# Reactivity in Shiny WebApps

*Fu-Sheng Chou, MD, PhD*

Reactivity is the heart and soul of Shiny WebApps. It allows the WebApps to respond to user input with immediate response. There are three components in a reactivity system:

1. Reactive source
2. Reactive conductor
3. Reactive endpoint

In a simple one-to-one reactivity system, the reactive conductor can be left out of a reactivity system's design. However, in more sophisticated WebApps with multiple sources and endpoints, the design of one or multiple reactive conductors will allow computation to be more efficient and make the reactivity system work more seamlessly.

**Reactive source and endpoint**

Figure 1A illustrates the simplest form of a reactivity system, where an input corresponds to an output. For example, scoring of each neonatal abstinence syndrome (NAS) items (as input) results in reporting of total scores (as output). Another example would be inputting bilirubin levels and hours of life, resulting in the output of the risk category. In a more sophisticated reactivity system, values of one or multiple variables provided by the user enter a statistic model to calculate a predicted value as a reactive endpoint for output (Figure 1B).

***"One can make an analogy of reactivity to the autosave function of Google Doc documents. Many readers likely have a Google account and have used Google Doc at least once in the past. Different from saving word processing documents in Microsoft Word, where a "Save" button is clicked to save the changes to the document, in the modern-day Google Doc, the document is saved as soon as the document is modified, without the need to hit "Save." That is Reactivity."***

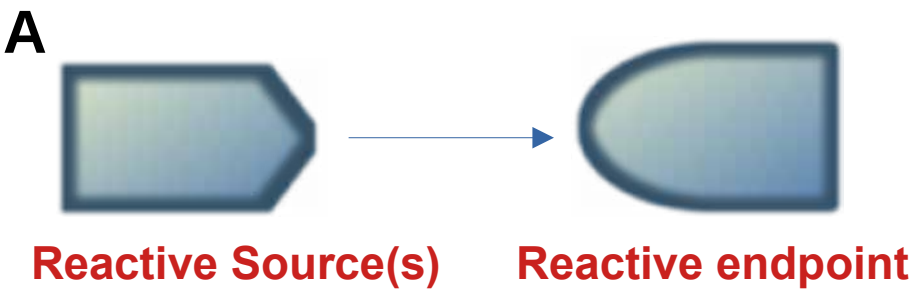
One can make an analogy of reactivity to the *autosave* function of Google Doc documents. Many readers likely have a Google account and have used Google Doc at least once in the past. Different from saving word processing documents in Microsoft Word, where a "Save" button is clicked to save the changes to the document, in the modern-day Google Doc, the document is saved as soon as the document is modified, without the need to hit "Save." That is *Reactivity*. Of course, in the Shiny WebApp, the flexibility still exists to build a button to trigger the flow from input to output (such as the SEARCH button of Neonatology Today's new website), but the use of a reactivity system provides the opportunity to assess the output dynamically. This access is particularly useful when the input is a continuous numeric variable or a multi-level categorical variable. With reactivity, data presentation is no longer static constrained in a tabular or a 2-dimensional figure format but now has a "depth."

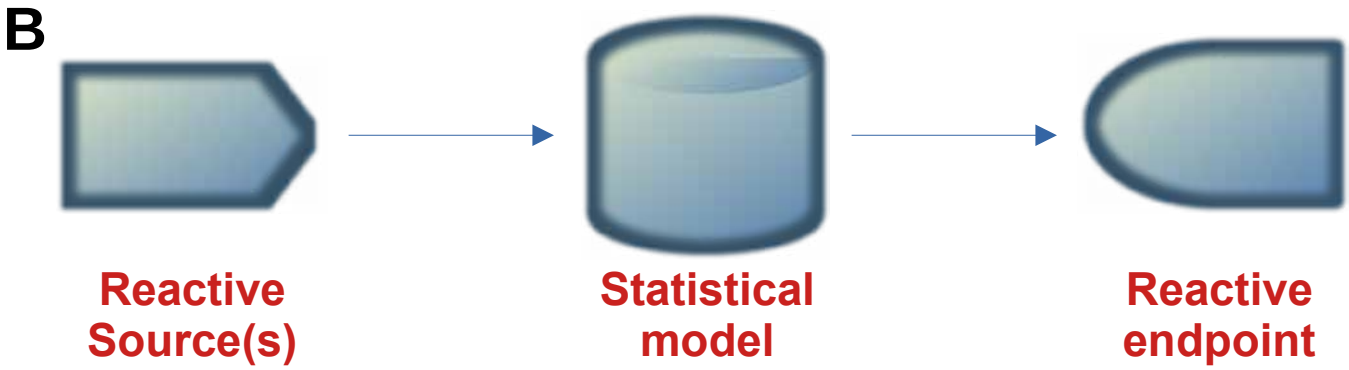
The NAS WebApp has the simplest form of reactivity. As you can see, when an option for an item is clicked, The Total Score (at the top) is updated immediately to reflect the selection. Another example is the PAST ISSUES box of Neonatology Today's new website. When the Year pull-down menu is updated, the Month list is updated immediately (the Month list options are dynamic in accordance to which year is selected: when 2021 is selected, the Month list only displays January and February, but when 2020 is selected, the Month list has all months available for selection. Similarly, changes in year and month result in automatic updates of the table of contents resulting from reactivity.

***" It is relatively easy to conceptualize in a one-to-one or multiple-to-one reactivity system. Nonetheless, the system can handle much more sophisticated situations. There can be one-to-multiple or multiple-to-multiple reactivity systems, with one source going out to multiple endpoints and each endpoint receiving input from multiple sources, sort of like neuronal connections in the brain."***

**Reactivity Conductor**

It is relatively easy to conceptualize in a one-to-one or multiple-to-one reactivity system. Nonetheless, the system can handle much more sophisticated situations. There can be one-to-multiple or multiple-to-multiple reactivity systems, with one source going out to multiple endpoints and each endpoint receiving input from multiple sources, sort of like neuronal connections in the brain. In such a case, designing the flow to include one or multiple reactivity conductors as the information relay station is helpful (Figure 2A). It may seem redundant to have a conductor between the source and the endpoint if data stored in the input source is transferred to the endpoint without any modification. However, in a case where the input source is fed into a very complex statistical algorithm that takes minutes or even hours to run, having a reactivity conductor to store the output from the very complex statistical calculation will make computation much more efficient. Why so? Let us say we developed a statistical model that predicts the probability of successful extubation based on 15 variables. The calculation takes 5 min. The endpoints are multiple, including 1) displaying the probability, 2) calculating the sensitivity and specificity of the model, and 3) plotting the receiver's operating characteristic (ROC) curve. It would probably be wise to have the calculation performed once (takes 5 min), store the predicted values, and render the outputs based on the stored values instead of performing a calculation




every time an output requests the predicted values (takes 15 min). The *reactivity conductor* is the object that stores the values; in other words, the conductor reacts to the input, the value stored in the conductor is constantly monitored for an update. The endpoints then monitor the conductor for further updates (Figure 2B). In this case, the *conductor* is a mediator, which can be treated as an output (to the input sources) and an input (to the endpoints).

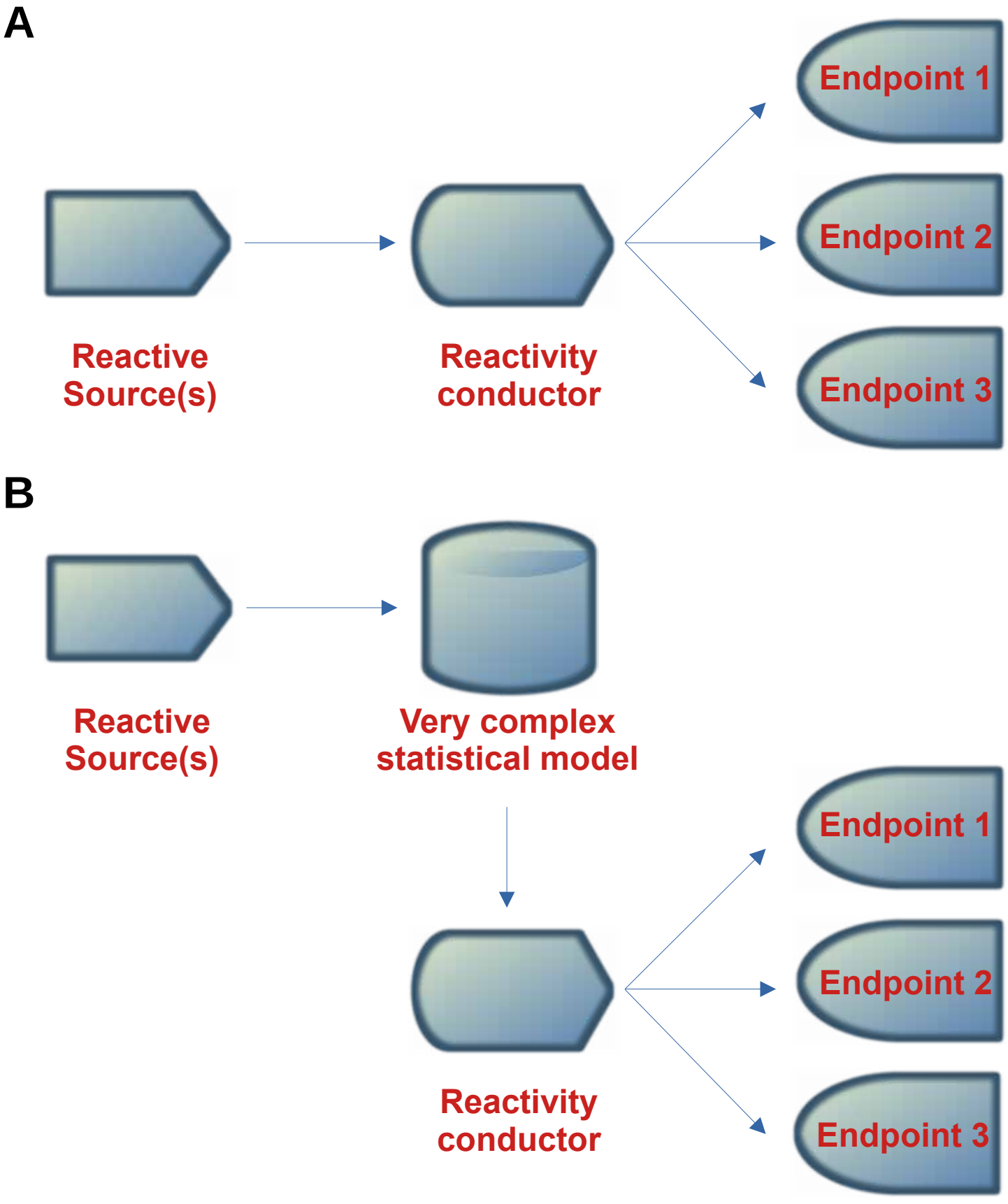
**How does reactivity work?**

In the previous section, we hinted at how reactivity works. First, the arrows' direction in Figure 1 and Figure 2 is easy for humans to understand, but it is not how reactivity works behind the scene. Rather than passively waiting for input sources to send out a notice to the conductor or the output endpoint upon update of an input source, the receiving endpoints constantly check the computer memory to see if any upstream variable contributes to the output endpoints is updated. The upstream variables can be the input sources or the reactivity conductors. There is a computer science approach to the monitoring task, so it does not exhaust the computing power quickly. We, as end-users, probably do not need to know all the details.

**In practical terms**

In our NAS example, because of the reactive nature of the renderUI() function, it constantly monitors whether input values are changed. If so, the user interface, which displays the total score value, will be re-rendered to provide the updated total sum to the variable *total* in the output:

```
server <- function(input, output) {
output$total <- renderUI({
   score = as.numeric(input$high_pitch_cry) +
      as.numeric(input$sleep) +
      as.numeric(input$moro) +
      as.numeric(input$tremor) +
      as.numeric(input$tone) +
      .....
})
}
```


A
Endpoint 1
Endpoint 2
Endpoint 3
Reactive Source(s)
Reactivity conductor
B
Reactive Source(s)
Very complex statistical model
Endpoint 1
Endpoint 2
Endpoint 3
Reactivity conductor

Therefore, the users do not have to click a button to tell the system to re-calculate the total score; it automatically does the job. As one can imagine, the accuracy of the information provided by the input source is critical. One has to choose the correct answer for each NAS item because the total score is updated immediately without a security check. It is like stepping away from working on a manuscript in Google Docs, and your cat jumps onto the computer and decides to walk on the keyboard. You are left with extra random symbols on the document, or even worse, a paragraph is completely deleted. Google Doc does not know it is the cat walking, not human typing, so all the changes are saved. In the old days, when all the changes were not saved in document history, it was an absolute nightmare. The true story, thanks to my lovely cat Aimee.

***"Google Doc does not know it is the cat walking, not human typing, so all the changes are saved. In the old days, when all the changes were not saved in document history, it was an absolute nightmare. The true story, thanks to my lovely cat Aimee."***

One last thing to point out is that the *Shiny* package comes with a function called reactiveValues() which functions similarly to the list() function we introduced in the January 2021 article, (1) except it allows the values stored in each element to be monitored "reactively." Let us say you decide to treat the NAS score as a reactive conductor because you want the output to be the management plan (for example, if the total score is > 12, you want the endpoint to be displaying the unit protocol for NAS pharmacological treatment). In such a case, a reactive list using the reactiveValues() function (see below where we created an element named *NAS* within the reactive list named *myReactiveValues*) can be created to store the total score value. The observeEvent() function can then be used to monitor any update on the value stored in myReactiveValues$NAS to produce the endpoint. In the example below, you can see that observeEvent() has two arguments: the first one is the reactive conductor to monitor, and the second is the *expression* flanked by curvy brackets. Note that the first argument can also be an input value, meaning that observeEvent() can monitor the input source too.

```
server <- function(input, output) {
myReactiveValues <- reactiveValues(NAS = NA)
myReactiveValues$NAS <-
      score = as.numeric(input$high_pitch_cry) +
      as.numeric(input$sleep) +
      as.numeric(input$moro) +
      .....
observeEvent(myReactiveValues$NAS, {
        if (myReactiveValues$NAS > 12) {
                display the protocol for NAS pharmacological
treatment
        }
})
```

**Summary**

In this article, we discussed what reactivity is and how it works. We introduced a couple of functions to store and monitor reactive values. Reactivity systems should hopefully make the WebApps more responsive to the users. However, if user input verification is critical to your app, building a security checkpoint (a button, a message dialog, etc.) may not be a bad idea either.

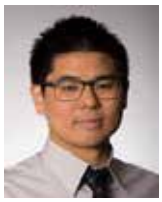
***References:***

1. *Chou F-S. Data types and different ways to store data in R. Neonatol Today. 2021;16(1):22-26. doi:10.51362/neonatology.today/202111612226*

*Disclosure: The author identifies no conflict of interest*

**NT**

*Corresponding Author*



*Fu-Sheng Chou, MD, PhD -*
*Senior Associate Editor,*
*Director, Digital Enterprise*
*Neonatology Today*
*Assistant Professor of Pediatrics*
*Division of Neonatology, Department of Pediatrics*
*Loma Linda University Children's Hospital*
FChou@llu.edu