SBMap_Segments

A.Bock

July 15, 2016

This document is a summary of work and code being developed for the CDI project "Automated Mapping of SB OBjects"

This script will acess SB data from a given ScienceBase ID. We are using Kathy's PRMS data for this application. Functions are a part of package specificed (*sbtools::item_get item_get* is function in *sbtools* library)

1. Accessing ScienceBase Items

This portion of the script will access the SB item and its content fiven a SB item id (the last part of the ScienceBase item page URL).

```
# Step 1 retrieve the SB files based on SB ID
# The ID will be a user-promped by number, this will be fixed for our project

# SB items we are interested in:
#-------------------------------
# This is the SB id for the CDI Project - 56f419c5e4b0f59b85e0bc5d
# This is the SB id for Kathy's Project - 5522bdf7e4b027f0aee3d039
# This is the child SB id for Kathy's Streamflow data from the public PRMs
#     release - 55b69c6ae4b09a3b01b5f653
# We will use this to retrieve the streamflow data (Appendix 2)
test_item<-sbtools::item_get("5522bdf7e4b027f0aee3d039")
names(test_item)
```

```
##  [1] "link"              "relatedItems"      "id"
##  [4] "identifiers"       "title"             "subTitle"
##  [7] "summary"           "body"              "citation"
## [10] "purpose"           "provenance"        "hasChildren"
## [13] "parentId"          "contacts"          "webLinks"
## [16] "browseTypes"       "systemTypes"       "tags"
## [19] "dates"             "spatial"           "extents"
## [22] "files"             "distributionLinks" "previewImage"
```

Note that the streamflow data is a "child item" of Kathy's page (Appendix 2), so we can use the upstream/downstream parent-child relationships to get at that information.

```
#parent<-sbtools::item_get(test_item$parentId)
children<-sbtools::item_list_children(test_item)
#Kathy's streamflow data, extract the sciencebase ID of item
sFlow_SBid<-tail(unlist(strsplit(children[[2]]$link$url,"/")),1)
#Create object for the SFlow data
child_item<-sbtools::item_get(sFlow_SBid)
print(sbtools::item_list_files(child_item)$fname[1:10])
```

```
##  [1] "Appendix_2.1_OF_1982_BASELINE.csv"
##  [2] "Appendix_2.1_OF_2030_ECHAM5.csv"
##  [3] "Appendix_2.1_OF_2030_GENMOM.csv"
##  [4] "Appendix_2.1_OF_2055_ECHAM5.csv"
##  [5] "Appendix_2.1_OF_2055_GENMOM.csv"
##  [6] "Appendix_2.1_OF_2055_GFDL.csv"
##  [7] "Appendix_2.1_OF_2080_ECHAM5.csv"
##  [8] "Appendix_2.1_OF_2080_GENMOM.csv"
##  [9] "Appendix_2.2_RW_1982_BASELINE.csv"
## [10] "Appendix_2.2_RW_2030_ECHAM5.csv"
```

2. Retrieving and Organizing Data from the SB Item

Now that we have the list of files as R objects, we can bring them into R and start aggregating them for the purpose of adding them to our Apps and Maps. There are two objectives here:

1. Summarize conditions for a single time-step for all features to show on the mapo
2. Summarize conditions for a single site across multiple time-steps that can be visualized and/or downloaded by the uers.

2A. Summarizing data across the entire set of features at a single time-setp

First we assume the interaction will focus on the user selecting a single time-period (such as shown in Kathy's example). Using the selected period (i.e. 2055), we perform a "grep" across the filetypes to get the necessary files (for either a single/multiple GCMs, here we just use a single GCM).

Note that these are the basin names for the abbreviations in the filenames: O'Fallon (OF), Redwater River (RW), Little Dry Creek (LD), Middle Musselshell (MM), Judith (JD), Cottonwood Creek (CD), Belt Creek (BT)

```r
# Operations for the entire dataset
sbFiles<-sbtools::item_list_files(child_item)
# find files based on grep of year, and gcm name
baseLine<-grep(paste(c("1982","BASELINE"),collapse="_"),sbFiles$fname)
future<-grep(paste(c("2055","GFDL"),collapse="_"),sbFiles$fname)
print(sbFiles$fname[baseLine])
```

```
## [1] "Appendix_2.1_OF_1982_BASELINE.csv" "Appendix_2.2_RW_1982_BASELINE.csv"
## [3] "Appendix_2.3_LD_1982_BASELINE.csv" "Appendix_2.4_MM_1982_BASELINE.csv"
## [5] "Appendix_2.5_JD_1982_BASELINE.csv" "Appendix_2.6_CD_1982_BASELINE.csv"
## [7] "Appendix_2.7_BT_1982_BASELINE.csv"
```

```r
print(sbFiles$fname[future])
```

```
## [1] "Appendix_2.1_OF_2055_GFDL.csv" "Appendix_2.2_RW_2055_GFDL.csv"
## [3] "Appendix_2.3_LD_2055_GFDL.csv" "Appendix_2.4_MM_2055_GFDL.csv"
## [5] "Appendix_2.5_JD_2055_GFDL.csv" "Appendix_2.6_CD_2055_GFDL.csv"
## [7] "Appendix_2.7_BT_2055_GFDL.csv"
```

This next set of functions opens and reads each file and then binds them together in a single data frame.

```r
baseFlow<-lapply(sbFiles$url[baseLine],RCurl::getURL)
baseAll<-lapply(baseFlow,function(i){
  read.csv(text=unlist(i),row.names=1)
})
# NOTE for this we need use the .id argument in bind_cols to add
# the site names to the column names of the DF, do this next week
baseData<-dplyr::bind_cols(baseAll)
row.names(baseData)<-rownames(baseAll[[1]])
print (dim(baseData))
```

```
## [1] 216 187
```

```r
futFlow<-lapply(sbFiles$url[future],RCurl::getURL)
futAll<-lapply(futFlow,function(i){
  read.csv(text=unlist(i),row.names=1)
})
futData<-dplyr::bind_cols(futAll)
rownames(futData)<-rownames(futAll[[1]])
print (dim(futData))
```

```
## [1] 216 187
```

Calculate the mean for the period (i.e. 2055). We can probably automate to pick the statistic (i.e. Mean, SD, Variability) if we want.

```r
#Means
baseMeans<-colMeans(baseData)
futMeans<-colMeans(futData)
sfMeans<-head(baseMeans-futMeans,-2)
```

2B. Summarizing data for a single site across the entire period of record

The first step is to grab all GCMs for all POR fora single site. We are just going to grab one period of record for now. Right now this is setup to get the mean monthly data for the site.

```r
baseLine_OF<-grep(paste(c("OF","1982","BASELINE"),collapse="_"),sbFiles$fname)
future_OF<-grep(paste(c("OF","2055"),collapse="_"),sbFiles$fname)
```

This is the same set of functions we used above

```r
# retrieve baseflow data for all sites
baseOF <- RCurl::getURL(sbFiles$url[baseLine_OF])
baseOF2 <- read.csv(text=baseOF)
names(baseOF2)
```

```
##  [1] "date"   "seg.1"  "seg.2"  "seg.3"  "seg.4"  "seg.5"  "seg.6"
##  [8] "seg.7"  "seg.8"  "seg.9"  "seg.10" "seg.11" "seg.12" "seg.13"
## [15] "seg.14" "seg.15" "seg.16" "seg.17" "seg.18" "seg.19" "seg.20"
## [22] "seg.21" "seg.22" "seg.23" "seg.24" "seg.25" "seg.26"
```

```r
futOF<-lapply(sbFiles$url[future_OF],RCurl::getURL)
futAllOF<-lapply(futOF,function(i){
  read.csv(text=unlist(i),row.names=1)
})
# look to use the .id argument for bind_cols to identity GCM's with their specific columns
futDataOF<-dplyr::bind_cols(futAllOF)
futDataOF$date<-rownames(futAllOF[[1]])
```

Here, the data is converted to a zoo object, and the mean monthly values are calculated.

```r
# convert to zoo object for annual and seasonal aggregation if necessary
baseZoo<-zoo::read.zoo(baseOF2,index.column=1,format="%Y-%m-%d")
futZoo<-zoo::read.zoo(futDataOF,index.column=length(colnames(futDataOF)),
                      format="%Y-%m-%d")

# get the mean monthly data, turn off warnings.
options(warn=-1)
FutMM <- aggregate(futDataOF, list(data.table::month(as.Date(futDataOF$date))),
                   mean,na.rm=T)
baseMM <- aggregate(baseOF2, list(data.table::month(as.Date(futDataOF$date))),
                    mean, na.rm=T)
options(warn=0)
```

3. Matching Kathy's segments up with the Geospatial Fabric

This section retrieves the segments for the Geospatial Fabric for Region 10u.
Note this is hosted on one of my SB item pages. I need to talk to Roland when he gets back from Sacremento to see if there is an alternative location to access this

```r
# WFS for R10U
layer<-sbtools::item_get_wfs("571559c2e4b0ef3b7ca864c7")
```

```
## Loading required namespace: rgdal
```

```
## Loading required namespace: xml2
```

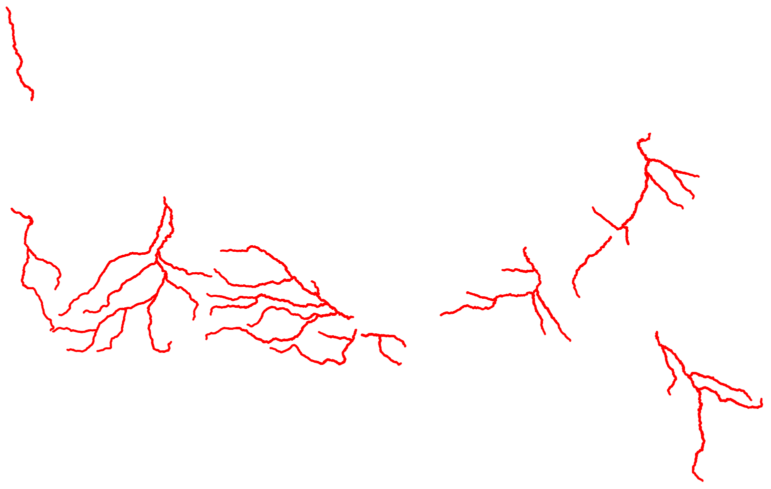
```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\abock\AppData\Local\Temp\1\RtmpMjanT8/file11fc3cd01d53", layer: "nsegment"
## with 5193 features
## It has 16 fields
```

```r
layer_dd<-sp::spTransform(layer,"+init=epsg:4326")
```

The csv below is the key for matching Kathy's segments up to the GeoFab. Right now this is accessed locally. Notes: 1. We need to remove the two extraneous segments from the mussellshell River Kathy found on Friday 2. If we are going to buffer the polylines, we need to do so before we project the shapefile from albers to geographic. 3. Right now we are re-projecting to WGS84, we need to determine if it is possible to re-project to WGS84 web mercator in R.

```
# can only buffer shapefiles with projected coordinates?
#buffer<-rgeos::gBuffer(GF_segs,byid=FALSE,width=3,capStyle="ROUND",joinStyle="ROUND")

# need to order the basin names and segs like they are in line 41 sbmaps_segments
segMap<-read.csv("d:/abock/CDI_Mapping/SB_Mapping/PRMSdata/Streamsegments_Qchange_Buffer.csv",header=T,
# remove the dummy rows, and two rows to get to 185 (fix this next week)
segIDs<-head(segMap$POI_ID,-4)
# reproject to wgs84
GF_segs<-layer_dd[which(layer_dd$POI_ID %in% segIDs),]
sp::plot(GF_segs,col="red")
```



This next section will take the accessed SB data and GIS features, map them onto a leaflet map, and create several summary graphs and downloadable files. Note that the "shiny" app will be built upon this functionality. Functions are a part of package specificed (*sbtools::item_get item_get* is function in *sbtools* library)

1. Plotting the Data

Import the necessary libraries

```
library(leaflet)
library(RColorBrewer)
```

A couple of different ways to present color ramps to symbolize the segments with

```
#pal<-colorNumeric(
#  palette = "Blues",
#  domain=sfMeans
#)

pal2 <- colorQuantile("YlGn", NULL, n = 10)
```

This next section determines what information is in the popup balloons when a user clicks on a feature/segment. This needs to be changed to the stream GNIS name. (what else?)

```
popup <- paste0("<strong>Name: </strong>",
                GF_segs@data$seg_id)
```

Here is the map! Note that we can offer an option to change the basemap (I believe) Things to Do before carrying map over to shiny: 1. Add buffer to shapefile 2. Customize popup 3. Add legend to map 4. Set Bounds for map based on feature extent

Note: need to install the library "webshot" to view data

```
#leaflet(GF_segs) %>% addTiles() %>% addPolylines(color=~pal2(sfMeans),weight=2,popup=~popup)
```