Destination Field Description:</b> <input type="text" value="Based on employee's favourite food, what animal do they resemble the most"/>
--	---

Afterwards, our table of destination fields should resemble:

Current Destination Fields:	
destination_field_name	destination_field_description
acquisition_month	Month of capture
acquisition_day	Day of capture
record_priority	Priority of record (1 being the highest)
employee_animal_resemblance	Based on employee's favourite food, what animal do they resemble the most

Showing 21 to 24 of 24 entries

Previous 1 2 3 Next

---

## Add Standardizing Module

---

### Description

The add standardizing modules page is mainly used for developer purposes, this page allows you to add a new standardizing module that source fields can select as the module it will use for standardizing.

This is used in conjunction with the **Add Destination Fields** page as creating a custom module involves selecting a destination field to output to.

### Details

The user will be presented with a list of all current standardizing modules, along with four input fields.

First, is the **standardizing module name**, this name should be the post fix function name that it will call in the standardizing script after you've written the code you're planning on using in the module. Names should be concatenated with an underscore symbol.

Second, the **standardizing module description** you provide should be brief and be an overview of what exactly the standardizing module would do with the input values.

Third, the **destination field** is what the output field is to have its standardized field name become after it finishes being processed by the standardizing module.

Last, is whether this field should be included along with the non-linkage fields file.

### Example

Using the destination field we previously defined, consider that the code we would write in the standardizing script would be called using the function name **pre\_process\_fav\_food**.

The input fields should look like this before we submit, and it's important to note that when regarding the standardizing module name, that name will be **appended** as a postfix to the **pre\_process** prefix, so we will only need the end of the name.

<b>Standardizing Module Name:</b> <input type="text" value="fav_food"/> ?	<b>Standardizing Module Description</b> <input type="text" value="The field that will be used in a test to determine what animal resembles an employee."/> ?	<b>Select Destination Field:</b> <input type="text" value="employee_animal_resemblance"/> ?	<b>Include in Non-Linkage File?</b> <input type="text" value="No"/> ?
--	---	--	--

Afterwards, the table containing the standardizing modules will look like:

Current Standardizing Modules:			
standardizing_module_name	description	destination_field_name	output_program_data
acquisition_date	The field containing the compounded acquisition date of this record. This will require additional information in the form of a compound format.		0
acquisition_year	The field containing the acquisition year of the record on its own.	acquisition_year	0
acquisition_month	The field containing the acquisition month of the record on its own.	acquisition_month	0
acquisition_day	The field containing the acquisition day of the record on its own.	acquisition_day	0
compound_name	The field containing the compounded name of a person. This will require additional information in the form of a compound format.		0
numeric_date	The field containing the birthdate of a person or acquisition date of the record in a date-time format. This will require additional information in the form of a numeric format.		1
pass_through_field	The field that is used to perform linkages in a way where the values of the field do not need to be standardized.		1
record_priority	The field that is used to break ties, replaces values with priorities ranging from 1 -> N where 1 is the highest priority and N is lowest.	record_priority	0
fav_food	The field that will be used in a test to determine what animal resembles an employee.	employee_animal_resemblance	0
Showing 21 to 29 of 29 entries			
		Previous	1 2 3 Next

---

## data\_standardization\_ui.R

---

### Description

The data standardization GUI provides an easy interface for standardizing data using the metadata information. On a singular page the user can enter all the information for source file, output, and flag values.

To start this application, run the **startDataStandardizationUI()** function.

### Details

The available specifications/pages are:

- Upload Source Data File
- Given Name & Surname Fields Flag Options
- Location Fields Flag Options
- Sex Flag Options
- Output Flag Options
- Chunked Reading Flag Options
- Standardize Data

---

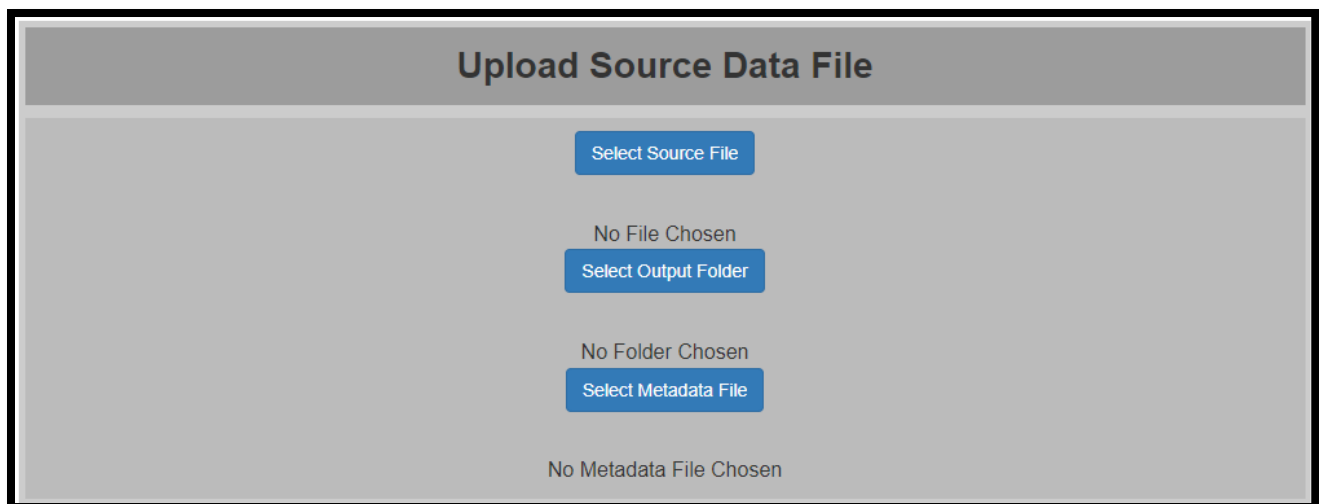
## Upload Source Data File

---

### Description

In this text block, the user is given two buttons to select an input file, output folder, and metadata file with labels below to provide user with the knowledge that the correct file was supplied.

### Details



The screenshot shows a web-based interface titled "Upload Source Data File". It features a light gray background with a dark gray header bar containing the title. Below the header, there are four blue buttons arranged vertically, each with a corresponding label above it: "Select Source File" (above "No File Chosen"), "Select Output Folder" (above "No Folder Chosen"), and "Select Metadata File" (above "No Metadata File Chosen"). The labels "No File Chosen", "No Folder Chosen", and "No Metadata File Chosen" are displayed in a smaller, lighter gray font.



## Given Name & Surname Fields Flag Options

### Description

In this text block, the user is given seven select input fields for flag options regarding people's names in the dataset, hovering over the [?] icon will provide additional information about what certain flags do and the expected outputs.

### Details

### Given Name & Surname Fields Flag Options

What Case Should Names Convert To?

Original Case

?

Should White Space Between Names Become Compressed?

No

?

Remove Punctuation From Names?

No

?

Convert Persons Name to ASCII?

No

?

Add Output Column Containing All Given Names of a Person?

No

?

Add Output Column Containing All Surnames of a Person?

No

?

Add Output Column Containing All Given Names & Surnames of a Person?

No

?

---

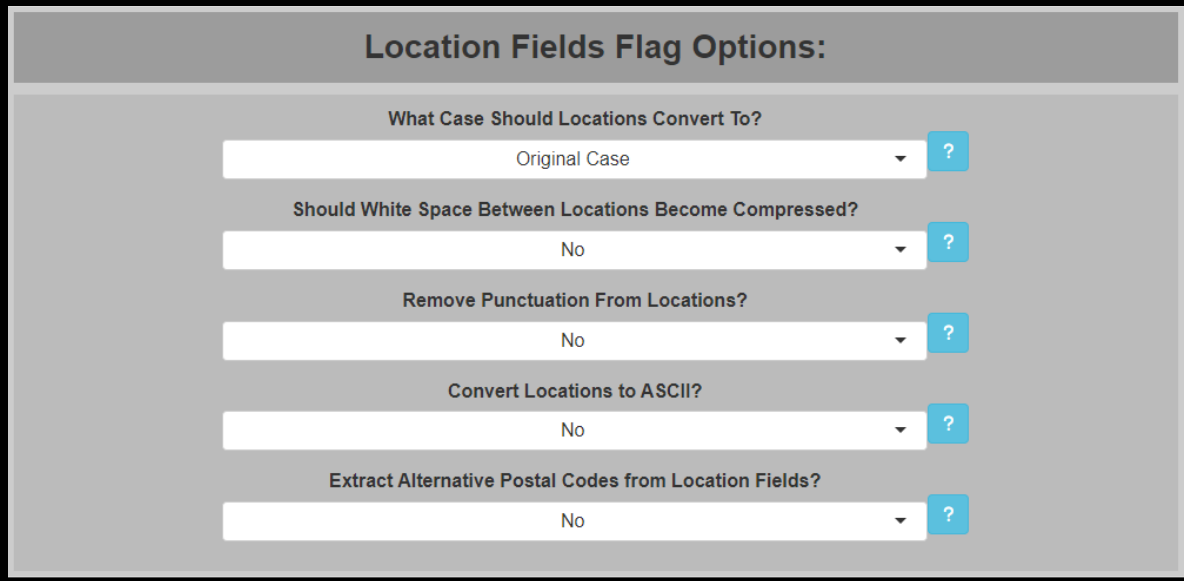
## Location Fields Flag Options

---

### Description

In this text block, the user is given five select input fields for flag options regarding address and location-based fields in the dataset, hovering over the [?] icon will provide additional information about what certain flags do and the expected outputs.

### Details



The screenshot shows a dialog box titled "Location Fields Flag Options:". It contains five rows, each with a question, a dropdown menu, and a blue question mark icon. The questions and their current values are: "What Case Should Locations Convert To?" (Original Case), "Should White Space Between Locations Become Compressed?" (No), "Remove Punctuation From Locations?" (No), "Convert Locations to ASCII?" (No), and "Extract Alternative Postal Codes from Location Fields?" (No).

Question	Value	Info Icon
What Case Should Locations Convert To?	Original Case	?
Should White Space Between Locations Become Compressed?	No	?
Remove Punctuation From Locations?	No	?
Convert Locations to ASCII?	No	?
Extract Alternative Postal Codes from Location Fields?	No	?

---

## Sex Flag Options

---

### Description

In this text block, the user is given two select input fields and an option for uploading a custom sex imputation file for flag options regarding the imputation of missing sex values in the dataset, hovering over the [?] icon will provide additional information about what certain flags do and the expected outputs.

### Details



The screenshot shows a dialog box titled "Sex Flag Options:". It contains two rows with questions, dropdown menus, and blue question mark icons. The questions and their current values are: "Impute Missing Sex?" (No) and "How Should Sex be Imputed?" (Custom File Imputation). Below the dropdowns is a blue button labeled "Select Sex Imputation File" and the text "No File Chosen".

Question	Value	Info Icon
Impute Missing Sex?	No	?
How Should Sex be Imputed?	Custom File Imputation	?

Select Sex Imputation File  
No File Chosen

---

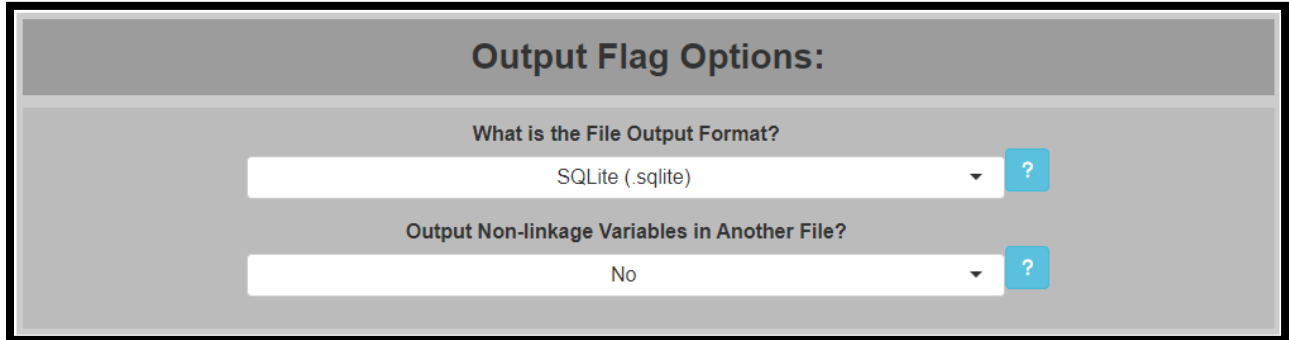
## Output Flag Options

---

### Description

In this text block, the user is given two select input fields flag options regarding the output of the clean dataset after processing has finished, hovering over the [?] icon will provide additional information about what certain flags do and the expected outputs.

### Details



The screenshot shows a dialog box titled "Output Flag Options:". It contains two questions, each with a dropdown menu and a help icon (a blue square with a white question mark). The first question is "What is the File Output Format?" with a dropdown menu showing "SQLite (.sqlite)". The second question is "Output Non-linkage Variables in Another File?" with a dropdown menu showing "No".

---

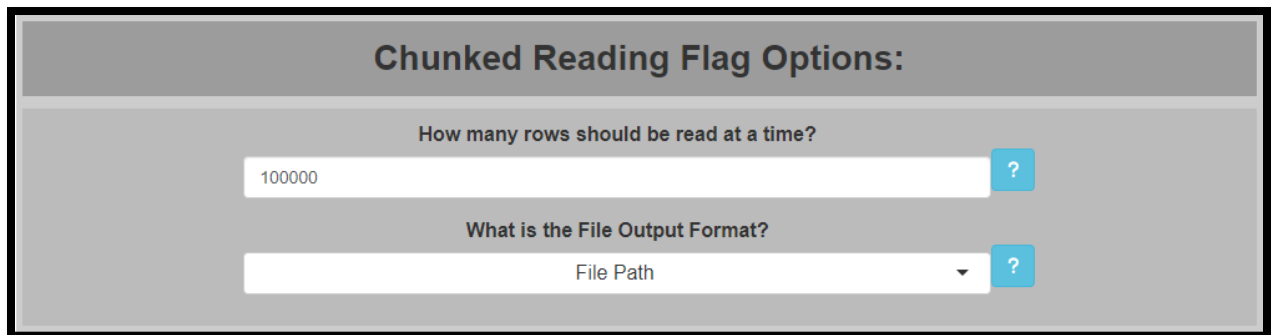
## Chunked Reading Flag Options

---

### Description

In this text block, the user is given two input fields, they may enter an integer greater than **10,000** and less than **1,000,000** which dictates how many rows will be read in at a time, and as well as a select input field for whether the file should be read using **shell commands**, or by using the regular **file path** reading mode.

### Details



The screenshot shows a dialog box titled "Chunked Reading Flag Options:". It contains two questions. The first question is "How many rows should be read at a time?" with a text input field containing "100000" and a help icon. The second question is "What is the File Output Format?" with a dropdown menu showing "File Path" and a help icon.

---

## Standardize Data

---

### Description

Once input files and output folders have all been supplied, and once the flag options have been set to the desired values, the user may press the green **Standardize Data** button at the bottom of the screen to begin data standardization.

Using the user inputs, it will call the standardizing script functions to process the data, and during processing, will share notifications to the top of the screen containing either a neutral notification about how many rows have been read from the source file, a success message indicating that the file has finished reading and processing the file correctly, and that the user may check the folder they provided the output to be in to get the file, or they may receive a red error notification about what went wrong during processing and what they may be able to do to fix it.