The Statistical Algorithms Importer

Giancarlo Panichi, Gianpaolo Coro
Istituto di Scienza e Tecnologie dell'Informazione A. Faedo
{panichi,coro}@isti.cnr.it

In this document we describe the Statistical Algorithms Importer Web interface. The Statistical Algorithms Importer (SAI) is a tool to import algorithms in the D4Science e-Infrastructure[1]. Currently, it supports R scripts integration. SAI separates R scripts development from its deployment in the infrastructure in a very flexible way. After the first deployment, made in collaboration with the e-Infrastructure team, script developers can modify and update their scripts by themselves, without the intervention of the e-Infrastructure team.

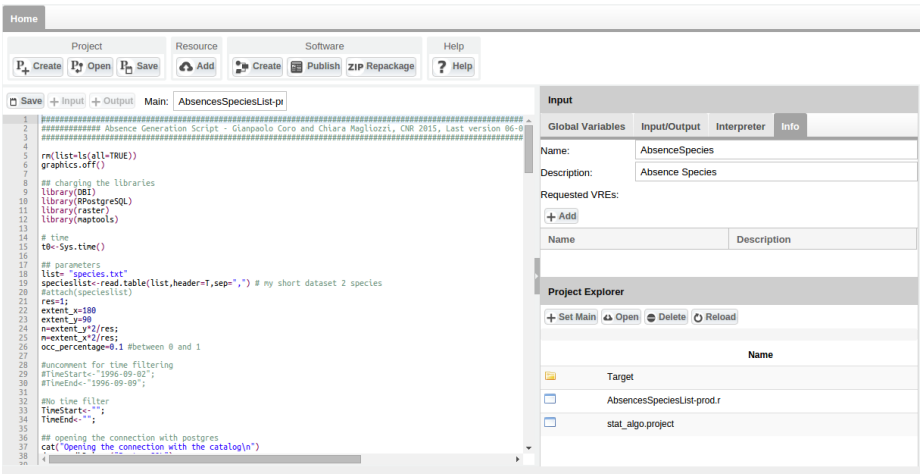

Figure 1: Statistical Algorithms Importer (SAI) portlet. Main interface.

Creating a new Project

This section explains how to create a project using the SAI portlet.

[1] www.d4science.org

Project Folder

The fist step is to create or select an empty folder on the e-Infrastructure Workspace. Then, using the Create Project button in the main menu, the system creates an empty project in that folder.

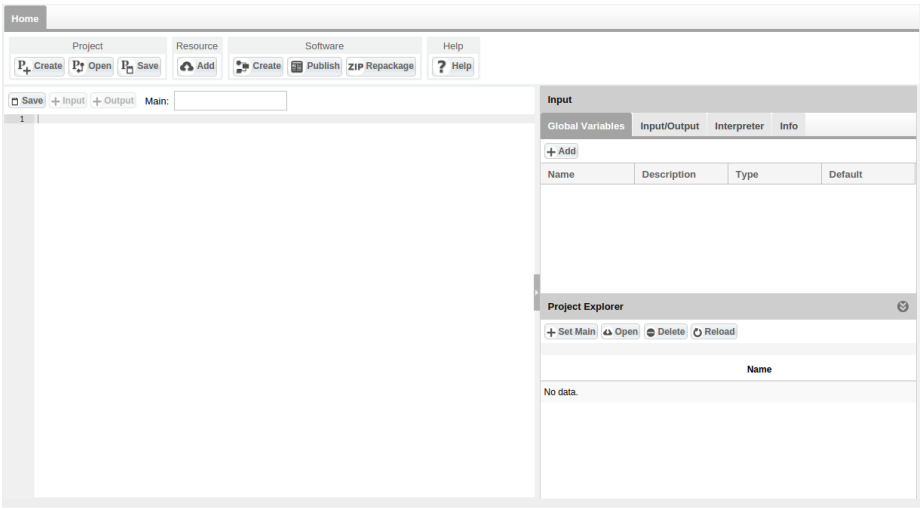

Figure 2: Creating a new Project with SAI.

Import Resources

Any resource needed to run the script can be imported in the Project folder. Resources can be added either via the Workspace or using the Add Resource button in Main Menu, or by dragging and dropping files in the folder window.

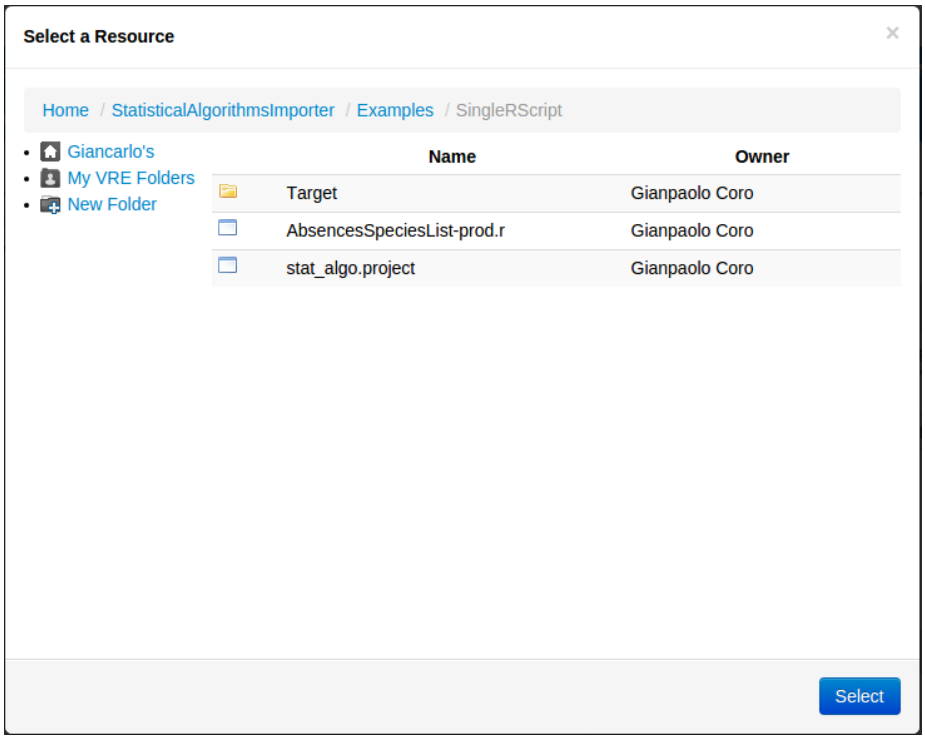

Figure 3: Adding a resource in SAI.

Thus, if the resource is on the user's local file system, (s)he can use Drag and Drop, also with multiple files.

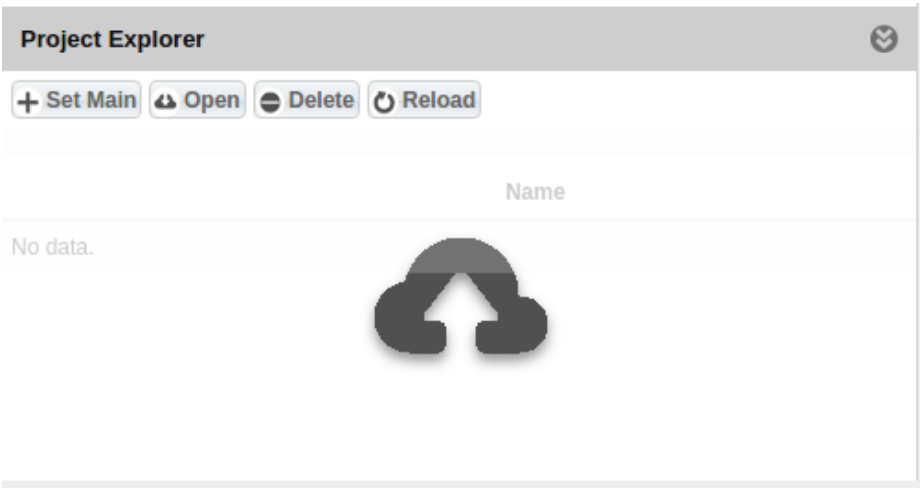

Figure 4: Adding resources with Drag and Drop.

Set the Main Code

After adding the scripts and resources, one of the script files should be indicated as the Main code. The D4Science e-Infrastructure will run this code, which is supposed to import and orchestrate the other scripts. Indicating a script as the Main code can be done by clicking the Set Main button in Project Explorer. The file will be loaded in the Editor. In this phase, the system also reads possible annotations inside the script (e.g. WPS4R annotations[2]). At this point, the user can change the code and save it using the Save button on the Editor panel. Alternatively, the user can also use Copy and Paste by writing the code directly in the editor and then save it, still using the Save button in Editor menu (A file name will be requested).

[2]https://wiki.52north.org/pub/Geostatistics/WPS4R/WPS4R_2.2_prsentation.pdf

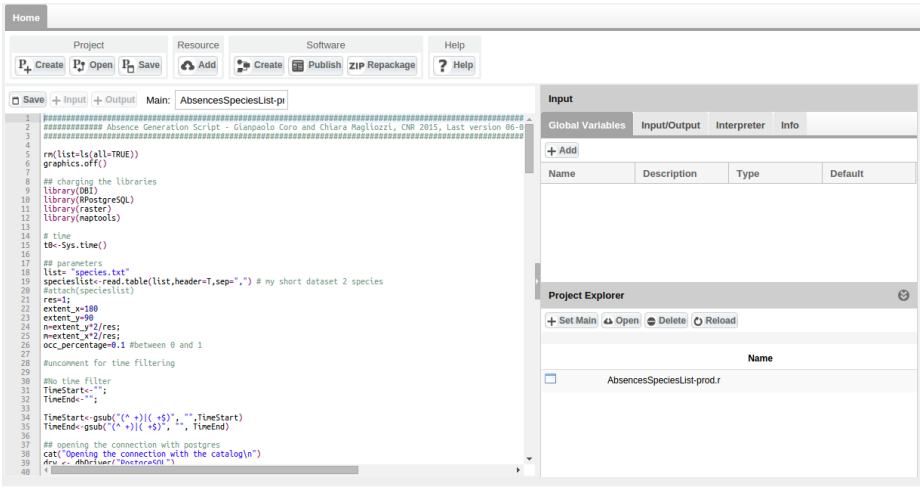

Figure 5: Set Main Code and Editor facilities.

Input

In this area the system collects all the information required by the system to create software for the e-Infrastructure and communicate with the e-Infrastructure team. Metadata, input/output information, global parameters and required packages are collected here.

Global Variables

In this panel the user can add any Global Variable that is used by the script as pre-requisite.

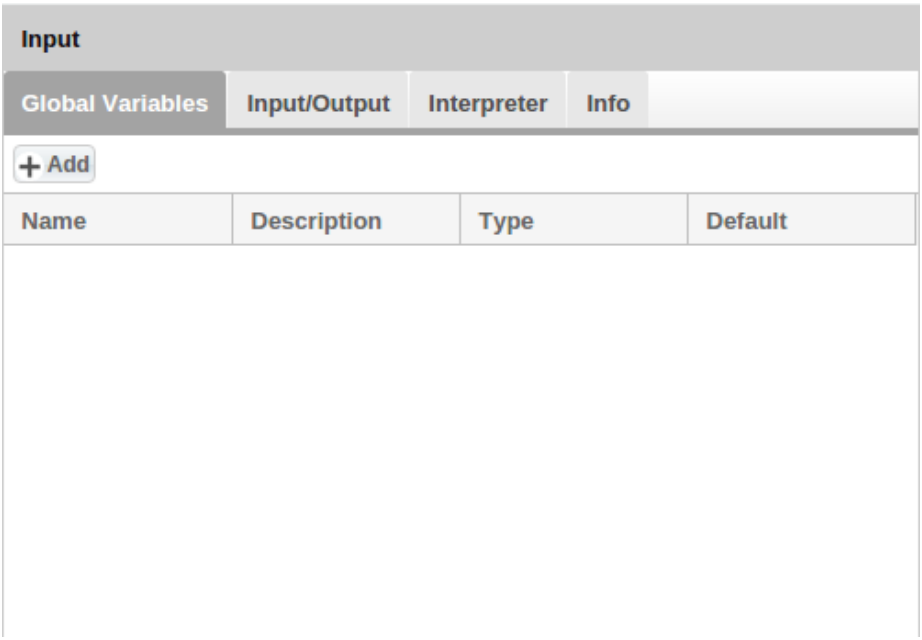

Figure 6: Global Variables panel.

Input/Output

In this area, selected input and output from the script is collected. In order to add a new I/O, the user should select a row in the code (from the Editor) and then click the +Input (or +Output) button in the Editor Menu.

A new row is added to the Input/Output list. The system parses the code behind the scenes and guesses the best type, description and name of the parameter. Once a row has been created in the Input/Output window, the user can change information by clicking on the row. At least one input is required for compiling the project. **The name of the input variable and the default value should not be changed unless a parsing error occurred**. The reason is that the infrastructure will discover the variables inside the script by using the name and the default value.

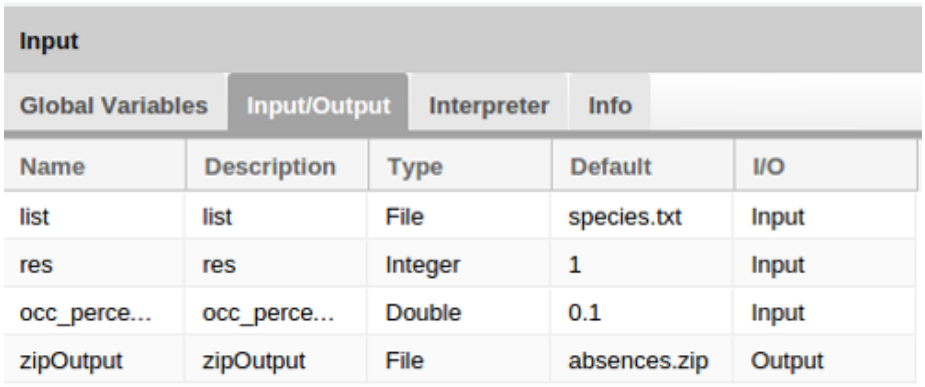

Figure 7: Input/Output panel.

Interprer Info

A user can add Version and Packages information in the Interpreter Info panel. The version number is mandatory for the project. Here, for example, a user should specify the version of the R interpreter and the packages needed to run the script. These will be installed on the e-Infrastructure machines during the first deployment session.

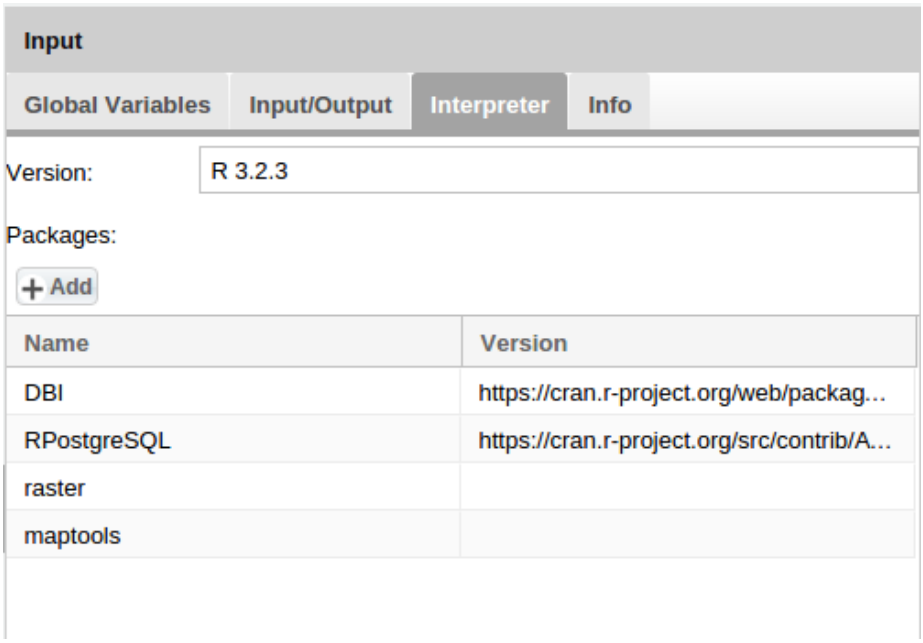

Figure 8: Interpreter Information panel.

Project Info

A name and a description of the project are mandatory. These will be displayed to the user of the e-Infrastructure and should also contain proper citation of the algorithm. Special characters are not allowed for the algorithm name. The user can include a list of the VREs this algorithm should be visible to.

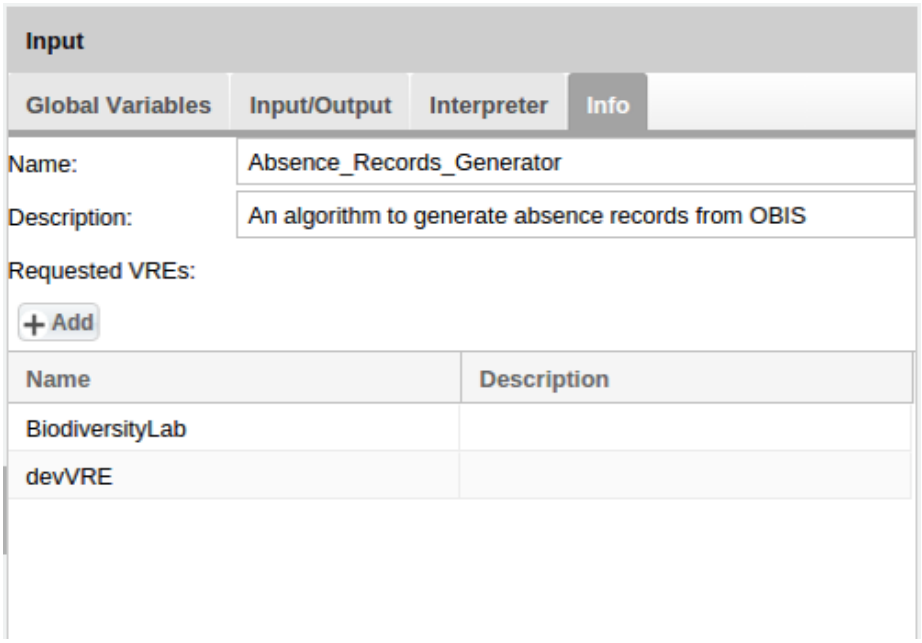

Figure 9: Project Information panel.

Saving the Project

A user can save a project by clicking on the Save button in the Main Menu. A file called stat_algo.project is added to Project Folder.

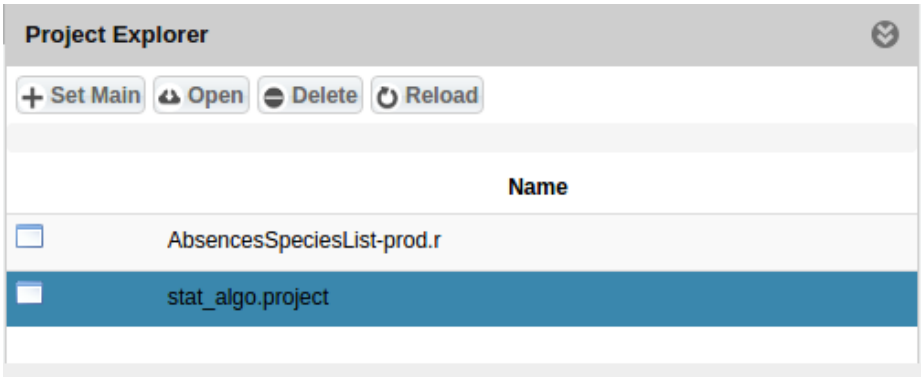

Figure 10: The .project file created for the SAI project.

Using WPS4R Annotations

SAI automatically parses R code containing WPS4R annotations, the system automatically transforms annotations into the Input/Output panel and the Project Info panel. The name of the algorithm is mandatory in the annotations. We report a full example of annotated algorithm and attach the complete algorithm in a zip package [3]:

```
#52North WPS annotations
# wps.des: id = Absence_generation_from_OBIS, title = Absence_
    generation_from_OBIS, abstract = A script to estimate absence
    records from OBIS;

rm(list=ls(all=TRUE))
graphics.off()

## charging the libraries
library(DBI)
library(RPostgreSQL)
library(raster)
library(maptools)

# time
t0<-Sys.time()

## parameters
# wps.in: id = list, type = text/plain, title = list of species
    beginning with the speciesname header,value="species.txt";
list= "species.txt"
specieslist<-read.table(list,header=T,sep=",") # my short dataset 2
     species
#attach(specieslist)
# wps.in: id = res, type = double, title = resolution of the
    analysis,value=1;
res=1;
extent_x=180
extent_y=90
n=extent_y*2/res;
m=extent_x*2/res;
# wps.in: id = occ_percentage, type = double, title = percentage of
     observations occurrence of a viable survey,value=0.1;
occ_percentage=0.1 #between 0 and 1

#uncomment for time filtering

#No time filter
TimeStart<-"";
TimeEnd<-"";

TimeStart<-gsub("(^ +)|( +$)", "",TimeStart)
TimeEnd<-gsub("(^ +)|( +$)", "", TimeEnd)

## opening the connection with postgres
```

[3] https://wiki.gcube-system.org/images_gcube/2/29/AbsencesSpeciesList_prod_annotated.zip

```
cat("Opening␣the␣connection␣with␣the␣catalog\n")
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname="", host="", port="", user="",
    password="")

cat("Analyzing␣the␣list␣of␣species\n")
counter=0;
overall=length(specieslist$scientificname)

cat("Extraction␣from␣the␣different␣contributors␣the␣total␣number␣of
    ␣obs␣per␣resource␣id...\n")

timefilter<-""
if (nchar(TimeStart)>0 && nchar(TimeEnd)>0)
  timefilter<-paste("␣where␣datecollected>'",TimeStart,"'␣and␣
      datecollected<'",TimeEnd,"'",sep="");

queryCache <- paste("select␣drs.resource_id,␣count(distinct␣
    position_id)␣as␣allcount␣from␣obis.drs", timefilter, "␣group␣by
    ␣drs.resource_id",sep="")
cat("Resources␣extraction␣query:",queryCache,"\n")

allresfile="allresources.dat"
if (file.exists(allresfile)){
  load(allresfile)
} else{
  allresources1<-dbGetQuery(con,queryCache)
  save(allresources1,file=allresfile)
}

files<-vector()
f<-0
dir.create("data")

for (sp in specieslist$scientificname){
  f<-f+1
  t1<-Sys.time()
  graphics.off()
  grid=matrix(data=0,nrow=n,ncol=m)
  gridInfo=matrix(data="",nrow=n,ncol=m)
  outputfileAbs=paste("data/Absences_",sp,"_",res,"deg.csv",sep="")
      ;
  outputimage=paste("data/Absences_",sp,"_",res,"deg.png",sep="");

  counter=counter+1;
  cat("analyzing␣species",sp,"\n")
  cat("***Species␣status",counter,"of",overall,"\n")

  ## first query: select the species
  cat("Extraction␣the␣species␣id␣from␣the␣OBIS␣database...\n")
  query1<-paste("select␣id␣from␣obis.tnames␣where␣tname='",sp,"'",
      sep="")
  obis_id<- dbGetQuery(con,query1)
  cat("The␣ID␣extracted␣is␣", obis_id$id, "for␣the␣species", sp, "\
      n", sep="␣")

  if (nrow(obis_id)==0) {
```

```
  cat("WARNING: there is no reference code for", sp,"\n")
  next;
}

## second query: select the contributors
cat("Selection of the contributors in the database having
    recorded the species...\n")
query2<- paste("select distinct resource_id from obis.drs where
    valid_id='",obis_id$id,"'", sep="")
posresource<-dbGetQuery(con,query2)

if (nrow(posresource)==0) {
  cat("WARNING: there are no resources for", sp,"\n")
  next;
}


## third query: select from the contributors different
    observations
merge(allresources1, posresource, by="resource_id")-> res_ids

## forth query: how many obs are contained in each contributors
    for the species
cat("Extraction from the different contributors the number of obs
    for the species...\n")
query4 <- paste("select drs.resource_id, count(distinct position_
    id) as tgtcount from obis.drs where valid_id='",obis_id,"'
    group by drs.resource_id ",sep="")
tgtresources1<-dbGetQuery(con,query4)
merge(tgtresources1, posresource, by="resource_id")->
    tgtresourcesSpecies

## fifth query: select contributors that has al least 0.1
    observation of the species
#### we have the table all together: contributors, obs in each
    contributors for at leat one species and obs of the species
    in each contributors
cat("Extracting the contributors containing more than 10% of
    observations for the species\n")
tmp <- merge(res_ids, tgtresourcesSpecies, by= "resource_id",all.
    x=T)
tmp["species_10"] <- NA
tmp$tgtcount / tmp$allcount -> tmp$species_10

viable_res_ids <- subset(tmp,species_10 >= occ_percentage, select
    =c("resource_id","allcount","tgtcount", "species_10"))
#cat(viable_res_ids)

if (nrow(viable_res_ids)==0) {
  cat("WARNING: there are no viable points for", sp,"\n")
  next;
}

numericselres<-paste("'",paste(as.character(as.numeric(t(viable_
    res_ids["resource_id"]))),collapse="','"),"'",sep="")
```

```
## sixth query: select all the cell at 0.1 degrees resolution in
    the main contributors
cat("Select the cells at 0.1 degrees resolution for the main
    contributors\n")
query6 <- paste("select position_id, positions.latitude,
    positions.longitude, count(*) as allcount ",
                "from obis.drs ",
                "inner join obis.tnames on drs.valid_id=tnames.id
                    ",
                "inner join obis.positions on position_id=
                    positions.id ",
                "where resource_id in (", numericselres,") ",
                "group by position_id, positions.latitude,
                    positions.longitude, resource_id")
all_cells <- dbGetQuery(con, query6)

## seventh query:  select all the cell at 0.1 degrees resolution
    in the main contributors for the selected species
cat("Select the cells at 0.1 degrees resolution for the species
    in the main contributors\n")
query7 <- paste("select position_id, positions.latitude,
    positions.longitude, count(*) as tgtcount ",
                "from obis.drs",
                "inner join obis.tnames on drs.valid_id=tnames.id
                    ",
                "inner join obis.positions on position_id=
                    positions.id ",
                "where resource_id in (", numericselres,") ",
                "and drs.valid_id='",obis_id,"'",
                "group by position_id, positions.latitude,
                    positions.longitude")
presence_cells<-dbGetQuery(con, query7)

## last query: for every cell in the sixth query if there is a
    correspondent in the seventh query I can put 1 otherwise 0
data.df<-merge(all_cells, presence_cells, by= "position_id",all.x
    =T)
data.df$longitude.y<-NULL
data.df$latitude.y<-NULL
data.df[is.na(data.df)] <- 0

######### Table resulting from the analysis
pres_abs_cells <- subset(data.df,select=c("latitude.x","longitude
    .x", "tgtcount","position_id"))
positions<-paste("'",paste(as.character(as.numeric(t(pres_abs_
    cells["position_id"]))),collapse="','"),"'",sep="")

query8<-paste("select position_id, resfullname, digirname, abstract
    , temporalscope, date_last_harvested",
              "from ((select distinct position_id, resource_id
                  from obis.drs where position_id IN (",
                  positions,
              ") order by position_id) as a",
              "inner join (select id, resfullname, digirname,
                  abstract, temporalscope, date_last_harvested from
                  obis.resources where id in (",
```

```
                numericselres,"))␣as␣b␣on␣b.id␣=␣a.resource_id)␣as␣
                    d")

resnames<-dbGetQuery(con,query8)
#sorting data df
pres_abs_cells<-pres_abs_cells[with(pres_abs_cells, order(
    position_id)),]

nrows = nrow(pres_abs_cells)
######## FIRST Loop inside the rows of the dataset
cat("Looping␣on␣the␣data\n")
for(i in 1:nrows) {
  lat<-pres_abs_cells[i,1]
  long<-pres_abs_cells[i,2]
  value<-pres_abs_cells[i,3]
  resource_name<-paste("\"",paste(as.character(t(resnames[i,]))),
      collapse="\",\""),"\"",sep="")#resnames[i,2]
  k=round((lat+90)*n/180)
  g=round((long+180)*m/360)
  if (k==0) k=1;
  if (g==0) g=1;
  if (k>n || g>m)
    next;
  if (value>=1){
    if (grid[k,g]==0){
      grid[k,g]=1
      gridInfo[k,g]=resource_name
    }
    else if (grid[k,g]==-1){
      grid[k,g]=-2
      gridInfo[k,g]=resource_name
    }
  }
  else if (value==0){
    if (grid[k,g]==0){
      grid[k,g]=-1
      #cat("resource abs",resource_name,"\n")
      gridInfo[k,g]=resource_name
    }
    else if (grid[k,g]==1){
      grid[k,g]=-2
      gridInfo[k,g]=resource_name
    }

  }
}
cat("End␣looping\n")

cat("Generating␣image\n")
absence_cells<-which(grid==-1,arr.ind=TRUE)
presence_cells_idx<-which(grid==1,arr.ind=TRUE)
latAbs<-((absence_cells[,1]*180)/n)-90
longAbs<-((absence_cells[,2]*360)/m)-180
latPres<-((presence_cells_idx[,1]*180)/n)-90
longPres<-((presence_cells_idx[,2]*360)/m)-180
resource_abs<-gridInfo[absence_cells]
```

```
  absPoints <- cbind(longAbs, latAbs)
  absPointsData <- cbind(longAbs, latAbs,resource_abs)

  if (length(absPoints)==0)
  {
    cat("WARNING no viable point found for ",sp," after processing!
        \n")
    next;
  }
  data(wrld_simpl)
  projection(wrld_simpl) <- CRS("+proj=longlat")
  png(filename=outputimage, width=1200, height=600)
  plot(wrld_simpl, xlim=c(-180, 180), ylim=c(-90, 90), axes=TRUE,
      col="black")
  box()
  pts <- SpatialPoints(absPoints,proj4string=CRS(proj4string(wrld_
      simpl)))

  ## Find which points do not fall over land
  cat("Retreiving the poing that do not fall on land\n")
  pts<-pts[which(is.na(over(pts, wrld_simpl)$FIPS))]
  points(pts, col="green", pch=1, cex=0.50)
  datapts<-as.data.frame(pts)
  colnames(datapts) <- c("longAbs","latAbs")

  abspointstable<-merge(datapts, absPointsData, by.x= c("longAbs","
      latAbs"), by.y=c("longAbs","latAbs"),all.x=F)


  header<-"longitude,latitude,resource_id,resource_name,resource_
      identifier,resource_abstract,resource_temporalscope,resource_
      last_harvested_date"
  write.table(header,file=outputfileAbs,append=F,row.names=F,quote=
      F,col.names=F)

  write.table(abspointstable,file=outputfileAbs,append=T,row.names=
      F,quote=F,col.names=F,sep=",")
  files[f]<-outputfileAbs
  cat("Elapsed:  created imaged in ",Sys.time()-t1," sec \n")
  graphics.off()
}

# wps.out: id = zipOutput, type = text/zip, title = zip file
    containing absence records and images;
zipOutput<-"absences.zip"
zip(zipOutput, files=c("./data"), flags= "-r9X", extras = "",zip =
    Sys.getenv("R_ZIPCMD", "zip"))

cat("Closing database connection")
cat("Elapsed:  overall process finished in ",Sys.time()-t0," min \n
    ")
dbDisconnect(con)
graphics.off()
```

The following screenshot reports the result of importing this script into SAI:

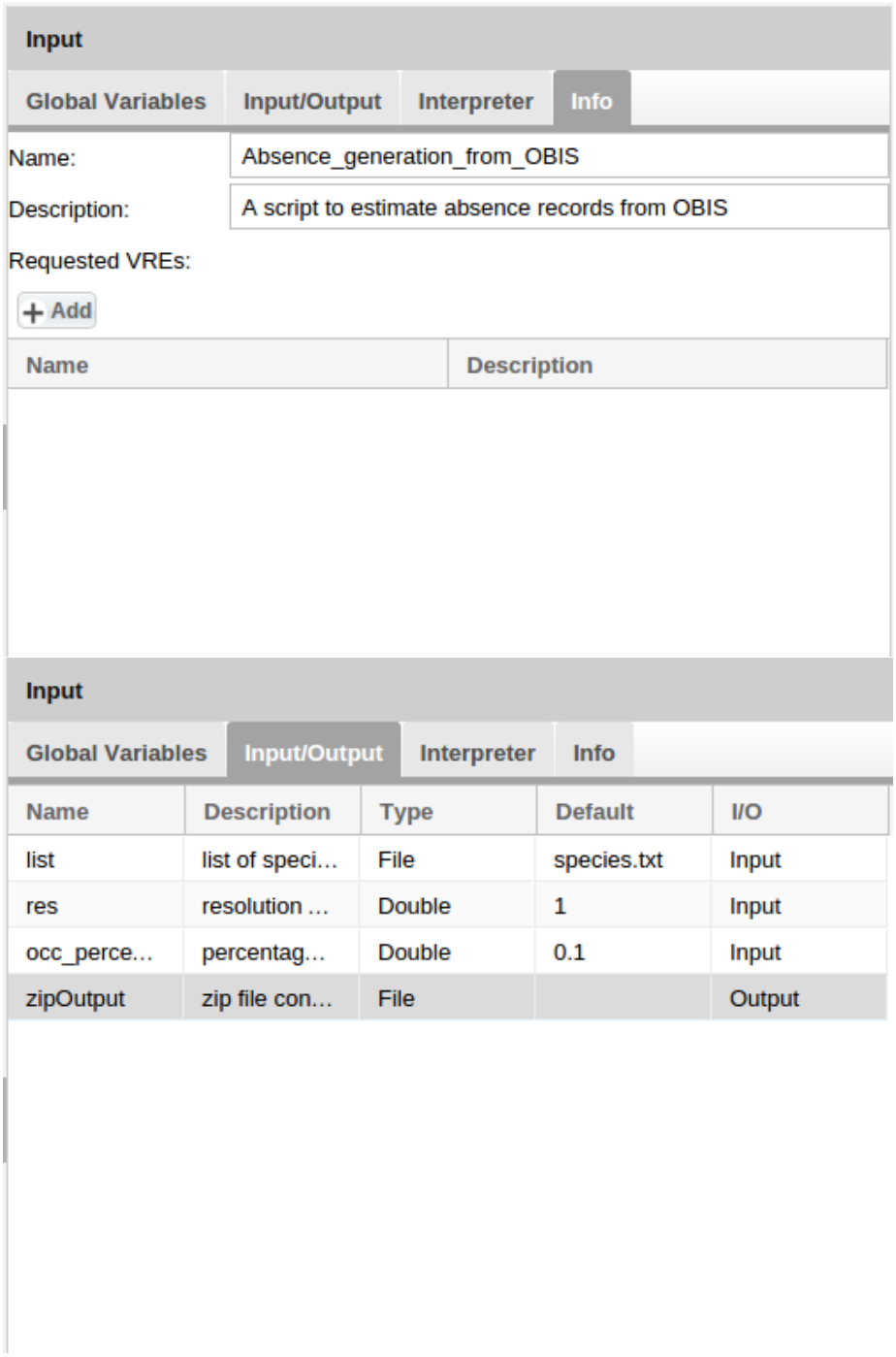

Figure 11: Example of information attached to an R Script.

Creating new Software

With the term "software" we mean a complete Java component that implements an as-a-Service version of the user provided scripts. The Java software will automatically download the scripts and the other required resources, will execute the script and will eventually return the result to the user. A Web GUI will be automatically created based on the I/O definitions in the corresponding SAI panel. The software can be created by pressing the Create button. The system will check the availability of at least one input and one output definition, the algorithm name and description and the interpreter version. A package will be created in the Target folder under the Compile folder. This will include Java code, metadata information and a JAR file to be deployed on the D4Science machines. Once the user is confident about the software and the interface, (s)he may pass to the publication phase.

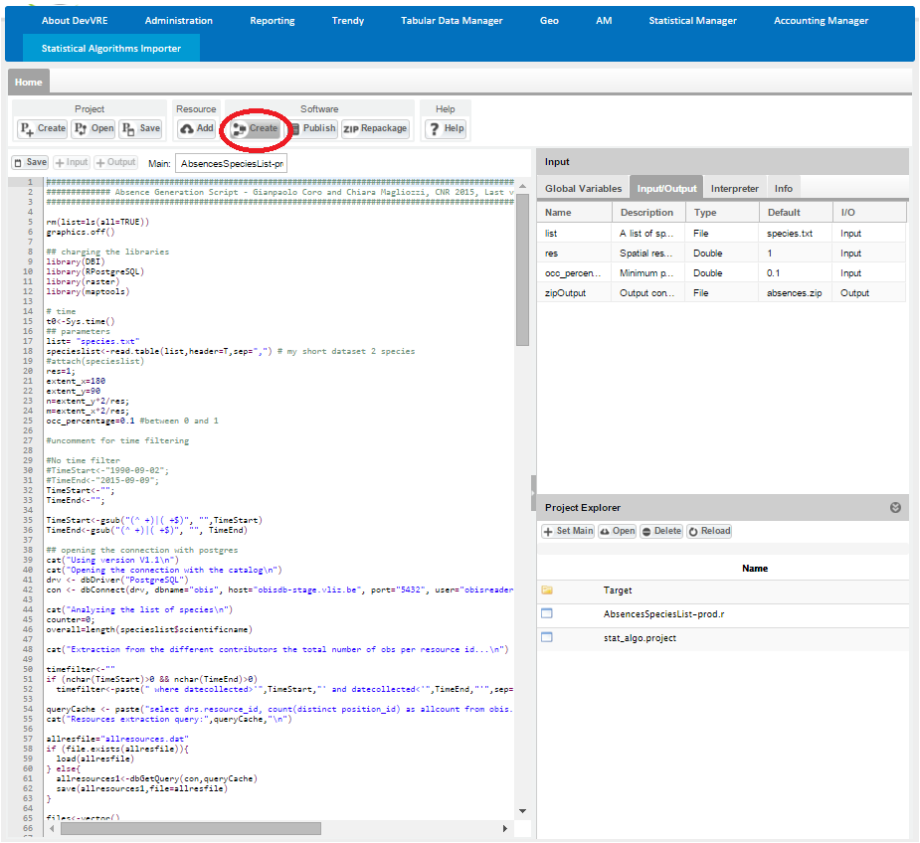

Figure 12: Button to create new Software.

Publishing an algorithm

This section explains how to publish algorithms created using the Statistical Algorithms Importer (SAI). Publishing an algorithm means making it available as-a-Service on the e-Infrastructure, for some Virtual Research Environment (VRE). In turn, this requires the algorithm to undergo a coherence check with respect to the rest of the infrastructure and be shipped on the D4Science machines. Only the JAR file will be deployed on the machines, leaving the scripts on the private Workspace of the user. This mechanism saves policy requirements of those users who do not want to share their code. Only the compiled Java code will be deployed on the D4Science machines (inside the JAR file), thus the e-Infrastructure staff will not be able to get the scripts code. The compiled Java file contains a link to a Workspace location, accessible only via authentication (owned only by the D4Science services). The service will download the scripts and resources on-the-fly, execute the main script and later delete everything.

The software publication can be run by pressing the Publish button. At this point, a link to the JAR file and to the algorithm metadata will be sent via e-mail to the e-Infrastructure managers. Once the algorithm has been deployed, the manager will alert the user via the e-Infrastructure messaging system.

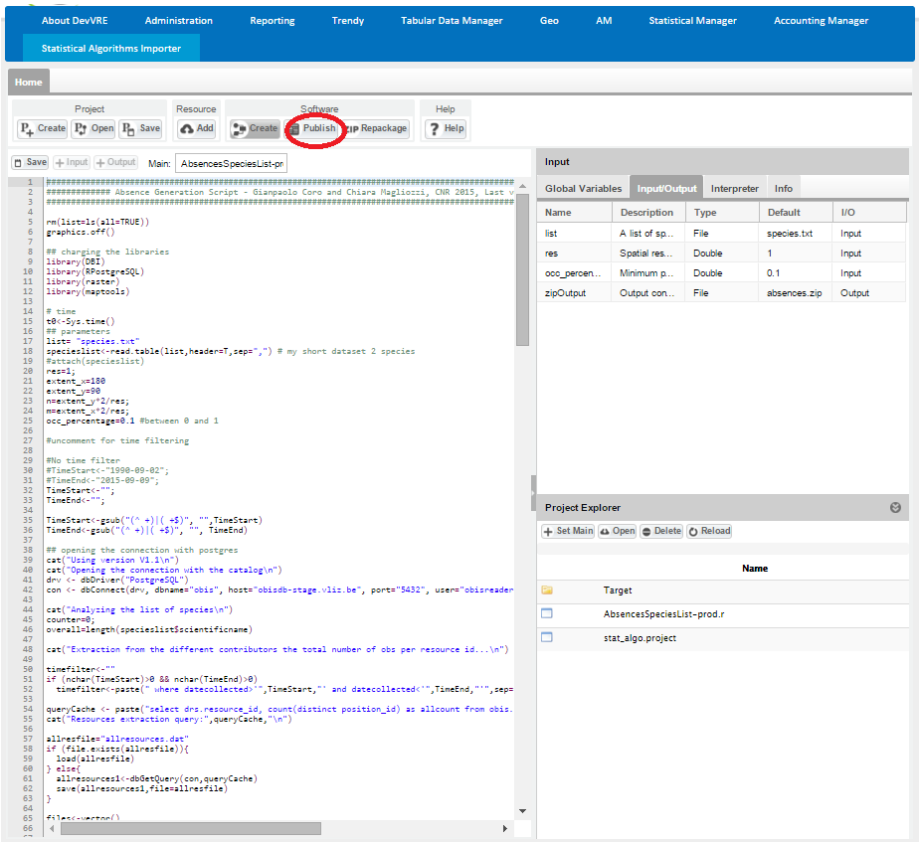

Figure 13: Button to publish a new algorithm.

Repackaging an algorithm

After the first deployment, the user can be independent on the e-Infrastructure managers in updating and redeploying the algorithm. The Repackaging function creates a new ZIP package containing the scripts and the required resources. This ZIP file will have the same Http link as the previously produced ZIP file of the first deployment phase. This means that the Repackaging feature updates the ZIP package that is currently downloaded by the JAR file on the D4Science machines. As a consequence, the scripts can be modified and the main code saved autonomously.

After saving the new scripts, by pressing the Repackage button the modification will be directly used by the D4Science without repeating the deployment phase.

Re-deployment is necessary if either the I/O or the required packages or the algorithm metadata are changed.

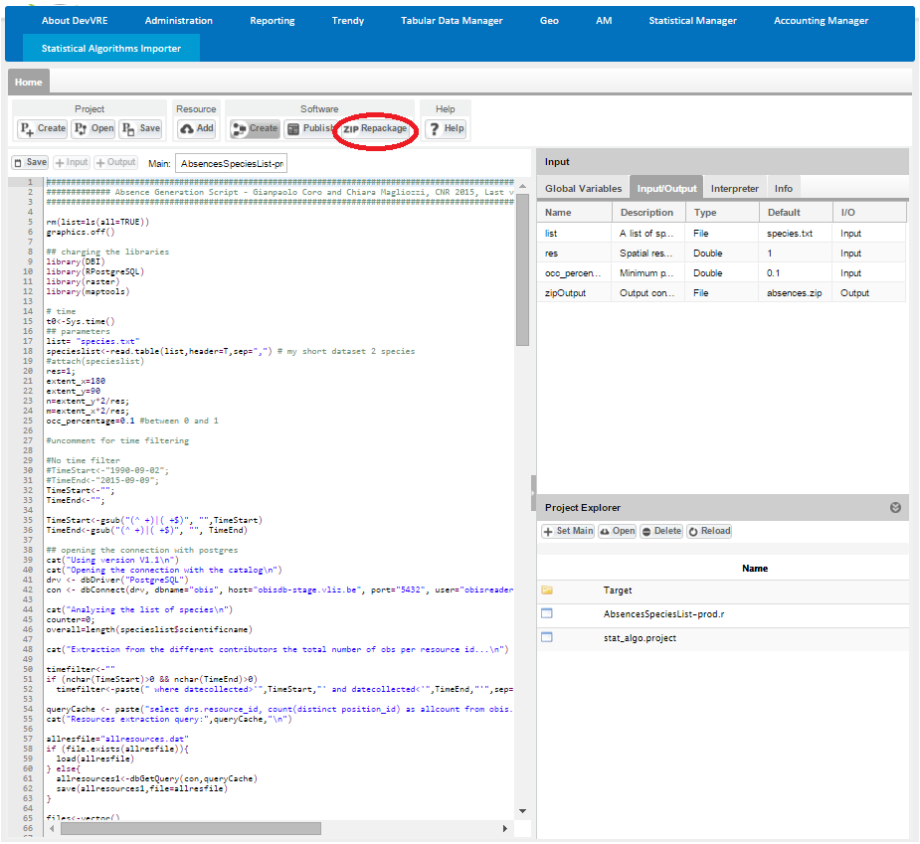

Figure 14: Button to repackaging an algorithm.

F.A.Q.

Please, read our best practices first, should you find issues:

https://wiki.gcube-system.org/gcube/Statistical_Algorithms_Importer:_FAQ

Demonstration

A demonstration video is available at this link:

http://data.d4science.org/SkhVR3AyTUNLaStCV2tNdUhsL2VIQ3AwMG1tTjVka3dHbWJQNStIS0N6Yz0