Masters Thesis

Nick Zinck

May 2017

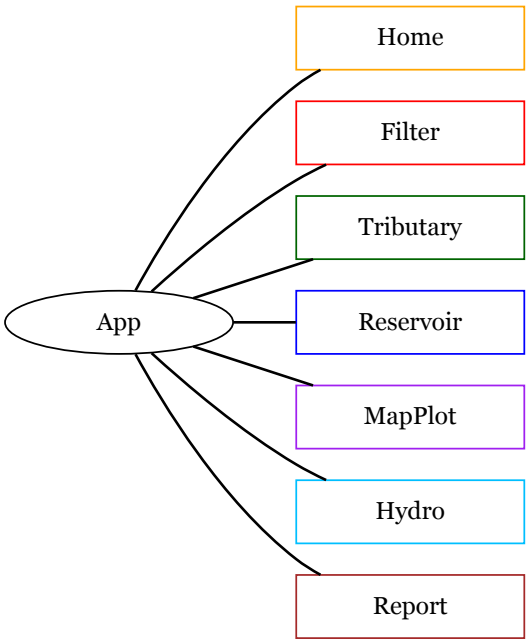
Contents

1 DWSPA Developement Manual

This text is meant for a person with some knowlege of R and Shiny. Please read and be familiar with the overall Shiny Developement Guide before attempting to understand this. It will be beneficial to follow some of the tutorials and leaner directions, rather than trying to dive right in.

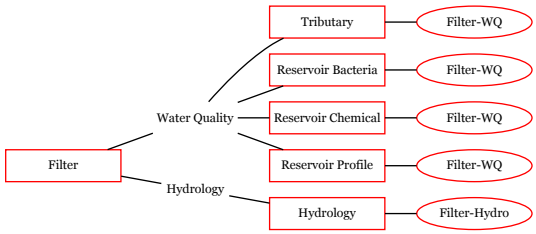
There are multiple script files that the Application comprises of. The App needs each of these files to be located in the correct location in order for the App to run. These files are organized into four main categories: Main, Functions, Modules, and Sources. The App is told to source all of these vairous types of script in the beginning of the App.R script, which essentially means these sripts are open, then read and run by RStudio (or computer?).

1.1 App Layout

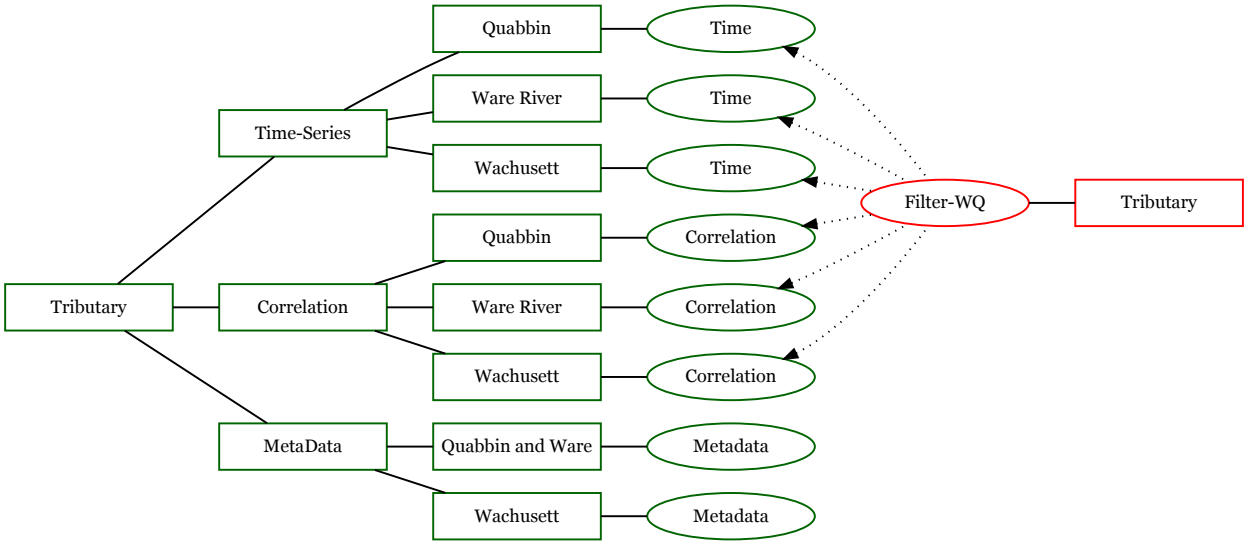


1.1.1 Home

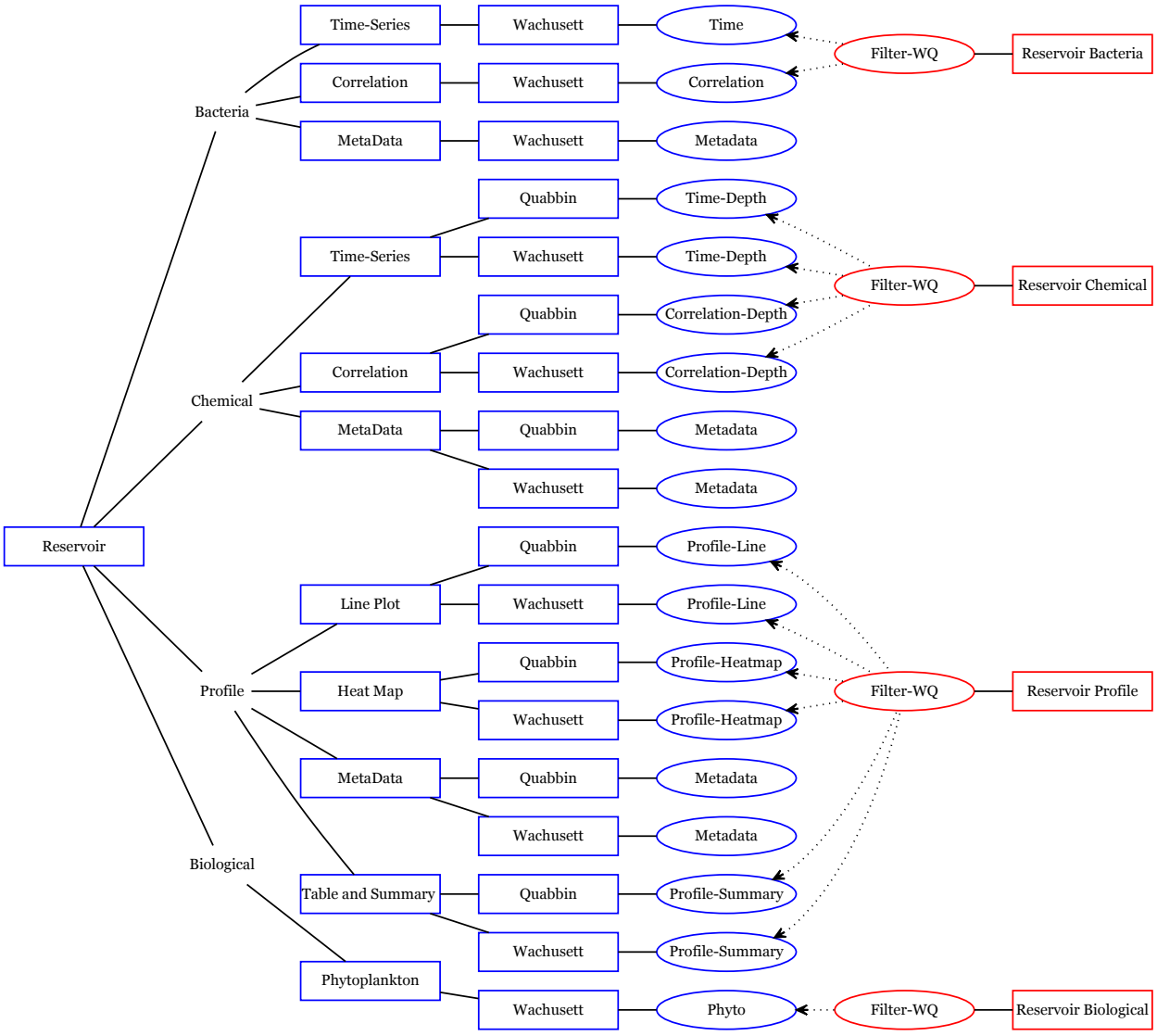
1.1.2 Filter



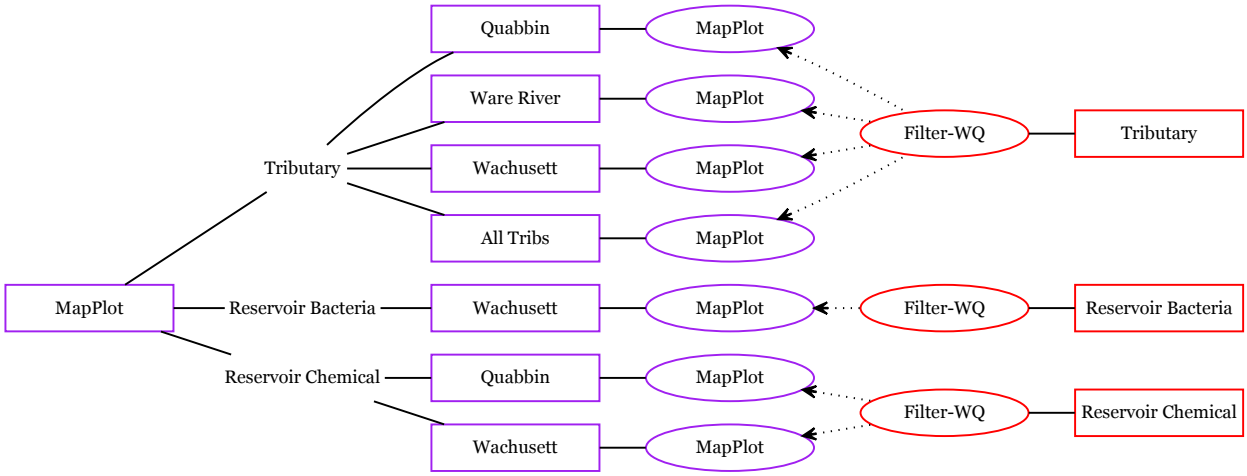
1.1.3 Tributary



1.1.4 Reservoir



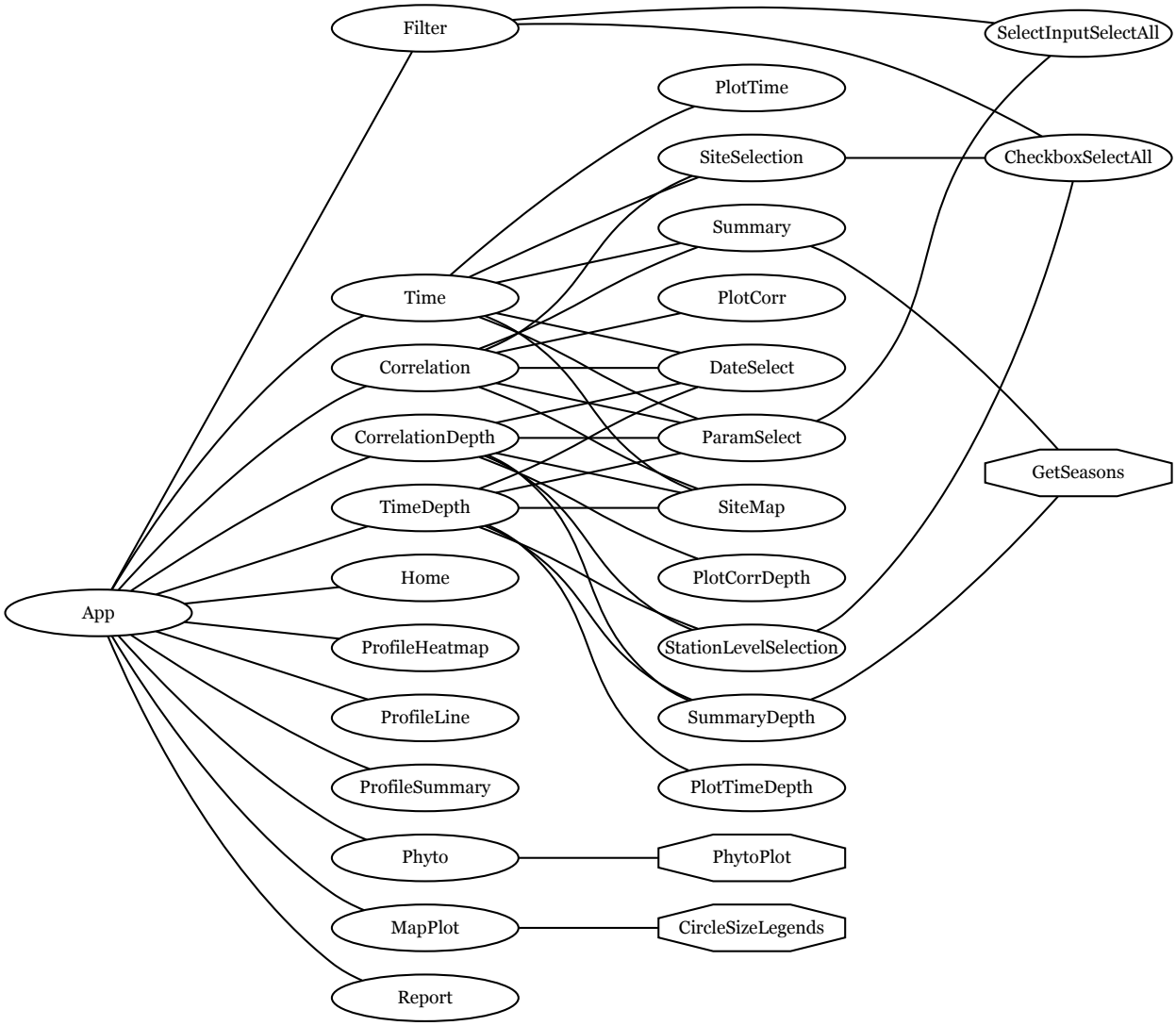
1.1.5 MapPlot


MapPlot
Tributary
Quabbin
Ware River
Wachusett
All Tribs
Reservoir Bacteria
Wachusett
Reservoir Chemical
Quabbin
Wachusett
MapPlot
MapPlot
MapPlot
MapPlot
MapPlot
MapPlot
MapPlot
Filter-WQ
Filter-WQ
Filter-WQ
Tributary
Reservoir Bacteria
Reservoir Chemical

1.1.6 Hydro

1.1.7 Report

1.2 Main Script



1.2.1 App.R

1.2.2 LoadMSAccess.R

1.2.3 LaunchApp.R

1.3 Modules (General)

1.3.1 Module overview

1.3.2 Module Syntax

1.3.3 Module Naming

1.3.4 Passing Reactives in and out of Modules

It can be tricky when dealing with passing reactive expressions into and out of modules. Reactive expressions must be passed in the unresolved reactive expression form ("input1") rather than the resolved reactive expression form ("input1()"). Since reactive expressions are a type of function in R, one is essentially passing the unresolved function into the module instead of the resolved value that the function produces. If one incorrectly passes a resolved reactive expression value through modules, the reactivity will not work correctly.

Passing a reactive expression out of a module is useful when one wants to access a value in an upper level module that is created inside an inner module. An example of when this is used in the App is when the selected value from Check Box Select All UI Widget (checkBoxSelectAll.R).

It is best to understand the underlying principle of modules when dealing with this task.

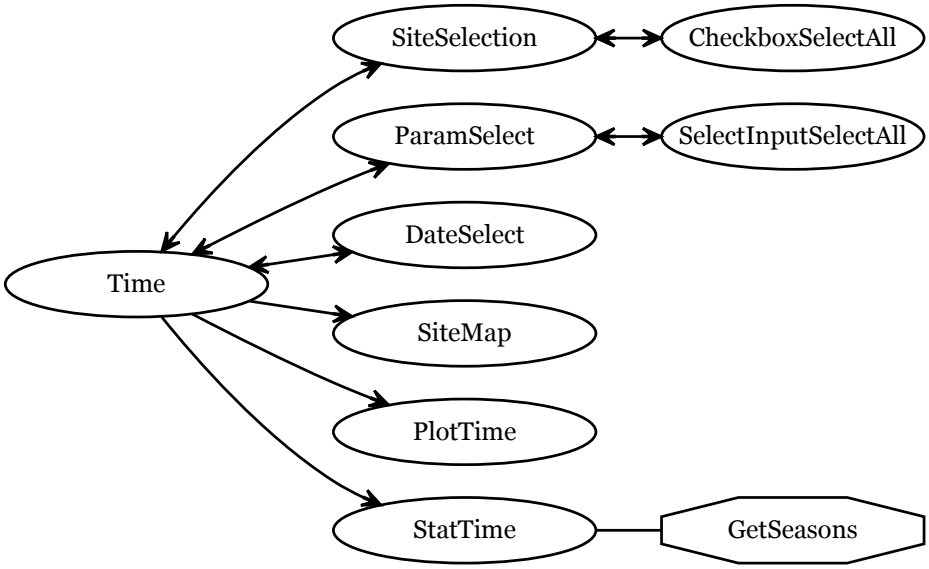
1.4 Primary Modules

1.4.1 Home.R

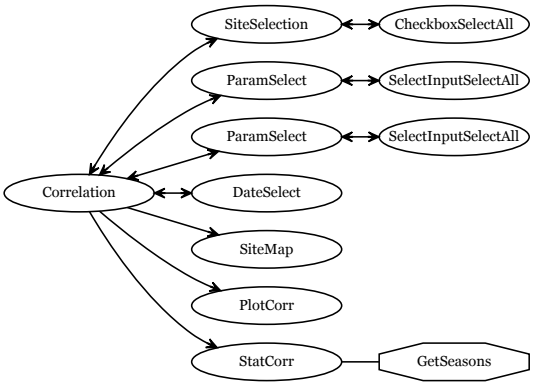
1.4.2 Filter-WQ.R

1.4.3 Time.R

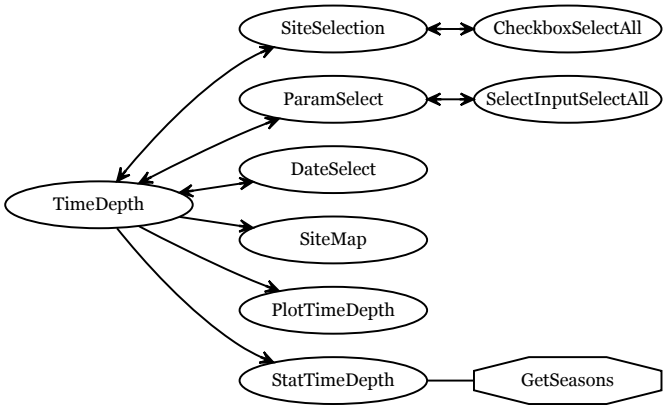
The parameter widget display all the parameters that the data has based on the site selected. The parameter list updates each time a site location is added or removed to the list. The parameter list was built in a way to remember which parameter was selected, so when the site list is changed, the parameter list updates but the previously selected parameter will be reselected automatically. If the previously selected parameter is not in the new updated list, than no parameter is selected and the user will be prompted to select a parameter. If a parameter was selected and than the user deselects all of the sites, the code would want to set the parameter list to an empty list, and therefore the parameter could not be saved. A work around for this was to tell the program that if no site is selected than make the parameter list consist of just the previously selected parameter and have this parameter be selected.



1.4.4 Correlation



1.4.5 Time-Depth



1.4.6 Corrlation-Depth

1.4.7 Profile-Line

1.4.8 Profile-Heatmap

1.4.9 Profile-Summary

1.4.10 Phyto

1.4.11 Mapplot (module)

The MapPlot module is incorporated in the App in under the MapPlot tab.

When the Base Leaflet Map contains the Map Tiles and the Circle Markers for the Site Locations. It is not neccesary to to specify the starting min/max Lat/Long coordinates becuase the location of the markers tell it where to start. Leaflet and RenderUI do not work well together. If this is needed, an action button is a solution but from my experience still causes some problems

1.4.12 Hydro.R

1.4.13 Forestry.R

1.4.14 Report.R

1.5 Secondary Modules - Input

Secondary Input modules serve the purpose of creating more complex input widgets that are used in multiple places throughout the App. Just like all modules, these are separated into their own modules to create more efficient writing of code.

1.5.1 CheckboxSelectAll.R

1.5.2 SelectInputSelectAll.R

1.5.3 SiteCheckbox.R

1.5.4 StationLevelCheckbox.R

1.5.5 Date Select.R

1.5.6 ParameterCheckbox.R

1.5.7 ParameterSelect.R

1.5.8 Saving Inputs (Widget Memory)

Related Scripts:

The functioning of these aforementioned input widgets, is actually much more complex than previously mentioned due to the want for the App to remember an input widget's previously selected value when an input is updated. For example, when a user changes the selected site(s) on the Site Input Widgets, the Parameter Input Widget choices update to a new list of corresponding parameters contained within the new

Site's data. The updated Parameter Input Widget will not automatically select the previously selected choice. This will not be problematic, if the user follows the prefered order of selection:

1. Select the "full" or "filtered" data options. If the "filtered" data option is selected than the user should select any filters on the Filter Tab.
2. Select desired site(s).
3. Select Parameter and Date Range. The order of selection between these two do not matter.

Figure 2 shows a simplistic view of how these widgets react with eachother.

To unconstrain the user from this specific work flow, the Application is programmed to have more or less a memory, of these Input Widget's selected values. A few questions arise that the developer must ask himself/herself when programming the App to memorize the previously selected values and automatically select them:

1. If one does program an Input Widget to have an initial selected value be the previously selected value, what happens when the App is first started and there is no previously selected value? How will this value be defined in the Apps beginning state?

The seemingly best resolution to the first question lies in the set of input updater functions (e.g. *updateCheckboxGroupInput, updateSelectInput, updateDateRangeInput*) that Shiny has created for this type of problem. The input updater funtions send a message to change the settings of the output object ((**???**)). The input updater functions should be contained within an observer function, whether be observe() or observeEvent(). The updater function essentially allows to seperate the initial rendering of the Input Widget with the updating renderings of the Input Widget based on the changing of inputs and reactive expressions that the Input Widget depends on. Code segment 1 shows a simplified portion of the ParamSelect.R script for the purpose of illustraing the seperate selectInput() and updateSelectInput() functions for the same Input Object. In this code, the *selectInput* function's "selected" argument is set undefined and thus defaulted to NULL for a multiple select selectInput Object. The *updateSelectInput* function's "selected" argument is defined as **save.selected** which is the previously selected value. The"save selected" value will be discusses at depth in the next section "***". . Please note that the code that defines the param.choices reactive expression is not shown in this segment for"***_____" reasons but essentially is all the parameter types within the active dataframe filtered for the selected sites.

```
# Parameter Selection UI
output$type.ui <- renderUI({
  ns <- session$ns # see General Note 1
  selectInput(ns("type"), "Parameter:", choices=c(param.choices()), multiple = TRUE)
})


# Update Parameter Selection for new choices
observe({
    updateSelectInput(session, inputId = "type", label = "Parameter:",
                      choices=c(param.choices()),
                      selected = save.selected)
})
```

2. How do you set the selected value of an Input Widget to it's own previously selected value, when in this process the App destroys the previous Input Widget before it builds the updated Input Widget?

To answer the second question, one may first attept to assign the selected value of an Input Widget to it's own reactive expression which we will call "Saved_Value_Reactive". This however does not work, due to "Saved_Value_Reactive" being reactively dependent on the value of the Input Widget. Once the Input Widget is deemed "***", than all of its reactive dependencies are also deemed***"" . Due to the order that Shiny executes reactive objects, the . Even if Shiny did not have this property , this would theortically create a reactive loop, due the the Input Widget depending on the value of the "Saved_Value_Reactive" to know what value to have inially selected and the "Saved_Value_Reactive" depending on the selected value

of the Input Widget. To solve this problem, there actually needs to be a forced break in reactivity with the *isolate* function. The **save.selected** variable is set to the "current" value of the Input Widget before the input updater function is executed, thus saving what will become the "previous" value once the input updater function is executed in the next code chunk. Becuase **save.selected** is surrounded by the *isolate* function, it will not try to update itself during the updating of the Input Widget. Code Segment 2 shows how the *isolate* function is used to assign an Input Widget value as a means of saving a previous value.

```
# Parameter Selection UI
output$type.ui <- renderUI({
  ns <- session$ns # see General Note 1
  selectInput(ns("type"), "Parameter:", choices=c(param.choices()), multiple = TRUE)
})


# Update Parameter Selection for new choices and TO AUTOMATICALLY SELECT THE PREVIOUSLY SELECTED INPU
observe({

  # sAVE THE INPUT WHEN CHOICES CHANGE, BEFORE INPUT IS UPDATED. ISOLATE DISRUPTS REACTIVITY
  isolate({
      save.selected <- input$type
  })

    updateSelectInput(session, inputId = "type", label = "Parameter:",
                      choices=c(param.choices()),
                      selected = save.selected)
})
```

3. What should happen if the previously selected value does not exist in the updated Input Widget's choices?

If an Input Widget is told to start with an inital selection that is not within the choices that the widget has, the widget will start with nothing selected. We will think about the situation when a user is viewing data for the Turbidity parameter and changes then selects a site that does not contain any Turbidity data. Keep in mind that when the user selected the site, he/she may not have been aware that the site does not contain Turbidity data. Due to the Turbidity parameter not being one of the updated parameter choices, The Input Widget will have nothing inially selected (it be blank). This naturally behavior of Shiny is seems to be desirable for this application, and yet no additional code was written in response to this question, yet the bahavior is good to keep in mind. If the user is scanning through various sites with set parameters and/or date ranges, he/she would like to be notified when a site by seeing blank data, rather than seeing other data than is expected to see and addicentally mistaking this data for the previous parameter and date ranges. Generally, notifying the user when inputs have changed that are not directly caused by the user's click is preferable than to not notifying the user and risking that he/she is unaware of the change. This is behavior is likely the cause anytime, an input object with this type of "memory" becomes blank during the session.

4. What if the user deselects all sites?

Based on the previously written code (code segment 2), an issue arises if the user deselects all sites (all sites are unselected) which would create an Input Widget with zero parameter choices (an empty list). This does not seem troublesome at first thought, but when the Input Widget is generated with an empty list, it therefore loses it ability to store the previously selected value. Let's take the situation where a user was curious about Turbidity data and has site "A" at has the proper Input Widgets selected to view this data. If the user now wants to select site "B", the user has two approaches of doing so:

- Approach 1: The user deselects Site "A" and then selects Site "B"
- Approach 2: The user selects Site "B" and then deselects Site "A"

Given approach 1 is taken, Once Site "A" is selected than the Input Widget is generated with an empty

choice list. Becuase the previously stored value of Turbidity (**save.selected**) is not contained within this empty site list, the selected value will be set to nothing (blank). When the user then selecsts Site "B" then the input updater function updates the Input Widget and it thinks that the previously selected value is nothing (blank). To alleviate this unpleasantness, an if statement is added for the two discrete situations where the site list is empty (NULL) and isn't empty (NULL).

```
# Parameter Selection UI

output$type.ui <- renderUI({
  ns <- session$ns # see General Note 1
  selectInput(ns("type"), "Parameter:", choices=c(param.choices()), multiple = TRUE)
})



# To fill back in previously selected

observe({

  # save the Parameter Type input for when the site selection changes.
  # Isolate so does not cause reactivity
  isolate({
      save.selected <- input$type
  })

  # If Site list is changed but not empty then generate a Select Input with the...
  # parameters for that site and autoselect previous selected parameter
  if(!is.null(site())){

    updateSelectInput(session, inputId = "type", label = "Parameter:",
                      choices=c(param.choices()),
                      selected = save.selected)

    # If site list is empty than make a parameter list of just the previously listed item to save it.
  } else {
    updateSelectInput(session, inputId = "type", label = "Parameter:",
                      choices= save.selected,
                      selected = save.selected)
  }

})
```

It is common

This difference in simplicity occurs due to the fact that the Selection Widgets choices only depends on the full or filtered data the complication of Parameter Widget and Date Range Widget arises becuase the choices that the user is allowed to select is limited to that

The Parameter Input Widget was built in a way to remember which parameter was selected, so when the site list is changed, the parameter list updates but the previously selected parameter will be reselected automatically. If the previously selected parameter is not in the new updated Parameter Widget choices, than no parameter is selected and the user will be prompted to select a parameter. If a parameter was selected and than the user deselects all of the sites, the program will create a parameter list just containing the previously selected parameter for saving purposes, even though there should actually be no parameters contained within the empty site list.

The Date Range Widget acts similar to the Parameter Widget. The Date Range Widget is also updated each time a location is added or removed from the list. The Date Range was built to remember which Date Range was previously selected and when the site list is changed by the user, the Date Range updates but the previously selected Dates will be selected automatically. If the previously date range is not within the limits of the new updated Date Rnage Widget choice range, than no date range is selected and the user will be promted to select a date. If a date range was

- need to make Site Selection reactive to the filter data

1.6 Secondary Modules - Output

1.6.1 Plot-Time.R

1.6.2 Plot-Time-Depth.R

1.6.3 Plot-Correlation.R

1.6.4 Plot-Correlation-Depth.R

1.6.5 Plot-Profheat.R

1.6.6 Plot-Profline.R

1.6.7 Summary.R

1.6.8 Summary-Depth.R

1.6.9 Summary-Profile.R

1.7 Secondary Modules - UI - Get rid of this and make into output

1.8 Functions

1.8.1 CircleSizeLegends.R

1.8.2 GetSeasons.R

1.8.3 Phytoplots.R

1.9 General Notes

1.9.1 1. Modules and Namespaces

1.9.2 1. Conditional Panel in Module

Conditional panels and shiny modules don't work well togethr due to the ns() wrapper requrement. Javascript must be used in the conditions of the conditional panel.

1.9.3 2. Req()

The most common use of req() is when you want to delay the execution/creation of an output object until a user input is not empty. For example, if a plot output depends on two inputs (e.g. x and y parameters), there is no need to generate the plot output until these two inputs are both selected. Moreover, this will likely cause an error and leave unwanted red error text on the screen. Req() can provide a soltion to delay the execution of the plot output object. There is a documentation on the req() on Shiny website that covers this topic somewhat differently than this document does. I encourage the redaer to read both, but this will emphasize two key properties of req() which the developer is likely to get stuck on.

The first key property of req() is that it not only stops the execution of the reactive expression it is in but also but stops the execution of other reactive expressions and objects that use this first reactive expression (or in other words depend on this first reactive expression). It stops all of the objects up the reactive chain all the way to the ouput objects that use them. Therefore, the best practice is to use req() directly in output objects. Req() can can also be used in reactive expressions, but caution should be taken on the effects that this req() will have upstream to all the output objects that use (depend on) this reactive expression. If a reactive expression is used in mulitple output objects and the desired execution/creation timing of these output objects are the same then it is probably okay to use the req() in this reactive expression rather than writing multiple req() statements in each of the outputs.

The second key propoerty to req() is how it determines if a value is "empty"" or "not empty". Due to the various types of input widgets and the corresponding empty values, the code to say if a input is empty or not empty is a bit more complex than previously mentioned. For example an empty textinput value is an empty string ("") value, an empty numericinput is NA, and an empty fileInput is NULL. Shiny uses the terms "falsy" or "truthy" which correspond to whether a user input is "empty" or "not empty", respectively. Shiny has made "Falsy" values contain all possible empty input widget values for the developer's conveniance. The full list can be seen in the req() documentation on the Shiny website, but the developer should not usually need these values if one follows the suggested methods listed below:

Delay when input "x" is empty (falsy):

```
output$plot <- renderPlot({
  req(input$x)
  ggplot(df) +
    geom_point(aes(x = input$x, y= input$y, shape = input$z))
})
```

Delay when either input "x", input "y", or input "z" is empty (falsy). This needs both inputs to be non empty (truthy) to execute:

```
output$plot <- renderPlot({
  req(input$x, input$y, input$z)
  ggplot(df) +
    geom_point(aes(x = input$x, y= input$y, shape = input$z))
})
```

A more complicated situation arises when one would like for an output to be delayed if all declared inputs are empty, but if just one is selected for it to execute. This best approach to write this code is to use isTruthy within the req() function. We pass a *vector* (c()) of the list of inputs to the first and only argument that isTruthy() takes.

```
output$plot <- renderPlot({
  req(isTruthy(c(input$x, input$y, input$z)))
  ggplot(df) +
    geom_point(aes(input$x, input$y, input$z))
})
```

An even more complicated situation can be solved with isTruthy(). Say if we want an output to execute

when a user selects a value for input X OR if a user selects a value for input Y AND input Z. We would use:

```
output$plot <- renderPlot({
  req(isTruthy(input$x) | isTruthy(input$y) & isTruthy(input$z))
  ggplot(df) +
    geom_point(aes(input$x, input$y, input$z))
})
```

1.10 Future Work

2 DWSPI Developer Manual

3 Appendix B - Database Tables

3.1 Tributary Data

3.2 Reservoir Data

3.3 Site Location Data

3.4 Parameter Data

4 Appendix C -Reports

4.1 Annual Water Quality Report

4.2 Monthly Water Quality Report

4.3 Custom Report

5 Appendix D - Application Scripts

References

"Shiny - Tutorial." 2018. Accessed January 7. https://shiny.rstudio.com/tutorial/.

"Shiny - Widget Gallery." 2018. Accessed January 7. https://shiny.rstudio.com/gallery/widget-gallery.html.