# Importing and Exporting Fixed-Width Data with R

Blake Madden, Institutional Data Analyst

A common format used for reporting data is fixed-width columns. In this format, columns are set to a specific width, with no delimiters between them. This is an optimal way to store data, but can prove to be difficult when importing and exporting it. For example, most spreadsheet applications (e.g., Microsoft® Excel) offer fixed-width importing, but using this feature is tedious and error prone. Likewise, exporting tabular data into columns of specific widths requires truncating and padding each column—an equally arduous process.

In this article, we will discuss more elegant solutions which use R to entirely automate these processes.

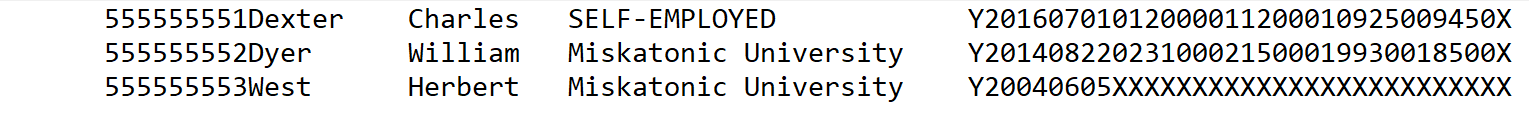
## Importing Fixed-Width Data

As an example, we will import some fictitious data similar to what we may receive from a job-services agency. (Data such as this may be cross-referenced with graduated students to track success in their respective fields.) We will parse its columns into tabular data, remove columns that we don’t need, reformat the data, and finally rearrange the columns.

Before delving into any R code, let’s first review how to import this type of data. The steps for importing fixed-width data are as follows:

1. Understand the file specification.
2. Decide which columns to import (and which to ignore).
3. Review any idiosyncrasies that may occur in the data.
4. Plan how to transform and reformat the data.
5. Decide how to reorder the columns (this may be optional).

In our example, the input file will appear as such:

This data’s format specification is:

* Column 1: 6 spaces (a placeholder).
* Column 2: the person’s social security number (9 characters).
* Column 3: the person’s last name, which will be 10 characters.
* Column 4: the person’s first name, which will be 10 characters.
* Column 5: the company that the person currently works for (25 characters).
* Column 6: the letter ‘Y’ (a control character).
* Column 7: the year that he/she started working there (8 characters).
* Columns 8 through 11: his/her pay for quarters 4 through 1 for the last year (6 characters, each).
* Column 12: the letter ‘X’ (a control character).

At this point, we will decide which columns we need to import, as well as how to import them. This is the import logic that we will use:

* Column 1: skip this.
* Columns 2 through 5: import these as text (also, we will combine the person’s first and last name into one column).
* Column 6: skip this.
* Column 7: import as a date.
* Columns 8 through 11: import as integers (something to note is that if no income is available, then these columns are filled with "XXXXXX").
* Column 12: skip this.

Now that we have our specifications, let’s began actually coding it in R. The first step in any R script is to define the libraries that we need:

suppressMessages(**library**(lubridate)) *#used for reformatting dates*  
suppressMessages(**library**(dplyr)) *#data transformation functions*  
suppressMessages(**library**(magrittr)) *#used for pipe (data assignment) operations*  
suppressMessages(**library**(stringr)) *#string manipulation functions*

Next, we will begin importing our data from "C:/data/EmploymentData.txt" with the **read.fortran()** function and assign it to a dataset named **EmploymentData**:

EmploymentData <- read.fortran("C:/data/EmploymentData.txt",

The next argument to this function is where we define the columns to import, as well as how to import them:

EmploymentData <- read.fortran("C:/data/EmploymentData.txt",  
 c("X6","A9","2A10","A25","X1","A8","4I6","X1"),

This argument will be an array of strings that provide the column definitions. Each of these strings follow the format:

* An optional number specifying how many times to repeat this definition.
* A character defining the column datatype.
* A number defining the width of the column.

Our first definition, "X6", will skip the first six characters (‘X’ means to skip the column).

Next, the "A9" definition will import the social security number as a string (‘A’ means alphanumeric).

After that, "A10" would import the next 10 characters as letters. However, because the both the first and last name columns are 10 characters, we can instruct the parser to use the same column definition for the next two columns. We can accomplish this by placing a ‘2’ in front of the "A10"—this will tell the parser to read the next two columns as 10-character-wide columns. In other words, "A10","A10" is equivalent to "2A10".

After reading the individuals’ names, we can then read in the company names with an "A25" column specification (the company name being a 25-character-wide alphanumeric column).

Next, "X1" will skip the 1-character (control-character) column between the company name and the date of hire.

Next, we will read in the date of hire as an alphanumeric column with an "A8" definition (we will convert this to a more usable date format at a later step).

The next four columns represent the person’s income for each quarter from the previous year. We will read these columns as six-character-wide integers (i.e., whole numbers). We could request that the next four columns be imported in this format by entering "I6" four times ("I6", "I6", "I6", "I6"); however, by using "4I6" instead we can accomplish the same thing with a single string.

Finally, we will skip the end-of-line marker (‘X’) by specifying "X1".

Now that we have defined the columns to import, let us apply some useful names to them. The optional parameter **col.names** in **read.fortran()** enables us to define these column names, which will be an array of strings:

EmploymentData <- read.fortran("C:/data/EmploymentData.txt",  
 c("X6","A9","2A10","A25","X1","A8","4I6","X1"),  
 col.names = c("SSN","Last","First","Company",  
 "DateOfHire",  
 "Q4\_Pay","Q3\_Pay","Q2\_Pay","Q1\_Pay"),

Our final step in our call to **read.fortran()** will involve dealing with missing data in our quarterly payment columns. According to the file’s specification, missing data in these columns will be written as **XXXXXX**. To instruct **read.fortran()** to see this as **NA** (i.e., missing data in R), we will define this with the **na.strings** parameter:

EmploymentData <- read.fortran("C:/data/EmploymentData.txt",  
 c("X6","A9","2A10","A25","X1","A8","4I6","X1"),  
 col.names = c("SSN","Last","First","Company",  
 "DateOfHire",  
 "Q4\_Pay","Q3\_Pay","Q2\_Pay","Q1\_Pay"),  
 na.strings="XXXXXX")

At this point, we will have our employment data; however, we may still need to apply some additional transformations to get the data into a format that we can work with. The first step to transforming data after importing it is to “pipe” (i.e., reroute) the dataset to a series of downstream functions. To pipe our data, add a **%>%** after the call to **read.fortran()**:

…  
na.strings="XXXXXX") **%>%**

Now, we can add our various post-import functions to edit the data. In R, a useful method for transforming and creating columns is the **mutate()** function. (In some ways, **mutate()** can be thought of as a spreadsheet formula in Excel.) Our first operation will be to convert the hire-date column from a string to an actual date:

mutate(DateOfHire = ymd(DateOfHire)) %>%

In the above code (which should be directly after the call to **read.fortran()**), the existing column **DateOfHire** will be converted from a *yyyymmdd* formatted string into an actual *yyyy-mm-dd* date column. This will make this column easier to read and work with. (Note that we included another **%>%** after this function call, meaning that the output from this operation will be rerouted to yet another transformation function.)

Because we imported the alphanumeric columns using specific column widths, some of our textual data (e.g., company names) may have unnecessary whitespace after it. To remove this, we can apply the **str\_trim()** function to each of these columns:

mutate(Last = str\_trim(Last), First = str\_trim(First),  
 Company = str\_trim(Company)) %>%

Another change we would like to make to the data is to combine the persons’ first and last names into one column.

mutate(Name = sprintf("%s, %s", Last, First)) %>%

In the above code, a new column, **Name**, is created by combining the last name, a comma, and then first name for each person.

Our final step will be to remove the **Last** and **First** columns (since we now have a **Name** column to include this information), as well as rearrange the quarterly payment columns into a more logical order. The function **select()** (similar to the SELECT function from SQL) enables us to pick which columns to include in our final output, as well as their order:

select(SSN, Name, Company, DateOfHire,  
 Q1\_Pay, Q2\_Pay, Q3\_Pay, Q4\_Pay)

In the above code, we have requested only the columns that we want in the final dataset, as well as rearranging the payment columns in a Q1…Q4 order.

Finally, we will have our employment dataset. This data can now be viewed by either entering its name (**EmploymentData**) or by calling **View(EmploymentData)** (if you are using RStudio).

At this point, we could also export this data to a tab-delimited file by using the **write.table()** function:

write.table(EmploymentData, "C:/data/EmploymentDataTabbed.txt",  
 sep="\t", quote=FALSE, row.names = FALSE, na = "")

Here, we are writing our dataset to **EmploymentDataTabbed.txt**, where tabs ("\t") are the column separator, none of the columns are quoted, no row names are included in the output, and any missing data will be written as blanks.

By creating this R script, we will now be able to completely automate importing any future fixed-width data (in this particular format) and exporting it to a tab-delimited file.

The full R script is as follows:

suppressMessages(**library**(lubridate)) *#used for reformatting dates*suppressMessages(**library**(dplyr)) *#data transformation functions*suppressMessages(**library**(magrittr)) *#used for pipe (data assignment) operations*suppressMessages(**library**(stringr)) *#string manipulation functions*

EmploymentData <- read.fortran("C:/data/EmploymentData.txt",  
 c("X6","A9","2A10","A25","X1","A8","4I6","X1"),  
 col.names = c("SSN","Last","First","Company",  
 "DateOfHire",  
 "Q4\_Pay","Q3\_Pay","Q2\_Pay","Q1\_Pay"),  
 na.strings="XXXXXX") %>%  
 mutate(DateOfHire = ymd(DateOfHire)) %>%  
 mutate(Last = str\_trim(Last), First = str\_trim(First),  
 Company = str\_trim(Company)) %>%  
 mutate(Name = sprintf("%s, %s", Last, First)) %>%  
 select(SSN, Name, Company, DateOfHire,  
 Q1\_Pay, Q2\_Pay, Q3\_Pay, Q4\_Pay)

write.table(EmploymentData, "C:/data/EmploymentDataTabbed.txt",  
 sep="\t", quote=FALSE, row.names = FALSE, na = "")

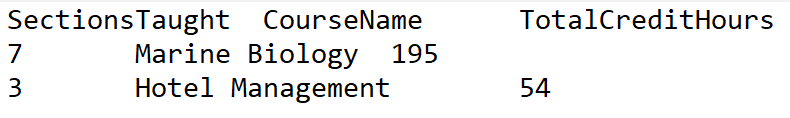
## Exporting Fixed-Width Data

Along with importing fixed-width data, sometimes we may need to convert our data into this format as well. For example, state agencies often require data reports to be uploaded as fixed-width data.

For exporting tabular data into fixed-width format, the major issue to deal with is ensuring that each column is the proper width. This will involve:

1. Truncating text that is too long for its column.
2. Whitespace (or zero) padding data so that it fills its entire column (this will also require knowing which side of the data should be padded).

Consider the following tab-delimited data:



Our goal will be to import this and then export it with the following specification:

1. A header row which consists of these columns:
   1. The name of our college (20 characters, should be right padded with spaces).
   2. The report year (4 characters).
   3. The report semester code (2 characters).
   4. The number of records in the report (a number, 4 characters wide, should be filled with leading zeros).
2. The rows of each section taught (from our tabular data), consisting of these columns:
   1. The name of the course (30 characters, should be right padded with spaces).
   2. The number of sections taught (a number, 2 characters wide, should be filled with leading zeros).
   3. The credits earned from each section (a number, 4 characters wide, should be filled with leading zeros).

The first step in our R script will be to define the information that we will use in our header:

AcademicYear <- 2018 *#Report year*ReportingSemester <- "AU" *#Report semester (autumn)*College <- "INNSMOUTH COLLEGE"

Next, we will import our tab-delimited data using the **read.delim()** function:

CoursesTaught <- read.delim("C:/data/SchoolInput.txt")

At this point, our data will be in a dataset named **CoursesTaught**.

Now, let’s format the header that will be the first line of our output file:

Header <- sprintf("%s%d%s%04d", str\_pad(College,20,"right"),  
 AcademicYear, ReportingSemester, nrow(CoursesTaught))

Here we are using the **sprintf()** function to format text and numbers into a single line of text, as well as controlling how some of this data is printed. **sprintf()** takes a format string as its first argument, where:

* **%s** indicates a placeholder for a string.
* **%d** indicates a placeholder for an integer.
* **%04d** indicates a placeholder for an integer that needs to be 4 characters wide (and will be left-filled with zeros if shorter than that).

In the case of our format string "%s%d%s%04d", **sprintf()** will expect us to provide a string, integer, string, and a final integer. The first string that we provide is the college name, which will be right padded with spaces to a length of 20 using the **str\_pad()** function. After that, we provide our **AcademicYear** variable (an integer), our **ReportingSemester** variable (a string), and finally the number of courses in the data that we imported. Something to note about this last argument is that we get the number of records from our data by calling the **nrow()** (“number of rows”) function on our dataset. Also, the **%04d** syntax in our format string will ensure that it is 4-characters wide and filled with leading zeros.

Our next step will be to format the rest of the data into a single array of text and then combine it with our header row. Although our dataset consists of 3 columns, our goal is to combine each row into a single string (where each column has been properly padded and aligned). To do this, we will again use the **sprintf()** and **str\_pad()** functions to accomplish this; however, instead of calling these functions on single variables, we will use them on entire columns. (An advantage of the R language is that many of its functions are vectorized, meaning that they can be ran on both individual values and also arrays.)

Recall that we are formatting these rows to be a 30-character course name, a 2-character section count, and a 4-character credit hour count. Hence, we will use the format string "%s%02d%04d" for **sprintf()**, and then provide the columns **CourseName**, **SectionsTaught**, and **TotalCreditHours** from our **CoursesTaught** dataset. Our call to **sprintf()** (which will convert our entire dataset to a text array) will appear like this:

sprintf("%s%02d%04d",  
 str\_trunc(str\_pad(CoursesTaught$CourseName,30,"right")  
 ,30,"right",ellipsis = ""),  
 CoursesTaught$SectionsTaught, CoursesTaught$TotalCreditHours)

Note that along with using **str\_pad()** to whitespace pad the course name, we also use the function **str\_trunc()** to ensure that it isn’t longer than 30 characters. If a course name is longer than 30 characters, then this function will truncate it (in this case, it will truncate the right side of the string).

Now, we will combine our header row with the array of text (which contains our data) that **sprintf()** returned:

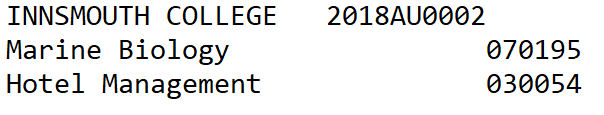
FinalReport <- c(Header,  
 sprintf("%s%02d%04d",  
 str\_trunc(str\_pad(CoursesTaught$CourseName,30,"right")  
 ,30,"right",ellipsis = ""),  
 CoursesTaught$SectionsTaught, CoursesTaught$TotalCreditHours))

The **c()** function will combine its arguments into a new array, and in the above example sends that to the variable **FinalReport**.

Now that we have our header and data in a fixed-column format, we can now export it:

write.table(as.data.frame(FinalReport), "C:/data/FinalReport.txt",  
 col.names = FALSE, row.names = FALSE, quote = FALSE)

Here we are converting our text array to a simple, 1-column dataset and then passing it to the **write.table()** function. We instruct this function that our data does not have any column or row names and to not quote the columns. Once this function runs, our output will appear as such:



Our data is now properly formatted and ready to submit to our state agency. Also, we now have an R script to automate creating these reports for any future submissions.

The final script is as follows:

suppressMessages(**library**(stringr)) *#string manipulation functions*

AcademicYear <- 2018 *#Report year*ReportingSemester <- "AU" *#Report semester (autumn)*College <- "INNSMOUTH COLLEGE"

CoursesTaught <- read.delim("C:/data/SchoolInput.txt")

Header <- sprintf("%s%d%s%04d", str\_pad(College,20,"right"),  
 AcademicYear, ReportingSemester, nrow(CoursesTaught))

FinalReport <- c(Header,  
 sprintf("%s%02d%04d",  
 str\_trunc(str\_pad(CoursesTaught$CourseName,30,"right")  
 ,30,"right",ellipsis = ""),  
 CoursesTaught$SectionsTaught, CoursesTaught$TotalCreditHours))

write.table(as.data.frame(FinalReport), "C:/data/FinalReport.txt",  
 col.names = FALSE, row.names = FALSE, quote = FALSE)