One of the things I really like about shiny is the ease with which you can incorporate other Javascript based tools and libraries.

**What is annyang?**

“annyang is a tiny javascript library that lets your visitors control your site with voice commands. annyang supports multiple languages, has no dependencies, weighs just 2kb and is free to use.”

Annyang works with all browsers that implement the speech recognition interface of the Web Speech API and makes it extremely easy to get started. Here is some JavaScript that you can include inside raw HTML to add a voice command that listens for the phrase **Jumping Rivers**.

**More complex commands**

In addition to explicit phrases like the above, annyang can also understand named variables, optional phrases and multi-word text (splats).

**Include it in Shiny**

**Note: This will not function in the RStudio browser if you run the code yourself open the app in another browser, preferably Chrome. You will likely have to give permission to the browser to use the microphone too.**

Shiny makes it easy to include snippets of Javascript code in your apps through the HTML tags. We could include the above command into a Shiny app with the following code

## Simple Shiny App - just say "Jumping Rivers"

library("shiny")

ui = fluidPage(

singleton(

tags$head(

tags$script(src = "//cdnjs.cloudflare.com/ajax/libs/annyang/2.6.0/annyang.min.js"),

tags$script(HTML("

if(annyang) {

var commands = {

'Jumping Rivers' : function() {

window.open('https://www.jumpingrivers.com', '\_blank');

}

};

annyang.addCommands(commands);

annyang.start();

}"

))

)

)

)

server = function(input, output, session) {}

shinyApp(ui, server)

It’s potentially easier to keep your JavaScript file separate and include it using includeScript() in your ui.R. So we have two files:

* **voice.js**
* if(annyang){
* var commands = {
* 'Jumping Rivers': function() {
* window.open('https://www.jumpingrivers.com','\_blank');
* }
* };
* annyang.addCommands(commands);
* annyang.start();
* }
* **app.R**
* library("shiny")
* ui = fluidPage(
* singleton(
* tags$head(
* tags$script(src = "//cdnjs.cloudflare.com/ajax/libs/annyang/2.6.0/annyang.min.js"),
* includeScript("voice.js")
* )
* )
* )
* server = function(input, output, session) {}
* shinyApp(ui, server)

**What about changing R variables?**

Now I think that by itself is kind of cool. But it would be much more useful if we could use this to interact with R objects. Fortunately Shiny makes this possible too. The Javascript Shiny.onInputChange('var', value) function can be used to set a value on a reactive under input$var.

As a simple example, let’s create a counter that increments when I say **“Count”**. Our voice.js file would be

if(annyang) {

var value = 0;

var commands = {

'count': function() {

value += 1;

Shiny.onInputChange('counter', value);

}

};

annyang.addCommands(commands);

annyang.start();

}

and a Shiny app that displays the counter might be

library("shiny")

ui = fluidPage(

singleton(

tags$head(

tags$script(src = "//cdnjs.cloudflare.com/ajax/libs/annyang/2.6.0/annyang.min.js"),

includeScript("voice.js")

)

),

textOutput("out")

)

server = function(input, output, session) output$out = renderText(input$counter)

shinyApp(ui, server)

**But I don’t know any Javascript and don’t want to know**

The above is all well and good if you are happy to write the JavaScript code for each of your apps. But if you really don’t want to or don’t feel comfortable doing that then we could define an R function which registers a keyword to start listening (think Alexa), takes all the spoken word after the keyword and stores the string as an R object. This way all processing of the result can be done in the comfort of the R language. One such implementation of this using the annyang splat is

# I have used the keyword jarvis as default because I like iron man

# default value is an empty string

# necessary to choose an "inputId" to bind the value to

voiceInput = function(inputId, keyword = "jarvis", value = "") {

tagList(

singleton(

tags$head(

tags$script(src = "//cdnjs.cloudflare.com/ajax/libs/annyang/2.6.0/annyang.min.js"),

tags$script(HTML("annyang.start({ autoRestart: true, continuous: true});")))),

tags$head(tags$script(HTML(

glue::glue("var = '';

if(annyang){

var commands = {

' \*val': function(val) {

= val;

Shiny.onInputChange('', );

}

}

annyang.addCommands(commands);

}", .open = "<", .close = ">")

)

)

)

)

}

We could now use this in a similar way to other shiny input functions.

ui = fluidPage(

voiceInput("text"),

textOutput("out")

)

server = function(input, output, session) {

output$out = renderText(input$text)

}

shinyApp(ui, server)

Multiple voice inputs with different keywords are also fine. Personally, I think this particular design pattern is a bit of a pain since it requires stating a keyword then processing text in R. But as a single R function that allows voice capture with flexibility in what we do with the output I think it’s OK and the plus side is outside of this function definition we don’t need any Javascript.

**How well does it actually work in practice**

I have found that performance across all devices and browsers is definitely not equal. By far the best browser I have found for viewing the apps is Google Chrome. I have also tended to find that my Ubuntu machines don’t do as well as Microsoft machines in picking up words correctly. A chat I had with someone recently suggested this might be down to drivers under Ubuntu for the microphones but that is not my area of expertise. Voice recognition was also fine on both of my Blackberry phones (one running BB OS 10, the other running Android 7).

It is worth noting that this does require an internet connection to function, in Chrome the voice to text is performed in the cloud.

The other thing I have noticed is that annyang seems relatively sensitive to background noise. This isn’t so bad for functions called using specific phrases but does sometimes have a large effect on the multi-word splats. This is because the splats are greedy and the background noise makes the recognition engine think that you are still talking long after you finished which gives the appearance of the application hanging.

**The grand plan**

I have been playing around with annyang quite a lot and have started making an R package to make it more accessible to other R users. It’s currently on a private GitHub repo, but I will finish up soon and do a first public release there with an associated blog post. For now please feel free to borrow the above R function definition.

I do think there is a lot of scope for the wider utility of voice-controlled applications. They do a phenomenal job of making certain aspects of the digital world more accessible to those that don’t consider themselves tech-savvy. I have been toying around with Alexa skills too, although primarily in Python and am yet to discover whether there can be a pure R solution to getting Alexa to do things. But perhaps that’s for another post.