The Shiny set of tools, and, by extension, Flexdashboard, give professional analysts tools to rapidly put interactive versions of their work in the hands of clients. Frequently, an end user will interact with data by either uploading or downloading a new set in its entirety (typically from a .csv or other similarly structured source), or do so ‘on the fly’ interactively, using tools like RHandsonTable. What if you want to do both at the same time? That is – what if you want to be able to change data interactively, or completely upload a new database *without stopping the current instance* or other ‘clunky’ things?

It turns out that you can, and the implementation is relatively straightforward. In this post, our aims are to:  
a. Make the reader more familiar with reactivity, particularly with respect to initializing and containerizing variables  
b. Show how to manage a single object that can be ‘touched’ by the uploadHandler and rHandsonTable  
c. Provide a worked example in Flexdashboard

We need to create an environment where a data object that may be manipulated by functions inside an app is initialized to a pre-loaded set. At runtime, the object may be downloaded to a .csv, manipulated directly in a spreadsheet-like interface, or replaced completely with an upload.

In the following sections, we highlight the working parts of the minimal example (also provided) based on the attributes of different ships in Star Wars.

**Step 1: Initialization**

library(flexdashboard)

library(dplyr)

library(magrittr)

library(rhandsontable)

After calling the applicable libraries (above), we load a starter dataset so that the application isn’t empty when launched. We also create a reactiveValues object called values that sets the handsontable hot to NULL. This is necessary because the handsontable object is self-referential in the sense that it is both an output and input device.

BFL = read.csv("StarterExampleData.csv")

BFL$Exclude = FALSE

values <- reactiveValues(hot = NULL)

The **main** trick is to hold the R dataframe in a reactive environment that responds to both the handsontable object and the upload handler. This is done by a series of nested if statements, that, in order, make the object equal:

* the initialization file, if the object is empty (from the previous code chunk),
* the uploaded file (from the upload handler, below),
* and finally the handsontable object.

Note the unglamorous statement read.csv(input$InputBFL$datapath). This is necessary when reading a file provided by an upload handler. Simply reading the handle attached to the csv will point to the ‘box’, not what is ‘in’ the box.

This is different than the approaches advocated on many forums where the initialization is done in the rhandsontable code. That approach is perfectly valid, but does not offer the flexibility of an uploaded file.

BFLD = reactive({

if(is.null(input$hot)){BFL}

else if(!is.null(input$InputBFL)){read.csv(input$InputBFL$datapath)}

else{

hot\_to\_r(input$hot)

}

})

**Step 2: File Handlers**

Now that we have set up the reactive structure for the object BFLD, which can be accessed by other functions inside our code, we add the functionality for the upload and download handlers.

The download function is two separate items: the downloadButton that triggers the action, and the downloadHandler that performs the interface. In Shiny apps, the linkage is explicit. In Flexdashboard, the linkage is made by putting the handler immediately after the button. *Note: The handlers do not always work in the RStudio App window; it may be necessary to ‘open in browser’*

downloadButton("StarWarsDownload", label = "Star Wars Download")

downloadHandler(

filename = function(){"Star\_Wars\_Download.csv"},

content = function(file){

write.csv(BFLD(), file)

}

)

The upload function is a single block. Here the input File is linked to the reactive variable built in Step 1 by the handle input$InputBFL.

fileInput("InputBFL", "Choose CSV File",

multiple = FALSE,

accept = c("text/csv",

"text/comma-separated-values,text/plain",

".csv"))

**Step 3: Hands On Table**

In the final step, we transform the stored reactive object BFLD into a handsontable object, and output. Note that because it is a reactive object, we call it as BFLD() with parentheses, and inside a ‘render’ context.

output$hot = renderRHandsontable({

rhandsontable(BFLD(), height = 550) %>% hot\_rows()

})

rHandsontableOutput("hot")

**Conclusion**

We hope that this minimal example will help you use both rhandsontable and upload/download handlers in flexdashboard. There should be enough code and explanations here for simpler cases. These code chunks should help you worry less about IO handling in your applications and spend more time on graphics and analysis.

While we did not develop it here, it seems that this same construct could be used to add a web-based data source.